Computational Astrophysics I: Introduction and basic concepts

Helge Todt

Astrophysics
Institute of Physics and Astronomy
University of Potsdam

SoSe 2025, 30.6.2025

Aims and contents I

Recommended prerequisites:

- basic knowledge of programming, especially in C/C++ $\rightarrow$ e.g., "Tools for Astronomers"
- basic knowledge in astrophysics

How to get a certificate of attendance / 6 CP/LP/ECTS ($\hat{=}$ 4 semester periods per week):

- without mark, e.g., Master of Astrophysics, module PHY-765: Topics in Advanced Astrophysics (this module has in total 12 CP! and an oral exam at the end):

  $\boxed{\rightarrow \text{at least 1./3. of the points of the exercises}}$

Attention!

PULS is strict: It is absolutely necessary to enroll for this lecture until **10.05.2025**!

- with a mark (other Master courses):
  little programming project at the end of the semester

Please note that the focus for this course is on the exercises!

Aims and contents II

Specialization track "Computational Astrophysics"

the regulations for obtaining the computational astrophysics specialization *certificate* are as follows:

1. Computational Astrophysics I (4 SWS, SoSe) :
   Computational Astrophysics: Introduction
   Computational Astrophysics: Basic Concepts
2. Computational Astrophysics II (3 SWS, WiSe)
   Advanced Computational Astrophysics: Concepts and Applications
3. Seminar Computational Astrophysics (2 SWS)
   Advanced Computational Astrophysics: Seminar
4. a 4th course of the computational curriculum, e.g.,
   Computational Astrophysics: Advanced Programming (2 SWS)

Aims and contents III

Aims & Contents of CA I:

- enhance existing basic knowledge in programming (C/C++)
- brief introduction to Fortran $\rightarrow$ relatively common in astrophysics
- work on astrophysical topics which require computer modeling:
  - solving ordinary differential equations

    $\rightarrow$ from the two-body problem to N-body simulations
    $\rightarrow$ stellar structure, the Lane-Emden equation
  - solving equations: linear algebra, root finding, data fitting
  - data analysis

    $\rightarrow$ data analysis and simulations
  - simulation of physical processes

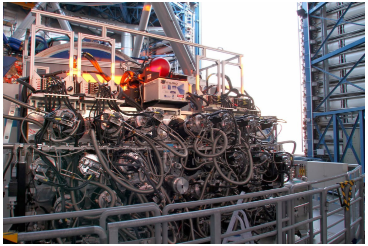
    $\rightarrow$ Monte-Carlo simulations and radiative transfer

+ introduction to parallelization (e.g., OpenMP)

Computational Astrophysics I

What are computers used for in astrophysics?

- control of instruments/telescopes/satellites:



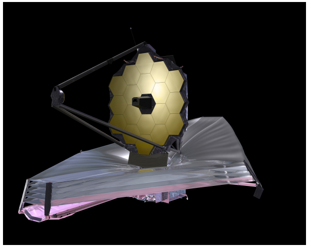
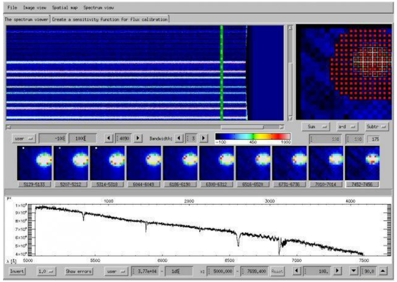
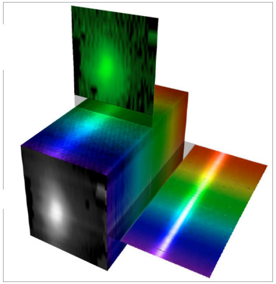


Figure: Multi Unit Spectroscopic Explorer (MUSE), Very Large Array (VLA), James Webb Space Telescope (JWST)

- data analysis / data reduction





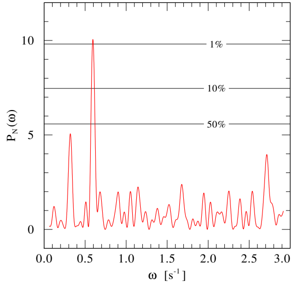
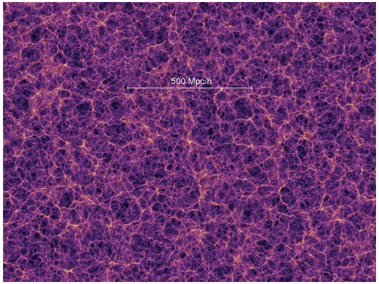
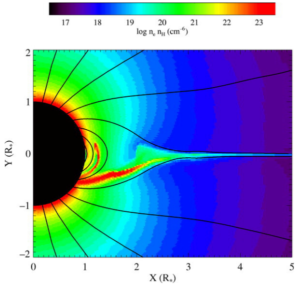


Figure: IDL, 3dCube / FITS, Fourier analysis

Computational Astrophysics III

- modeling / numerical simulations





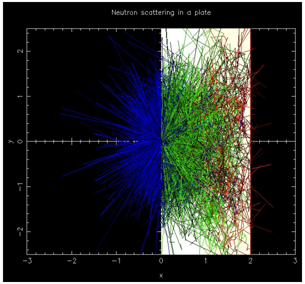


Figure: N-body simulation, hydrodynamics , Monte-Carlo

Conventions of used fonts

Meaning of the fonts / shapes

font/shape	meaning	example
`xvzf` (typewriter)	text to be entered literally (e.g., commands)	`man ls`
argument (italic)	place holder for own text	`file` *myfile*

The computer lab I

User accounts

useful for the lecture: your own account for this computer lab
(room 0.087 & 1.100)

Please, get your own account!
Sysad: Helge Todt, room 2.004

Guest account

$\rightarrow$ see left-hand side whiteboard
only valid per computer and in room 0.087

Attention: Unix/Linux is case sensitive!
Hint: You can choose the session type (e.g., Xfce, IceWM) at login screen.

The computer lab II

Security advice

As soon as you got your own account:

`passwd` Change the user password

(Enter the command in a terminal, Xterm, Konsole, or similar.)

Change your initial NIS password(!) to a strong password, use

- at least 9 characters, comprising of:
- capital AND lowercase letters, but not single words
- AND numbers
- AND special characters (Attention! Mind the keyboard layout!)

e.g., `$cPhT-25@comP2` or `tea4Pollen+Ahead`
But: prefer length over complexity!

The initial password expires after 14 days.

The computer lab III

Computers:

- 17 *NFS*^{1} mounted Linux computers (openSUSE 15.4/15.5),
  several Intel Core i7-2600K, i7-4770, i7-7700, i7-8700
  + 1 Xeon Gold 6152 *44*-core compute server
- home server ($\sim$`user`) always-on:
  - bell
  - mahler
  - weber

room 0.087:

- only for lectures
- Please, do not eat or drink in this room.

student's computer lab in room 1.100:

- open during the day
- b/w printer (500 pages / semester) and color printer (100 pages / semester)

NFS home directories

$$\begin{tikzcd} & \text{NFS server} \arrow[dl] \arrow[dr] & \\ \text{NFS client} & & \text{NFS client} \end{tikzcd}$$

NFS server: provides (home) directories (physical on disk)

NFS clients: mount NFS (home) directories in their root directory

As also other users might have their home directory on your computer

Never switch off the computers!

Linux

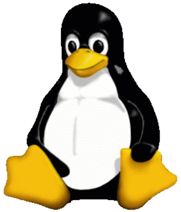
Operating system

Linux is a derivative of the operating system UNIX. It is a multi-user and multitasking operating system.

It was written in 1991 as a UNIX for PCs, now available for (almost) every platform, e.g., as Android or in Wireless routers and under permanent development.

Linux is ...

- for free
- open source (program code can also be modified)
- the combination of a *monolithic*[1] kernel and (GNU) software
- dominant in supercomputers (more than 90%)



[1] i.e., kernel contains also hardware driver

Graphical desktop environments

Important X-Window based environments under Linux: GNOME and KDE,
here: Xfce

Desktop environment (session type) can be chosen during local login, e.g., Xfce (nice) or IceWM (simple)

Desktop environment $\neq$ Linux

Desktop environment:	KDE	Xfce	GNOME	...
Linux distributions:	Ubuntu (Debian)		openSUSE	...

Important tools/programs

xterm or terminal: input of Linux shell commands, e.g., `cd`, `ls`

emacs or kate: editor for ASCII text files, e.g., `hello.cpp`

g++ or gfortran: gcc compilers, e.g., `g++ -o hello hello.cpp`

Shell and shell commands

Shell and shell commands I

Unix provides by the *shell* (command line) an extremely powerful tool. Within the shell Unix commands are executed.

Unix command syntax

`command [-option] [`*argument*`] <ENTER>`

Attention! Mind the blanks!

Open an xterm or similar terminal/console and enter following: (finish each line with `<ENTER>`):

```
echo hello
```

and

```
echo -n hello
```

What's the difference?

Shell and shell commands II

Tip 1:

Command history

By $\uparrow$ (arrow key up) you can repeat the last commands entered in the shell.

A list of the last commands can be shown via the command

```
history
```

Tip2:

Moreover, you can save typing by using the $\boxed{\texttt{TAB}}$ key, it completes commands or file names:

ech $\boxed{\texttt{TAB}}$

is completed to

```
echo
```

Copying text with the mouse

Tip 3:
Linux: Copy by Selection

mark the text with the mouse:

press left mouse button, keep it pressed

move mouse cursor until end of the region you want to mark

$\rightarrow$ marked region will be highlighted

marked text was copied to the *clipboard*

paste the copied text:

move cursor to the intended position

press the middle(!) mouse button (or wheel)

$\rightarrow$ the previously copied text was inserted

Directories

The filesystem tree

```
/                                        → root of the FS tree
|
|-- /home/                               → Home directories
|      |
|      |-- /home/weber/                  → Homes on weber
|                |
|                |--/home/weber/htodt/   → Helge's home
|
|
|-- /etc/                                → et cetera (config)
|
|
|-- /dev/                                → devices
```

Navigation through directories I

`pwd`	shows the current directory path (absolute) e.g., `/home/weber/htodt`
`cd` *name*	change to directory *name*
`.`	means the current directory
`..`	the parent directory, e.g., $\boxed{\texttt{cd ..}}$
`/`	root of the FS tree
`~`	the home directory, e.g., $\boxed{\texttt{cd} \sim}$ or just $\boxed{\texttt{cd}}$
`~`*user*	the home directory of *user*

Navigation through directories II

`mkdir` *name*	create directory *name*
`rmdir` *name*	remove directory *name*
`ls`	show (list) the content of the directory

Navigation through directories III

`ls`	show the content of the current directory
`ls -a`	also show hidden files (starting with a `.`)
`ls -l`	show the file attributes, owner, creation time

File attributes

```
drwxr-xr-x  2 htodt users    4096 14. Oct 13:35 Documents
```

`d` = directory	`r` = readable
`w` = writeable	`x` = executable
`htodt` = owner	`users` = group
`4096` = size in byte	`14. Oct 13:35` = creation time

`Documents` = name of the file (here: of the directory)

Hint: `ls -lc` → time of last modification ; `ls -lu` → time of last access

Navigation through directories IV

`man ls`	Manual pages (help for the command ls)
`info ls`	Info pages (alternative help for the command ls)
`ls --help`	Help for the command ls
`ls --help | less`	if more than one screen page

man page navigation – also `less`, `more`

`q`	quit		
`<SPACE>`	next page	`b`	previous page
`/`	forward search	`?`	backward search
`n`	next occurrence	`N`	previous occurrence
`>`	jump to the end	`<`	jump to the beginning

Working with a text editor

→ to create pure ASCII files (e.g., as input for g++)

$\boxed{\texttt{emacs } \textit{file} \texttt{ \&}}$

Starting programs in background:

The ampersand `&` at the end of a command let the command run in the background (bg) of the shell.

Hence, the input line of the shell can be still used.

If forgotten: `<CTRL>+z` followed by `bg <ENTER>`.

The text editor emacs

`emacs`	available on almost every system (must be installed), window- or terminal-based (`emacs -nw`)

`<STRG> + x <STRG> + c`	close (quit/exit)
`<STRG> + x <STRG> + s`	save
`<STRG> + k`	kill (cut, from cursor to end of line)
`<STRG> + y`	yank (paste)
`<STRG> + <SPACE>`	mark
`<STRG> + w`	cut marked region
`<ESC> w`	copy marked region
`<STRG> + a`	go to beginning of line
`<STRG> + e`	go to end of line

Files

Files in Unix

Remark: In Linux almost everything is a file (also directories and devices, see `ls -l /dev/`).

`mv` *source target*	move (rename) files
`cp` *source target*	copy files
`rm` *file*	remove file
`rm -rf` *directory*	remove directory

Compress files and directories

`tar` *action archive files* — use *action* on *archive*

tar actions

`c`	create archive from file/directory
`x`	extract archive
`v`	show executed actions (verbose)
`z`	zipp archive
`t`	show content of archive
`f`	archive is a file (default: tape device)

Example: Untar a tarball

```
tar xvzf muCommander.tar.gz
```

Connecting to other hosts (computers)

Login on other hosts (computers) I

`hostname`

this command shows the name of the host you're currently logged in

Login on other hosts (computers) II

Connection to another host (*remote host*) under Linux/Unix with the *secure shell*, within the same domain (e.g., within the computer lab cluster)

```
ssh host name
```

After successful *login*, in the same terminal/window a shell is shown that runs on the remote host.

Login on other hosts (computers) III

The SecureSHell

Client-server system for establishing a secure connection (encrypted), login on the remote host (remote host = SSH server)

if SSH client and SSH server support X11:

```
ssh host name -Y
```

allows the SSH server to open a graphical window (e.g., for `evince` or `kate`) on the SSH client

Besides the interactive use of the SSH one can also just let a program run on the remote host via ssh:

```
ssh hostname "ls -l"
```

The connection will be automatically closed after program/command is finished.

Login on other hosts (computers) IV

Login from outside (e.g., from home):

```
ssh username@bell.stud.physik.uni-potsdam.de
```

There are SSH clients for Windows that are for free, e.g., PuTTY. Moreover, MobaXterm, Xming, X2Go (requires also installation on the server) or with help of the Windows Subsystem for Linux (requires the installation of Linux distribution) it is also possible to perform a graphical SSH login from Windows to Unix/Linux.

Hint:

With help of the graphical login you can, e.g., use Mathematica on the computer lab cluster at home.

Login on other hosts (computers) V

For login without password:

1. run `ssh-keygen` on the client, answer all question just with `<ENTER>`
2. add the resulting `~/.ssh/id_rsa.pub` from the client host to `~/.ssh/authorized_keys` on the *remote host*

Remote copy I

With help of the SSH protocol it is also possible to transfer files between different computers:

```
scp document.txt username@bell.stud.physik.uni-potsdam.de:
```

secure copy to the remote host

```
scp username@bell.stud.physik.uni-potsdam.de:document.txt .
```

secure copy from the remote host (mind the dot!)

After the colon $\boxed{:}$ is a path given, either absolute or relative to the home directory

Remote copy II

To copy only files that have been modified (comparison of source and target):

`rsync -rtvz` *username@host.domain:directory/* `.`

secure copy from the remote host, only modified files

some useful options:	`-r`	recursive: also directories
	`-t`	time: keep time stamps of transferred files
	`-v`	verbose: print information during transfer
	`-z`	zip: compressed file transfer (faster for slow connections)
	`-c`	checksum: use check sums (instead of time stamps) for comparison

Copy files via `konqueror` from other hosts

`konqueror` allows to show directories of remote hosts with help of the `fish` protocol. So, enter in the address bar, e.g.,

```
fish://user@weber.stud.physik.uni-potsdam.de
```

More Unix commands

`df -h`	shows free space on hard drive
`du -hs`	shows total size of current directory
`ps ux`	shows running processes of the current user
`top`	shows load and running processes (interactive)
`htop`	
`kill -9` *PID*	"kills" the process with the given process ID (PID)

Computing jobs on the stud cluster I

The ressources of the stud cluster (CPUs, RAM, disk space) can be used by all users, the users share these ressources.

$\rightarrow$ Therefore, please, think of other users:

- Log out, when leaving the computer, do not just lock the screen. Never switch off/shutdown the computer.
- Have an eye on the disk usage of you home directory (see below), delete regularly data that are no longer required.
- If you intend to start a job that will run a bit longer, then *nice* this job (see below).

Computing jobs on the stud cluster II

Nice and renice jobs/processes

The command `top` shows the priority and the consumption of ressources of running processes:

```
  PID USER         PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
26054 htodt        39  19 1103308 179636   8648 R 100,0 0,552   0:28.93 53_steal.exe
14763 htodt        20   0 1749032 358808  74328 S 3,322 1,102  12:23.34 Xvnc
```

Jobs that might run longer than just a few minutes and put some load on the CPU, should be niced when started, e.g.:

```
nice -19 ./53_steal.exe
```

$\rightarrow$ The priority is decreased from 20 (default) by 19 to 39 (higher values mean actually lower priority).

A job can also be niced when already running with help of `renice` and the process ID (PID), e.g.:

```
renice +19 26054
```

The program `top` in its interactive mode can also renice a process by pressing the key $\boxed{\texttt{r}}$.

Computing jobs on the stud cluster III

Checking disk space

The command `df -hT` shows an overview of the available and used disk space on the current host:

```
weber/htodt> df -hT
Filesystem          Type       Size     Used  Avail   Use% Mounted on
/dev/sdb1           xfs        3,7T     981G   2,7T    27% /home
bell:/home/bell     nfs4       1,8T     1,1T   702G    60% /nfs/bell
```

Moreover, the command `du -hs ~` displays the disk usage of your own home directory.

```
weber/htodt> du -hs ~
30G /home/weber/htodt
```

Instead of the tilde ~ you can also use other (own) subdirectories as an argument, to check their disk usage.

→ If a disk shows a use of 100%, you cannot longer write on this disk or copy data to it.

Computing jobs on the stud cluster IV

Show CPU performance

The type, its parameters, and its current clock rate(s) of the installed CPU are shown in the file `/proc/cpuinfo`, which you can read with help of `cat` or `less` (info is duplicated for each thread/logical core):

```
weber/htodt>cat /proc/cpuinfo
model name        : Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
cpu MHz           : 3491.946
```

The number of logical cores/threads (= either number of physical cores or number of physical cores $\times 2$ for Hyperthreading) can be also seen in the program `top`, if you press the key $\boxed{1}$.

Show available RAM

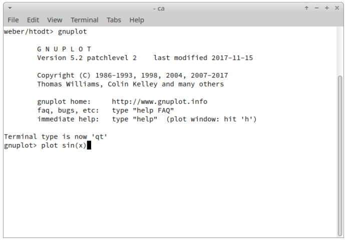
The command `free -h` shows the amount of free/available RAM:

```
weber/htodt> free -h
              total        used        free      shared  buff/cache   available
Mem:          187Gi        66Gi        51Gi       3,2Gi        69Gi       116Gi
```

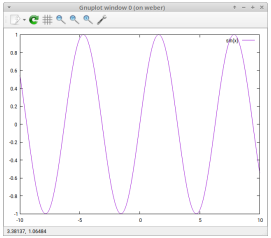
Brief introduction to gnuplot

Starting gnuplot

- gnuplot is available for almost every platform (operating system): Linux, MacOS X, Windows, . . .
- download, e.g, from $\boxed{\texttt{http://gnuplot.info/}}$
- under Linux: start interactive session in termial via
  `gnuplot`



```
weber/htodt> gnuplot

        G N U P L O T
        Version 5.2 patchlevel 2    last modified 2017-11-15

        Copyright (C) 1986-1993, 1998, 2004, 2007-2017
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> plot sin(x)
```



- quit gnuplot by command
  `exit`

Plotting functions

- gnuplot can plot basic functions (independent variable / dummy variable is x) and combinations of them, default plot symbol for functions: solid line
- examples
  - `plot sin(x)`
  - `plot x**3 + 0.5*sqrt(2)`
- plotting more than one function by using comma separated list:
  `plot sin(x), cos(1/x), tanh(x+2)`
- plotting only over a specific x-range (also for fitting):
  `plot [-2.5:+3] cos(x)`
- plotting only over a specific y-range:
  `plot [] [-1:+1] sin(x)`
- plotting only over a specific x- and y-range:
  `plot [-2.5:+3] [-1:1] cos(2*x)`

Labels

- x- and y-axis labels:
```
set xlabel "d in pc"
set ylabel "t in Ga"
```
- key (legend): is automatically generated, can be written by option `title`:
```
plot "data.txt" title "observation (1998)" \
, f(x) t "model 17-04"
```

→ requires execution of previous plot command or just type `replot`

Plotting data

`gnuplot` plots data from files in ascii table format, i.e.

```
# this is a comment
  4.5 91 -0.5
  5.6 70  0.8
  19  200 1.1
```

- Columns are separated by blanks. Can be changed before plotting, e.g.,
  `set datafile separator ","` # (comma seprated)
  `set datafile separator "\t"` # (separated by tabs)
- `plot "file.txt"`
  → default: plots 2nd column over 1st column
- `plot 'filexyz.txt' using ($2):($3)`
  → plots 3rd column over 2nd column
- `plot 'data.txt' u (log10($1)):(log10($2))`
  → plots the decadic logarithm of the data in columns 1 and 2 (double-logarithmic plot)

Creating a histogram I

Histogram

is the **graphical** representation of the frequency distribution of some quantity x.

- $\rightarrow$ requires the division of the data (quantity) into **bins** of a **width** Δx (can be constant or variable)
- $\rightarrow$ representation usually by rectangles of width Δx and height corresponding to frequency of occurrence
- $\rightarrow$ can be used to estimate the **probability density function** $p(x)$ of a continuous random variable x

Creating a histogram II

Example: normal distribution / Gaussian distribution

A data set of 10^3 random variables drawn from a Gaussian distribution with

$$N(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where the mean value was set to $\mu = 0$ and the variance to $\sigma = 1$, hence the distribution is $N(x,0,1) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}\cdot x^2) \rightarrow$ so-called standard normal distribution in some file `gauss.dat` (embedded).

10-DM banknote with a Gaussian distribution

Creating a histogram III

Graphical representation with gnuplot:

1. Define the (constant) width of the bins ("bin width", Δx):

```
gnuplot> bw=0.2
gnuplot> set boxwidth bw
```

   ... and a so-called "binning" function:

```
gnuplot> bin(x,s)=s*ceil(x/s)
```

   The function ceil(x) rounds **up** the value of x to next larger integer

2. The number of data points (for normalization):

```
gnuplot> N=1000
```

   one can also use

```
gnuplot> stats "gauss.dat" ; N = STATS_records
```

Creating a histogram IV

3. The histogram is then created via:

```
gnuplot> plot "gauss.dat" u (bin($1,bw)-0.5*bw):(1./(N*bw)) \
         smooth frequency with boxes lc rgb "blue"
```

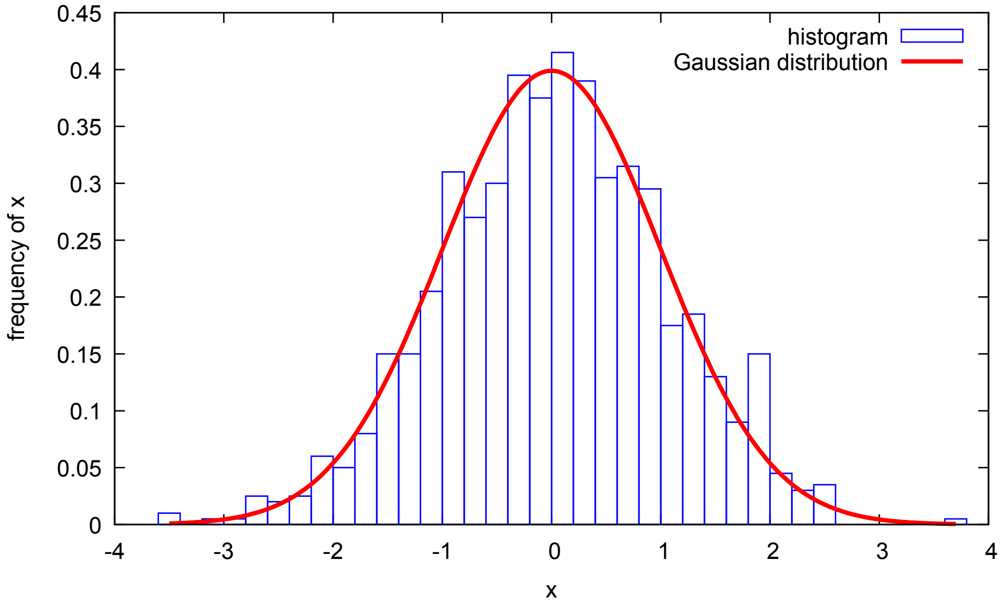
4. Plot together with the underlying probabilty density function

```
gnuplot> gauss(x) = 1./sqrt(2*pi) * exp(-0.5*x**2)
```

in the same diagram:

```
gnuplot> replot gauss(x) with lines linewidth 3 linecolor rgb "red"
```

Creating a histogram V



Fitting data

With help of the Levenberg-Marquardt algorithm `gnuplot` can fit any function with free parameters to data:

1. define function:
   ```
   f(x) = a * x + b
   ```
2. fitting examples:
   ```
   fit f(x) "data.txt" via a, b
   fit f(x) "data.txt" u (log10($1)):(log10($2)) via a, b
   ```
3. plotting data and function:
   ```
   plot "data.txt", f(x)
   ```

If the fit should be done only for a specific x-range:

```
fit [100:300] f(x) "data.txt" via a, b
```

Creating PDFs for output

gnuplot supports many different output formats (see → `help terminal`)

1. `set terminal pdf enhanced color`
   → sets terminal (output *format*) to colorized pdf with special characters
2. `set output "myplot.pdf"`
   → name of the file for output (don't forget it!)
3. `plot "data.txt", f(x)`
   or: `replot`
4. either: `set term qt` (resetting terminal to previous output format)
   or: `quit`
   → this assures that the plot is *written* to the file (otherwise: empty or incomplete PDF file)

Special characters

If output is written to PDF or PS file, via option `enhanced`:

Input	Output in PDF/PS
`T_0`	T_0 (subscript)
`e^{-x}`	e^{-x} (superscript)
`{/Symbol Qp}`	$\Theta\pi$

Scripts in gnuplot

In addition to the interactive mode, `gnuplot` supports also non-interactive script mode

- write all instructions into an ASCII text file (e.g., “myplot.gplt”)
  comments begin with a # (like in makefile and shell)
  line continuation via backslash \
- execute `gnuplot` script from shell:
  `gnuplot myplot.gplt`

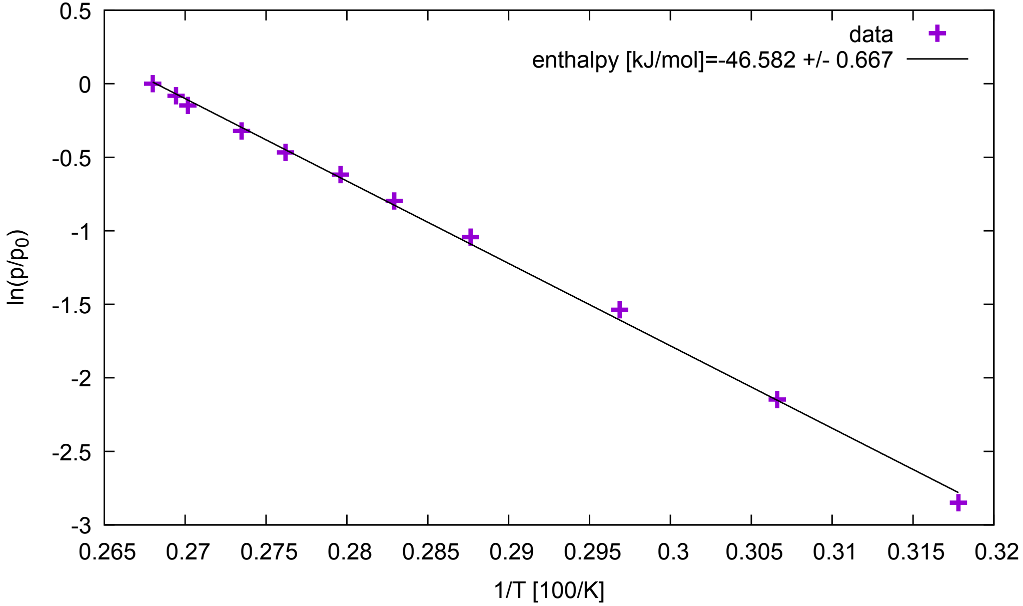
→ useful for automated PDF creation
→ easy re-use of formatting and plot instructions (labels, sizes, . . .)

Example for fitting and pdf output I

embedded file : enthalpie.dat

```
set terminal pdf enhanced color
set xlabel "1/T [100/K]"
set ylabel "ln(p/p_0)"
ln_p(x) = b + a*x
set fit errorvariables
R=8.314
p_0=1.019
fit [*:*] ln_p(x) "enthalpie.dat" \
using (1./(($2)+273.15)):(log((1.019+($1))/1.019)) via a,b
set output "enthalpy.pdf"
plot "enthalpie.dat" \
using (1e2/(($2)+273.15)):(log((p_0+($1))/p_0)) \
with points ps 1 linewidth 3 title "data" \
 , ln_p(1e-2*x) with lines linecolor "black" \
t sprintf("enthalpy [kJ/mol]=%5.3f +/- %5.3f",a*R*1e-3,R*a_err*1e-3)
```

Example for fitting and pdf output II



C/C++ Programming

Programming languages I

One can, e.g., distinguish:

scripting languages

- bash, csh $\rightarrow$ Unix shell
- Perl, Python
- IRAF, IDL, Midas $\rightarrow$ especially for data reduction in astrophysics

compiler-level languages

- C/C++ $\rightarrow$ very common, therefore our favorite language
- Fortran $\rightarrow$ very common in astrophysics, especially in radiative transfer

Programming languages II

	scripting language	compiler-level language
examples	shell (bash, tcsh), Perl, Mathematica, MATLAB, ...	C/C++, Fortran, Pascal, ...
source code	directly executable	translated to machine code, e.g., `0x90` $\rightarrow$ no operation (NOP)
runtime behavior	interpreter runs as a program $\rightarrow$ full control over execution $\rightarrow$ error messages, argument testing	error handling difficult $\rightarrow$ task of the programmer, often only crash
speed	usually slow $\rightarrow$ analysis tools	very fast by optimization $\rightarrow$ simulations, number crunching

$\rightarrow$ moreover, also bytecode compiler (JAVA) for virtual machine,
Just-in-time (JIT) compiler (JavaScript, Perl)

C/C++ I

- C is a *procedural* (imperative) language
- C++ is an *object oriented* extension of C with the same syntax
- C++ is because of its additional structures (template, class) $\gg$ C

Basic structure of a C++ program

```
#include <iostream>
using namespace std ;
int main () {
    instructions of the program ;
    // comment
    return 0 ;
}
```

every instruction must be finished with a $\boxed{;}$ (semicolon) !

C/C++ II

Compiling a C++ program:

source file

`.cpp, .C`

$\Downarrow$

compiler + linker

`.o, .so, .a`

$\Downarrow$

executable program

`a.out, program`

C/C++ III

Command for compiling + linking:

```
g++ -o program program.cpp
```

(GNU compiler for C++)

- only compiling, do not link:

  ```
  g++ -c program.cpp
  ```

  creates *program*.o (object file, not executable)
- option -o *name* defines a name for a file that contains the executable program, otherwise program file is called: a.out

  the name of the executable program can be arbitrarily chosen

C/C++ IV

Task 2.1 Compiling

Use a text editor to create a file `nothing.cpp`, which contains *only* the empty function `int main(){}`, compile it and execute the resulting program.

Simple program for output on screen I

Example: C++ output via streams

```
#include <iostream>

using namespace ::std ;

int main () {

    cout << endl << "Hello world!" << endl ;

    return 0 ; // all correct

}
```

Simple program for output on screen II

- `<iostream>` . . . is a C++ library (input/output)
- `main()` . . . program (function)
- `return 0` . . . returns the return value 0 to main (all ok)
- source code can be freely formatted, i.e., it can contain an arbitrary number of spaces and empty lines (white space) → useful for visual structuring
- comments are started with `//` - everything after it (in the same line) is ignored, C has only `/* comment */` for comment blocks
- `cout` . . . output on screen/terminal (C++)
- `<<` . . . output/concatenate operator (C++)
- string `"Hello world!"` must be set in quotation marks
- `endl` . . . manipulator: new line and stream flush (C++)
- a block several instructions which are hold together by curly braces

Simple program for output on screen III

Task 2.2 Hello world!

Use a text editor to create a file $\boxed{\texttt{hello.cpp}}$, which prints out "Hello World!" in the terminal, compile it and execute the resulting program.

Functions I

C/C++ is a procedural language
The procedures of C/C++ are *functions*.

- Main program: function with specific name `main(){}`
- every function has a type (for return), e.g.: `int main (){}`
- functions can get arguments by call, e.g.:
  `int main (int argc, char *argv[]){}`
- functions must be *declared before* they can be called in the main program,
  e.g., `void swap(int &a, int &b) ;`
  or included via a header file:
  `#include <cmath>`
- within the curly braces `{ }`, the so-called function body, is the *definition* of the function (what shall be done how), e.g.:
  `int main () { return 0 ; }`

Functions II

Example

```
#include <iostream>
using namespace std ;

float cube(float x) ;

int main() {
  float x = 4. ;
  cout  << "The cube of x is: " << cube(x) << endl ;
  return 0 ;
}

float cube(float x) {
  return x * x * x ;
}
```

Functions III

Task 2.3 Calling a function

Use a text editor to create a file $\boxed{\texttt{cubemain.cpp}}$, which contains the source code from the previous slide (copy & paste).

1. Compile it and execute the resulting program.
2. Modify the source code so that the program reads in a number from the user with the help of `cin`:

```
float x  ;
cout << "type in a number: " ;
cin >> x ;
```

Functions IV

`inline` functions

- usually for compiled program: functions as code sections with own address; calling a function = jump to this address, pass arguments → overhead for argument passing, address for jumping back from function (`return`) must be stored:

Example

```
nm cubemain | grep " T "
0000000004008b7 T main
0000000004007de T _start
00000000040090d T _Z4cubef
```

→ calling many *small* functions is expensive

Functions V

- solution: use keyword `inline` → compiler replaces function *call* by function *code*, each time the function is called → increases size of compiled code

Example

```
      inline  float cube(float x) {
         return x * x * x ; }
```

→ definition must be in the same source text file where function is called
→ not all functions can be inlined by the compiler

- methods *defined* in class headers are automatically `inline`

The `cmath`-library I

In C/C++ only basic mathematical operations `+`, `-`, `*`, `/`, `%` available.

By including the `cmath`-library in the beginning:

```
#include <cmath>
```

many mathematical functions become available:

`cos();`	`sin();`	`tan();`
`asin();`	`atan();`	`acos();`
`cosh();`	`sinh();`	`tanh();`
`exp();`	`fabs();`	`abs();`

`log();`	... natural logarithm (base e)
`log10();`	... decadic logarithm (base 10)
`pow(x,y);`	... x^y †
`sqrt();`	$\sqrt{}$

The cmath-library II

Moreover, there are also predefined mathematical constants:

`M_E`	...	e
`M_PI`	...	π
`M_PI_2`	...	$\pi/2$
`M_PI_4`	...	$\pi/4$
`M_2_PI`	...	$2/\pi$
`M_SQRT2`	...	$+\sqrt{2}$

Variables

- A variable is a piece of memory.
- in C/C++ data types are explicit and static

We distinguish regarding visibility ("scope"):

- global variables → declared outside of any function, before `main`
- local variables → declared in a function or in a block `{ }` , only there visible

...regarding data types → intrinsic data types:

- `int` → integer, e.g., `int n = 3 ;`
- `float` → floats (floating point numbers),
  e.g., `float x = 3.14, y = 1.2E-4 ;`
- `char` → characters, e.g., `char a_character ;`
- `bool` → logical (boolean) variables, e.g., `bool btest = true ;`

Integer data types I

Integer numbers are represented *exactly* in the memory with help of the binary number system (base 2), e.g.

$$13 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \mathrel{\widehat{=}} \boxed{1}\boxed{1}\boxed{0}\boxed{1}\,[1] \quad \text{(binary)}$$

In the assignment

```
a = 3
```

3 is an integer literal (literal constant). Its bit pattern ($3 = 1 \cdot 2^0 + 1 \cdot 2^1 \mathrel{\widehat{=}} \boxed{1}\boxed{1}$) is inserted at the corresponding positions by the *compiler*.

[1]doesn't correspond necessarily to the sequential order used by the computer → "Little Endian": store least significant bit first, so actually: 1011

Integer data types II

on 64-bit systems

`int`	compiler reserves 32 bit (= 4 byte) of memory "1 bit for sign" (see below) and $2^{31} = 2\,147\,483\,648$ values (incl. 0): $\rightarrow$ range: `int` $= -2\,147\,483\,648 \ldots +2\,147\,483\,647$
`unsigned int`	32 bit, no bit for sign $\rightarrow 2^{32}$ values (incl. 0) `unsigned int` $= 0 \ldots 4\,294\,967\,295$
`long`	on 64 bit systems: 64 bit (= 8 byte), "1 bit for sign": $-9.2 \times 10^{18} \ldots 9.2 \times 10^{18}$ (quintillions)
`unsigned long`	64 bit without sign: $0 \ldots 1.8 \times 10^{19}$

and also: `char` (1 byte), smallest addressable (!); `short` (2 byte) ; `long long` (8 bytes)

Integer data types III

Two's complement

Table: Representation: unsigned value (0s), value and sign (sig), two's complement (2'S) for a nibble (1/2 byte)

binary	unsigned	signed	2'S
0000	0	0	0
0001	1	1	1
	...		
0111	7	7	7
1000	8	-0	-8
1001	9	-1	-7
	...		
1111	15	-7	-1

Disadvantages of representation as value and sign:
$\exists$ 0 *and* -0; Which bit is sign? ($\rightarrow$ const number of digits, fill up with 0s);

Advantage of 2'S:
negative numbers[†] always with highest bit=1
$\rightarrow$ cf. $+1 + -1$ bitwise for value & sign vs. 2'S

Binary arithmetic: $1 + 1 = 2$

$$\begin{array}{rr} & 0001 \\ + & 0001 \\ \hline = & 0010 \end{array}$$

[†]How to write negative numbers in 2'S? $\rightarrow$ start with corresponding positive number, invert all bits, and add 1 ignoring any overflow

Floating point data types I

Floating point numbers are an approximate representation of real numbers.
Floating point numbers can be declared via, e.g.,:

```
  float radius, pi, euler, x, y ;
  double  radius, z ;
```

Valid assignments are, e.g.,:

```
x = 3.0 ;
y = 1.1E-3 ;
z = x / y ;
```

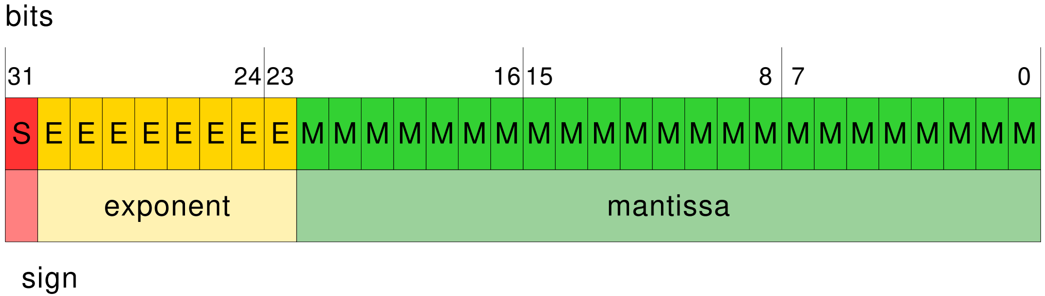
Floating point data types II

- representation (normalization) of floating point numbers are described by standard IEEE 754 :

$$x = s \cdot m \cdot b^e \qquad (1)$$

with base $b = 2$ (IBM Power6: also $b = 10$), sign s, and normalized significand (mantissa) m, bias

- So for 32 Bit (Little Endian[†]), 8 bit exponent, 23 bit mantissa:



([†] least significant bit at start address, read each part: $\rightarrow$)

Floating point data types III

- mantissa is *normalized* to the form (e.g.)

$$1.0100100 \times 2^4$$

  i.e. with a 1 before the decimal point. This 1 is not stored, so $m = 1.f$

  Moreover, a bias (127 for 32 bit, 1023 for 64 bit) is added to the exponent (results in non-negative integer)

Example: Conversion of a decimal number to IEEE-32-Bit

172.625	base 10
10101100.101×2^0	base 2 $(0.625 = 1 \cdot 1/2 + 0 \cdot 1/4 + 1 \cdot 1/8)$
1.0101100101×2^7	base 2 normalized

add bias of 127 to exponent $= 134 = 1 \cdot 2^7 + \ldots + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$

0 10000110 01011001010000000000000

Floating point data types IV

- single precision (32 bit) `float`: exponent 8 bit, significand 23 bit

$$-126 \leq e \leq 127 \text{ (basis 2)}$$

$$\rightarrow \approx 10^{-45} \ldots 10^{38}$$

$$\text{digits: 7-8 } (= \log 2^{23+1} = 24 \log 2)$$

- for 64 bit (double precision) – `double`: exponent 11 bit, significand 52 bit

$$-1022 \leq e \leq 1023 \text{ (basis 2)}$$

$$\rightarrow \approx 10^{-324} \ldots 10^{308}$$

$$\text{digits: 15-16 } (= \log 2^{52+1})$$

Floating point data types V

some real numbers cannot be presented exactly in the binary numeral system, e.g.:

$$0.1 \approx 1.10011001100110011001101 \times 2^{-4} \tag{2}$$

$\rightarrow$ cf. 1/3 in decimal: all fractions with denominator not product of prime factors (2,5) of the base 10, e.g., 1/3, 1/6, ...
In binary numeral system only one prime factor: 2

Warning

Do not compare two floating point numbers blindly for equality (e.g., `0.362 * 100.0 == 36.2`), but rather use an accuracy limit:
`abs( x - y ) <= eps`, better: relative error
`abs(1-y/x) <= eps`

Floating point data types VI

Floating point arithmetic

Subtraction of floating point numbers

consider $1.000 \times 2^5 - 1.001 \times 2^1$ (only 3 bit mantissa)
$\rightarrow$ *bitwise subtraction*, requires same exponent

$$
\begin{array}{rll}
 & 1.000\,0000 & \times 2^5 \\
- & 0.000\,1001 & \times 2^5 \\
\hline
 & 0.111\,0111 & \times 2^5 \text{ infinite precision} \\
 & 1.110\,111 & \times 2^4 \text{ shifted left to normalize} \\
 & 1.111 & \times 2^4 \text{ rounded up, as last digits} > 1/2 \text{ ULP}^\dagger
\end{array}
$$

†unit in the last place = spacing between subsequent floating point numbers

Floating point data types VII

Properties of floating point arithmetic (limited precision):

- loss of significance / catastrophic cancellation: occurs for subtraction of almost equal numbers

Example for loss of significance

$\pi - 3.141 = 3.14159265\ldots - 3.141$ with 4-digit mantissa;
maybe expected: $= 0.00059265\ldots \approx 5.927 \times 10^{-4}$;
in fact: 1.0000×10^{-3}, because π is already rounded to 3.142

- absorption (numbers of different order of magnitude): addition of subtraction of a very small number does not change the larger number

Example for absorption

for 4-digit mantissa: $100 + 0.001 = 100$:
$1.000 \times 10^2 + 1.000 \times 10^{-3} = 1.000 \times 10^2 + 0.000\,01 \times 10^2 = 1.000 \times 10^2 + 0.000 \times 10^2 = 1.000 \times 10^2$, same for subtraction

Floating point data types VIII

- distributive and associative law usually not fulfilled, i.e. in general

$$(x + y) + z \neq x + (y + z) \tag{3}$$
$$(x \cdot y) \cdot z \neq x \cdot (y \cdot z) \tag{4}$$
$$x \cdot (y + z) \neq (x \cdot y) + (x \cdot z) \tag{5}$$
$$(x + y) \cdot z \neq (x \cdot z) + (y \cdot z) \tag{6}$$

- solution of equations, e.g., $(1 + x) = 1$ for 4-bit mantissa solved by any $x < 10^{-4}$ (see absorption) $\rightarrow$ *smallest* float number ϵ with $1 + \epsilon > 1$ called machine precision

Multiplication and division of floating point numbers:
mantissas multiplied/divided, exponents added/subtracted
$\rightarrow$ no cancellation or absorption problem

Floating point data types IX

Guard bit, round bit, sticky bit (GRS)

- in floating point arithmetic: if mantissa shifted right $\rightarrow$ loss of digits
- therefore: during calculation 3 extra bits (GRS)
  Guard bit: 1st bit, just extended precision
  Round bit: 2nd (Guard) bit, just extended precision (same as G)
  Sticky bit: 3rd bit, set to 1, if any bit beyond the Guard bits non-zero, stays then 1(!)
  $\rightarrow$ sticky
- example

```
                                               G R S
Before 1st shift: 1.11000000000000000000100 0 0 0
After 1 shift:    0.11100000000000000000010 0 0 0
After 2 shifts:   0.01110000000000000000001 0 0 0
After 3 shifts:   0.00111000000000000000000 1 0 0
After 4 shifts:   0.00011100000000000000000 0 1 0
After 5 shifts:   0.00001110000000000000000 0 0 1
After 6 shifts:   0.00000111000000000000000 0 0 1
After 7 shifts:   0.00000011100000000000000 0 0 1
After 8 shifts:   0.00000001110000000000000 0 0 1
```

Floating point data types X

GRS bits – possible values and stored values

extended sum	stored value	why
1.0100 000	1.0100	truncated because of GR bits
1.0100 001	1.0100	truncated because of GR bits
1.0100 010	1.0100	rounded down because of GR bits
1.0100 011	1.0100	rounded down because of GR bits
1.0100 100	1.0100	rounded down because of S bit
1.0100 101	1.0101	rounded up because of S bit
1.0100 110	1.0101	rounded up because of GR bits
1.0100 111	1.0101	rounded up because of GR bits

Floating point data types XI

IEEE representation of 32 bit floats:

Number name	sign, exp., f	value
normal	$0 < e < 255$	$(-1)^s \times 2^{e-127} \times 1.f$
subnormal	$e = 0, f \neq 0$	$(-1)^s \times 2^{-126} \times 0.f$
signed zero (± 0)	$e = 0, f = 0$	$(-1)^s \times 0.0$
$+\infty$	$s = 0, e = 255, f = 0$	`+INF`
$-\infty$	$s = 1, e = 255, f = 0$	`-INF`
Not a number	$e = 255, f \neq 0$	`NaN`

- if float $> 2^{128}$ $\rightarrow$ overflow, result may be `NaN` or unpredictable
- if float $< 2^{-128}$ $\rightarrow$ underflow, result is set to 0

If not default by compiler: enable floating-point exception handling (e.g., `-fpe-all0` for `ifort`)

Automatic type conversion

In C/C++ many data type conversions are already predefined, which will be invoked automatically:

```
int main () {
   int a = 3 ;
   double b ;
   b = a ;       // implicit conversion of a to double
   b = 1. / 3 ; // implicit conversion of 3 to double
   return 0.2 ; // implicit conversion of 0.2 to integer 0
}
```

Explicit type conversions (casts) I

Moreover, a type conversion/casting can be done explicitly:

C cast

```
int main () {
  int a = 3 ;
  double b  ;
  b = (double) a ; // type cast
  return 0 ;
}
```

- obviously possible: integer ↔ floating point
- but also : pointer (see below) ↔ data types
- Caution: For such C casts there is no type checking during runtime!

Explicit type conversions (casts) II

The better way: use the functions of the same name for type conversion

```
int i, k = 3 ;
float x = 1.5, y ;
i = int(x) + k ;
y = float(i) + x ;
```

Task 2.4 Integer conversion

What is the result for `i` and `y` in this example above?

Logical variables

```
bool b ;
```

intrinsic data type, has effectively only two different values:

```
bool btest, bdo ;
bdot = false ; // = 0
btest = true ; // = 1
```

but also:

```
btest = 0. ; // = false
btest = -1.3E-5 ; // = true
```

Output via `cout` yields `0` or `1` respectively. By using `cout << boolalpha << b ;` is also possible to obtain `f` and `t` for output.

Note: minimum addressable piece of memory is 1 byte $\rightarrow$ `bool` needs more memory than necessary

Character variables I

```
char character ;
```

are encoded as integer numbers:

```
char character = 'A' ;
char character = 65 ;
```

mean the same character (ASCII code)

Assignments of character literals to character variables require single quotation marks `'`:

```
char yes = 'Y';
```

Character variables II

Character input

```
char character ;
int  number ;
cout << "Character input: " ;
cin >> character ;
cout << "character is: " << character
     << " corresponds to " << int(character) << endl;
cout << "Number input: " ;
cin >> number ;
cout << "Number " << number
     << " corresponds to " << char(number) << endl;
```

Task 2.5 Characters

Complete this code example to a C++ program, compile and execute it. Which (decimal) ASCII code have }, Y and 1 ? Which character has the code 97 ?

Execution control - `for`-loops I

Executable control constructs modify the program execution by selecting a block for repetition (loops, e.g., `for`) or branching to another statement (conditional, e.g., `if`/ unconditional, e.g., `goto`).

Repeated execution of an instruction/block:

`for` loop

```
for (int k = 0 ; k < 6 ; ++k ) sum = sum + 7 ;

// also possible: non-integer loop variable -> not recommended
for (float x = 0.7 ; x < 17.2 ; x = x + 0.3) {
    y = a * x + b ;
    cout << x << " " << y << endl;
}
```

Execution control - `for-loops` II

Structure of the loop control (header) of the `for` loop:

There are (up to) three arguments, separated by semicolons:

1. initialization of the loop variable (loop counter), if necessary with declaration, e.g.:
   `int k = 0 ;`[†]
   $\rightarrow$ is executed *before the first* iteration
2. condition for termination of the loop, usually via arithmetic comparison of the loop variable, e.g.,
   `k < 10 ;`
   is tested *before each* iteration
3. expression: incrementing/decrementing of the loop variable, e.g.,
   `++k` or `--k` or `k += 3`
   is executed *after each* iteration

[†]interestingly also: `int k = 0, j = 1;`, i.e. multiple loop variables of same type

Increment operators

`sum += a`

$\rightarrow$ `sum = sum + a`

`++x`

$\rightarrow$ `x = x + 1` (increment operator)

`--x`

$\rightarrow$ `x = x - 1` (decrement operator)

Note that there is also a *post* increment/decrement operator: `x++`, `x--`, i.e. incrementing/decrementing is done *after* any assignment, e.g., `y = x++`.

Logical operators I - Comparisons/inequalities

→ return either(!) `true` or `false`:

`a > b`	greater than
`a >= b`	greater than or equal
`a == b`	equal
`a != b`	not equal
`a <= b`	less than or equal
`a < b`	less than

Caution!

The exact equality == should not be used for float-type variables because of the limited precision in the representation.

Logical operators II - Logical operations

`!(a < b)`	not	(2)
`(a < b) && (c != a)`	and	(14)
`(a < b) || (c != a)`	or	(15)

It is recommend to use parentheses ( ) for combination of operations for unambiguousness.

Otherwise: Operator Precedence (incomplete list)

Precedence	Operator
5	$*$ / %
6	+ −
9	< <= > >=
10	== !=
14	&&
15	||

Loops I

Moreover, there exist also:

`while` loops

```
while (x < 0.) x = x + 2. ;

do x = x + 2. ;  // do loop is executed
while (x < 0.) ; // at least once!
```

Instructions for loop control

```
break ;    // stop loop execution / exit current loop
continue ; // jump to next iteration
```

Loops II

In C/C++: no real “for loops”

$\rightarrow$ loop variable (counter, limits) can be changed in loop body

slow, harder to optimize for compiler/processor

Recommendation: *local* loop variables

$\rightarrow$ declaration in loop header
$\rightarrow$ scope limited to loop body

Loops III

Our example with the `float` loop variable

```
for (float x = 0.7 ; x < 17.2 ; x = x + 0.3) { // = 55 iterations
    y = a * x + b ;
    cout << x << " " << y << endl;
}
```

can be rewritten with integer loop variables (number of iterations clear)

```
float x = 0.7 , x_inc = 0.3, x_max = 17.2 ;
int it_max = ((x_max - x) / x_inc) + 0.5 ;  // +0.5 for correct rounding
for (int i = 0 ; i < it_max ; ++i) { // it_max = 54
    y = a * x + b ;
    cout << x << " " << y << endl;
    x+= x_inc ;
}
```

$\rightarrow$ note that when converting `float` $\rightarrow$ `int`, digits after decimal point just cut off $\rightarrow$ add `+0.5` before conversion for correct rounding

Execution control – conditional statements I

Conditional execution via `if`:

```
if (z != 1.0) k = k + 1 ;
```

Conditional/branching

```
if (a == 0) cout << "result" ; // one-liner

if (a == 0) a = x2 ; // branching
else if (a > 1) {
     a = x1 ;
}
else a = x3 ;
```

switch (...) case I

If the variable used for branching has only discrete values (e.g., `int`, `char`, but not floats!), it is possible to formulate conditional statements via `switch/case`:

Branching II

```
switch (epxression) {
        case value1 : instruction ; break ;
        case value2 : instruction1 ;
                      instruction2 ; break ;
        default     : instruction ;
}
```

Heads up!

Every case instruction section should be finished with a `break`, otherwise the next case instruction section will be executed automatically.

switch (...) case II

Example: switch

```
int k ;
cout << "Please enter number, 0 or 1: " ;
cin >> k ;
switch (k) {
  case    0 : cout << "pessimist" << endl ; break ;
  case    1 : cout << "optimist"  << endl ; break ;
  default   : cout << "neutral" << endl ;
}
```

Arrays in C/C++

Static array declaration for a one-dimensional array of type `double`:

`double a[5] ;`	one-dimensional array with 5 elements of type double (e.g., vectors)

Access to individual elements:

```
total = a[0] + a[1] + a[2] + a[3] + a[4] ;
```

Heads up!

In C/C++ the index for arrays starts always at 0 and runs in this example until 4, so the last element is `a[4]`.

A common source of errors in C/C++ !!!

Note: While the size of the array can be set during runtime, the size cannot be changed after declaration (**static** declaration).

Two-dimensional arrays I

an $m \times n$ matrix (rows $\times$ columns) :

$$\begin{array}{cc} & n \text{ columns} \rightarrow \\ \begin{array}{c} m \\ \text{rows} \\ \downarrow \end{array} & \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & \dots & & \\ \dots & & & \\ a_{m1} & & & a_{mn} \end{pmatrix} \end{array}$$

`int a[m][n]` ... static allocation of two-dimensional array, e.g., for matrices (m, n must be constants)

access via, e.g., `a[i][j]`

`i` is the index for the rows,
`j` for the columns.

Two-dimensional arrays II

$$\text{e.g.,}\quad a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Note that in C/C++ the second (last) index runs first, i.e. the entries of `a[2][3]` are in this order in the memory :

```
a[0][0]  a[0][1]  a[0][2]  a[1][0]  a[1][1]  a[1][2]
1        2        3        4        5        6
```

(row-major order $\rightarrow$ stored row by row)

Two-dimensional arrays III

Task 2.6 Internal order of arrays

The cache, which is the memory closest to the CPU and usually on the same chip, is limited ($\sim$MB). Therefore it is important to design programs in a way that for a specific task data that must be read into the cache are in a subsequent order.

Let's assume for a cosmological simulation with 10^6 particles, for each particle the coordinates and velocities (3D) should be saved in an array `particle[][]`. A function loops over all particles and needs to access for each particle all $\vec{x}, \vec{v}$-data.

How should this array be dimensioned in C/C++: `particle[6][1000000]` or `particle[1000000][6]` ?

Initialization of arrays

An array can be initialized by curly braces:

```
int array[5] = {0, 1, 2, 3, 4} ;

short field[] = {0, 1} ; // array field is automatically
                         // dimensioned

float x[77] = {0} ; // set all values to 0
```

Strings I

There are no string variables in C. Therefore strings are written to one-dimensional character arrays:

```
char text[6] = "Hello" ;
```

The string literal constant `"Hello"` consists of 5 printable characters and is terminated automatically by the compiler with the null character `\0`, i.e. the array must have a length of 6 characters! Note the double quotation marks!

Example

```
char text[80] ;
cout << endl << "Please enter a string:" ;
cin  >> text ;
cout << "You have entered " << text << " ." << endl ;
```

Strings II

Task 2.7

1. What is the difference between `'Y'` and `"Y"`?
2. Which of these two literals is correct: `'Yes'` oder `"Yes"`?
3. What's wrong here: `char text[2] = "No" ;`?

String comparison

C-Strings (character arrays) cannot be compared directly with ==, in this case the operator would compare the start addresses of the arrays.
Instead: Use `strcomp(string1,string2)` from library `string.h`, this will return 0 if strings are equal (arrays can have different lengths).

Declaration and visibility of variables I

Declarations of variables should be at the beginning of a **block**, exception: loop variables

```
float x, y ; // declaration of x and y
int n = 3 ; // declaration and initialization of n
```

Local variables / variables in general

- are only visible within the block (e.g., in `int main() { }`), where they have been declared $\rightarrow$ **scope**
- are **local** regarding this block, can only be accessed within this block
- are unknown outside of this block, i.e., they don't exist there
- are automatically deallocated when leaving the scope, except those with modifier `static`

Declaration and visibility of variables II

Global variables

- must be declared outside of any function, e.g., before `main()`
- are visible/known to all following functions within the same program
- have file wide visibility (i.e., if you split your source code into different files, you have to put the declaration into every file)
- are only removed from memory when execution of the program is ended

A locally declared variable will hide a global variable of the same name. The global variable can be still accessed with help of the scope operator `::`, e.g., `cout << ::m ;`

Declaration and visibility of variables III

Global and local variables

```
int m = 0 ;       // global variable

void calc() {
  int k = 0;      // local variable
  m = 1 ;         // ok, global variable
  ++j ;           // error, as j only known in main
}

int main() {
  int j = 3 ;
  ++j ; // ok
  for (int i = 1 ; i < 10 ; ++i) {
     j = m + i ; // ok, all visible
  }
  m = j - i ;     // error: i not visible outside loop
  return j ;
}
```

Defining constants

Values (e.g., numbers) that do not change during the program execution, should be *defined* as constants:

```
const float e = 2.71828 ;
const int prime[] = {2,3,5,7} ;
```

Constants must be initialized during declaration.

After initialization their value cannot be changed.

Use `const` whenever possible!
(The compiler will replace any occurrence of the constant name by the value before "translation" $\rightarrow$ no memory addressing necessary as for variables.)

Pointer I

Pointer variables – or pointer for short – allow a direct access (i.e. not via the name) to a variable.

Declaration of pointers

```
int     *pa   ; // pointer to int
float   *px   ; // pointer to float

int    **ppb ; // pointer to pointer to int
int ***pppb ; // pointer to pointer to pointer to int
   ...
```

C++ standard : at least 255 (static) ; in C: at least 12 (static)
but: infinite dynamic (linked lists)

Pointer II

A pointer is a variable that contains an address, i.e. it points to a specific part of the memory.
As every variable in C/C++ a pointer variable must have a data type.
The value at address (memory) to which the pointer points, must be of the declared data type.

address	value	variable
1000	0.5	x
1004	42	n
1008	3.141. . .	d
1012	. . . 5926	
1016	H E Y !	salutation
1020	1000	px
1024	1008	pd
1028	1004	pn
1032	1016	psalutation
1036	1028	pp

Pointer III

Pointers must be always initialized before usage!

Initialization of pointers

```
int  *pa ; // pointer to int
int b ;    // int
pa = &b ;  // assigning the address of b to a
```

The character & is called the address operator (“address of”)
(not to be confused with the reference `int &i = b ;`).

Declaration and initialization

```
int b ;
int *pa = &b ;
```

$\rightarrow$ content of `pa` = address of `b`

Pointer IV

With help of the dereference operator * it is possible to get access to the value of the variable b, one says, pointer pa is dereferenced:

Dereferencing a pointer

```
int b, *pa = &b ;
*pa = 5 ;
```

Here, * . . . is the dereference operator and means "value at address of . . .".

The part of the memory to which pa points, contains the value 5, that is now also the value of the variable b.

```
cout << b << endl ; // yields 5
cout << pa << endl ; // e.g., 0x7fff5fbff75c
// and with pointer to int-pointer:
int **ppa ; ppa = &pa ; cout << **ppa << endl ; // yields also 5
```

Pointer V

Once again:

Pointer declaration:

```
float *pz, a = 2.1 ;
```

Pointer initialization:

```
pz = &a ;
```

Result – output:

```
cout << "address of variable a (content of pz): "
     << pz << endl ;
cout << "content of variable a: "
     << *pz << endl ;
*pz = 5.2 ; // change value of a
```

References

```
int &n = m ;
m2 = n + m ;
```

- A reference is a new name, an alias for a variable. So, it is possible to address the same part of the memory (variable) by different names within the program. Every modification of the reference is a modification of the variable itself - and vice versa.
- References are declared via the & character (reference operator) and must be initialized instantaneously:

```
int a ;
int &b = a ;
```

- This initialization cannot be changed any more within the program!

(At this stage a reference seems to be rather useless.)

Passing variables to functions I

Structure of functions – definition

type name (arg1, ...) { ... }

example: `int main (int argc, char *argv[]) { }`

- in parentheses (): arguments of the function / formal parameters
- when function is called: copy arguments (values of the given variables) to function context $\rightarrow$ *call by value* / *pass by value*

```
setzero (float x) { x = 0. ; }
int main () {
    float y = 3. ;
    setzero (y) ;
    cout << y ; // prints 3. }
```

Passing variables to functions II

Call by value

Pros:

- the value of a passed variable cannot be changed unintentionally within the function

Cons:

- the value of a passed variable can also not be changed on purpose
- for every function call all value must be *copied*
  $\rightarrow$ extra overhead (time)
  (exception: if parameter is an array, only *start address* is passed $\rightarrow$ pointer)

Passing variables to functions II

Call by reference (C++)

```
void swap(int &a, int &b) ;
```

Passing arguments as references:

The variables passed to the function swap are changed in the function and keep these values after returning from swap.

```
void swap (int &a, int &b) {
 int t = a ;  a = b ; b = t ; }
```

$\rightarrow$ and called via: `swap (n, m) ;`

Thereby we can pass an arbitrary number of values back from a function.

Hint: The keyword `const` prevents that a passed argument can be changed within the function:
`sum (int const &a, int const &b) ;`

Passing variables to functions

Call by pointer

A function for swapping two `int` variables can also be written by using pointers:

```
void swap(int *a, int *b) { // pointers as formal parameters
    int t = *a ; *a = *b ; *b = t ; // remember: *a -> value at address of a
}
```

Call in `main()`:

```
  swap (&x, &y) ;  // Passing addresses(!) of x and y
```

Passing arrays to functions

In contrast to (scalar) variables, arrays are automatically passed by address (pointer) to functions (see below), e.g.,
`myfunc ( float x[] )`

Pointers and references

Pointer variables

- store addresses
- must be dereferenced (to use the value of the spotted variable)
- can be assigned as often as desired to different variables (of the same, correct type) within the program

References

- are alias names for variables,
- can be used by directly using their names (without dereferencing)
- the (necessary!) initialization at declaration cannot be changed later
- (actually only useful as function arguments or result)

Passing arrays to functions in C++ I

Declaration of a 1d-array:

```
int m[6] ; // statically dimensioned†
```

Declaration of a function with an array type argument:

```
int sumsort (int m[], int n) ; // n = length of m
```

Calling a function with an array type argument:

```
sum = sumsort (m, 6) ;
```

→ passing the array is implicitly done by a pointer, i.e. only the *start address* of the array will be passed to the function

†an array can also be declared dynamically, so with size fixed at runtime, but only *locally* and arrays with more than 1 dimension must have fixed sizes at compile time if they are passed to functions (see below)

Passing arrays to functions in C++ II

Correspondence of pointers and arrays

→ see exercise

- the assignment

```
a[i] = 1 ;
```

  is equivalent to

```
*(a + i) = 1 ;
```

- when passing 1d-arrays to functions the start address and the data type (size of the entries) is sufficient

Passing arrays to functions in C++ III

Problem:

When using multi-dimensional arrays, passing of the start address alone is not sufficient. Every dimensioning after the first one must be explicitly (integer constant!) written.

Therefore:

```
float absv  (float vector[], int n) ;  // 1d-array
float trace (float matrix[][10]) ;     // 2d-array
float maxel (float tensor[][13][13]) ; // 3d-array
```

→ more flexibility by using pointers as arguments, e.g., for an array `a[3][4]`:
`float *a[3] ; ... ; a[i] = new float[4] ; float function (float **a, ...)`

→ special matrix-*classes* simplify the passing to functions

→ in Fortran, passing arrays to functions is much easier (i.e. only start address is passed)

Structs and classes – defining new data types I

Besides the intrinsic (/basic) data types there are many other data types, which can be defined by the programmer

struct

```
struct complex {
     float re ;
     float im ;
} ; [a]
```

[a] Note the necessary semicolon after the } for structs

In this example the data type `complex` is defined, it contains the *member variables* for real and imaginary part.

struct vs. class

The constructs `struct` and `class` are identical in C++ with the exception that access to `struct` is public by default and for `class` it is private. They can be defined outside or inside a function (e.g., `main`).

Structs and classes – defining new data types II

Structs can be imagined as collections of variables.

struct

```
struct star {
     char full_name[30] ;
     unsigned short binarity ;
     float luminosity_lsun ;
} ;
```

These (self defined) data types can be used in the same way as intrinsic data types:

Declaration of `struct` objects

```
complex z, c ;
star sun ;
```

Structs and classes – defining new data types III

Concrete structs which are declared in this way are called *instances* or *objects* (→ object-oriented programming) of a class (struct).

Declaration and initialization

```
complex z = {1.1 , 2.2} ;
star sun  = {"Sun", 1, 1.0 } ;
```

The access to *member variables* is done by the *member selection operator* **.** (dot):

Access to members

```
 real_part = z.re ;
 sun.luminosity_lsun = 1.0 ;
```

Structs and classes – defining new data types IV

It is also possible to define functions (so-called *methods*) within structs:

Member functions

```
struct complex {
     ...
     float absolute () {
          return (sqrt(re*re + im*im)) ;
     }
} ;
complex c = {2., 4.} ;

cout << c.absolute() << endl ;
```

The call of the *member function* is also done with the `.` , the function (method) is associated with the object.

Structs and classes – defining new data types V

And even operators:

Operator overloading

```
complex operator+ (const complex & c) {
  complex z ;
  // calling object is referenced with  this->
  z.re = this->re + c.re ;
  z.im = this->im + c.im ;
  return z ;
}
  ...
complex w, z, c ;
  ...
w = z + c ;
// object on left side (z) of operator calls +
// object on the right side (c) is "argument" for call
```

Structs and classes – defining new data types VI

In our example for the absolute of a complex number, the call is `c.absolute()` instead of the common `absolute(c)`
The latter call can be achieved with help of a static member function, that is shared by all objects and exists independently of them

Static member functions

```
  static double abs (const complex & c)
    return ( sqrt(c.re * c.re + c.im * c.im) ) ;


  ...
  complex::abs(c) ;
```

Static functions must be called with the class name (here: complex) and the scope operator ::
Static functions have no `this->` pointer

Classes – Example: writing/reading files I

Output to a file by using library `fstream`:

1. `#include <fstream>`
2. create an object of the class `ofstream`:

   `ofstream fileout ;`
3. method open of the class `ofstream`:

   `fileout.open("graphic.ps") ;`
4. writing data: e.g.
   `fileout << x ;`
5. close file via method `close`:

   `fileout.close() ;`

Simple alternative (Unix): Use `cout` and redirection operator > or >> of the shell:

```
./program > output.txt
```

Classes – Example: writing/reading files II

By including the `<fstream>` library, one can also read from a file

Input from a file

```
char line[132] ;
ifstream filein ; // create ifstream object
filein.open("data.txt") ; // open the file
while ( filein.good() ) {
    filein.getline(line,132) ; // read in line;
                               // use buffer size (132)
    x[i] = atof(line) ;        // read into float array
}
```

The method `good()` checks, whether the end of file (EOF) is reached or an error occurred.

Private and public

- `class` : by default all members are private → accessible elements must be declared as public

```
class complex {
  float real, imag ; // implicitly private
  public : getreal () { return this->real ; }
};
```

- member variables usually set private, access to them via public methods (e.g., get. . . , set. . .)
- keywords `public` and `private` (with :) valid until next of those occurs

Constructors

- each class has a default constructor with empty argument list if no constructor is explicitly defined:

```
struct complex {
  ...
};
...
complex z ; // default constructor
z = {x , 1.} ; // initialization (only if constructor is public)
```

- one may define more constructors, e.g.:

```
struct complex {
 public : complex (double x, double y) {real = x ; imag = y ;}
 ...
};
complex z (x, y) ; // constructor initializes real and imaginary part
```

Templates I

Templates allow to create universal definitions of certain structures. The final realization for a specific data type is done by the compiler.

Function templates

```
template <class T> // instead of keyword 'class' also 'typename' allowed
T sqr (const T &x) {
 return x * x ; }
```

The keyword `template` and the angle brackets < > signalize the compiler that `T` is a template parameter. The compiler will process this function if a specific data type is invoked by a function call, e.g.,

```
double w = 3.34 ; int k = 2 ;
cout << sqr(w) << " " << sqr(k) ;
```

$\rightarrow$ for full convenience, templates must be already defined before the call, e.g., already in the header file (i.e. the compiler needs to know which concrete versions must be created)

Templates II

Moreover, templates can be used to create structs/classes. For example, the class `complex` of the standard C++ library (`#include <complex>`) is realized as template class:

Class templates

```
template <class T>
class std::complex {
   T re, im ;
 public:
      ...
   T real() const return re ;
}
```

Therefore, the member variables `re` and `im` can be arbitrary (numerical) data types.

Templates III

We can also have function templates of different types

Function template for multiple types

```
template <class T, class U>
 auto max (const T &x, const U &y) {
 return (x > y) ? x : y ; // return maximum of both arguments
}
   ...
cout max(2, 1) << " " << max(3.3, 4.4) << " " << max(1, 2.) << endl ;
   ...
```

→ `max( , )` can now be called with mixed arguments, e.g., `int` and `double`: `max(1, 2.)`
→ keyword `auto` instructs compiler to select return type automatically, e.g., `double` if arguments are `double` and `int`
In C++20 the function header above can be shorter written as

```
auto max (const auto &x, const auto &y)
```

? is the ternary conditional operator, meaning *condition* ? result_if_true : result_if_false

Typ definitions via typedef

By using `typedef` *datatype alias name* one can declare new names for data types:

```
typedef unsigned long   large ;
typedef char*  pchar ;
typedef std::complex<double>  complex_d ;
```

These new type names can then be used for variable declarations:

```
large  mmm ;
pchar  Bpoint ;
complex_d  z = complex_d (1.2, 3.4) ;
```

In the last example, the constructor for the class template `complex` gets the same name as the variable through the `typedef` command.

Exception handling – exceptions I

A major strength of C++ is the ability to handle runtime errors, so called exceptions:

Throwing exceptions: `try` – `throw` – `catch`

```
try {
   cin >> x ;
   if ( x < 0.) throw string("Negative value!") ;
   y  = g(x) ;
}
catch (string info) { // catch exception from try block
   cout << "Program stops, because of: " << info << endl ;
   exit (1) ;
}
...
double g (double x) {
  if (x > 1000.) throw string("x too large!") ; ... }
```

Exception handling – exceptions II

`try { ...}`

- within a `try` block an arbitrary exception can be thrown

`throw` *e* ;

- throw an exception *e*
- the data type of *e* is used to identify to the corresponding `catch` block to which the program will jump
- exceptions can be intrinsic or self defined data types

Exception handling – exceptions III

`catch ( type e ) { ...}`

- after a `try` one or more `catch` blocks can be defined
- from the data type of *e* the first matching `catch` block will be selected
- any exception can be caught by `catch (...)`
- if after a `try` no matching `catch` block is found, the search is continued in the next higher call level
- if no matching block at all is found, the `terminate` function is called; its default is to call `abort`

Exception handling – exceptions IV

Data types for exception throwing

In contrast to the simple example above, it is recommended to use specific (not built-in) data types *e* for `throw`, e.g., from class `exception`.

```
#include <exception>
 ...
 try {
    cin >> x ;
    if ( x < 0.) throw runtime_error("Negative Number!");
    y = g(x) ;
 }
 catch (const runtime_error& ex) { // catch exception from try block
    cout << "Program stops, because of: " << ex.what()  << endl ;
    exit (1) ;
  }
```

Reading arguments from program call

Sometimes it is more convenient to pass the parameters the program needs directly at the call of the program, e.g,

```
./rstarcalc 3.5 35.3
```

this can be realized with help of the library `stdlib.h`

Read an integer number from command line call

```
#include "stdlib.h"
int main (int narg, char *args[]) {
  int k ;
  // convert char array to integer
  if (narg > 1) k = atoi(args[1]) ;
}
```

- if the string cannot be converted to `int`, the returned value is 0
- there exist also `atol` and `atof` for conversion to `long` and `float`

Summary

Common mistakes in C/C++:

- forgotten semicolon `;`
- wrong dimensioning/access of arrays
  `int m[4] ; imax = m[4] ;` → `imax = m[3] ;`
- wrong data type in instructions / function calls
  `float x ; ... switch (x)` → `int i ; ... switch (i)`
  `void swap (int *i, int *j) ; ... int m, n ; ... swap(n, m) ;`
  → `swap(&n, &m) ;`
- confusing assignment operator `=` with the equality operator `==`
  `if(i = j)` → `if(i == j)`
- forgotten function parenthesis for functions without parameters
  `clear ;` → `clear();`
- ambiguous expressions
  `if (i == 0 && ++j == 1)`
  no increment of `j`, if `i` $\neq 0$

Some recommendations I

- use always(!) the `.` for floating point literals: `x = 1./3.` instead of `x = 1/3`
- white space is for free $\rightarrow$ use it extensively for structuring your source code (indentation, blank lines)
- comment so that you(!) understand your source code in a year
- use self-explaining variable names, e.g., `Teff` instead of `T` (think about searching for this variable in the editor)
- use integer loop variables:

  ```
  for (int i = 1; i < n ; ++i) {
   x = x + 0.1 ; ... }
  ```

  instead of

  ```
  for (float x = 0.; x < 100. ; x = x + 0.1) {... }
  ```

- take special care of user input, usually: $t_{\text{input}} \ll t_{\text{calc}}$, so exception catching for input is never wasted computing time

Some recommendations II

Tips for High Performance Computing / Number Crunching

- The more flexible your program is, the harder it is for the compiler to optimize it.
  Hence:
- Use `const` whenever possible (values, arguments).
- Avoid pointers (except for argument passing).
- (Avoid dynamic allocations.)
- Use keyword `inline` (see Sect. 1) for *small* functions (vs. code size see below).
  Avoid many (nested) function calls.
- Keep loops simple, avoid too many branchings and jumps. Use matrix classes/functions instead of looping over elements.

Execution speed vs. flexibility:

$\rightarrow$ flexibility increases $\rightarrow$

Assembler   Fortran   C   C++   Python

$\leftarrow$ speed increases $\leftarrow$

Some recommendations III

Table: Latencies of memory operations in relation to each other, see github

operation	real time	scaled time ($\times 10^9$)
Level 1 cache access	0.5 ns	0.5 s ($\sim$ heart beat)
Level 2 cache access	7 ns	7 s
Multiply two floats	10 ns	10 s (estimated)
Devide two floats	40 ns	40 s (estimated)
RAM access	100 ns	1.5 min
Send 2kB over Gigabit network	20 000 ns	5.5 h
Read 1MB from RAM	250 000 ns	2.9 d
Read 1MB from SSD	1 000 000 ns	11.6 d
Read 1MB from HDD	20 000 000 ns	7.8 months
Send packet DE$\rightarrow$ US$\rightarrow$ DE	150 000 000 ns	4.8 years

Numerical precision

Machine precision I

- as seen for `float x = 7 + 1.E-7`: because of only 23 bit for mantissa result is 7
- therefore: machine precision ϵ_m is maximum possible number for which $1_\mathrm{c} + \epsilon_\mathrm{m} = 1_\mathrm{c}$, where c means computer representation
- hence: for any number x_c
  $x_\mathrm{c} = x(1 \pm \epsilon), \quad |\epsilon| \leq \epsilon_\mathrm{m}$
- remember: for all 32 bit floats $\rightarrow$ error in $\simeq$ 6th decimal place,
  for 64 bit doubles $\rightarrow$ error in $\simeq$ 15th place

Machine precision II

Determining machine precision

```
float eps = 1.f ;
for (int i = 1 ; i < 100 ; ++i){
    eps = eps / 2.f ; // float literal 2.f
    cout << i << " " << eps << " "
         << setprecision(9)
         << 1.f + eps << endl ;
}
```

e.g., for `float`:

```
23 1.1920929e-07  1.00000012
24 5.96046448e-08 1
```

Similarly for `double`:

```
52 2.2204460492503130808e-16 1.0000000000000002220
53 1.1102230246251565404e-16 1
```

Types of errors I

We may distinguish:

1. *random errors:* caused by non-perfect hardware, e.g., aging of RAM cells; can be minimized by using "checksums", e.g., by ECC (Error correction code) techniques (corrects 1 bit errors, recognizes 2 bit errors), checksums in protocol headers (e.g., TCP/UDP), Btrfs scrub on RAID1 against Bit rot on HDD (typical bit error rate 1:10^{14})
$\rightarrow$ likelihood increases with runtime
2. *approximation errors:* because of finiteness of computers, e.g., stopping series calculation, finite integration steps, ...

$$e^{-x} = \sum_{n=0}^{\infty} \frac{(-x)^n}{n!} \approx \sum_{n=0}^{N} \frac{(-x)^n}{n!} = e^{-x} + \mathcal{E}(x, N) \tag{7}$$

where $\mathcal{E}$ vanishes for $N \rightarrow \infty$, hence we require $N \gg x$, expecting here large $\mathcal{E}$ for $x \approx N$

Types of errors II

3. *roundoff errors:* limitation in the representation of real numbers (finite number of digits), e.g., if only three decimals are stored: 1/9=0.111 and 5/9=0.556, hence

$$5\left(\frac{1}{9}\right) - \frac{5}{9} = 0.555 - 0.556 = -0.001 \neq 0 \qquad (8)$$

$\rightarrow$ error is intrinsic and *accumulates with the number of calculation steps*
$\rightarrow$ some algorithms unstable because of roundoff errors

again: for a *float* number like

$$x = 1122334455667788990\text{0}. = 1.1223344556677889900 \times 10^{19} \qquad (9)$$

only the first part (32 bit: 1.12233) is stored, while exponent is stored exactly

Absorption and subtractive cancellation I

Absorption

- adding floating point numbers of very different magnitude may result in absorption, e.g., `float x = 7. + 1.E-7` gives 7
- absorption may even result in instable behavior in combination with floating point loop counters (therefore never use them!)

```
float y = 100000010., inc = 1. ;
for (float x = 100000001. ; x <= y ; x += inc) { ... }
```

$\rightarrow$ loop may run infinitely

Absorption and subtractive cancellation II

- absorption may lead to saturation in summation schemes, hence:

Kahan summation algorithm

E.g., summing over an array `input[n]`:

```
float y, t, sum = 0.,
      c = 0. ; // compensation
for (int i = 0 ; i < n ; ++i) {
   y = input[i] - c ;  // c is zero in 1st iteration
   t = sum + y ;       // sum >> y
   c = (t - sum) - y ; // (t - sum) cancels high-order part of y;
                       // subtracting y recovers negative (low part of y)
   sum = t ;
}
```

$\rightarrow$ alternatively: higher precision (`double`),
pairwise summation (e.g., default in NumPy, used in many FFT algorithms)

Absorption and subtractive cancellation III

Subtractive cancellation

- consider computer representation x_c of an exact number x:

$$x_c \simeq x(1+\epsilon_x) \tag{10}$$

with relative error ϵ_x in x_c (similar to machine precision)

- so for subtraction

$$a = b - c \rightarrow\ a_c \simeq b_c - c_c \simeq b(1+\epsilon_b) - c(1+\epsilon_c) \simeq b - c + b\epsilon_b - c\epsilon_c \ \big|\ : a \tag{11}$$

$$\rightarrow \frac{a_c}{a} \simeq \frac{b-c}{a} + \epsilon_b \frac{b}{a} - \epsilon_c \frac{c}{a} \simeq 1 + \epsilon_b \frac{b}{a} - \epsilon_c \frac{c}{a} \tag{12}$$

(weighted errors) and if $b \simeq c$

$$\frac{a_c}{a} = 1 + \epsilon_a \simeq 1 + \frac{b}{a}(\epsilon_b - \epsilon_c) \simeq 1 + \frac{b}{a}\max(|\epsilon_b|, |\epsilon_c|) \tag{13}$$

as $b \simeq c \rightarrow \frac{b}{b-c} \gg 1 \rightarrow \frac{b}{a} \gg 1 \rightarrow$ relative error in a blown up

Absorption and subtractive cancellation IV

Warning

When subtracting two large numbers resulting in a small number, significance is lost.

Examples:

- computation of derivatives according to $\frac{f(x+h)-f(x)}{h}$
- the original Verlet method: $v_n = \dfrac{x_{n+1} - x_{n-1}}{2\Delta t}$
- solution of quadratic equation for $b \gg 4ac$:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad \text{or} \quad x_{1,2} = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}} \tag{14}$$

- in e^{-x} for large x: the first terms $(1 - x + x^2/2 - \ldots)$ are large $\rightarrow$ as result is small $\rightarrow$ subtraction by other large terms $\rightarrow$ improve algorithm by calculating $1/e^x$

Errors in algorithms I

Roundoff error accumulation, e.g., for multiplication:

$$a = b * c \rightarrow \; a_\mathrm{c} = b_\mathrm{c} * c_\mathrm{c} = b(1+\epsilon_b) * c(1+\epsilon_c) \mid : a \tag{15}$$

$$\rightarrow \; \frac{a_\mathrm{c}}{a} = (1+\epsilon_b)(1+\epsilon_c) \simeq 1 + \varepsilon_\mathrm{b} + \varepsilon_\mathrm{c} \tag{16}$$

(neglecting very small ϵ^2 terms) $\rightarrow$ as for physical error-propagation: adding up relative errors (no cancellation)

So, model for error-propagation: similar to random-walk (see later) where accumulated distance after N computation steps of length ℓ is $\approx \sqrt{N}\ell$, roundoff error *may* accumulate randomly:

$$\epsilon_\mathrm{roundoff} \approx \sqrt{N}\,\epsilon_\mathrm{m} \tag{17}$$

$\rightarrow$ if no detailed error analysis available;
otherwise, if not random: $\epsilon_\mathrm{roundoff} \approx N\epsilon$

Errors in algorithms II

Usually: if A is correct result and numerical approximation is $A(N)$, accuracy of $A(N)$ improves by adding more terms, i.e. approximation error drops with larger N

$$\epsilon_{\mathrm{appr}} \simeq \frac{\alpha}{N^{\beta}} \tag{18}$$

with some constants α, β depending on algorithm

However, each calculation step might increase roundoff error, so

$$\epsilon_{\mathrm{tot}} = \epsilon_{\mathrm{appr}} + \epsilon_{\mathrm{roundoff}} \simeq \frac{\alpha}{N^{\beta}} + \sqrt{N}\epsilon_{\mathrm{m}} \tag{19}$$

Hopefully: ϵ_{appr} dominant, but $\epsilon_{\mathrm{roundoff}}$ grows slowly
$\rightarrow$ stop calculation (optimum N) for minimum ϵ_{tot}

Errors in algorithms III

Minimize the error

Let's assume that some algorithm behaves like

$$\epsilon_{\mathrm{appr}} \simeq \frac{1}{N^2} \rightarrow \epsilon_{\mathrm{tot}} \simeq \frac{1}{N^2} + \sqrt{N}\,\epsilon_{\mathrm{m}} \tag{20}$$

Then the best result (minimum total error) is achieved for an N from

$$\frac{d\epsilon_{\mathrm{tot}}}{dN} = 0 = -2\,N^{-3} + \frac{1}{2}N^{-1/2}\epsilon_{\mathrm{m}} \rightarrow N^{\frac{5}{2}} = \frac{4}{\epsilon_{\mathrm{m}}} \tag{21}$$

So, for single precision ($\epsilon_{\mathrm{m}} \simeq 10^{-7}$)

$$N^{\frac{5}{2}} = \frac{4}{10^{-7}} \rightarrow N \simeq 1099 \rightarrow \epsilon_{\mathrm{tot}} = 4 \times 10^{-6} \tag{22}$$

$\rightarrow$ total error dominated by ϵ_{m}, typical for single precision

Errors in algorithms IV

Minimize the error II

So, if another algorithm

$$\epsilon_{\mathrm{appr}} \simeq \frac{2}{N^4} \rightarrow \epsilon_{\mathrm{tot}} \simeq \frac{2}{N^4} + \sqrt{N}\,\epsilon_{\mathrm{m}} \tag{23}$$

And again minimum error obtained for an N

$$\frac{d\epsilon_{\mathrm{tot}}}{dN} = 0 \rightarrow N^{\frac{9}{2}} = \frac{16}{\epsilon_{\mathrm{m}}} \rightarrow N \simeq 67 \rightarrow \epsilon_{\mathrm{tot}} = 9 \times 10^{-7} \tag{24}$$

So, need less steps and also obtain better precision

The better algorithm is not more elegant but needs less calculation steps and achieves a better precision.

Libraries

Excursus: Libraries I

→ collection of functions, variables, operators

```
#include <iostream>
```

- already seen: even simple input/output needs an additional library (e.g., `iostream`)
- idea of C/C++ in contrast to many other languages: only a few builtin instructions (e.g., `return`),
  everything else realized by corresponding libraries
  ⇒ high flexibility because of "outsourcing"
- also mathematical functions only available by corresponding libraries (e.g., `cmath` for sin and power)
- libraries allow easily the reuse of functions in different programs

Excursus: Libraries II

Including libraries in C++:

- at compile time:
  automatic call of the C preprocessor (cpp) by g++:
  read all instructions which start with a number sign #, especially

  ```
  #include <iostream>
  ```

  → look in the specified (default) directory paths (e.g.,`/usr/include/`)
  for header files, usually with extension `.h`,
  here: `iostream`
  → include the corresponding header file
  → pass output to compiler

The `<iostream>` header

The header file for the `iostream` library is in `/usr/include/c++/x.x/iostream`, where `x.x` depends on the specific version. It basically contains further `include` instructions.

→ for compilation the header file must be present, in openSUSE usually the corresponding *`libname`*`-devel` package must be installed manually

Excursus: Libraries III

The C preprocessor

CPP statements start with #, *no* semicolon ; at the end, but can be commented out via //

If the preprocessor is called explicitly:

```
cpp rcalc.cpp output
```

then from the source file `rcalc.cpp`, it generates an output file `output`, in which, e.g., `#define` instructions are resolved

- at link time:
  look for the libraries which belong to the header files, translate the names (symbols)[†] used in the library to (relative) memory addresses;

  static linking: include the necessary library symbols in the program

[†] the list of symbol names of the compiled code can be printed out with `nm` *`file.o`*

Excursus: Libraries IV

Dynamic libraries

The Unix command $\boxed{\texttt{ldd}}$ lists the dynamically linked-in libraries for a given program (or object file/library), e.g., `ldd -v rcalc`:

```
linux-vdso.so.1 (0x00007fff72bff000) †
libstdc++.so.6 => /usr/lib64/libstdc++.so.6 (0x00007ff2d9c0b000)
```

The path to the library and the memory address is printed.

- at runtime:
  dynamic linking: loading program and libraries to memory (RAM)
  advantage (over static linking): library is loaded only once and can be used by other programs

†vdso = virtual dynamic shared object

Excursus: Libraries V

$$\begin{tikzcd}
\text{C Preprocessor (cpp)} \arrow[d, Rightarrow] \\
\text{Compiler (g++)} \arrow[d, Rightarrow] \\
\text{Linker (ld)}
\end{tikzcd}$$

Excursus: Libraries VI

Overview: Unix commands for developers

- cpp: C preprocessor for the #-instructions
- g++: C++ compiler
- `ld`: link editor (usually called by the compiler)
- `ldd`: lists the used libraries of an object file (also program or library)
- `nm`: lists the *symbols* of an object file (etc.)

Symbols

In a C++ program `main` belongs to the symbols labeled with letter `T`. I.e., it is a symbol from the text (code) section of the file.

Linking and libraries I

- sometimes necessary for using some specific libraries: explicit specification (name) of the library at link time
- specification of a library `libpthread.so` via lower case `l`:

```
        -lpthread
```

  when calling the compiler for creation of the executables

  Example: `g++ -o programm program.cpp -lX11`
- specification of the path to the library via upper case `L`:

```
        -L/usr/lib/ -lpthread
```

  when calling the compiler for creation of the executables
  Heads up: The path must be given before the library!
- Important: the corresponding header file must be in a standard path, the current directory, or the path is specified via `-I`*`path-to-include-file`*

Linking and libraries II

- dynamic libraries must be located in a default system path (e.g., `/lib`) or the the path must be added to the environment variable

```
LD_LIBRARY_PATH
```

E.g. for the `bash` via

```
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:.
```

and for the `csh` respectively

```
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:.
```

$\rightarrow$ extending the path to dynamic libraries for the current working directory

Static linking I

- static libraries (file extension `.a`) are *archives* of object files
- these objects files are fixed included in the binary output during the procedure of static linking $\rightarrow$ can lead to large program files
- possible advantage: compact binaries with lean libraries (e.g., diet libc)

Sequence for static linking

If a library/program `libA` needs symbols from the library `libB`, the name of `libA` must be given before that of `libB` at link time for static linking: `-lA -lB`

- (complete) static linking isn't supported anymore by modern OSs (e.g. MacOS) at normal developer level
- but against some libraries (e.g., `libgfortran`, MKL) it can be selectively statically linked

Graphics with X11

Graphical output with X11

- there are many libraries for graphical output:
  - `Qt`, e.g., for `Mathematica`
  - `Simple DirectMedia Layer` for simple games
  - ...
- Pros: large support, comprehensive literature, often platform independent (e.g. via ports)
- Cons: often huge frameworks even for simplest tasks, huge libraries (memory consumption), usually high thresholds for beginners
- always available under Unix/Linux: `X11` or just `X` with many abilities:
  - creation of windows incl. internal structures (panels)
  - simple routines for drawing lines, circles, colors
  - keyboard and mouse inquiry
  - graphical forwarding (`ssh -Y`)

$\rightarrow$ We want to use `X11` more or less directly with help of the library `Xgraphics`.

`make`

make I

Purpose of `make`:

- automatic determination of the program parts (usually source files) that must be re-compiled via
    - a given definition of the dependencies / prerequisites (implicit, explicit)
    - comparison of time stamps (file system)
- calling the required commands for re-compilation:

typical use: `./configure ; make ; make install`
useful especially for large programs with many source files

make II

Main idea of `make` is the *rule*:

```
Target : Dependencies
<TAB> command for creation of the target
```

e.g.,

```
myprogram : myprogram.o
<TAB> g++ -o $@ $?
```

Note

- *explicit rules* are defined via an ASCII file, the so-called *makefile*
- every command belonging to a rule must start with a `<TAB>`!
- the macros `$@` and `$?` are called *automatic variables*, i.e., they are replaced by `make`:
  `$@` is replaced by the target,
  `$?` by the dependencies that are *newer* than the target
  `$^` $\rightarrow$ all dependencies (separated by blanks)

make III

Implicit rules:

- some rules for compilation are re-occurring, e.g., for C++ `.o` files are always created from `.cpp` files
- `make` has therefore a number of implicit rules, hence `make` can also be used without a `makefile`

Example

```
echo 'int main() {}' > myprog.cpp
make myprog
```

executes `g++ -o myprog myprog.cpp`[1]

- `make` uses implicit rules if no explicit rule for creation of the target has been found

[1] `make` invokes g++ automatically, or the C++ compiler that is specified in the environment variable `CXX`

make IV

Explicit rules

- an explicit rule is usually specified in a text file that has one of the following default names: `makefile`, `Makefile`
- every rule must define at least one target
- it is possible to define several dependencies for one target
- a rule can contain an arbitrary number of commands

Moreover, explicit rules overwrite implicit rules:

```
.c.o :
<TAB> $CPP -c $?

$(PROJECT) : $(OBJECTS)
<TAB> $(CPP) $(CPPLAGS) -o $(@) $(OBJECTS)
```

make V

Usual run of a `make` call:

1. after calling `make` the `makefile` is parsed (read)
2. read and substitute variables (see below) and determination of the highest target(s) (given in the beginning), evaluation of the dependencies
3. creation of a *tree* of dependencies
4. determination of the time stamps for all dependencies of the corresponding files and comparison with those of the next step in the tree
5. targets whose dependencies are newer than the targets are re-compiled

make VI

Variables

- during processing of the rules `make` uses automatic variables, e.g., `$@` and `$?` (see above)
- variables can also be defined explicitly before the first rule, syntax is shell-like:

```
CC = gcc
CFLAGS = -O3
PROJECT = galaxy
```

- variables can, as in the shell, be held together with help of curly braces `${OBJECTFILES}`, or alternatively with help of round parentheses `$(CFLAGS)`

make VII

Usual pseudo targets → Call via `make` *pseudo target*

- don't create a file, or don't have dependencies, e.g.
- `clean`, for `make clean`, defines explicitly how the intermediate and final products (targets) of the compilation shall be removed
- `all` creates all project files
- `install` if the targets (programs, libraries) shall be copied to a specific directory (or similar), it should be stated in `install`

Pseudo targets (e.g., `clean`) can only be used if defined in the `makefile`.

make VIII

Example of a makefile

```
CXX = g++ -O3
CPFLAGS = -Wall
LIBRARIES = -lX11

OBJECTS = componentA.o componentB.o
PROJECT = myprogram

$(PROJECT) : $(OBJECTS)
        ${CXX} $(CPFLAGS) $(OBJECTS) -o $@ ${LIBRARIES}

.cpp.o :
        ${CXX} -c ${CPFLAGS} $?

clean :
        rm -f $(OBJECTS)
```

make IX

`Makefile` uses a shell-like syntax:

- comments are started with a #:

  ```
  # a comment
  ```

- one command per line, multiple commands via ; and line continuation via \

  ```
  $FC $?  ; ldconfig
  ```

- every command corresponds to a shell command, and is printed before execution:

  ```
  .c.o :
  	echo "Hello ${USER}"
  ```

  the print-out of commands can be suppressed with @ before the command

  ```
  	@echo "Hi ${DATE}"
  ```

make X

- variables are set without \$ and used/referenced with a \$

```
progname = opdat
PROJECT = $(progname).exe
```

Variable names that contain multiple characters should always be held together with parentheses () or curly braces {}.

make XI

Special targets:

- problem: pseudo target `clean` is not executed, if a *file* with that name exists (why?)
- solution: pseudo targets can be marked as such via the *special target* `.PHONY`:

  ```
  .PHONY: clean install
  ```

- special targets start with a `.`

make XII

Some more special targets:

- `.INTERMEDIATE` : dependencies are only created if another dependency before the target is newer, or if a dependency of an intermediate file is newer than the actual target. The intermediate target is deleted after the target was created:

  ```
  .INTERMEDIATE : colortable.o

  xapple.exe : xapple.cpp colortable.o
          $(CXX) -o xapple.exe xapple.cpp colortable.o

  colortable.o : colortable.cpp
          $(CXX) -c colortable.cpp
  ```

  Here, `colortable.o` is only created if `xapple.cpp` or if `colortable.cpp` are newer than `xapple.exe`. After the creation of `xapple.exe` the target `colortable.o` will be removed.

make XIII

- `.SECONDARY` : like `.INTERMEDIATE`, but the dependencies are not removed automatically
- `.IGNORE` : errors during creation of the specified dependencies will not lead to an abort of the make procedure

Hint

The tool `make` is not bound to programming languages, but can also be used for, e.g., automatic compilation of `.tex` files etc.

Advantage of using `make`

A `Makefile`

- can save compilation time
- documents the compiler options and necessary files of the project

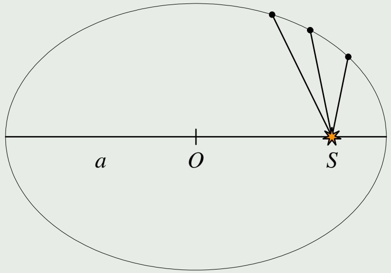
The two-body problem

Equations of motion I

We remember (?)

The Kepler's laws of planetary motion (1619)

1. Each planet moves in an elliptical orbit where the Sun is at one of the foci of the ellipse.
2. The velocity of a planet increases with decreasing distance to the Sun such, that the planet sweeps out equal areas in equal times. (*Consequence of which law?*)
3. The ratio P^2/a^3 is the same for all planets orbiting the Sun, where P is the orbital period and a is the semimajor axis of the ellipse. (*What defines value of ratio?*)



The 1. and 3. Kepler's law describe the shape of the orbit (Copernicus: circles), but not the time dependence $\vec{r}(t)$. This can in general not be expressed *analytically* by elementary mathematical functions (see below).
Therefore we will try to find a *numerical* solution.

Equations of motion II

Earth-Sun system

Step 1: $\rightarrow$ two-body problem $\rightarrow$ one-body problem via reduced mass of lighter body (partition of motion) via Newton's 3. & 2. law:

$$\vec{F}_{12} = -\vec{F}_{21} \Rightarrow m_1 \vec{a}_1 = -m_2 \vec{a}_2 \Rightarrow \vec{a}_2 = -\frac{m_1}{m_2}\vec{a}_1 \tag{25}$$

$$\vec{a}_{\rm rel} := \vec{a}_1 - \vec{a}_2 = \left(1 + \frac{m_1}{m_2}\right)\vec{a}_1 = \frac{m_2 + m_1}{m_1 m_2} m_1 \vec{a}_1 = \mu^{-1}\vec{F}_{12} \tag{26}$$

$$= \frac{d^2\vec{x}_{\rm rel}}{dt^2} = \frac{d^2}{dt^2}(\vec{x}_1 - \vec{x}_2) \tag{27}$$

$$\Rightarrow \mu = \frac{M\,m}{m + M} = \frac{m}{\frac{m}{M} + 1} \tag{28}$$

as $m_{\rm E} \ll M_\odot$ is $\mu \approx m$, i.e. motion is relative to the center of mass $\equiv$ only motion of m. Set point of origin $(0, 0)$ to the source of the force field of M.

Equations of motion III

Hence: Newton's 2. law (with $m \approx \mu$):

$$m\frac{d^2\vec{r}}{dt^2} = \vec{F} \tag{29}$$

$$m\frac{d^2}{dt^2}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} \tag{30}$$

and force field according to Newton's law of gravitation :

$$\vec{F} = -\frac{GMm}{r^3}\,\vec{r} \tag{31}$$

$$\begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} = -\frac{GMm}{r^3}\begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{32}$$

Equations of motion IV

Kepler's laws, as well as the assumption of a *central force* imply $\rightarrow$ *conservation of angular momentum* $\rightarrow$ motion is only in a *plane* ($\rightarrow$ Kepler's 1st law).
So, we use Cartesian coordinates in the xy-plane:

$$F_x = -\frac{GMm}{r^3}x \tag{33}$$

$$F_y = -\frac{GMm}{r^3}y \tag{34}$$

The equations of motion are then:

$$\frac{d^2x}{dt^2} = -\frac{GM}{r^3}x \tag{35}$$

$$\frac{d^2y}{dt^2} = -\frac{GM}{r^3}y \tag{36}$$

$$\text{where} \quad r = \sqrt{x^2+y^2} \tag{37}$$

Excursus: Analytic solution of the Kepler problem I

To derive the *analytic* solution for equation of motion $\vec{r}(t) \rightarrow$ use polar coordinates: ϕ, r

1. use conservation of angular momentum ℓ:

$$\mu r^2 \dot{\phi} = \ell = \text{const.} \tag{38}$$

$$\dot{\phi} = \frac{\ell}{\mu r^2} \tag{39}$$

2. use conservation of total energy $(\vec{v} = \dot{r}\vec{e}_r + r\dot{\phi}\vec{e}_\phi \rightarrow E_{\text{kin}} = \frac{\mu}{2}(\dot{r}^2 + r^2\dot{\phi}^2))$:

$$E = \frac{1}{2}\mu\dot{r}^2 + \frac{\ell^2}{2\mu r^2} - \frac{GM\mu}{r} \tag{40}$$

$$\dot{r}^2 = \frac{2E}{\mu} - \frac{\ell^2}{\mu^2 r^2} + \frac{2GM}{r} \tag{41}$$

$\rightarrow$ two coupled equations for r and ϕ

Excursus: Analytic solution of the Kepler problem II

3. decouple Eq. (39), use the orbit equation $r = \frac{\alpha}{1+e\cos\phi}$ with numeric eccentricity e ($= \overline{f_1\,O}/a$, *Value for circle?*) and $\alpha \equiv \frac{\ell^2}{GM\mu^2}$ gives separable equation for $\dot\phi$

$$\dot\phi = \frac{d\phi}{dt} = \frac{G^2M^2\mu^3}{\ell^3}(1+e\cos\phi)^2 \tag{42}$$

$$t = \int_{t_0}^{t} dt' = k\int_{\phi_0}^{\phi}\frac{d\phi'}{(1+e\cos\phi')^2} = f(\phi) \tag{43}$$

right-hand side integral can be looked up in, e.g., Bronstein:

$$t/k = \frac{e\sin\phi}{(e^2-1)(1+e\cos\phi)} - \frac{1}{e^2-1}\int\frac{d\phi}{1+e\cos\phi} \tag{44}$$

$\rightarrow e \neq 1$: parabola excluded; the integral can be further simplified
for the hyperbola ($e > 1$):

$$\int\frac{d\phi}{1+e\cos\phi} = \frac{1}{\sqrt{e^2-1}}\ln\frac{(e-1)\tan\frac{\phi}{2}+\sqrt{e^2-1}}{(e-1)\tan\frac{\phi}{2}-\sqrt{e^2-1}} \tag{45}$$

Excursus: Analytic solution of the Kepler problem III

for the ellipse ($0 \leq e < 1$):

$$\int \frac{d\phi}{1 + e\cos\phi} = \frac{2}{\sqrt{1-e^2}} \arctan \frac{(1-e)\tan\frac{\phi}{2}}{\sqrt{1-e^2}} \tag{46}$$

$\rightarrow$ Eq. (44) with Eqn. (46) & (45): $t(\phi)$ must be inverted to get $\phi(t)$!

(e.g., by numeric root finding)

$\rightarrow$ only easy for $e = 0 \rightarrow$ circular orbit

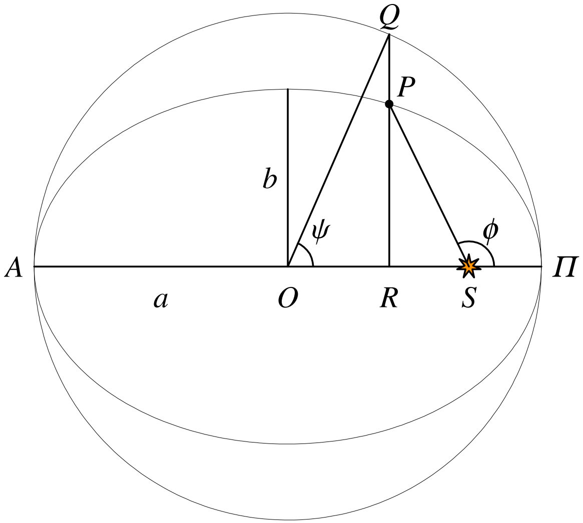
$$t = k \int d\phi' = k\phi \rightarrow \phi(t) = k^{-1} t = \frac{G^2 M^2 \mu^3}{\ell^3} t \tag{47}$$

and from orbit equation (for $e = 0$) $r = \alpha = \frac{\ell^2}{GM\mu^2} = \text{const.}$

For the general case, it is much easier to solve the equations of motion numerically.

Excursus: The Kepler equation I

Alternative formulation for time dependency in case of an ellipse ($0 \leq e < 1$):



Orbit, circumscribed by auxiliary circle with radius a ($=$semi-major axis); true anomaly ϕ, eccentric anomaly ψ. Sun at S, planet at P, circle center at O. Perapsis (perhelion) Π and apapsis (aphelion) A:

- consider a line normal to $\overline{A\Pi}$ through P on the ellipse, intersecting circle at Q and $\overline{A\Pi}$ at R.
- consider an angle ψ (or E, eccentric anomaly) defined by $\angle \Pi O Q$

Excursus: The Kepler equation II

Then: position in polar coordinates (r, ϕ) of the body P can be described in terms of ψ:

$$x_S(P) = r\cos\phi = a\cos\psi - ae \qquad (ae = \overline{OS}) \tag{48}$$

$$y_S(P) = r\sin\phi = a\sin\psi\sqrt{1-e^2} \qquad (= \overline{PR} = \overline{QR}\sqrt{1-e^2} = a\sin\psi\sqrt{1-e^2}) \tag{49}$$

(with $\overline{PR}/\overline{QR} = b/a = \sqrt{1-e^2}$), square both equations and add them up:

$$r = a(1 - e\cos\psi) \tag{50}$$

Now, to find $\psi = \psi(t)$, need relationship between $d\phi$ and $d\psi$, so combine Eqn. (49) & (50)

$$\sin\phi = \frac{b\sin\psi}{a(1-e\cos\psi)} \quad |d/dt \text{ \& quotient rule } \left(\frac{u}{v}\right)' = \frac{u'v - v'u}{v^2} \tag{51}$$

$$\cos\phi d\phi = \frac{b}{a}\frac{(\cos\psi(1-e\cos\psi)d\psi - e\sin^2\psi d\psi)}{(1-e\cos\psi)^2} \tag{52}$$

$$d\phi = \frac{b}{a(1-e\cos\psi)}d\psi \tag{53}$$

Excursus: The Kepler equation III

together with the angular momentum $d\phi = \frac{\ell}{\mu r^2} dt$, where r is replaced by Eq. (50):

$$(1 - e\cos\psi)d\psi = \frac{\ell}{\mu ab} dt \tag{54}$$

$$= \text{set } t = 0 \rightarrow \psi(0) = 0, \text{ integration:} \tag{55}$$

$$\psi - e\sin\psi = \frac{\ell t}{\mu ab} \tag{56}$$

use Kepler's 2nd law $\frac{\pi ab}{P} = \frac{\ell}{2\mu}$ with πab the area of the ellipse, we get $\ell/(\mu ab) = 2\pi/P \equiv \omega$ (orbital angular frequency), so:

Kepler's equation for the eccentric anomaly ψ (or E)

$$\psi - e\sin\psi = \omega t \tag{57}$$

$$E - e\sin E = M \quad \text{(astronomer's version)} \tag{58}$$

M: mean anomaly = angle for constant angular velocity $= 2\pi \dfrac{t - t_\Pi}{P}$

Excursus: The Kepler equation IV

Kepler's equation $E(t) - e \sin E(t) = M(t)$

- is a transcendental equation for the eccentric anomaly $E(t)$
- can be solved by, e.g., Newton's method
- because of $E = M + e \sin E$, also (Banach) fixed-point iteration possible (slow, but stable), already used by Kepler (1621):

```
E = M ;
for (int i = 0 ; i < n ; ++i)
   E = M + e * sin(E) ;
```

- can be solved, e.g., by Fourier series → Bessel (1784-1846):

$$E = M + \sum_{n=1}^{\infty} \frac{2}{n} J_n(ne) \sin(nM) \quad (59)$$

$$J_n(ne) = \frac{1}{\pi} \int_0^{\pi} \cos(nx - ne \sin x) dx \quad (60)$$

Circular orbits

A special case as a solution of the equations of motion (35) & (36) is the circular orbit. Then:

$$\ddot{r} = \frac{v^2}{r} \tag{61}$$

$$\frac{mv^2}{r} = \frac{GMm}{r^2} \quad \text{(equilibrium of forces)} \tag{62}$$

$$\Rightarrow \quad v = \sqrt{\frac{GM}{r}} \tag{63}$$

The relation (63) is therefore the condition for a circular orbit.
Moreover, Eq. (63) yields together with

$$P = \frac{2\pi r}{v} \tag{64}$$

$$\Rightarrow \quad P^2 = \frac{4\pi^2}{GM} r^3 \tag{65}$$

Astronomical units

For our solar system it is useful to use astronomical units (AU):

$$1\,\mathrm{AU} = 1.496 \times 10^{11}\,\mathrm{m}$$

and the unit of time is the (Earth-) year

$$1\,\mathrm{a} = 3.156 \times 10^{7}\,\mathrm{s} \quad (\approx \pi \times 10^{7}\,\mathrm{s}),$$

so, for the Earth $P = 1\,\mathrm{a}$ and $r = 1\,\mathrm{AU}$
Therefore it follows from Eq. (65):

$$GM = \frac{4\pi^2 r^3}{P^2} = 4\pi^2\,\mathrm{AU}^3\,\mathrm{a}^{-2} \tag{66}$$

I.e. we set $GM \equiv 4\pi^2$ in our calculations.
Advantage: handy numbers!
Thus, e.g. $r = 2$ is approx. 3×10^{11} m and $t = 0.1$ corresponds to 3.16×10^{6} s, and $v = 6.28$ is roughly 30 km/s.
cf.: our `rcalc` program with "solar units" for R, T, L; natural units in particle physics
$\hbar = c = k_\mathrm{B} = \epsilon_0 = 1 \rightarrow$ unit of m, p, T is eV (also for E)

The Euler method I

The equations of motion (35) & (36):

$$\frac{d^2\vec{r}}{dt^2} = -\frac{GM}{r^3}\vec{r} \tag{67}$$

are a system of differential equations of 2nd order, that we shall solve now.
Formally: *integration* of the equations of motion to obtain the *trajectory* $\vec{r}(t)$.

Step 1: reduction

Rewrite Newton's equations of motion as a system of differential equations of *1st order* (here: 1d):

$$v(t) = \frac{dx(t)}{dt} \quad \& \quad a(t) = \frac{dv(t)}{dt} = \frac{F(x,v,t)}{m} \tag{68}$$

The Euler method II

Step 2: Solving the differential equation

Differential equations of the form (initial value problem)

$$\frac{dx}{dt} = f(x,t), \quad x(t_0) = x_0 \tag{69}$$

can be solved numerically (discretization[1]) by as simple method:

Explicit Euler method (“Euler’s polygonal chain method”)

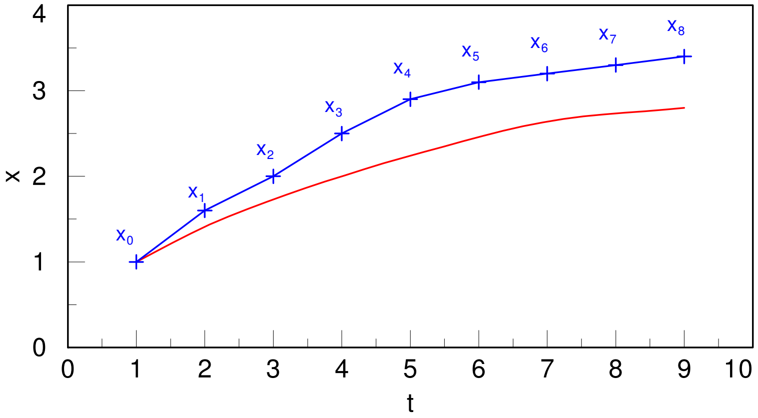
1. choose step size $\Delta t > 0$, so that $t_n = t_0 + n\Delta t, \quad n = 0, 1, 2, \ldots$
2. calculate the values (iteration):
   $x_{n+1} = x_n + f(x_n, t_n)\Delta t$ where $x_n = x(t_n)$ etc.

Obvious: The smaller the step size Δt, the more steps are necessary, but also the more accurate is the result.

[1] I.e. we change from calculus to algebra, which can be solved by computers.

The Euler method III

Why "polygonal chain method"?



Exact solution (🟥) and numerical solution (🟦).

The Euler method IV

Derivation from the Fundamental theorem of calculus

$$\text{integration of the ODE} \quad \frac{dx}{dt} = f(x,t) \quad \text{from } t_0 \text{ till } t_0 + \Delta t \tag{70}$$

$$\int_{t_0}^{t_0+\Delta t} \frac{dx}{dt} dt = \int_{t_0}^{t_0+\Delta t} f(x,t) dt \tag{71}$$

$$\Rightarrow x(t_0 + \Delta t) - x(t_0) = \int_{t_0}^{t_0+\Delta t} f(x(t),t) dt \tag{72}$$

apply rectangle rule for the integral:

$$\int_{t_0}^{t_0+\Delta t} f(x(t),t) dt \approx \Delta t\, f(x(t_0),t_0) \tag{73}$$

Equating (72) with (73) yields Euler step

$$x(t_0 + \Delta t) = x(t_0) + \Delta t\, f(x(t_0),t_0) \tag{74}$$

The Euler method V

Derivation from Taylor expansion

$$x(t_0 + \Delta t) = x(t_0) + \Delta t\, \frac{dx}{dt}(t_0) + \mathcal{O}(\Delta t^2) \tag{75}$$

$$\text{use} \quad \frac{dx}{dt} = f(x, t) \tag{76}$$

$$x(t_0 + \Delta t) = x(t_0) + \Delta t\, f(x(t_0), t_0) \tag{77}$$

while neglecting term of higher order in Δt

(*In which step did we neglect these higher order terms in the derivation from the fundamental theorem of calculus?*)

The Euler method VI

For the system Eqn. (68)

$$v(t) = \frac{dx(t)}{dt} \quad \& \quad a(t) = \frac{dv(t)}{dt} = \frac{F(x,v,t)}{m}$$

this means

Euler method for solving Newton's equations of motion

$$v_{n+1} = v_n + a_n \Delta t = v_n + a_n(x_n, t)\Delta t \tag{78}$$

$$x_{n+1} = x_n + v_n \Delta t \tag{79}$$

We note:

- the velocity at the end of the time interval v_{n+1} is calculated from a_n, which is the acceleration at the beginning of the time interval
- analogously x_{n+1} is calculated from v_n

The Euler method VII

Example: Harmonic oscillator $F = ma = -kx$

```
#include <iostream>
#include <cmath>
using namespace std ;
// set k = m = 1
int main () {
  int n = 10001, nout = 500 ;
  double t, v, v_old, x ;
  double const dt = 2. * M_PI / double(n-1) ;
  x = 1. ; t = 0. ; v = 0. ;
  for (int i = 0 ; i < n ; ++i) {
    t = t + dt ; v_old = v ;
    v = v - x * dt ;
    x = x + v_old * dt ;
    if (i % nout == 0) // print out only each nout step
      cout << t << " " << x << " " << v << endl ;
  }
  return 0 ;
}
```

The Euler-Cromer method

We will slightly modify the explicit Euler method, but such that we obtain the same differential equations for $\Delta t \to 0$.
For this new method we use v_{n+1} for calculating x_{n+1}:

Euler-Cromer method (semi-implicit Euler method)

$$v_{n+1} = v_n + a_n \Delta t \quad \text{(as for Euler)} \tag{80}$$

$$x_{n+1} = x_n + v_{n+1} \Delta t \tag{81}$$

Advantage of this method:

- as for Euler method, x, v need to be calculated only once per step
- especially appropriate for oscillating solutions, as energy is conserved much better (see below)

Excursus: Proof of stability for the Euler-Cromer method I

Proof of stability (Cromer 1981):

$$v_{n+1} = v_n + F_n \Delta t \quad (= v_n + a(x_n)\Delta t,\ m = 1) \tag{82}$$
$$x_{n+1} = x_n + v_{n+1}\Delta t \tag{83}$$

Without loss of generality, let $v_0 = 0$. Iterate Eq. (82) n times:

$$v_n = (F_0 + F_1 + \ldots + F_{n-1})\Delta t = S_{n-1} \tag{84}$$
$$x_{n+1} = x_n + S_n \Delta t \tag{85}$$
$$S_n := \Delta t \sum_{j=0}^{n} F_j \tag{86}$$

Note that for explicit Euler Eq. (85) is $x_{n+1} = x_n + S_{n-1}\Delta t$.

Excursus: Proof of stability for the Euler-Cromer method II

The change in the kinetic energy K between $t_0 = 0$ and $t_n = n\Delta t$ is because of Eq. (82) and $v_0 = 0$

$$\Delta K_n = K_n - K_0 = K_n = \frac{1}{2} S_{n-1}^2 \tag{87}$$

The change in the potential energy U:

$$\Delta U_n = -\int_{x_0}^{x_n} F(x) dx \tag{88}$$

Excursus: Proof of stability for the Euler-Cromer method III

Now use the trapezoid rule for this integral

$$\Delta U_n = -\frac{1}{2}\sum_{i=0}^{n-1}(F_i + F_{i+1})(x_{i+1} - x_i) \tag{89}$$

$$= -\frac{1}{2}\Delta t\sum_{i=0}^{n-1}(F_i + F_{i+1})S_i \qquad (\rightarrow \text{ Eq. } 85) \tag{90}$$

$$= -\frac{1}{2}\Delta t^2\sum_{i=0}^{n-1}\sum_{j=0}^{i}(F_i + F_{i+1})F_j \qquad (\rightarrow \text{ Eq. } 86) \tag{91}$$

Excursus: Proof of stability for the Euler-Cromer method IV

As j runs from 0 to i (instead of $i-1$):
$\rightarrow \Delta U_n$ has same squared terms as ΔK_n, using $S_n = \Delta t \sum_{j=0}^{n} F_j$:

$$\Delta U_n = -\frac{1}{2}\Delta t^2 \left(\sum_{i=0}^{n-1} F_i^2 + \sum_{i=0}^{n-1}\sum_{j=0}^{i-1} F_i F_j + \sum_{i=1}^{n}\sum_{j=0}^{i-1} F_i F_j \right) \tag{92}$$

$$= -\frac{1}{2}\Delta t^2 \left(\sum_{i=0}^{n-1} F_i^2 + 2\sum_{i=0}^{n-1}\sum_{j=0}^{i-1} F_i F_j + F_n \sum_{j=0}^{i-1} F_j \right) \tag{93}$$

$$= -\frac{1}{2} S_{n-1}^2 - \frac{1}{2}\Delta t\, F_n S_{n-1} \tag{94}$$

Hence the total energy changes as

$$\Delta E_n = \Delta K_n + \Delta U_n = \frac{1}{2} S_{n-1}^2 - \frac{1}{2} S_{n-1}^2 - \frac{1}{2}\Delta t\, F_n S_{n-1} \tag{95}$$

$$= -\frac{1}{2}\Delta t\, F_n S_{n-1} = -\frac{1}{2}\Delta t\, F_n v_n \tag{96}$$

Excursus: Proof of stability for the Euler-Cromer method V

For oscillatory motion: $v_n = 0$ at turning points, $F_n = 0$ at equilibrium points
$\rightarrow \Delta E_n = -\frac{1}{2}\Delta t\, F_n v_n$ is 0 four times of each cycle $\rightarrow \Delta E_n$ oscillates with $T/2$.
As F_n and v_n are bound $\rightarrow \Delta E_n$ is bound, more important: average of ΔE_n over half a cycle (T)

$$\langle \Delta E_n \rangle = \frac{\Delta t^2}{T} \sum_{n=0}^{\frac{1}{2}T/\Delta t} F_n v_n \simeq \frac{\Delta t}{T} \int_0^{\frac{T}{2}} F\, v\, dt = \frac{\Delta t}{T} \int_{x(0)}^{x(\frac{T}{2})} F\, dx \tag{97}$$

$$= -\frac{\Delta t}{T} \left(U(T/2) - U(0)\right) = 0 \tag{98}$$

as U has same value at each turning point
$\rightarrow$ energy conserved on average with Euler-Cromer for oscillatory motion

$\square$

Excursus: Proof of stability for the Euler-Cromer method VI

For comparison: with explicit Euler method ΔE_n contains term $\sum_{i=0}^{n-1} F_i^2$ which increases monotonically with n and

$$\Delta E_n = -\frac{1}{8}\Delta t^2 \left(F_0^2 - F_n^2\right) \tag{99}$$

with $v_0 = 0 \rightarrow F_0^2 \geq F_n^2 \rightarrow \Delta E_n$ oscillates between 0 and $-\frac{1}{8}\Delta t^2 F_0^2$ per cylce.
Energy is bounded as for Euler-Cromer, but $\langle \Delta E_n \rangle \neq 0$

Stability analysis of the Euler method I

Consider the following ODE

$$\frac{dx}{dt} = -cx \tag{100}$$

with $c > 0$ and $x(t=0) = x_0$. Analytic solution is $x(t) = x_0 \exp(-ct)$. The explicit Euler method gives:

$$x_{n+1} = x_n + \dot{x}_n \Delta t = x_n - c x_n \Delta t = x_n(1 - c\Delta t) \tag{101}$$

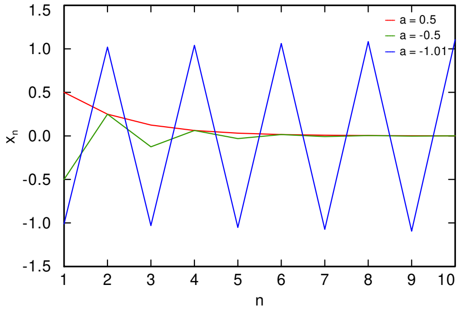
So, every step will give $(1 - c\Delta t)$ and after n steps:

$$x_n = (1 - c\Delta t)^n x_0 = (a)^n x_0 \tag{102}$$

But, with $a = 1 - c\Delta t$:

$$\begin{array}{rll} 0 < a < 1 & \Rightarrow \Delta t < 1/c & \text{monotonic decline of } x_n \text{ (correct)} \\ -1 < a < 0 & \Rightarrow 1/c < \Delta t < 2/c & \text{oscillating decline of } x_n \\ a < -1 & \Rightarrow \Delta t > 2/c & \text{oscillating increase of } x_n\,! \end{array} \tag{103}$$

Stability analysis of the Euler method II



Stability of the explicit Euler method for different $a = 1 - c\Delta t$

In contrast, consider implicit Euler method (Euler-Cromer):

$$x_{n+1} = x_n + \dot{x}_{n+1}\Delta t = x_n - c x_{n+1}\Delta t \tag{104}$$

$$\Rightarrow x_{n+1} = \frac{x_n}{1 + c\Delta t} \tag{105}$$

declines for all Δt (!)

Higher-Order Taylor series method I

In Taylor approximation Eq. (463) for $x' = f(x,t)$ we neglected terms of $\mathcal{O}(\Delta t^2)$:

$$x(t_0 + \Delta t) = x(t_0) + \Delta t\, x'(t_0) + \frac{\Delta t^2}{2!} x''(t_0) + \frac{\Delta t^3}{3!} x^{(3)}(t_0) + \frac{\Delta t^4}{4!} x^{(4)}(\zeta_0) \tag{106}$$

with $t_0 < \zeta_0 < t_1$, nectlect this term, then difference equation:

$$\rightarrow\ x(t_0 + \Delta t) = x(t_0) + \Delta t\, f(x_0, t_0) + \frac{\Delta t^2}{2} f'(x_0, t_0) + \frac{\Delta t^3}{6} f''(x_0, t_0) \tag{107}$$

Using chain rule for f' with partial derivatives f_t etc.:

$$x' = f(x,t) \tag{108}$$

$$x'' = f' = f_t \frac{dt}{dt} + f_x\, x' = f_t + f_x f \tag{109}$$

$$x^{(3)} = f'' = f_{tt} + 2 f_{tx} f + f_{xx} f^2 + f_t f_x + f_x^2 f \tag{110}$$

$\rightarrow$ replace f', f'' in Eq. (107) $\rightarrow$ third-order Taylor's method
problem: compute and find partial derivatives of f (for Newton: $\partial_{x,v,t} F(x,v,t)$)

Higher-Order Taylor series method II

Hence: replace $\sum_j^p \frac{\Delta t^j}{j!} f^{(j-1)}(t_n, x_n)$ with some function $ak_1 + bk_2$:

$$x_{n+1} = x_n + ak_1 + bk_2 \tag{111}$$
$$k_1 = \Delta t\, f(t_n, x_n) \tag{112}$$
$$k_2 = \Delta t\, f(t_n + \alpha \Delta t, x_n + \beta k_1) \tag{113}$$

and determine constants a, b, α, β so that error in Eq. (111) is minimum
$\rightarrow$ Eq. (111) $\hat{=}$ Taylor series:

$$x_{n+1} = x_n + \Delta t\, f(t_n, x_n) + \frac{\Delta t^2}{2} f'(t_n, x_n) + \ldots \tag{114}$$
$$\text{with } f' = f_t + f_x f \ : \tag{115}$$
$$x_{n+1} = x_n + \Delta t\, f + \frac{\Delta t^2}{2}(f_t + f_x f) + \mathcal{O}(\Delta t^3) \tag{116}$$

Now, Taylor expansion of $f(t_n + \alpha \Delta t, x_n + \beta k_1)$:

$$f(t_n + \alpha \Delta t, x_n + \beta k_1) = f(t_n, x_n) + \alpha \Delta t\, f_t + \beta k_1 f_x + \mathcal{O}(\Delta t^2) \tag{117}$$

Higher-Order Taylor series method III

$\rightarrow$ combine Eq. (117) with Eqn. (111 - 113)

$$\begin{aligned} x_{n+1} &= x_n + ak_1 + bk_2 = a\Delta t\, f(t_n, x_n) + b\Delta t\, f(t_n + \alpha\Delta t, x_n + \beta k_1) && (118)\\ &= x_n + \Delta t(a+b)f + b\Delta t^2(\alpha f_t + \beta f_x\, f) && (119)\\ &\overset{!}{=} x_n + \Delta t\, f + \frac{\Delta t^2}{2}(f_t + f_x f) \quad (\rightarrow \text{Eq. } (463)) && (120)\\ &\Rightarrow \quad a + b = 1 \quad \& \quad \alpha = \beta = \frac{1}{2b} && (121) \end{aligned}$$

$\rightarrow$ 3 equations for 4 unknowns $\rightarrow$ one variable can be chosen arbitrarily, e.g.,

$$a = b = \frac{1}{2} \quad \& \quad \alpha = \beta = 1 \tag{122}$$

$\rightarrow$ modified Euler method (so-called Runge-Kutta method of order 2)

$$x_{n+1} = x_n + \frac{1}{2}(k_1 + k_2) = x_n + \frac{1}{2}(\Delta t\, f(t_n, x_n) + \Delta t\, f(t_n + \Delta t, x_n + \Delta t\, k_1)) \tag{123}$$

$$x_{n+1} = x_n + \frac{\Delta t}{2}(f(t_n, x_n) + f(t_n + \Delta t, x_n + \Delta t\, f(t_n, x_n))) \tag{124}$$

(analogously: construct RK4-method $\rightarrow$ see later)

Higher-Order Taylor series method IV

Alternative choice: $\alpha = \beta = 1/2, a = 0, b = 1 \rightarrow$ midpoint method

$$x_{n+1} = x_n + ak_1 + bk_2 \tag{125}$$

$$= x_n + k_2 = x_n + \Delta t\, f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_1\right) \tag{126}$$

$$x_{n+1} = x_n + \Delta t\, f\left(t_n + \frac{\Delta t}{2}, x_n + \frac{\Delta t}{2} f(t_n, x_n)\right) \tag{127}$$

$\rightarrow$ also known as Euler-Richardson method

The Euler-Richardson method

Sometimes it is better, to calculate the velocity for the midpoint of the interval:

Euler-Richardson method ("Euler half step method")

$$a_n = F(x_n, v_n, t_n)/m \tag{128}$$

$$v_{\mathrm{M}} = v_n + a_n \frac{1}{2}\Delta t \tag{129}$$

$$x_{\mathrm{M}} = x_n + v_n \frac{1}{2}\Delta t \tag{130}$$

$$a_{\mathrm{M}} = F\left(x_{\mathrm{M}}, v_{\mathrm{M}}, t_n + \frac{1}{2}\Delta t\right)/m \tag{131}$$

$$v_{n+1} = v_n + a_{\mathrm{M}}\Delta t \tag{132}$$

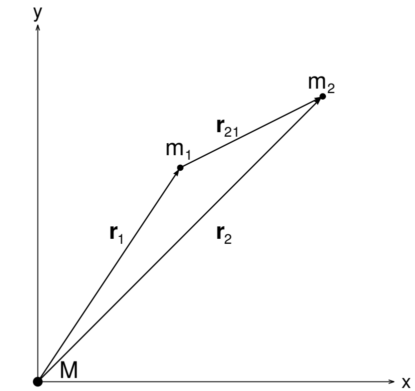
$$x_{n+1} = x_n + v_{\mathrm{M}}\Delta t \tag{133}$$

We need twice the number of steps of calculation, but may be more efficient, as we might choose a larger step size as for the Euler method.

The (special) three-body problem

The (special)[†] three-body problem I

We will not solve the general case of the three-body problem, but consider only the following configuration ($m_1, m_2 < M$):



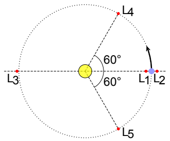
$$m_1 \frac{d^2 \vec{r}_1}{dt^2} = -\frac{GMm_1}{r_1^3}\vec{r}_1 + \frac{Gm_2 m_1}{r_{21}^3}\vec{r}_{21} \qquad (134)$$

$$m_2 \frac{d^2 \vec{r}_2}{dt^2} = -\frac{GMm_2}{r_2^3}\vec{r}_2 - \frac{Gm_1 m_2}{r_{21}^3}\vec{r}_{21} \qquad (135)$$

[†]not to be confused with the restricted three-body problem, where $m_1 \approx m_2 \gg m_3$

$\rightarrow$ Lagrangian points, e.g, L_1 for SOHO, L_2 for JWST

see also "The Three-Body Problem" (Novel)



The (special)† three-body problem II

It is useful to divide the Eqn. (134) & (135) each by m_1 and m_2 respectively:

$$\frac{d^2\vec{r}_1}{dt^2} = -\frac{GM}{r_1^3}\vec{r}_1 + \frac{Gm_2}{r_{21}^3}\vec{r}_{21} \quad (136)$$

$$\frac{d^2\vec{r}_2}{dt^2} = -\frac{GM}{r_2^3}\vec{r}_2 - \frac{Gm_1}{r_{21}^3}\vec{r}_{21} \quad (137)$$

Moreover we can set – using astronomical units – again:

$$GM \equiv 4\pi^2 \quad (138)$$

The terms

$$+\frac{Gm_2}{r_{21}^3}\vec{r}_{21} \quad \& \quad -\frac{Gm_1}{r_{21}^3}\vec{r}_{21} \quad (139)$$

can be written with help of mass ratios

$$\frac{m_2}{M} \quad \& \quad -\frac{m_1}{M} \quad (140)$$

The (special)[†] three-body problem III

so that

$$\texttt{ratio}[0] = \frac{m_2}{M} GM \quad \& \quad \texttt{ratio}[1] = -\frac{m_1}{M} GM \tag{141}$$

The *accelerations* are then calculated like this (in C/C++):

```
dx = x[1] - x[0]
    ...
dr3 = pow(dx * dx + dy * dy , 3./2. )
    ...
ax = -GM * x[i] / r3 + ratio[i] * dx / dr3
ay = -GM * y[i] / r3 + ratio[i] * dy / dr3
```

Methods for solving the Newtonian equations of motion

Numerical Integration I

Review $\rightarrow$ Newtonian equations of motion (2nd order ODE $\rightarrow$ reduction to 1st order)

$$\frac{dv}{dt} = a(t) \quad \& \quad \frac{dx}{dt} = v(t) \tag{142}$$

Numerical solution from Taylor expansion:

$$v_{n+1} = v_n + a_n \Delta t + \mathcal{O}((\Delta t)^2) \tag{143}$$

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 + \mathcal{O}((\Delta t)^3) \tag{144}$$

Euler method: account only for $\mathcal{O}(\Delta t)$ (for $\Delta t \rightarrow 0$):

$$v_{n+1} = v_n + a_n \Delta t \tag{145}$$

$$x_{n+1} = x_n + v_n \Delta t \tag{146}$$

therefore, only having $\mathcal{O}(\Delta t)$:
$\rightarrow$ local truncation error in one time step: $\sim (\Delta t)^2$
$\rightarrow$ global error over t: $\sim (\Delta t)$, because n steps with $n = \frac{t}{\Delta t} \sim \frac{1}{\Delta t}$,

Numerical Integration II

$\rightarrow$ so order of global error reduced by $\frac{1}{\Delta t}$

A method is of nth order, if the global (truncation) error is of the order of $(\Delta t)^n$.
The Euler method is of of 1st order.

Note, the Euler-Cromer method (semi-implicit Euler method) is also of 1st order, but conserves energy (symplectic integrator):

$$v_{n+1} = v_n + a_n(x_n)\Delta t \tag{147}$$
$$x_{n+1} = x_n + v_{n+1}\Delta t \tag{148}$$

but there is also a 2nd variant of the (semi-implicit) Euler method

$$x_{n+1} = x_n + v_n\Delta t \tag{149}$$
$$v_{n+1} = v_n + a_{n+1}(x_{n+1})\Delta t \tag{150}$$

$\rightarrow$ used for Verlet integration (see below Eqn. (160) & (161))

Numerical Integration III

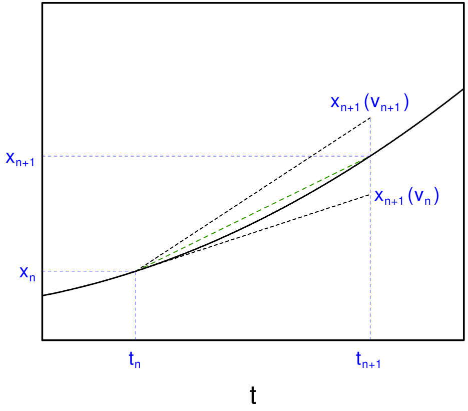
Possible improvement: use velocity from the *midpoint* of the interval

cf. Heun's method (Karl Heun, 1859-1929)

$$v_{n+1} = v_n + a_n \Delta t \quad \text{(as for Euler)} \tag{151}$$

$$x_{n+1} = x_n + \frac{1}{2}(v_n + v_{n+1})\Delta t \tag{152}$$

$$= x_n + v_n \Delta t + \frac{1}{2} a_n \Delta t^2 \tag{153}$$



→ accuracy of position is of 2nd order and velocity is of 1st order (only good for constant acceleration, not more accurate than Euler, as error increases with each time step)

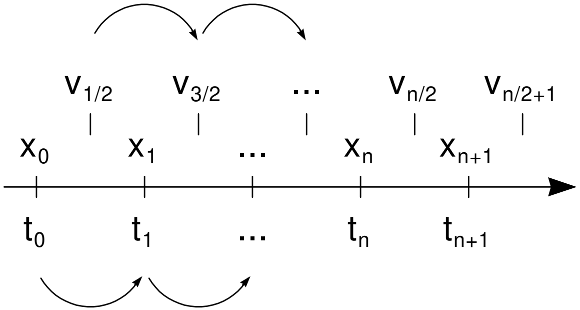
Numerical Integration IV

Better (stable for oscillatory motions with const. $\Delta t \leq 2/\omega$, therefore common, error bounded):

Halfstep method / Leapfrog integration

$$v_{n+\frac{1}{2}} = v_{n-\frac{1}{2}} + a_n \Delta t \tag{154}$$

$$x_{n+1} = x_n + v_{n+\frac{1}{2}} \Delta t \tag{155}$$



Numerical Integration V

$\rightarrow$ 2nd order with same number of steps as Euler (1st order), time-reversable, exact conservation of momenta, energy conserved up to 3rd order

But: not self starting, i.e. from Eq. (154) $\nrightarrow v_{\frac{1}{2}}$
therefore Euler method for the first half step:

$$v_{\frac{1}{2}} = v_0 + \frac{1}{2} a_0 \Delta t \tag{156}$$

Moreover, velocity steps can be eliminated by using Eq. (154) & (155):

$$(x_{n+1} - x_n) - (x_n - x_{n-1}) = (v_{n+\frac{1}{2}} - v_{n-\frac{1}{2}})\Delta t \tag{157}$$

$$x_{n+1} - 2x_n + x_{n-1} = a_n \Delta t \tag{158}$$

$$\rightarrow\ x_{n+1} = 2x_n - x_{n-1} + a_n \Delta t^2 \quad \text{(Størmer's method}^\dagger\text{)} \tag{159}$$

with start values $x_0, x_1 = x_0 + v_0 + \frac{1}{2} a_0(x_0)\Delta t^2$ (so v_0 is still required!)

† Carl Størmer (1874-1957), Norwegian physicist, theoretical description of aurora borealis $\rightarrow$ trajectories of charged particles in magnetic field

Numerical Integration VI

Or, by interpolation of intermediate values as combination of symplectic, semi-implicit Euler methods (Eq. (147)-(150))

$$\left.\begin{aligned} v_{n+\frac{1}{2}} &= v_n + a_n\,\tfrac{1}{2}\Delta t \\ x_{n+\frac{1}{2}} &= x_n + v_{n+\frac{1}{2}}\,\tfrac{1}{2}\Delta t \end{aligned}\right\} \quad (160) \qquad \left.\begin{aligned} x_{n+1} &= x_{n+\frac{1}{2}} + v_{n+\frac{1}{2}}\,\tfrac{1}{2}\Delta t \\ v_{n+1} &= v_{n+\frac{1}{2}} + a_{n+1}\,\tfrac{1}{2}\Delta t \end{aligned}\right\} \quad (161)$$

by substituting system (160) into system (161) one obtains Leapfrog with integer steps:

$$x_{n+1} = x_n + v_n\Delta t + \frac{1}{2}a_n\Delta t^2 \tag{162}$$

$$v_{n+1} = v_n + \frac{1}{2}(a_n + a_{n+1})\Delta t \tag{163}$$

$\rightarrow$ so-called Verlet integration† (see next slides)

†Loup Verlet (1931-2019), french physicist, pioneered computer simulations

Numerical Integration VII

Higher order methods

for that purpose: Taylor expansion of x_{n-1} (negative time step $-\Delta t$):

$$x_{n-1} = x_n - v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 - \mathcal{O}((\Delta t)^3) \tag{164}$$

$$+ \quad x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 + \mathcal{O}((\Delta t)^3) \tag{165}$$

$$= \quad x_{n+1} + x_{n-1} = 2x_n + a_n (\Delta t)^2 + \mathcal{O}((\Delta t)^4) \tag{166}$$

$$\Rightarrow \quad x_{n+1} = 2x_n - x_{n-1} + a_n (\Delta t)^2 \tag{167}$$

Analogously:

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 + \mathcal{O}((\Delta t)^3) \tag{168}$$

$$- \quad (x_{n-1} = x_n - v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 - \mathcal{O}((\Delta t)^3)) \tag{169}$$

$$= \quad x_{n+1} - x_{n-1} = 2 v_n \Delta t + \mathcal{O}((\Delta t)^3) \tag{170}$$

$$\Rightarrow \quad v_n = \frac{x_{n+1} - x_{n-1}}{2\Delta t} \qquad \text{(Verlet)} \tag{171}$$

Numerical Integration VIII

$\rightarrow$ method of 2nd order in v and 3rd order in x

But:

- not self starting (needs start values $x_0, x_1 = x_0 + v_0 + \frac{1}{2}a_0\Delta t^2$, see above)
- Eq. (171) contains differences of two values of same order of magnitude and expected $\Delta x \ll x \rightarrow$ round-off errors possible (subtractive cancelation)

Therefore, from Eq. (170)

$$x_{n-1} = x_{n+1} - 2v_n\Delta t \qquad \text{insert in Eq. (167):} \tag{172}$$

$$x_{n+1} = 2x_n - x_{n+1} + 2v_n\Delta t + a_n(\Delta t)^2 \tag{173}$$

Solve for x_{n+1}, yields:

Velocity Verlet

$$x_{n+1} = x_n + v_n\Delta t + \frac{1}{2}a_n(\Delta t)^2 \tag{174}$$

$$v_{n+1} = v_n + \frac{1}{2}(a_{n+1} + a_n)\Delta t \qquad \text{(see below for derivation)} \tag{175}$$

Numerical Integration IX

$\rightarrow$ self-starting
$\rightarrow$ minimizes round-off errors (no differences)
$\rightarrow$ 4th order in x (why? $\rightarrow$ Eq. (168) & (169))
Eq. (175) results from Eq. (171) for v_{n+1}:

$$\begin{aligned}
v_{n+1} &= \frac{x_{n+2} - x_n}{2\Delta t} && (176)\\
\text{and} \quad x_{n+2} &= 2x_{n+1} - x_n + a_{n+1}(\Delta t)^2 \quad \text{from Eq. (167)} && (177)\\
\Rightarrow \quad v_{n+1} &= \frac{2x_{n+1} - x_n + a_{n+1}(\Delta t)^2 - x_n}{2\Delta t} && (178)\\
&= \frac{x_{n+1} - x_n}{\Delta t} + \frac{1}{2}a_{n+1}(\Delta t)^2 \quad \& \ x_{n+1} \text{ from Eq. (174)} &&\\
&= \frac{x_n + v_n\Delta t + \frac{1}{2}a_n(\Delta t)^2 - x_n}{\Delta t} + \frac{1}{2}a_{n+1}\Delta t && (179)\\
&= v_n + \frac{1}{2}(a_{n+1} + a_n)\Delta t && (180)
\end{aligned}$$

Numerical Integration X

Alternatively:
(developed for liquid particles in a Lennard-Jones potential)

Beeman method (Schofield 1973; Beeman 1976)

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{6}(4a_n - a_{n-1})(\Delta t)^2 \quad (181)$$

$$v_{n+1} = v_n + \frac{1}{6}(2a_{n+1} + 5a_n - a_{n-1})\Delta t \quad (182)$$

$\rightarrow$ not self-starting
$\rightarrow$ locally: $\mathcal{O}(\Delta t)^4$ in x and $\mathcal{O}(\Delta t)^3$ in v, globally $\mathcal{O}(\Delta t)^3$
$\rightarrow$ better energy conservation than for Verlet, but more calculation steps

even better: $\rightarrow$ Runge-Kutta method of 4th order

The Runge-Kutta method

Runge-Kutta method of 4th order I

Remember:

Euler-Richardson method (Euler-halfstep method)

$$a_n = F(x_n, v_n, t_n)/m \tag{183}$$

$$v_\mathrm{M} = v_n + a_n \frac{1}{2}\Delta t \tag{184}$$

$$x_\mathrm{M} = x_n + v_n \frac{1}{2}\Delta t \tag{185}$$

$$a_\mathrm{M} = F\left(x_\mathrm{M}, v_\mathrm{M}, t_n + \frac{1}{2}\Delta t\right)/m \tag{186}$$

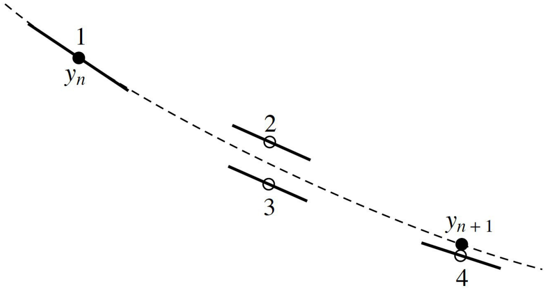
$$v_{n+1} = v_n + a_\mathrm{M}\Delta t \tag{187}$$

$$x_{n+1} = x_n + v_\mathrm{M}\Delta t \tag{188}$$

$\rightarrow$ calculation of F or a, respectively, for the whole step at the "midpoint" of the interval, instead of using the values from the beginning
$\equiv$ Runge-Kutta method *2nd order*

Runge-Kutta method of 4th order II

We will refine the halfstep method by using more supporting points:



Runge-Kutta method of 4th order III

With the Runge-Kutta method[†] the initial value problem

$$dy/dx = y' = f(x,y), \quad y(x_0) = y_0 \tag{189}$$

is solved by calculating approximate values y_i at selected supporting points x_i to obtain the wanted $y(x)$. These y_i are calculated with help of the following scheme (cf. Bronstein), where also only linear terms are calculated, but in form of a "polygonal line":

- supporting point at the beginning and at the end of the interval
- two additional supporting points in the middle of the interval with doubled weight

Derivation: include higher order terms in Taylor expansion, replace partial derivatives with coefficients to be determined ...

[†]Carl Runge (1856-1927), Wilhelm Kutta (1867-1944)

Runge-Kutta method of 4th order IV

Move from x_0 to $x_i = x_0 + ih$ (step size h, $i = 0, 1, 2, \ldots$) $\rightarrow$ single step method

x	y	$k = h \cdot f(x,y) = h \cdot dy/dx$
x_0	y_0	k_1
$x_0 + h/2$	$y_0 + k_1/2$	k_2
$x_0 + h/2$	$y_0 + k_2/2$	k_3
$x_0 + h$	$y_0 + k_3$	k_4
$x_1 = x_0 + h$	$y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$	

Cf.: Simpson's rule[†] (actually Kepler's rule , “Keplersche Fassregel”, 1615) for integration of $y'(x)$ via a parabola:

$$\int_a^b y'(x)dx \approx \frac{b-a}{6}\left(y'(a) + 4\,y'\left(\frac{a+b}{2}\right) + y'(b)\right) \tag{190}$$

[†] Thomas Simpson (1710-1761)

Runge-Kutta method of 4th order V

For the equations of motion this means therefore:

$$\vec{k}_{1v} = \vec{a}(\vec{x}_n, \vec{v}_n, t)\,\Delta t \quad (= \vec{a}_{\text{grav.}}(\vec{x}_n)\,\Delta t \quad \text{in our case}) \tag{191}$$

$$\vec{k}_{1x} = \vec{v}_n\,\Delta t \tag{192}$$

$$\vec{k}_{2v} = \vec{a}\left(\vec{x}_n + \frac{\vec{k}_{1x}}{2}, \vec{v}_n + \frac{\vec{k}_{1v}}{2}, t_n + \frac{\Delta t}{2}\right)\Delta t \tag{193}$$

$$\vec{k}_{2x} = \left(\vec{v}_n + \frac{\vec{k}_{1v}}{2}\right)\Delta t \tag{194}$$

$$\vec{k}_{3v} = \vec{a}\left(\vec{x}_n + \frac{\vec{k}_{2x}}{2}, \vec{v}_n + \frac{\vec{k}_{2v}}{2}, t_n + \frac{\Delta t}{2}\right)\Delta t \tag{195}$$

$$\vec{k}_{3x} = \left(\vec{v}_n + \frac{\vec{k}_{2v}}{2}\right)\Delta t \tag{196}$$

$$\vec{k}_{4v} = \vec{a}(\vec{x}_n + \vec{k}_{3x}, \vec{v}_n + \vec{k}_{3v}, t + \Delta t)\,\Delta t \tag{197}$$

$$\vec{k}_{4x} = (\vec{v}_n + \vec{k}_{3v})\,\Delta t \tag{198}$$

Runge-Kutta method of 4th order VI

So, finally

$$v_{n+1} = v_n + \frac{1}{6}(k_{1v} + 2k_{2v} + 2k_{3v} + k_{4v}) \quad (199)$$

$$x_{n+1} = x_n + \frac{1}{6}(k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x}) \quad (200)$$

$\rightarrow$ Runge-Kutta methods are self-starting

Numerical integration: Improvements I

Adaptive stepsize: step doubling

1. calculate new coordinates $(\vec{x}, \vec{v})$ via *two* Runge-Kutta steps each with Δt
2. calculate new coordinates $(\vec{x}, \vec{v})'$ via *one* Runge-Kutta step with $2\Delta t$

$\rightarrow$ calculation overhead increases only by 11/8, because of same derivatives on the beginning of the interval

Now, if

$$\frac{|(x, v) - (x, v)'|}{|(x, v)|} \geq \epsilon_{\text{max}} \tag{201}$$

with an accuracy criterion ϵ_{max} $\rightarrow$ decrease stepsize Δt
If

$$\frac{|(x, v) - (x, v)'|}{|(x, v)|} \leq \epsilon_{\text{min}} \quad \text{mit} \quad \epsilon_{\text{min}} < \epsilon_{\text{max}} \tag{202}$$

$\rightarrow$ increase Δt

Numerical integration: Improvements II

Predictor-corrector method

First *prediction* of the new position, e.g.:

$$x_{\mathrm{p}} = x_{n-1} + 2v_n\Delta t \tag{203}$$

$\rightarrow$ yields accleration a_{p} $\rightarrow$ *corrected* position by trapezoidal rule:

$$v_{n+1}^0 = v_n + \frac{1}{2}(a_{\mathrm{p}} + a_n)\Delta t \tag{204}$$

$$x_{n+1}^0 = x_n + \frac{1}{2}(v_{n+1} + v_n)\Delta t \tag{205}$$

$\rightarrow$ yields better value for a_{n+1} and hence

$$v_{n+1}^1 = v_n + a_{n+1}\Delta t \tag{206}$$

$$x_{n+1}^1 = x_n + v_{n+1}\Delta t \tag{207}$$

repeated iteration until $|x_{n+1}^{k+1} - x_{n+1}^{k}| < \epsilon$ with intended accuracy ϵ

Bulirsch-Stoer method I

Especially interesting for interactions of several bodys (*few-body problem*):

- resonances in planetary systems
- influence by one-time passage of a star
- influence of the galactic gravitational potential

$\rightarrow$ Requires:

- high numerical accuracy
- flexibility
- high computation rate

Idea: combination of

- modified midpoint method
- Richardson extrapolation
- extrapolation via rational functions

$\rightarrow$ Bulirsch-Stoer method (Stoer & Bulirsch 1980)†
cf. Numerical Recipes

† Roland Bulirsch (1932-2022), Josef Stoer (*1934)

Bulirsch-Stoer method II

1. Modified midpoint method

For an ODE $dx/dt = f(t,x)$ over a time step $H = Nh$ with N equidistant sub-steps

$$\begin{aligned} x_0 &= x(t) & (208)\\ x_1 &= x_0 + hf(t, x_0) & (209)\\ &\ldots & (210)\\ x_i &= x_{i-2} + 2hf(t + [i-1]h, x_{i-1}) \quad i = 2,\ldots,N & (211)\\ x(t+H) \approx \tilde{x} &= \frac{1}{2}[x_N + x_{N-1} + hf(t+H, x_N)] & (212) \end{aligned}$$

$\rightarrow$ 2nd order method, but with only one derivative per h-(sub)step
(where 2nd order Runge-Kutta has two derivatives per step)

Bulirsch-Stoer method III

Gragg[†] (1965): error in Eq. (212) $\rightarrow$ even power series:

$$\tilde{x} - x(t+H) = \sum_{i=1}^{\infty} \alpha_i h^{2i} \tag{213}$$

$\rightarrow$ for even N (so, $N = 2, 4, 6, \ldots$) all odd error terms cancel out $\rightarrow$ accuracy increases two orders at a time when combining two crossings of interval H with increasing N:

Let $x_{N/2}$ the result for $x(t+H)$ with half the number of steps:

$$x(t+H) \approx \frac{4\tilde{x}_N - \tilde{x}_{N/2}}{3} \tag{214}$$

$\rightarrow$ 4th order accuracy (as for RK4), but only with 1.5 derivatives
(RK4: 4)

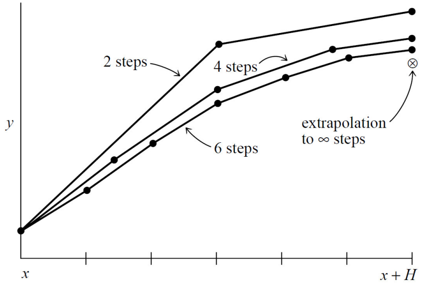
[†]William B. Gragg (1936-2016)

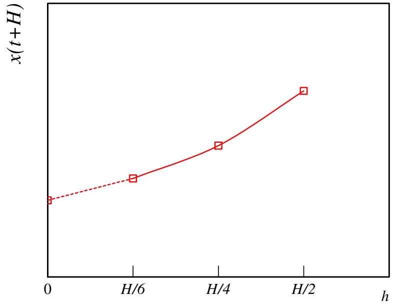
Bulirsch-Stoer method IV

2. Richardson extrapolation

Idea: result $x(t+H)$ is an analytic function of h with $h = H/N$:

1. calculate $x_{t+H}(h = 2, 4, 6, \ldots)$
2. fit function $x_{t+H}(h)$ to $x_{t+H}(N = 2)$, $x_{t+H}(N = 4)$, $\ldots$
3. extrapolate $x_{t+H}(h \to 0)$, corresponding to $N \to \infty$





Bulirsch-Stoer method V

3. Extrapolation via polynomial

Compute k-times x_{t+H} with $N = 2, 4, 6, \ldots$:

$$x_{t+H}(h) = a_0 + a_1 h + a_2 h^2 + \ldots + a_k h^{k-1} \tag{215}$$

where following Lagrange

$$\begin{aligned} x_{t+H}(h) &= \frac{(h-h_2)(h-h_3)\ldots(h-h_k)}{(h_1-h_2)(h_1-h_3)\ldots(h_1-h_k)} x_{t+H}(h_1) && (216)\\ &+ \frac{(h-h_1)(h-h_3)\ldots(h-h_k)}{(h_2-h_1)(h_2-h_3)\ldots(h_2-h_k)} x_{t+H}(h_2) && (217)\\ &+ \ldots + \frac{(h-h_1)(h-h_2)\ldots(h-h_{k-1})}{(h_k-h_1)(h_k-h_2)\ldots(h_k-h_{k-1})} x_{t+H}(h_k) && (218) \end{aligned}$$

In the original Bulirsch-Stoer method: rational function $(P(h)/Q(h))$ instead of Lagrange polynomial $\rightarrow$ much better approximation for functions with poles

Regularization I

Consider an N-body system with

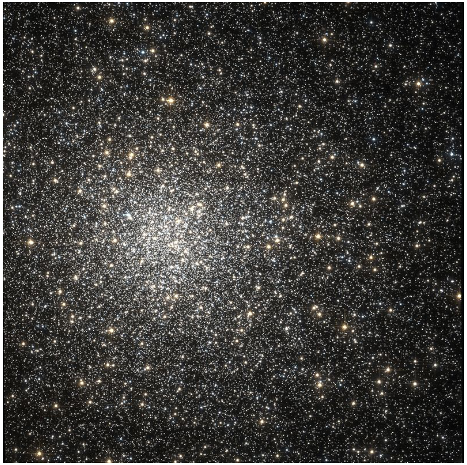
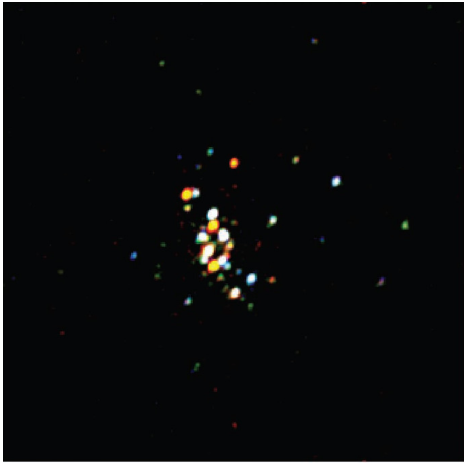
$$\frac{d^2\vec{x}_i}{dt^2} = -\sum_{j=1;j\neq i}^{N} \frac{Gm_j(\vec{x}_i - \vec{x}_j)}{|\vec{x}_i - \vec{x}_j|^3} = -\sum_{j=1;j\neq i}^{N} \frac{Gm_j(\vec{e}_i - \vec{e}_j)}{r_{ij}^2} \tag{219}$$

problem: $a_{ij} \propto \dfrac{1}{r_{ij}^2}$ for very small distances r_{ij} (close encounters)
$\rightarrow$ small distances $\rightarrow$ large accelerations $\rightarrow$ requires small Δt
$\rightarrow$ slows down calculations & increases numerical accumulation error

possibly uncomplicated for *one time* encounters

But in star clusters:
$\rightarrow$ formation of close binaries $\rightarrow$ *periodic*
so-called "binary hardening": transfer of the energy of the binary system to the cluster by consecutive close encounters

Regularization II

M 62 (NGC 6266). *Left:* optical HST. *Right:* X-ray CHANDRA

$\rightarrow$ above-average rate of close binary systems (e.g., low-mass X-ray binaries) in globular clusters (Pooley et al. 2003)

Regularization III

obvious (and inaccurate) idea: "softening" term in Eq. (219):

$$\vec{a}_i = \frac{G\, m_j(\vec{x}_i - \vec{x}_j)}{(\epsilon^2 + |\vec{x}_i - \vec{x}_j|^2)^{3/2}} \tag{220}$$

$$\text{such that} \tag{221}$$

$$\max|\vec{a}_i| = \frac{2G\, m_j}{3^{3/2}\,\epsilon^2} \quad \text{at } r = \frac{1}{\sqrt{2}}\epsilon \tag{222}$$

$\rightarrow$ adaptive Δt not arbitrarily small; but: close binary orbits and passages not resolvable

When is "softening" applicable?

$\rightarrow$ if close encounters are irrelevant
$\rightarrow$ collisionless systems, e.g., galaxy

Regularization IV

Illustration: distances in a galaxy

Galaxy: $\varnothing \approx 10^{23}\,\mathrm{cm}$ with 10^{11} stars with $R_* \approx 10^{11}\,\mathrm{cm} \rightarrow d \approx 10^{19}\,\mathrm{cm}$

10^{11} sand grains

$\rightarrow$ 100× width of

$\rightarrow$ average distance between sand grains $\approx 10\,\mathrm{km}$

$\rightarrow t_{*,\mathrm{coll}} \gg t_{\mathrm{Hubble}} \rightarrow$ collisionless

stars perceive only the average gravitational potential of the galaxy

Regularization V

Better: regularization (technique in physics to avoid ∞) with help of transformation of spacetime coordinates.
Consider vector $\vec{R}$ between two particles (center of mass frame):

$$\frac{d^2\vec{R}}{dt^2} = -G(m_1 + m_2)\frac{\vec{R}}{|\vec{R}|^3} + \vec{F}_{12} \tag{223}$$

with external force $\vec{F}_{12} = \vec{F}_1 - \vec{F}_2$ per mass, by other particles
1. regularized time τ

$$\begin{aligned} dt &= R^n d\tau & (224)\\ \frac{d^2}{dt^2} &= \frac{1}{R^{2n}}\frac{d^2}{d\tau^2} - \frac{n}{R^{2n+1}}\frac{dR}{d\tau}\frac{d}{d\tau} & (225)\\ \frac{d^2\vec{R}}{d\tau^2} &= \frac{n}{R}\frac{dR}{d\tau}\frac{d\vec{R}}{d\tau} - G(m_1+m_2)\frac{\vec{R}}{R^{3-2n}} + R^{2n}\vec{F}_{12} & (226) \end{aligned}$$

for $n = 1 \rightarrow R \propto dt/d\tau$ and without R^{-2}- singularity,
but with $\vec{R}/R$ term (indefinite for $R \rightarrow 0$)

Regularization VI

therefore:
2. regularized distance u, initially only for 1 dimension (already known by Euler), without external force (see Aarseth 2003):

$$\frac{d^2R}{d\tau^2} = \frac{1}{R}\left(\frac{dR}{d\tau}\right)^2 - G(m_1+m_2) \tag{227}$$

$$\tag{228}$$

and with *conservation of energy*, total energy h per reduced mass $\mu = m_1 m_2/(m_1+m_2)$:

$$h = \frac{1}{2}\left(\frac{dR}{dt}\right)^2 - \frac{G}{R}(m_1+m_2) \tag{229}$$

$\rightarrow h$ is fixed without external force, and with

$$\frac{dR}{dt} = \frac{1}{R}\frac{dR}{d\tau} \tag{230}$$

$$\Rightarrow \quad \frac{d^2R}{d\tau^2} = 2hR + G(m_1+m_2) \tag{231}$$

Regularization VII

$\rightarrow$ no more singularities. With $u^2 = R$:

$$\frac{d^2 u}{d\tau^2} = \frac{1}{2} h u \tag{232}$$

$\rightarrow$ harmonic oscilator (h is const.)
$\rightarrow$ easy to integrate
$\rightarrow$ method: change from (x, t) to (u, τ) below some distinct distance (for 1d collision!)

Regularization VIII

in 2 dimensions (Levi-Civita 1904)[†]:

$$\begin{aligned} x &= u_1^2 - u_2^2 && (233)\\ y &= 2u_1 u_2 && (234)\\ \text{or} \quad \vec{R} &= \mathcal{L}\vec{u} && (235)\\ \text{where } \mathcal{L} = \mathcal{L}(\vec{u}) &= \begin{pmatrix} u_1 & -u_2 \\ u_2 & u_1 \end{pmatrix} && (236) \end{aligned}$$

With the following properties:

$$\begin{aligned} \mathcal{L}(\vec{u})^T \mathcal{L}(\vec{u}) &= R\mathcal{I} && (237)\\ \frac{d}{dt}\mathcal{L}(\vec{u}) &= \mathcal{L}\left(\frac{d\vec{u}}{dt}\right) && (238)\\ \mathcal{L}(\vec{u})\vec{v} &= \mathcal{L}(\vec{v})\vec{u} && (239)\\ \vec{u}\cdot\vec{u}\mathcal{L}(\vec{v})\vec{v} - 2\vec{u}\cdot\vec{v}\mathcal{L}(\vec{u})\vec{v} &+ \vec{v}\cdot\vec{v}\mathcal{L}(\vec{u})\vec{u} = 0 && (240) \end{aligned}$$

[†] Tullio Levi-Civita (1873-1941), Italian mathematician and physicist

Regularization IX

With help of Eqn. (238 & 239) coordinates change to

$$\begin{aligned}\frac{d\vec{R}}{d\tau} &= 2\mathcal{L}(\vec{u})\frac{d\vec{u}}{d\tau} && (241)\\ \frac{d^2\vec{R}}{d\tau^2} &= 2\mathcal{L}(\vec{u})\frac{d^2\vec{u}}{d\tau^2} + 2\mathcal{L}\left(\frac{d\vec{u}}{d\tau}\right)\frac{d\vec{u}}{d\tau} && (242)\end{aligned}$$

Hence in Eq. (226) with $n = 1$ and with Eq. (240) and some transformations:

$$2\vec{u}\cdot\vec{u}\mathcal{L}(\vec{u})\frac{d^2\vec{u}}{d\tau^2} - 2\frac{d\vec{u}}{d\tau}\cdot\frac{d\vec{u}}{d\tau}\mathcal{L}(\vec{u})\vec{u} + G(m_1+m_2)\mathcal{L}(\vec{u})\vec{u} = (\vec{u}\cdot\vec{u})^3\vec{F}_{12} \quad (243)$$

further transformations lead to a form without singularities and indefinitenesses:

$$\frac{d^2\vec{u}}{d\tau^2} = \frac{1}{2}h\vec{u} + \frac{1}{2}R\mathcal{L}^T(\vec{u})\vec{F}_{12} \quad (244)$$

Regularization X

Binary star without external forces $\vec{F}_{12} \rightarrow$ energy h conserved
Binary star with external forces:

$$h = \left[2 \frac{d\vec{u}}{d\tau} \cdot \frac{d\vec{u}}{d\tau} - G(m_1 + m_2) \right] \Big/ R \tag{245}$$

The time evolution in usual coordinates

$$\frac{d}{dt} \left[\frac{1}{2} \left(\frac{dR}{dt} \right)^2 - \frac{G}{R}(m_1 + m_2) \right] = \frac{d\vec{R}}{dt} \cdot \vec{F}_{12} \tag{246}$$

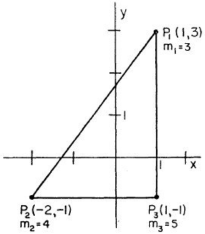
after transformation

$$\frac{dh}{d\tau} \quad = \quad 2 \frac{d\vec{u}}{d\tau} \cdot \mathcal{L}(\vec{u}) \vec{F}_{12} \tag{247}$$

can be solved continuously for $R = 0$ simultaneously with Eq. (244)

Regularization XI

Application of the 2d solution to the so-called Pythagoraian three-body problem ($\vec{L} = 0$) in Szebehely & Peters (1967):



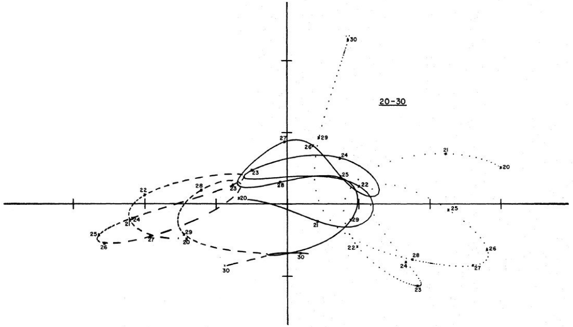


FIG. 4. Orbits between $t=20$ and $t=30$.

because of $\vec{L} = 0$ three-body collision possible $\rightarrow$ does not occur (3rd body gives perturbation $\vec{F}_{12}$)

Regularization XII

Regularization for 3 dimensions (Kustaanheimo & Stiefel 1965) requires transformation to 4d coordinates:

$$\begin{aligned}
R_1 &= u_1^2 - u_2^2 - u_3^2 + u_4^2 && (248)\\
R_2 &= 2(u_1 u_2 - u_3 u_4) && (249)\\
R_3 &= 2(u_1 u_3 + u_2 u_4) && (250)\\
R_4 &= 0 && (251)
\end{aligned}$$

and $\vec{R} = \mathcal{L}(\vec{u})\vec{u}$, such that

$$\mathcal{L} = \begin{bmatrix} u_1 & -u_2 & -u_3 & u_4 \\ u_2 & u_1 & -u_4 & -u_3 \\ u_3 & u_4 & u_1 & u_2 \\ u_4 & -u_3 & u_2 & -u_1 \end{bmatrix} \tag{252}$$

$\rightarrow$ yields again equations similar to (244) & (247)

see Bodenheimer et al. (2007) and Aarseth (2003)

N-body simulations for large N I

Problems:

1. number of interactions is $N(N-1)/2 \propto \mathcal{O}(N^2)$
2. multiple timescales for adaptive time steps for each particle i:

$$\Delta t_i \simeq k\sqrt{\frac{1}{|\vec{a}_i|}} \tag{253}$$

with acceleration $\vec{a}_i$ and small factor k

N-body simulations for large N II

possible solutions:

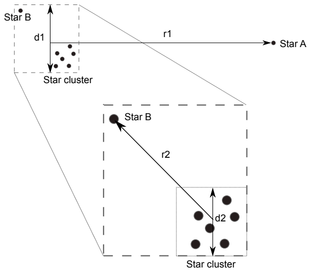
1. Tree method (Barnes & Hut 1986, 1989)
$\rightarrow$ hierarchical structure and calculation of multipoles of the potential $\rightarrow \mathcal{O}(N \log N)$

   Holmberg (Lund, 1941) even $\mathcal{O}(N)$ with help of light bulbs on 2d grid (flux $\propto 1/r^2$)

2. Fourier transformation: compute potential $\Phi(\vec{x})$ with FFT $\rightarrow$ CA 2
3. Leapfrog method (2nd order integ.):

$$\vec{r}_i^{\,n+1/2} = \vec{r}_i^{\,n-1/2} + \Delta t_i \vec{v}_i^{\,n} \quad (254)$$
$$\vec{v}_i^{\,n+1} = \vec{v}_i^{\,n} + \Delta t_i \vec{a}_i^{\,n+1/2} \quad (255)$$

with time step doubling $\Delta t_i = \Delta t_{\max}/2^{n_i}$ for each particle i



The gravitational effect excerted by the star cluster and the single star B on star A can be approximated by a point mass. (from Barnes-Hut Galaxy Simulator)

N-body simulations for large N III

Example: time step doubling with $\rightarrow$ leapfrog method

particle A: time step $\Delta t/2$, particle B: time step Δt
starting via

$$\vec{r}_i^{\,n+1/2} = \vec{r}_i^{\,n} + \frac{1}{2}\Delta t_i \vec{v}_i^{\,n} + \frac{1}{8}\Delta t_i^2 \vec{a}_i^{\,n} \quad \text{for } i = A, B. \tag{256}$$

1) Hence, we get $\vec{r}_A(\Delta t/4)$ and $\vec{r}_B(\Delta t/2)$ and from that

2) $\vec{a}_A(A[\Delta t/4], B[\Delta t/2])$ and analogously $\vec{a}_B \rightarrow$ time asymmetry

3) $\vec{a}_A \rightarrow \vec{v}_A(\Delta t/2) \rightarrow \vec{r}_A(3/4\Delta t)$
4) $\vec{a}_A(A[3/4\Delta t], B[\Delta t/2]) \rightarrow \vec{v}_A(\Delta t) \rightarrow$ reversed time asymmetry

5) Averaging of $\vec{r}_A(\Delta t/4), \vec{r}_A(3/4\Delta t)$ to $\vec{r}_A(\Delta t/2)$, then

6) $\rightarrow \vec{a}_B(A[\Delta t/2], B[\Delta t/2]) \rightarrow \vec{v}_B(\Delta t)$

7) from $\vec{v}_A(\Delta t), \vec{v}_B(\Delta t) \rightarrow \vec{r}_A(5/4\Delta t)$
i.e. next cycle starts, cf. 1) $\vec{r}_A(\Delta t/4)$ & $\vec{r}_B(\Delta t/2)$)

Summary

Methods to solve N-body interactions:

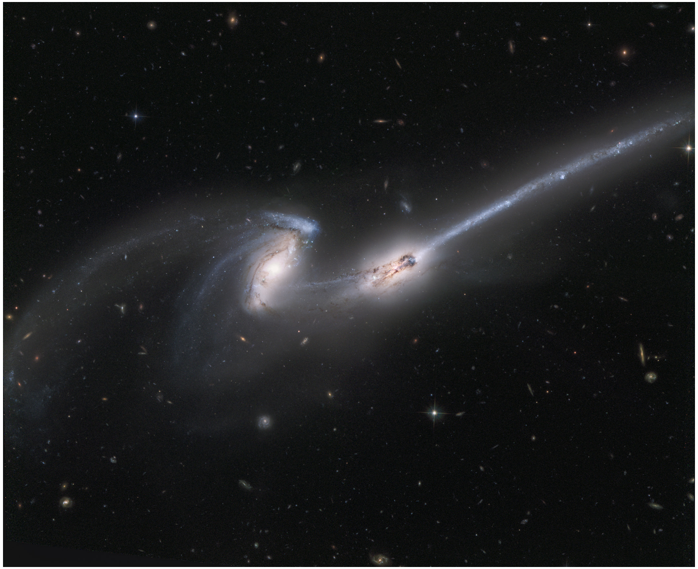
- Runge-Kutta (RK4): standard for any ODE
- 2nd order leapfrog: reasonable accuracy for extremely large number of particles, integration only over a few dynamical times (e.g., Sun orbiting Galactic center)
- Bulirsch-Stoer[†]: highly accurate, for few-body systems
- predictor-corrector: reasonable accuracy for moderate up to *large* number of particles
- for close encounters: softening (collisionless) or accurate regularization (collisions)

[†]alternatively for long-term evolution of few-body systems, e.g., over lifetime of Sun and whithout close encounters: symplectic map $\rightarrow$ split Hamiltonian $H = H_{\text{Kepler}} + H_{\text{interaction}}$, where analytic solution (ellipse) is used for H_{Kepler}, requires transformation to Jacobi coordinates

Outlook: Interacting galaxies I

Arp 271 (Gemini South)

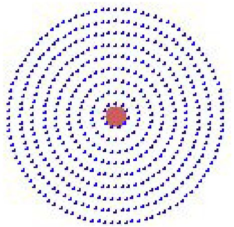
Outlook: Interacting galaxies II


NGC 4676 “Mice” (HST / NASA)

Galactic “Bridges” and “Tails” I

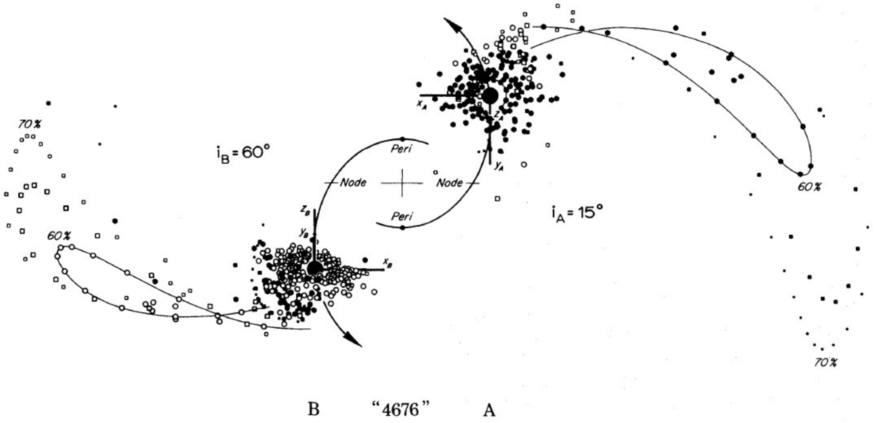
Toomre & Toomre (1972):

- *bridges* (connections between galaxies) and *tails* (structures on the opposite site of the interaction point) as the result of tidal forces between galaxies
- simplified model:
  - encounter of only two galaxies, parabolic (unbound)
  - galaxies as disks of non-interacting “test particles”, initially on circular orbits around a central point mass



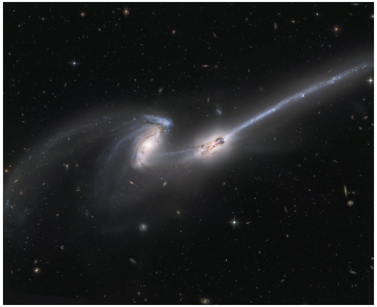
- result: mutual distortion of the galaxies just by gravitation, kinematic evolution to narrow, elongated structures

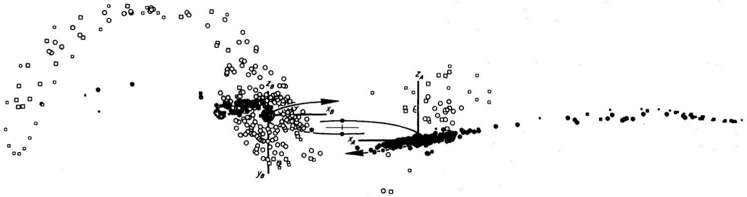
Galactic "Bridges" and "Tails" II



simulation of NGC 4676 from Toomre & Toomre (1972)
$\rightarrow$ two identical galaxies

Galactic "Bridges" and "Tails" III





NGC 4676 as before, but now seen *edge-on*

Applications:
The Lane-Emden equation

The Lane-Emden Equation I

We remember: Stellar structure equations

Example: Boundary values

First two equations of stellar structure (e.g., for white dwarf), with mass coordinate m (Lagrangian description)

$$\frac{\partial r}{\partial m} = \frac{1}{4\pi r^2 \rho} \quad \text{mass continuity, cf. shell } dm = 4\pi r^2 \rho dr \tag{257}$$

$$\frac{\partial P}{\partial m} = -\frac{GM}{4\pi r^4} \quad \text{hydrostatic equilibrium} \tag{258}$$

+ equation of state $P(\rho)$ (e.g., ideal gas $P(\rho, T) = RT\rho/\mu$), and boundary values

$$\text{center} \quad m = 0 : r = 0 \tag{259}$$

$$\text{surface} \quad m = M : \rho = 0 \rightarrow P = 0 \tag{260}$$

$\rightarrow$ solve for $r(m)$, specifically for $R_* = r(m = M_*)$, i.e. for given M_*

The Lane-Emden Equation II

Derivation of the Lane-Emden equation

(see also Hansen & Kawaler 1994)
$\rightarrow$ if equation of state (EOS) for pressure is only function of density, e.g., completely degenerate, nonrelativistic, electron gas (e.g., white dwarf)

$$P_\mathrm{e} = 1.004 \times 10^{13} \left(\frac{\rho[\mathrm{g\,cm}^{-3}]}{\mu_\mathrm{e}} \right)^{5/3} \mathrm{dyn\,cm}^{-2} \tag{261}$$

so, $P \propto (\rho/\mu_\mathrm{e})^{5/3}$ power law ...

($\mu_\mathrm{e} = [\sum Z_i X_i y_i/A_i]^{-1}$ mean molecular weight per electron, e.g., $\mu_\mathrm{e} \approx (\frac{1\cdot 0.7\cdot 1}{1} + \frac{2\cdot 0.3\cdot 1}{4}) \approx 1.2$ for fully ionized H-He plasma; Z_i: nucleus charge and X_i mass fraction of element i, y_i its relative ionziation fraction, i.e., 1 for fully inonized)

Polytropes are pseudo-stellar models where a power law for $P(\rho)$ is assumed a priori without reference to heat transfer/thermal balance

$\rightarrow$ only hydrostatic and mass continuity equation taken into account

The Lane-Emden Equation III

define a polytrope as

$$P(r) = K\rho^{1+\frac{1}{n}}(r) \tag{262}$$

with some constant K and the *polytropic index* n.
$\rightarrow$ polytrope must be in hydrostatic equlibrium, so hydrostatic equation (function of r only)

$$\frac{dP}{dr} = -\frac{GM_r}{r^2}\rho \quad | \cdot \frac{r^2}{\rho} \mid d/dr \tag{263}$$

with the continuity equation $\frac{dM_r}{dr} = 4\pi r^2\rho$ and the mass variable $M_r = \int_0^r dm(r)$, i.e., $M_r = 0$
$\rightarrow$ center $(r = 0,\ \rho = \rho_\mathrm{c})$ and $M_r = M_*$ $\rightarrow$ surface $(r = R_*,\ \rho = 0)$

$$\frac{d}{dr}\left(\frac{r^2}{\rho}\frac{dP}{dr}\right) = -G\frac{dM_r}{dr} = -4\pi Gr^2\rho \tag{264}$$

so finally:

The Lane-Emden Equation IV

$$\frac{1}{r^2}\frac{d}{dr}\left(\frac{r^2}{\rho}\frac{dP}{dr}\right) = -4\pi G\rho \tag{265}$$

$\rightarrow$ Poisson's equation of gravitation with $g(r) = d\Phi/dr = GM_r/r^2$, and $\frac{dP}{dr} = -\frac{GM_r}{r^2}\rho$
hence $\rightarrow$ $\nabla^2\Phi = 4\pi G\rho$ in spherical coordinates

find transformations to make Eq. (265) *dimensionless*. Define *dimensionless* variable θ by

$$\rho(r) = \rho_c\theta^n(r) \tag{266}$$

$\rightarrow$ then, power law for pressure from our definition of the polytrope Eq. (262)

$$P(r) = K\rho^{1+1/n}(r) = K\rho_c^{1+1/n}\,\theta^{n+1}(r) = P_c\,\theta^{1+n}(r) \tag{267}$$

$$\text{and} \quad \rightarrow \; P_c = K\rho_c^{1+1/n} \tag{268}$$

inserting Eqs. (266) & (268) into Eq. (265)

The Lane-Emden Equation V

$$\frac{(n+1)P_\mathrm{c}}{4\pi G\rho_\mathrm{c}^2}\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\theta}{dr}\right) = -\theta^n \tag{269}$$

together with dimensionless *radial* coordinate ξ

$$r = r_n\,\xi \quad \text{with (const.) scale length } r_n^2 = \frac{(n+1)P_\mathrm{c}}{4\pi G\rho_\mathrm{c}^2} \tag{270}$$

our Poisson's equation (265) becomes

$\rightarrow$ so called

The Lane-Emden Equation VI

Lane-Emden equation (Lane 1870; Emden 1907)

$$\frac{1}{\xi^2}\frac{d}{d\xi}\left(\xi^2\frac{d\theta}{d\xi}\right) = -\theta^n \tag{271}$$

with solutions "polytropes of index n" $\theta_n(\xi)$

Applications:

- describe i.g. self-gravitating spheres (of plasma)
- Bonnor-Ebert sphere ($n \to \infty$, so u, e^{-u} instead of θ, θ^n): stable, finite-sized, finite-mass isothermal cloud with $P \neq 0$ at outer boundary $\to$ Bonnor-Ebert mass (Ebert 1955; Bonnor 1956)
- characterize (full) stellar structure models, e.g., Bestenlehner (2020) ($n = 3$, removing explicit M_*-dependance of $\dot{M}$-CAK desription)
- composite polytropic models for modeling of massive interstellar clouds with a hot ionized core, stellar systems with compact, massive object (BH) at centre
- generalized-piecewise polytropic EOS for NS binaries (P. Biswas 2021)

Remarks:

if EOS is ideal gas $P = \rho N_\mathrm{A} k T/\mu$, one can get

$$P(r) = K' T^{n+1}(r), \quad T(r) = T_\mathrm{c}\theta(r) \tag{272}$$

$$\text{with } K' = \left(\frac{N_\mathrm{A} k}{\mu}\right)^{n+1} K^{-n}, \quad T_\mathrm{c} = K \rho_\mathrm{c}^{1/n} \left(\frac{N_\mathrm{A} k}{\mu}\right)^{-1} \tag{273}$$

$\rightarrow$ polytrope with EOS of ideal gas and mean molecular weight μ gives temperature profile, radial scale factor is

$$r_n^2 = \left(\frac{N_\mathrm{A} k}{\mu}\right)^2 \frac{(n+1)\,T_\mathrm{c}^2}{4\pi G P_\mathrm{c}} = \frac{(n+1) K \rho_\mathrm{c}^{1/n-1}}{4\pi G} \tag{274}$$

The Lane-Emden Equation VIII

Requirements for physical solutions:

+ central density $\rho_c \rightarrow \theta(\xi = 0) = 1$ (because of Eq. (266): $\rho(r) = \rho_c \theta^n(r)$)
+ spherical symmetry at center $(dP/dr|_{r=0}) \rightarrow \theta' \equiv d\theta/d\xi = 0$ at $\xi = 0$
  $\rightarrow$ suppresses divergent solutions of the 2nd order system $\rightarrow$ regular solutions (E-solutions)
+ surface $P = \rho = 0 \rightarrow \theta_n = 0$ (first occurrence of that!) at ξ_1

Boundary conditions for polytropic model

$\theta(0) = 1,\ \theta'(0) = 0$ at $\xi = 0$ (center)
$\theta(\xi_1) = 0$ at $\xi = \xi_1$ (surface)

So stellar radius

$$R = r_n \xi_1 = \sqrt{\frac{(n+1)P_c}{4\pi G \rho_c^2}}\,\xi_1 \qquad (275)$$

for given K, n, and either ρ_c or P_c ($P_c = K\rho_c^{1+1/n}$)

The Lane-Emden Equation IX

Analytic E-solutions

$\rightarrow$ analytic regular solutions exist for $n = 0, 1, 5$

$n = 0$ constant density sphere, $\rho(r) = \rho_\mathrm{c}\,\theta^n(\xi) = \rho_\mathrm{c}$, and

$$\theta_0(\xi) = 1 - \frac{\xi^2}{6} \quad \rightarrow \xi_1 = \sqrt{6} \tag{276}$$

so $P(\xi) = P_\mathrm{c}\,\theta^{1+n}(\xi) = P_\mathrm{c}\,\theta(\xi) = P_\mathrm{c}\left[1 - (\xi/\xi_1)^2\right]$.
For P_c we need M, R from Eq. (275): $P_\mathrm{c} = \frac{3}{8\pi}\frac{GM^2}{R^4}$ (Proof!)

$n = 1$ solution θ_1 is sinc function

$$\theta_1 = \frac{\sin\xi}{\xi} \quad \text{with } \xi_1 = \pi \tag{277}$$

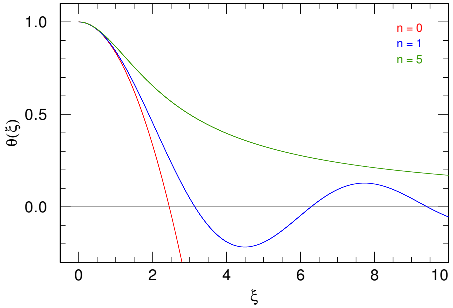
$\rightarrow \rho = \rho_\mathrm{c}\,\theta^n(\xi) = \rho_\mathrm{c}\,\theta(\xi)$ and $P = P_\mathrm{c}\,\theta^{1+n}(\xi) = P_\mathrm{c}\,\theta^2(\xi)$

The Lane-Emden Equation X

$n = 5$ finite central density ρ_c but infinite radius $\xi_1 \to \infty$:

$$\theta_5(\xi) = \frac{1}{\sqrt{1 + \frac{\xi^2}{3}}} \tag{278}$$

contains finite mass (there is also a solution with oscillatory behavior for $\xi \to 0$, see Srivastava 1962)



$\rightarrow$ solutions with $n > 5$ have infinite radius too, but also infinite mass

Numerical solution I

For the interesting cases $0 \leq n \leq 5 \rightarrow$ numerical solution

$$\frac{1}{\xi^2}\frac{d}{d\xi}\left(\xi^2 \frac{d\theta}{d\xi}\right) = \frac{2}{\xi}\frac{d\theta}{d\xi} + \frac{d}{d\xi}\frac{d\theta}{d\xi} = -\theta^n \tag{279}$$

Reduction: set $x = \xi$, $y = \theta$, $z = (d\theta/d\xi) = (dy/dx) \rightarrow \frac{2}{x}z + \frac{d}{dx}(\frac{dy}{dx}) = -y^n$

$$y' = \frac{dy}{dx} = z, \tag{280}$$

$$z' = \frac{dz}{dx} = \frac{d}{dx}\left(\frac{dy}{dx}\right) = -y^n - \frac{2}{x}z \tag{281}$$

Assume that we have values y_i, z_i at a point x_i, so that we can get with some step size h:
y_{i+1} & z_{i+1} at $x_{i+1} = x_i + h$

Numerical solution II

Then with RK4:

$$k_1 = h \cdot y'(x_i, y_i, z_i) = h \cdot (z_i) \tag{282}$$

$$\ell_1 = h \cdot z'(x_i, y_i, z_i) = h \cdot (-y_i^n - \frac{2}{x_i} z_i) \tag{283}$$

$$k_2 = h \cdot y'(x_i + h/2, y_i + k_1/2, z_i + \ell_1/2) = h \cdot (z_i + \ell_1/2) \tag{284}$$

$$\ell_2 = h \cdot z'(x_i + h/2, y_i + k_1/2, z_i + \ell_1/2) \tag{285}$$

$$= h \cdot \left(-(y_i + k_1/2)^n - \frac{2}{x_i + h/2}(z_i + \ell_1/2) \right) \tag{286}$$

$$k_3 = h \cdot y'(x_i + h/2, y_i + k_2/2, z_i + \ell_2/2) \tag{287}$$

$$\ell_3 = h \cdot z'(x_i + h/2, y_i + k_2/2, z_i + \ell_2/2) \tag{288}$$

$$k_4 = h \cdot y'(x_i + h, y_i + k_3, z_i + \ell_3) \tag{289}$$

$$\ell_4 = h \cdot z'(x_i + h, y_i + k_3, z_i + \ell_3) \tag{290}$$

$\rightarrow \quad y_{i+1} = y_i + \ldots$ and $z_{i+1} = z_i + \ldots$

Numerical solution III

Although $z' = -y^n - \frac{2}{x} z$ (Eq. (281)) is indeterminate for $\xi = 0$, integration can in principle be started for $\xi = 0$ for regular solutions (Cox & Giuli 1968; Hansen & Kawaler 1994) with help of power series expansion around $\xi = 0$:

$$\theta_n(\xi) = 1 - \frac{\xi^2}{6} + \frac{n}{120}\,\xi^4 - \frac{n(8n-5)}{15120}\,\xi^6 + \dots \tag{291}$$

$$\to \ \theta_n'(\xi) = -\frac{1}{3}\xi + \frac{n}{30}\,\xi^3 - \frac{n(8n-5)}{2520}\,\xi^5 + \dots \tag{292}$$

So for $\xi \to\ 0$ then $y' \to\ -1/3\,\xi = 0$.
However, better: choose $0 < \xi \ll 1$ and compute with help of Eq. (291) $y, y'(= z), z'$ (should also work for irregular solutions)

Applying the Lane-Emden equation to stars I

construct polytropes for $n < 5$ and *given* M, R
$\rightarrow$ possible as long as K not fixed
because of definition of θ from $\rho(r) = \rho_c \theta^n(r)$ (Eq. (266)) and $r = r_n \xi$
(Eq. (270))$\rightarrow dr = r_n d\xi$

$$m(r) = \int_0^r 4\pi \rho r^2 dr = 4\pi \rho_c \int_0^r \theta^n r^2 dr = 4\pi \rho_c \frac{r^3}{\xi^3} \int_0^\xi \theta^n \xi^2 d\xi \tag{293}$$

note that $r^3/\xi^3 = r_n^3$ is constant. From Lane-Emden equation (271)
$\frac{1}{\xi^2}\frac{d}{d\xi}\left(\xi^2 \frac{d\theta}{d\xi}\right) = -\theta^n \rightarrow \theta^n \xi^2 = -\frac{d}{d\xi}\left(\xi^2 \frac{d\theta}{d\xi}\right)$ follows direct integration, so

$$m(r) = 4\pi \rho_c \frac{r^3}{\xi^3} \int_0^\xi -\frac{d}{d\xi}\left(\xi^2 \frac{d\theta}{d\xi}\right) d\xi = 4\pi \rho_c r^3 \left(-\frac{1}{\xi}\frac{d\theta}{d\xi}\right) \tag{294}$$

$\rightarrow$ Eq. (294) contains ξ and r, related by Eq. (270): $r/\xi = r_n = R/\xi_1$, so for the surface:

Applying the Lane-Emden equation to stars II

$$M = 4\pi\rho_c R^3 \left(-\frac{1}{\xi}\frac{d\theta}{d\xi}\right)_{\xi=\xi_1} \tag{295}$$

With help of the mean density $\overline{\rho} := M/(\frac{4}{3}\pi R^3)$ this can be written as

$$\frac{\overline{\rho}}{\rho_c} = \left(-\frac{3}{\xi}\frac{d\theta}{d\xi}\right)_{\xi=\xi_1} \tag{296}$$

Note the right hand side depends only on n, can be computed. E.g., for $n = 0$
$\rightarrow (-\frac{3}{\xi}\frac{d\theta}{d\xi})_{\xi=\xi_1} = 1$, and for $n = 1 \rightarrow \frac{\overline{\rho}}{\rho_c} = \frac{3}{\pi^2}$
the larger $n \rightarrow$ the smaller $\frac{\overline{\rho}}{\rho_c} \rightarrow$ the higher the density concentration towards center.

Differential equations

Types of differential equations I

One can classify differential equations regarding their

- *order*, so the degree of the highest derivative. Note: $y \equiv y(t)$
  General form of a first-order differential equation:

$$\frac{dy}{dt} = f(y,t) \tag{297}$$

  for any arbitrary function f, e.g., $\frac{dy}{dt} = 2ty^8 - t^5 + \sin(y)$.
  A second-order differential equation has the form:

$$\frac{d^2y}{dt^2} + \lambda\frac{dy}{dt} = f(t, \frac{dy}{dt}, y) \tag{298}$$

  and so on.

Types of differential equations II

Reduction

By introducing auxillary variables/functions, every higher order differential equation can be reduced to a set of *first-order* differential equations

$$y^{(m)}(x) = f(x, y(x), y^{(1)}(x), \ldots, y^{(m-1)}(x)) \tag{299}$$

introduce functions z

$$\rightarrow z_1(x) := y(x) \tag{300}$$

$$z_2(x) := y^{(1)}(x) \tag{301}$$

$$\vdots \tag{302}$$

$$z_m(x) := y^{(m-1)}(x) \tag{303}$$

$$\rightarrow z' = \begin{bmatrix} z_1' \\ \vdots \\ z_m' \end{bmatrix} = \begin{bmatrix} z_2 \\ \vdots \\ f(x, z_1, z_2, \ldots, z_m) \end{bmatrix} \tag{304}$$

Types of differential equations III

One can distinguish

- *ordinary diffential equations (ODE)*, where only one independent variable is explicitly involved (typically time or location), e.g., hydrostatic equation for $P(r)$:

$$\frac{dP}{dr} = -\rho(r)\,g(r) \tag{305}$$

- *partial differential equations (PDE)*, where derivatives with respect to at least two variables occur, e.g., Poisson equation:

$$\Delta\rho = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)\rho(x,y,z) = f(x,y,z) \tag{306}$$

$\rightarrow$ The theory and (numerical) solution of PDEs is more complicated than for ODE.

Types of differential equations IV

Moreover, there are the classes of

- *linear* differential equations: only the first power of y or $d^n y/dt^n$ occurs, e.g. *wave equation*:

$$\left(\frac{1}{c^2}\frac{\partial}{\partial t} - \Delta\right) u = 0 \tag{307}$$

$\rightarrow$ special property: *law of linear superposition*, linear combinations of solutions are also solutions:

$$u_2(x, y, z, t) = a u_0(x, y, z, t) + b u_1(x, y, z, t) \tag{308}$$

$\rightarrow$ unperturbed superposition of waves

Types of differential equations V

- *nonlinear* differential equations: contain higher powers or other functions of y or $d^n y/dt^n$, e.g., simple gravity pendulum:

$$\frac{d\theta^2}{dt^2} = \frac{l}{g}\sin\theta \tag{309}$$

$\rightarrow$ clear: linear combinations of solutions are not automatically solutions too, e.g.

$$\frac{dy}{dt} = \lambda y(t) - \lambda^2 y^2(t) \tag{310}$$

$$y(t) = \frac{a}{1+be^{-\lambda t}} \quad \text{one solution} \tag{311}$$

$$y_1(t) = \frac{a}{1+be^{-\lambda t}} + \frac{c}{1-de^{-\lambda t}} \quad \text{not a solution} \tag{312}$$

$\rightarrow$ nonlinear differential equations in general became feasible with the rise of computers

Boundary values I

As general solution of (ordinary) differential equation contains arbitrary constant per order, problems involving differential equations can be characterized by the type of conditions:

1. *initial* values/conditions must be given: constant for 1st order differential equation (usually time-dependent) fixed by giving $y(t)$ for some time t_0, so giving $y_0 = y(t_0)$; for 2nd order by giving additionaly $y'(t_0)$ and so on (Note, that we solve usually for $t > t_0$, but this is not a requirement), e.g., Kepler problem

$$\vec{v}(t) = \dot{\vec{r}}(t) \quad \& \quad \vec{a}(t) = \dot{\vec{v}}(t) = \vec{F}_{\mathrm{G}}(r)/m \tag{313}$$

$$x(t_0) = x_0, y(t_0) = 0; v_x(t_0) = 0, v_y(t_0) = v_{y,0} \tag{314}$$

For the initial value problem (Cauchy problem), the theorem by Picard-Lindelöf guarantees a unique solution:

Boundary values II

Existence and uniqueness of the solution for the initial value problem

$$y' = f(y, x), \quad y(x_0) = y_0 \tag{315}$$

If f is continuous on the stripe $S := \{(x, y)|a \leq x \leq b, y \in \mathbb{R}^n\}$ with finite a, b and a constant L, such that

$$||f(x, y_1) - f(x, y_2)|| \leq L||y_1 - y_2|| \tag{316}$$

for all $x \in [a, b]$ and for all $y_1, y_2 \in \mathbb{R}^n$ (Lipschitz continuous), then exists for all $x_0 \in [a, b]$ and for all $y_0 \in \mathbb{R}^n$ a unique function $y(x)$ for $x \in [a, b]$ with

1. $y(x)$ is continuous and continuously differentiable for $x \in [a, b]$;
2. $y'(x) = f(x, y(x))$ for $x \in [a, b]$;
3. $y(x_0) = y_0$

Boundary values III

Note that the Lipschitz condition (bounded slope) of $f(y,x)$ is required for uniqueness, e.g., $y'(x) = \sqrt{|x|}$ with $y(0) = 0$ is fulfilled by $y_1(x) \equiv 0$ and also by $y_2(x) = \frac{x^2}{4}$, that is because $f'(y,x) = \frac{1}{\sqrt{|x|}}$ and hence $\lim_{x\to 0} f' = \infty$.

Without Lipschitz condition the *Peano existence theorem* guarantees at least the existence of a solution.

Proof concept

Integrating Eq. (315) gives a fixed point equation:

$$y(x) - y(x_0) = \int_{x_0}^{x} f(s, y(s))ds \tag{317}$$

with Picard-Lindelöf iteration

$$\phi_0(x) = y_0 \quad \text{and} \quad \phi_{k+1} = y_0 + \int_{x_0}^{x} f(s, \phi_k(s))ds \tag{318}$$

Boundary values IV

Example: Picard iteration

For the Cauchy problem

$$y'(x) = 1 + y(x)^2, \quad y(x_0) = y(0) = 0 \tag{319}$$

$$\phi_0(x) = 0 \tag{320}$$

$$\phi_1(x) = 0 + \int_0^x (1 + 0^2)ds = x \tag{321}$$

$$\phi_2(x) = 0 + \int_0^x (1 + s^2)ds = x + \frac{1}{3}x^3 \tag{322}$$

$\rightarrow$ Taylor series expansion of $y(x) = \tan(x)$

so following *Banach fixed point theorem* ϕ_k converges uniquely to the solution $y(x)$. The existence of $y(x)$ (Peano) is proven by constructing a piecewise continuous function with help of the Euler method (polygonal curve) that converges uniformly for $\Delta x \rightarrow 0$.

Boundary values V

2. *boundary* values/conditions can be given, (additionally to initial conditions) to restrict further the solutions, i.e., constrain it to fixed values at the boundaries of the solution space, usually for 2nd order differential equation

$$u''(x) = f(u, u', x) \tag{323}$$

where u or u' is given at boundaries, by transformation, e.g.,

$$x' = (x - x_1)/(x_2 - x_1) \tag{324}$$

at $x = 0$ and $x = 1$. Then $\rightarrow$ 4 possible types of boundary conditions

1. $u(0) = u_0$ and $u(1) = u_1$
2. $u(0) = u_0$ and $u'(1) = v_1$
3. $u'(0) = v_0$ and $u(1) = u_1$
4. $u'(0) = v_0$ and $u'(1) = v_1$

Boundary values VI

Usually: reduce to set of 1st order differential equations and start integration with given $u(0)$ *and* $u'(0)$. But for boundary-value problem: only $u(0)$ ***or*** $u'(0)$ given, $\rightarrow$ not sufficient for any initial-value algorithm

Example: Boundary values

First two equations of stellar structure (e.g., for white dwarf)

$$\frac{\partial r}{\partial m} = \frac{1}{4\pi r^2 \rho} \quad \text{mass continuity} \tag{325}$$

$$\frac{\partial P}{\partial m} = -\frac{G\,M}{4\pi r^4} \quad \text{hydrostatic equilibrium} \tag{326}$$

+ equation of state $P(\rho)$ (e.g., ideal gas $P = RT\rho/\mu$), and boundary values

$$\text{center} \quad m = 0 : r = 0 \tag{327}$$

$$\text{surface} \quad m = M : \rho = 0 \rightarrow \; P = 0 \tag{328}$$

$\rightarrow$ solve for $r(m)$, specifically for $R_* = r(m = M_*)$

Boundary values VII

3. *eigenvalue problems:* solution for selected parameters (λ) in the equations; usually even more complicated and solution not always exist, sometimes *trial-and-error search* necessary. E.g.,

$$u'' = f(u, u', x, \lambda) \tag{329}$$

for eigenvalue λ plus a set of boundary conditions. Eigenvalue λ can only have some selected values for valid solution.

E.g., Schrödinger equation for particle confined in a potential:
eigenfunctions $\rightarrow$ wavefunction ϕ_k;
eigenvalues $\rightarrow$ discrete energies $E_k \rightarrow \hat{H}\phi_k = E_k\phi_k$

Boundary values VIII

Eigenvalue problem: Stationary elastic waves

Displacement $u(x)$ by

$$u'' = -k^2 u \tag{330}$$

Allowed values of wavevector $k = \omega/c \rightarrow$ eigenvalues of the problem
both ends fixed: $u(0) = u(1) = 0$ or one end fixed, other end free: $u(0) = 0$ and $u'(1) = 0$.
Fortunately, analytical solutions:

$$u_n(x) = \sqrt{2}\sin(k_n x) \quad \& \quad k_n = n\pi \quad n = \pm 1, \pm 2, \ldots \tag{331}$$

Moreover, complete solution of longitudinal waves along elastic rod: linear combination of *all* eigenfunctions with their initial solutions (fixing c_n)

$$u(x,t) = \sum_{n=-\infty}^{\infty} c_n u_n(x) e^{in\pi ct} \tag{332}$$

The shooting method I

Simple method for *boundary-value* and *eigenvalue* problems: shooting method (origin from artillery), cf. Pang (1997)
e.g., for boundary-value problem $u'' = f(u, u', x)$ with $y_1 \equiv u$ and $y_2 \equiv u'$

$$\frac{dy_1}{dx} = y_2 \tag{333}$$
$$\frac{dy_2}{dx} = f(y_1, y_2, x) \tag{334}$$

plus boundary conditions, e.g., $u(0) = y_1(0) = u_0$ and $u(1) = y_1(1) = u_1$.
Idea: introduce adjustable parameter δ, so that we have an initial value problem. E.g., $u'(0) = \delta \rightarrow$ together with given $u(0) = u_0$; integrate for given intial values up to $x = 1$ with result $u(1) = u_\delta(1)$, so that

$$F(\delta) = u_\delta(1) - u_1 \overset{!}{=} 0 \tag{335}$$

$\rightarrow$ use root search algorithm to determine (approximative) δ

The shooting method II

Shooting method for boundary value problem (Stoer & Bulirsch 2005)

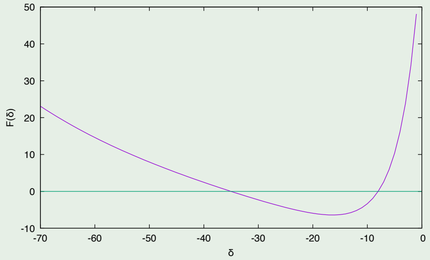
$$u''(x) = \frac{3}{2}u^2, \quad u(0) = 4, \quad u(1) = 1 \tag{336}$$

$$\text{set } y_1 \equiv u \text{ and } y_2 \equiv u' \quad y_1(0) = 4, \quad y_2(0) = \delta = -1, \ldots - 70 \tag{337}$$

$$\rightarrow y_{1,k+1} = y_{1,k} + \Delta x \cdot y_{2,k} \tag{338}$$

$$y_{2,k+1} = y_{2,k} + \Delta x \cdot 3./2. * y_{1,k}^2 \tag{339}$$

plot $F(\delta) = y_{1,n} - u(1)$, roots give missing initial values $u'(0)$



The shooting method III

Similarly, for given

- $u'(0) = v_0$ and $u(1) = u_1 \to u(0) = \delta$, find root of $F(\delta) = u_\delta(1) - u_1$
- $u'(0) = v_0$ and $u'(1) = v_1 \to F(\delta) = u'_\delta(1) - v_1$

Moreover, for eigenvalue problem:

- if $u(0) = u_0$ and $u(1) = u_1$ given, start integration with $u'(0) = \delta$ with small δ
- search root $F(\lambda) = u_\lambda(1) - u_1 \to$ approximated eigenvalue λ and eigenvector from normalized solution $u_\lambda(x) \to \delta$ automatically modified to be correct $u'(0)$ through normalization of eigenfunctions

Direct solution of 2nd order ODE I

Although, always possible $\rightarrow$ reduce 2nd order ODE to set of coupled 1st order ODEs, however, sometimes direct solution has advantages

Example: Radiative Transfer Equation

For the 1d case:

$$\frac{dI^{\pm}}{d\tau} = \pm(S - I^{\pm}), \quad d\tau = \kappa dz \tag{340}$$

with inward ($-$) and outward ($+$) intensities $I = dE/d\Omega\, dA\, dt\, d\nu$, optical depth τ and source function $S = \eta/\kappa$.
Introducing Feautrier variables (Schuster 1905; Feautrier 1964):

$$u = \frac{1}{2}(I^{+} + I^{-}) \quad \text{(intensity-like)} \tag{341}$$

$$v = \frac{1}{2}(I^{+} - I^{-}) \quad \text{(flux-like)} \tag{342}$$

Direct solution of 2nd order ODE II

we get system of two coupled 1st order ODE:

$$\frac{du}{d\tau} = v \quad \text{and} \quad \frac{dv}{d\tau} = u - S \tag{343}$$

or, combining them:

$$\frac{d^2u}{d\tau^2} = u - S \tag{344}$$

discretization on a τ grid (τ_i) with numerical derivatives (see below):

$$\left.\frac{d^2u}{d\tau^2}\right|_{\tau_i} \approx \frac{\left.\frac{du}{d\tau}\right|_{\tau_{i+1/2}} - \left.\frac{du}{d\tau}\right|_{\tau_{i-1/2}}}{\tau_{i+1/2} - \tau_{i-1/2}} \approx \frac{\frac{u_{i+1}-u_i}{\tau_{i+1}-\tau_i} - \frac{u_i-u_{i-1}}{\tau_i-\tau_{i-1}}}{\frac{1}{2}(\tau_{i+1}-\tau_i) - \frac{1}{2}(\tau_i - \tau_{i-1})} \tag{345}$$

Direct solution of 2nd order ODE III

$\rightarrow$ set of linear equations for u_i for $i = 2, \ldots, i_{\max} - 1$:

$$-A_i u_{i-1} + B_i u_i - C_i u_{i+1} = S_i \tag{346}$$

with the coefficients

$$A_i = \left(\frac{1}{2}(\tau_{i+1} - \tau_{i-1})(\tau_i - \tau_{i-1})\right)^{-1} \tag{347}$$

$$C_i = \left(\frac{1}{2}(\tau_{i+1} - \tau_{i-1})(\tau_{i+1} - \tau_i)\right)^{-1} \tag{348}$$

$$B_i = 1 + A_i + C_i \tag{349}$$

Direct solution of 2nd order ODE IV

$\rightarrow$ tridiagonal matrix, efficiently solvable by standard linear algebra solvers (e.g., Gauß-Seidel elimination)

$$\begin{bmatrix} B_1 & -C_1 & & & & & \\ -A_2 & B_2 & -C_2 & & & & \\ \dots & & & & & & \\ & & -A_i & B_i & -C_i & & \\ \dots & & & & & & \\ & & & & B_{i_\mathrm{max}} & -C_{i_\mathrm{max}} \end{bmatrix} \circ \begin{bmatrix} u_1 \\ u_2 \\ \dots \\ u_i \\ \dots \\ u_{i_\mathrm{max}} \end{bmatrix} = \begin{bmatrix} W_1 \\ W_2 \\ \dots \\ W_i \\ \dots \\ W_{i_\mathrm{max}} \end{bmatrix} \tag{350}$$

Note: $W_i = S_i$ exept for $i = 1$ and i_max $\rightarrow$ boundary conditions

Direct solution of 2nd order ODE V

Advantage of Feautrier scheme

- direct solution of 2nd order ODE saves memory
- at large optical depths $I^+ \approx I^- \rightarrow$ radiative flux $\sim I^+ - I^-$ inaccurate because of roundoff error, Feautrier scheme uses instead averaged quantities u, v for higher accuracy ($\rightarrow$ stability in an iterative scheme for $S(I), \tau(I)$)

Root finding - Iterative techniques

Transcendent equations I

Problem: Finding roots for equations that cannot be solved analytically, i.e. finding x_0 for $f(x_0) = 0$

Transcendent equation: quantum states in a square well

The 1d potential $V(x)$ for the Schrödinger equation

$$V(x) = \begin{cases} -V_0 & , |x| \leq a \\ 0 & , |x| \geq a \end{cases} \tag{351}$$

has bound states with energies $E = -E_\mathrm{B} < 0$

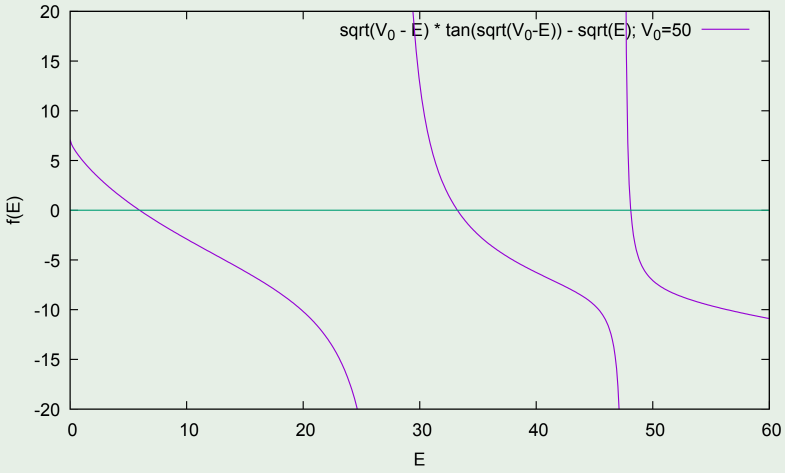
$$\sqrt{2m(V_0 - E_\mathrm{B})} \tan\left[a\sqrt{2m(V_0 - E_\mathrm{B})}\right] = \sqrt{2mE_\mathrm{B}} \tag{352}$$

$\rightarrow$ e.g., for $2m = 1$, $a = 1$ we want to find the roots E_B of

$$f(E_\mathrm{B}) = \sqrt{V_0 - E_\mathrm{B}} \tan\left(\sqrt{V_0 - E_\mathrm{B}}\right) - \sqrt{E_\mathrm{B}} \overset{!}{=} 0 \tag{353}$$

Transcendent equations II

Hints: Transcendent equation: quantum states in a square well



Note: the function $f(E_\mathrm{B})$

- is not continuous
- has multiple roots

Transcendent equations III

Roots of numerically derived functions

Some functions cannot even be written analytically, e.g.

- $x(t)$ for the Kepler problem
- solutions of the Lane-Emden equation $\theta_n(\xi)$ for $n \neq \{0, 1, 5\}$

$\rightarrow$ roots can be found numerically by trial-and-error algorithms, i.e. iteratively until some specified level of precision is reached

Bisection I

$\rightarrow$ very stable (root is always found if conditions fulfilled), but also very slow iterative procedure
$\rightarrow$ needs *two* start values $[x_1, x_2]$ for estimating x_0
If $f(x)$ continuous on $[a, b]$ and $f(a) \cdot f(b) < 0$, then the intermediate value theorem guarantees the existence of an $x_0 \in [a, b]$ with $f(x_0) = 0$.

Bisection algorithm

1. start with interval $[x_1, x_2]$ on which $f(x)$ *changes sign* (so $f(x_1) \cdot f(x_2) < 0$)
   $\rightarrow$ contains root
2. choose new x_3 as the midpoint of the interval $x_3 = \frac{x_1+x_2}{2}$ (Beware of round-off errors!)
3. calculate $f(x_3)$: either $f(x_3)$ is sufficiently close to 0 $\rightarrow$ root is x_3
   or x_3 is a new interval endpoint:
   if $f(x_3) \cdot f(x_1) > 0 \rightarrow$ new interval is $[x_3, x_2]$
   or if $f(x_3) \cdot f(x_1) \leq 0 \rightarrow$ new interval is $[x_1, x_3]$
4. goto step 2

$\rightarrow$ nested intervals enclosing the root
$\rightarrow$ as interval is halved every step, gain ≈ 1 digit each 3 steps (2^3)

The secant method I

$\rightarrow$ similar to Newton's method (see below), actually approximation with finite differences

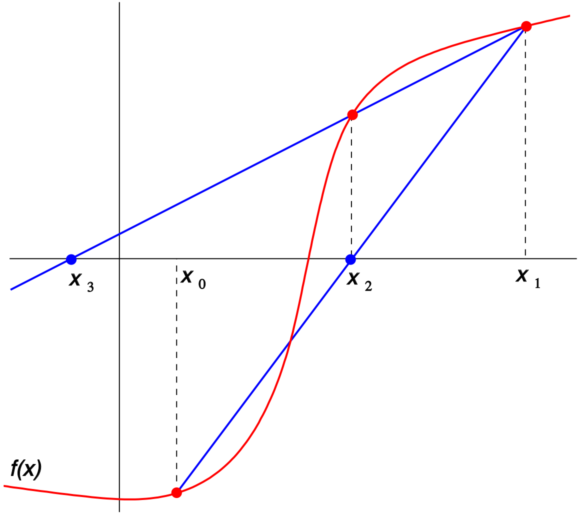
Requirement: $f(x)$ continuous and $\exists x_0 \in [a, b]$ with $f(x_0) = 0$.
Then: line trough $(x_0, f(x_0))$ and $(x_1, f(x_1))$, so that

$$y = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_1) + f(x_1)$$

with root

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

$\rightarrow$ new point $(x_2, f(x_2))$ repeat with x_1, x_2 instead of x_0, x_1



The secant method II

Secant method

1. start with interval $x_1 \neq x_2$ close to the root
2. iterate

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n) \tag{354}$$

$\rightarrow$ superlinear convergence, per iteration about 1.6 more correct digits
$\rightarrow$ convergence not assured
$\rightarrow$ numerically limited by subtractive cancelation, as fraction $\rightarrow 0/0$

Regula falsi method I

$\rightarrow$ refinement of bisection by combining it with the secant method

Regula falsi (False position method)

1. as for bisection: start with interval $[x_1, x_2]$ with $f(x_1) \cdot f(x_2) < 0$
2. calculate the zero of the secant

$$x_3 = x_1 - \frac{x_2 - x_1}{f(x_2) - f(x_1)} f(x_1) = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)} \tag{355}$$

3. if $f(x_3) = 0 \rightarrow$ stop, else
4. if $f(x_1) \cdot f(x_3) > 0 \rightarrow$ replace $x_1 = x_3$
   if $f(x_2) \cdot f(x_3) > 0 \rightarrow$ replace $x_2 = x_3$
5. goto 2

$\rightarrow$ superlinear convergence (usually more than one significant digit per iteration)
$\rightarrow$ advantage: numerically stable (converges always), no evaluation of derivatives required,
computation of function values is reused
$\rightarrow$ preferred method for 1d problems

Newton's method I

or *Newton-Raphson method* (Newton 1669, Raphson 1690) to solve numerically non-linear equations or systems of equations

$\rightarrow$ faster convergence than for bisection, but sometimes problematic

Idea: start with approximation x_0, draw tangent at $(x_0, f(x_0))$, determine intersection with x-axis $\rightarrow$ new approximation for root

Derivation: evaluate function $f(x)$ around x_0 (Taylor expansion)

$$f(x_0 + \Delta x) \simeq f(x_0) + f'(x_0) \cdot \Delta x \tag{356}$$

$$\text{(linear approximation = tangent } t \text{ on } x_0 \text{ shall vanish)} \tag{357}$$

$$\rightarrow \; t(x) = f(x_0) + f'(x_0) \cdot \Delta x \overset{!}{=} 0 \tag{358}$$

$$\rightarrow \; \Delta x = -\frac{f(x_0)}{f'(x_0)} \tag{359}$$

the correction Δx added on x_0 gives improved guess for root

Newton's method II

Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{360}$$

Convergence:

If $f : [a, b] \to \mathbb{R}$ is a $\mathcal{C}^2$ function with

1. f has a root ξ in $[a, b]$
2. $f'(x) \neq 0$ for $x \in [a, b]$
3. f is *either* convex ($f'' \geq 0$) *or* concave ($f'' \leq 0$) in $[a, b]$
4. the iterated x_1 for $x_0 = a$ and $x_0 = b$ are in $[a, b]$

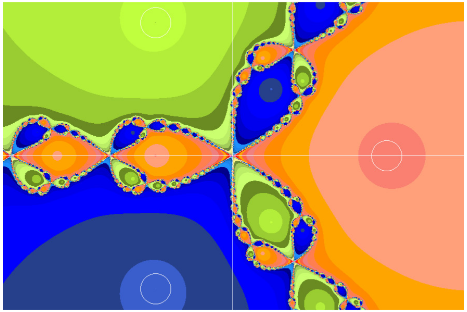
Then: For any $x_0 \in [a, b]$ the values $x_1, x_2, \ldots$ from Eq. (360) are in $[a, b]$ and the sequence converges monotonically to ξ.

Newton's method III

Remarks:

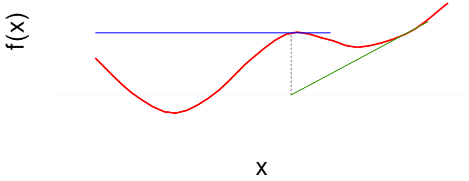
- only locally convergent,

  i.e. result depends on start approximation for x_0
  $\rightarrow$ Newton fractal for $z^3 - 1 = 0$



- in some situations Newton's method may fail (see requirements):
  - if x_n is at local extremum

    with $f(x_n) \neq 0 \rightarrow$ tangent with slope 0,
    i.e. $f'(x_n) = 0 \rightarrow$ infinite correction
    $\rightarrow$ solution: start over with different x_0

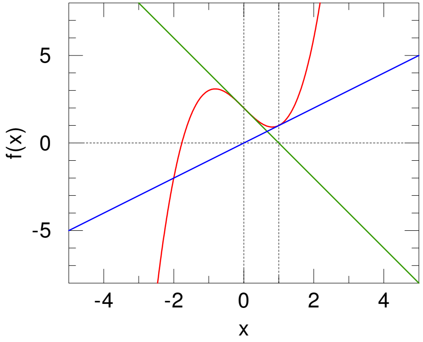


Newton's method IV

- infinite loop,

e.g., $f(x) = x^3 - 2x + 2$
with $x_0 = 0 \rightarrow f(0) = 2,\ f'(0) = -2$
$\rightarrow x_1 = 0 - \frac{2}{-2} = 1$ and
for $x_0 = 1 \rightarrow f(1) = 1,\ f'(1) = 1$
$\rightarrow x_1 = 1 - \frac{1}{1} = 0$

$\rightarrow$ happens if x_0 in region where $f(x)$ not "linear enough" (vizualization may help to find better initial guess)



- convergence is quadratic, i.e. with every step two more significant digits
- instead of analytic $f'(x)$ numeric approximation $f'(x_n) \simeq \frac{f(x_n+h)-f(x_n)}{h}$ sufficient
  $\rightarrow$ even rough (or constant!) approximation may be sufficient
- if convergent, method is stable

Newton's method V

Backtracing

$\rightarrow$ solution to some problems (i.e. infinite loop) with large corrections

So: if for new guess $x_0 + \Delta x$

$$|f(x_0 + \Delta x)|^2 > |f(x_0)|^2 \tag{361}$$

$\rightarrow$ backtrack, try smaller guess, e.g., $x_0 + \Delta x/2$, if still condition (361), try $x_0 + \Delta x/4$ and so on

$\rightarrow$ because tangent line will lead to *decrease* in $f(x)$, even small step Δx sufficient

Newton's method VI

Extension to multidimensional case

for multidimensional function f : $\mathbb{R}^n \to \mathbb{R}^n$

$$f(\boldsymbol{x}+\boldsymbol{h}) = f(\boldsymbol{x}) + J(\boldsymbol{x})\cdot\boldsymbol{h} + \mathcal{O}(||\boldsymbol{h}||^2) \tag{362}$$

where $\boldsymbol{J}(\boldsymbol{x}) = \boldsymbol{f}'(\boldsymbol{x}) = \frac{\partial f}{\partial \boldsymbol{x}}(\boldsymbol{x})$ the Jacobi matrix, the matrix of the partial derivatives w.r.t. $\boldsymbol{x}$:

$$\boldsymbol{J}(\boldsymbol{x}) := \left(\frac{\partial f_i}{\partial x_j}(\boldsymbol{x})\right)_{ij} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \cdots & & & \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{363}$$

Therefore

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \boldsymbol{J}^{-1}(\boldsymbol{x})\, f(\boldsymbol{x_n}) \to \Delta\boldsymbol{x}_n = -\boldsymbol{J}^{-1}(\boldsymbol{x}_n) f(\boldsymbol{x}_n) \tag{364}$$

As direct inversion of $\boldsymbol{J}$ is expensive (see lin. algebra), usually solve system of linear equations:

$$J(\boldsymbol{x}_n)\Delta\boldsymbol{x}_n = -f(\boldsymbol{x}_n) \tag{365}$$

to get $\Delta\boldsymbol{x}_n$ and then $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta\boldsymbol{x}_n$

Newton's method VII

$\rightarrow$ Newton-Raphson method in n dimensions (i.e. system of equations) is expensive, therefore often used: *quasi Newton methods*

Example: statistical equilibrium

In the non-LTE case population numbers of ions n from statistical equations with transition rates P_{ij}, stationary: $\sum_{i\neq j} n_i\,P_{ij} = \sum_{j\neq i} n_j\,P_{ji}$ with $P_{ii} := -\sum_i P_{ji} \rightarrow \mathrm{n}\cdot\mathrm{P}(\mathrm{n}, J, T_\mathrm{e}) = 0$, matrix has block structure (but coupling extra line from charge conservation / electron density):

$$\mathrm{P} \;=\; \left[\begin{array}{c|c|c} \mathrm{H} & & \\ \hline & \mathrm{He} & \\ \hline & & \mathrm{N} \end{array}\right] \tag{366}$$

together with $J = \Lambda S(\mathrm{n})$. When using net-radiative brackets or accelerated Λ iteration:
$\rightarrow$ non-linear system of N equations $\rightarrow N^3$ derivatives (N derivatives for $N\times N$ rates)

Quasi Newton's methods: Broyden method I

Instead of calculating n^3 derivatives use modified *secant* equation

$$x_{k+1} = x_k - f(x_k) B_k^{-1} \tag{367}$$

$$\text{with "slope" } B_{k+1} = \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k} = \frac{\Delta y_k}{\Delta x_k} \qquad \to \Delta y_k = B_{k+1} \Delta x_k \tag{368}$$

But: Eq. (368) defines B only as $n-1$ dimensional subspace $\to$ need further constraints.
Broyden (1965): use updating algorithm

$$B_{k+1} = B_k + \frac{\Delta x_k^T \otimes (\Delta y_k - \Delta x_k B_k)}{|\Delta x_k|^2} \tag{369}$$

with dyadic product of two vectors (columns $\times$ rows) yielding matrix elements:
$(u^T \otimes v)_{ij} = u_i v_j$
Advantage: Broyden's formula (369) can be inverted analytically by help of

Quasi Newton's methods: Broyden method II

Sherman-Morrison-Woodbury lemma

$$(A + u^\mathsf{T} \otimes v)^{-1} = A^{-1} - \frac{A^{-1} u^\mathsf{T} \otimes v A^{-1}}{1 + v\, A^{-1} u^\mathsf{T}} \tag{370}$$

with row-vectors u, v and an invertible matrix A the required B_{k+1}^{-1} can be directly obtained from previous B_k^{-1}:

$$B_{k+1}^{-1} = B_k^{-1} + \frac{(B_k^{-1} \Delta x_k^\mathsf{T}) \otimes (\Delta x_k - \Delta y_k B_k^{-1})}{(\Delta y_k B_k^{-1}) \cdot \Delta x_k^\mathsf{T}} \tag{371}$$

$\rightarrow$ no operations between full matrices involved $\rightarrow$ only $\sim N^2$ multiplications

Quasi Newton's methods: Broyden method III

Broyden method

1. select starting point x_0 (e.g., initial guess on n from LTE population numbers) and starting matrix $B_0^{-1} = (f')^{-1}$ (Newton step)
2. $\mathrm{x}_{k+1} = \mathrm{x}_k - f(\mathrm{x}_k) B_k^{-1}$
3. stop if $|\Delta \mathrm{x}| < \epsilon$
4. else update Broyden matrix Eq. (371)

$$B_{k+1}^{-1} = B_k^{-1} + \frac{(B_k^{-1} \Delta x_k^{\mathrm{T}}) \otimes (\Delta x_k - \Delta y_k B_k^{-1})}{(\Delta y_k B_k^{-1}) \cdot \Delta x_k^{\mathrm{T}}}$$

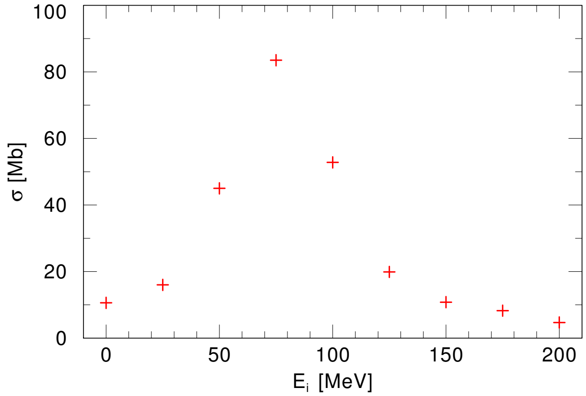
5. $k = k + 1$ goto 2

Interpolation

Interpolating data I

Consider following measurement of a cross section

E_i [MeV]	0	25	50	75	100	125	150	175	200
$\sigma(E_i)$ [Mb]	10.6	16.0	45.0	83.5	52.8	19.9	10.8	8.25	4.7



The cross section can be described by Breit-Wigner formula

$$f(E) = \frac{f_r}{(E - E_r)^2 + \Gamma^2/4} \qquad (372)$$

Interpolating data II

Interpolation problem

Task: Determine $\sigma(E)$ for values of E which lie between measured values of E

By, e.g.,

- numerical interpolation (assumption of data representation by polynomial in E):
  - piecewise constant $\rightarrow$ step function (easy to implement, error goes as $\sim y_i'(x_{i+1} - x_i)$)
  - piecewise linear (special case of polynomial)
  - polynomial (Lagrange)
  - piecewise Lagrange, cubic spline

  $\rightarrow$ ignores errors in measurement (noise)
- fitting parameters of an underlying model, e.g., Breit-Wigner with f_r, E_r, Γ, (taking errors into account), i.e., minimizing χ^2
- Fourier analysis

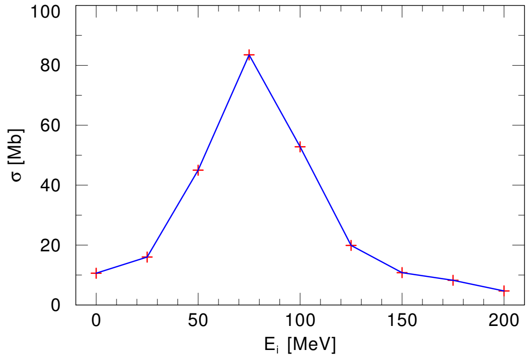
Interpolating data III

Linear interpolation

tabulated function $y_i = y(x_i)$, $i = 1 \ldots N$, e.g., for interval x_i, x_{i+1}, linear interpolation in this interval is by

$$y = A(x)y_i + B(x)y_{i+1} \tag{373}$$

$$A \equiv \frac{x_{i+1} - x}{x_{i+1} - x_i} \qquad B \equiv 1 - A = \frac{x - x_i}{x_{i+1} - x_i} \tag{374}$$



or:

$$y = y_i + (y_{i+1} - y_i)\frac{x - x_i}{x_{i+1} - x_i}$$

disadvantages:

- not differentiable at nodes x_i
- error $\sim y_i''(x_{i+1} - x_i)^2$

Interpolating data IV

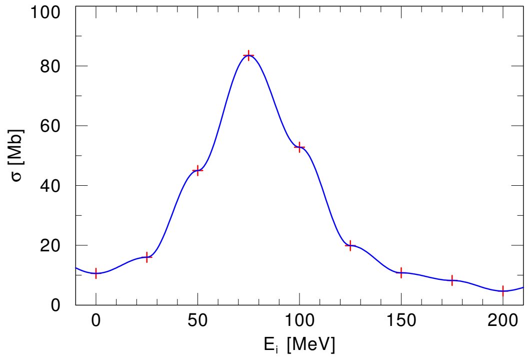
Cosine interpolation

a smoother transition between intervals can be achieved by piecewise cosine functions:

$$t = \frac{x - x_i}{x_{i+1} - x_i} \quad \text{(mapping on unit interval } [0,1]\text{)} \tag{375}$$

$$B = (y_{i+1} + y_i)/2 \; ; \qquad A = y_i - B \tag{376}$$

$$y = A \cos(\pi\, t) + B \tag{377}$$



note, that at the nodes x_i because of $\cos'(0) = 0 = \cos'(\pi) \rightarrow y'_i = 0$

Interpolating data V

Lagrange interpolation (global)

- fit $(n-1)$th degree polynomial through n data points (x_i, y_i)

$$p(x) = y_1\lambda_1(x) + y_2\lambda_2(x) + \ldots + y_n\lambda_n(x) \tag{378}$$

$$\lambda_i(x) = \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} = \frac{x - x_1}{x_i - x_1}\frac{x - x_2}{x_i - x_2}\cdots\frac{x - x_n}{x_i - x_n} \tag{379}$$

  where $\sum_{i=1}^{n} \lambda_i(x) = 1$

- practical: the λ_i are independent from the values of the function values $f_i \rightarrow$ for same nodes $x_i \rightarrow$ same λ_is (e.g., when measuring different y_is for same x_is)
- so, for $n = 9 \rightarrow (n-1) =$ 8th degree polynomial
- note that $\lambda_i(x_j) = \delta_{ij}$

Interpolating data VI

Example: Lagrange interpolation polynomial $n = 3$

$n = 3$ data points $\rightarrow n - 1 = 2$ degree polynomial, e.g., for points $P_1 = (-1; 4), P_2 = (0; 1), P_3 = (2; 5)$
$(x_1 = -1;\ x_2 = 0;\ x_3 = 2)$

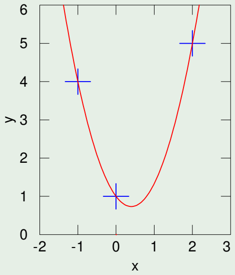
$$\lambda_1 = \frac{x - x_2}{x_1 - x_2} \cdot \frac{x - x_3}{x_1 - x_3} = \frac{(x-0)}{(-1-0)} \cdot \frac{(x-2)}{(-1-2)} = \frac{x^2 - 2x}{3} \quad (380)$$

$$\lambda_2 = \frac{x - x_1}{x_2 - x_1} \cdot \frac{x - x_3}{x_2 - x_3} = \frac{(x-(-1))}{(0-(-1))} \cdot \frac{(x-2)}{(0-2)} = \frac{x^2 - 2 - x}{-2} \quad (381)$$

$$\lambda_3 = \frac{x - x_1}{x_3 - x_1} \cdot \frac{x - x_2}{x_3 - x_2} = \frac{(x-(-1))}{(-2-(-1))} \cdot \frac{(x-0)}{(2-0)} = \frac{x^2 + x}{6} \quad (382)$$

$$p(x) = y_1 \cdot \lambda_1 + y_2 \cdot \lambda_2 + y_3 \cdot \lambda_3 = 4 \cdot \frac{x^2 - 2x}{3} + 1 \cdot \frac{x^2 - 2 - x}{-2} + 5 \cdot \frac{x^2 + x}{6} \quad (383)$$

$$= \frac{5}{3}x^2 - \frac{4}{3}x + 1 \quad (384)$$



Check: $\lambda_1 + \lambda_2 + \lambda_3 = \frac{x^2 - 2x}{3} + \frac{x^2 - 2 - x}{-2} + \frac{x^2 + x}{6} = 1$

Interpolating data VII

Application: Newton-Cotes formulae for integration

Idea: interpolate $f(x)$ in $\int_a^b f(x)dx$ with polynomial of degree n and integrate this polynomial exactely (note: now $n =$degree, start with $j = 0$):

$$\int_a^b f(x)dx \approx \int_a^b p_n(x)dx = \int_a^b \sum_{i=0}^{n} f(x_i) \cdot \lambda_i(x) \tag{385}$$

$$\lambda_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} \quad \xrightarrow{x=a+ht} \quad \phi_i(t) := \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{t-j}{i-j} \tag{386}$$

Note that the transformation $x = a + ht$ means that $x_0 = a + h \cdot 0$, $x_1 = a + h \cdot 1, \ldots$
(equidistant subintervals h on x-axis)
Therefore the integration of $p_n(x)$ yields

$$\int_a^b \sum_{i=0}^{n} f(x_i) \cdot \lambda_i(x) = h \sum_{i=0}^{n} f_i \int_0^n \phi_i(t)dt = h \sum_{i=0}^{n} f_i w_i \tag{387}$$

Interpolating data VIII

Example: Newton-Cotes formula $n = 1$

$$w_0 = \int_0^n \phi_0(t)dt = \int_0^1 \frac{t-1}{0-1}dt = \int_0^1 (1-t)dt = \frac{1}{2} \tag{388}$$

$$w_1 = \int_0^n \phi_1(t)dt = \int_0^1 \frac{t-0}{1-0}dt = \int_0^1 t\,dt = \frac{1}{2} \tag{389}$$

$$\int_a^b p_1(x)dx = h\sum_{i=0}^{1} f_i w_i = h\,(f_0\frac{1}{2} + f_1\frac{1}{2}) = \frac{h}{2}(f_0 + f_1) \tag{390}$$

$\rightarrow$ trapezoid rule

Analogously for $n = 2$, e.g.,

$$w_0 = \int_0^2 \frac{t-1}{0-1}\cdot\frac{t-2}{0-2}dt = \frac{1}{2}\int_0^2 (t^2 - 3t + 2)dt = \frac{1}{3} \tag{391}$$

and $w_1 = \frac{4}{3}$, $w_2 = \frac{1}{3} \rightarrow \int_a^b p_2(x)dx = \frac{h}{3}(f_0 + 4f_1 + f_2) \rightarrow$ Simpson's rule

Interpolating data IX

$\rightarrow$ closed Newton-Cotes formulae with nodes t_i on $[0,1]$: $t_0 = 0, t_i = \frac{i}{n}, t_n = 1$, use mapping $x_i = a + t_i(b-a)$, so

$$\int_a^b f(x)dx = \int_a^b p_n(x)dx + E_f = (b-a)\sum_{i=0}^{n} w_i f(x_i) + E_f \tag{392}$$

n	name	nodes t_i	weights w_i	E_f
1	trapezoid rule	$0\ 1$	$\frac{1}{2}\ \frac{1}{2}$	$-\frac{(b-a)^3}{12}f''$
2	Simpson's rule	$0\ \frac{1}{2}\ 1$	$\frac{1}{6}\ \frac{4}{6}\ \frac{1}{6}$	$-\frac{(b-a)^5}{2880}f^{(4)}$
3	3/8 rule	$0\ \frac{1}{3}\ \frac{2}{3}\ 1$	$\frac{1}{8}\ \frac{3}{8}\ \frac{3}{8}\ \frac{1}{8}$	$-\frac{(b-a)^5}{6480}f^{(4)}$
4	Milne rule	$0\ \frac{1}{4}\ \frac{2}{4}\ \frac{3}{4}\ 1$	$\frac{7}{90}\ \frac{32}{90}\ \frac{12}{90}\ \frac{32}{90}\ \frac{7}{90}$	$-\frac{(b-a)^7}{1935360}f^{(6)}$
	...			

for $n \geq 8$ some weights w_i are also negative $\rightarrow$ subtractive cancellation $\rightarrow$ useless

Note, again: $\sum w_i = 1$. The error: $E_f = h^{p+1} \cdot K \cdot f^{(p)}(\xi)$, $\xi \in (a,b)$

Interpolating data X

Trick: Neville's algorithm (sometimes confused with Aitken's method)

Instead of computing the whole Lagrange polynomial: nested linear interpolations

Where the $f_{i\ldots j}$ are recursively computed, e.g.,

$$\begin{array}{llll}
x_1 & f_1 & & \\
 & & f_{12} & \\
x_2 & f_2 & & f_{123} \\
 & & f_{23} & \\
x_3 & f_3 & &
\end{array}$$

$$f_{i\ldots j} = \frac{x - x_j}{x_i - x_j} f_{i\ldots j-1} + \frac{x - x_i}{x_j - x_i} f_{i+1\ldots j} \tag{393}$$

$$f_{123} = \frac{x - x_3}{x_1 - x_3} f_{12} + \frac{x - x_1}{x_3 - x_1} f_{23} \tag{394}$$

$\rightarrow$ sequence of ... linear interpolations = interpolation with polynomial of $n-1$ degree

$\rightarrow$ error can be estimated from $\frac{|f_{i\ldots j} - f_{i..j-1}| + |f_{i\ldots j} - f_{i+1..j}|}{2}$,

e.g, $\frac{|f_{12345} - f_{1234}| + |f_{12345} - f_{2345}|}{2}$

Interpolating data XI

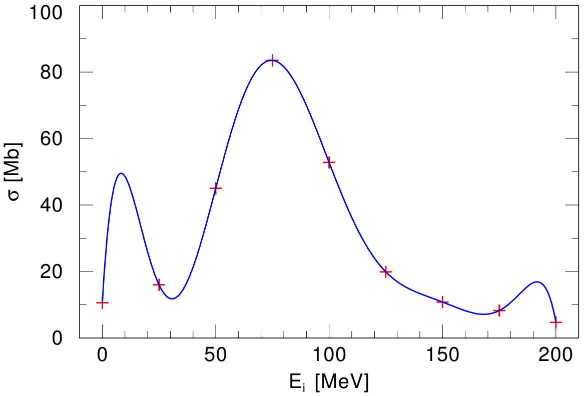
Neville's algorithm: code

```
// input : given points xi[], fi[], value of x for interpolation
// output: f at position x, error estimate df

for (i = 1 ; i <= n ; ++i) ft[i] = fi[i] ;

for (i = 1 ; i <= n-1 ; ++i) {
  for (j = 1 ; j <= n-i ; ++j) {
    x1 = xi[j] ; x2 = xi[j+1] ;
    f1 = ft[j] ; f2 = ft[j+1] ;
    ft[j] = (x - x1)/(x2 - x1) * f2 + (x - x2)/(x1 - x2) * f1
  }
}
f = ft[1] ;
df = (fabs(f - f1) + fabs(f - f2))/2. ;
```

Interpolating data XII

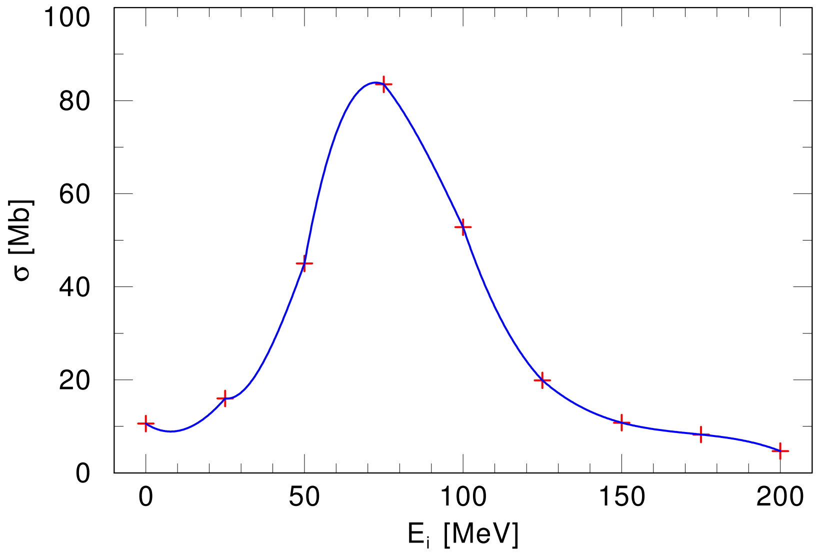
Runge's phenomenon:
polynomials $\rightarrow \pm\infty$ for $x \rightarrow \pm\infty$.

If function has different behavior (e.g., asymptotically constant) $\Rightarrow$ oscillations at intervall limits (e.g., for Runge's function $\frac{1}{1+x^2}$)



Interpolating data XIII

one possible solution for the problem of Runge's phenomenon: piecewise polynomials
here: 2nd degree polynomials (parabola, requires 3 points)



Problem:
not differentiable at x_i

Interpolating data XIV

better: Cubic Hermite spline

- remember: piecewise linear interpolation with functions

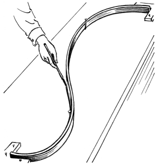
$$A(x) = \frac{x_{i+1} - x}{x_{i+1} - x_i} \qquad B(x) = 1 - A = \frac{x - x_i}{x_{i+1} - x_i} \tag{395}$$

$$\rightarrow\ y(x) = A(x)\,y_i + B(x)\,y_{i+1} \tag{396}$$

$\rightarrow$ 2nd derivative=0 in interval and undefined/infinite at interval points

- idea: get interpolation with smooth 1st derivative and continuous in 2nd derivative

A flat spline (lath) with fixed points (ducks) has minimum energy of bending $\rightarrow$ e.g., used for construction of hulls



Burmester stencils are splines of 3rd degree

Interpolating data XV

- if (assume!): not only y_i given, but also y_i'' $\rightarrow$ add cubic polynomial with 2nd derivative varying linearly between y_i'' to y_{i+1}'' *and* zero values for x_i and x_{i+1} (so y_i, y_{i+1} unchanged):

$$y(x) = A(x)\, y_i + B(x)\, y_{i+1} + C(x)\, y_i'' + D(x)\, y_{i+1}'' \tag{397}$$

$$C(x) \equiv \frac{1}{6}(A^3(x) - A(x))(x_{i+1} - x_i)^2 \qquad D(x) \equiv \frac{1}{6}(B^3(x) - B(x))(x_{i+1} - x_i)^2 \tag{398}$$

$\rightarrow x$ dependence only through $A(x)$, $B(x)$ $\rightarrow$ cubic x-dependence in $C(x)$, $D(x)$

- check: now y_i'' is 2nd derivative of interpolating polynomial (calculating $dA/dx, \ldots$):

$$\frac{dy}{dx} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{3A^2 - 1}{6}(x_{i+1} - x_i)y_i'' + \frac{3B^2 - 1}{6}(x_{i+1} - x_i)y_{i+1}'' \tag{399}$$

$$\frac{d^2y}{dx^2} = Ay_i'' + By_{i+1}'' \tag{400}$$

note that $A = 1$ and $B = 0$ at x_i; and $A = 0$ and $B = 1$ at x_{i+1}, so y'' is ok ($\checkmark$)

Interpolating data XVI

- however: in most cases y_i'' not known
  idea $\rightarrow$ 1st derivative shall be continuous across interval boundaries $\rightarrow$ gives equation for 2nd derivatives
- so: Eq. (399) shall be same for x_i on $[x_{i-1}, x_i]$ and on $[x_i, x_{i+1}]$ (for $i = 2, \ldots, N-1$) yielding $N-2$ equations

$$\frac{x_i - x_{i-1}}{6} y_{i-1}'' + \frac{x_{i+1} - x_{i-1}}{3} y_i'' + \frac{x_{i+1} - x_i}{6} y_{i+1}'' = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \tag{401}$$

  with N unknown y_i'' $\rightarrow$ need further constraint
- often: y_1'' and y_N'' set to 0 $\rightarrow$ natural cubic spline
- advantage of cubic splines: linear set of equations and also tridiagonal, each y_i'' couples only to nearest neighbors

Interpolating data XVII

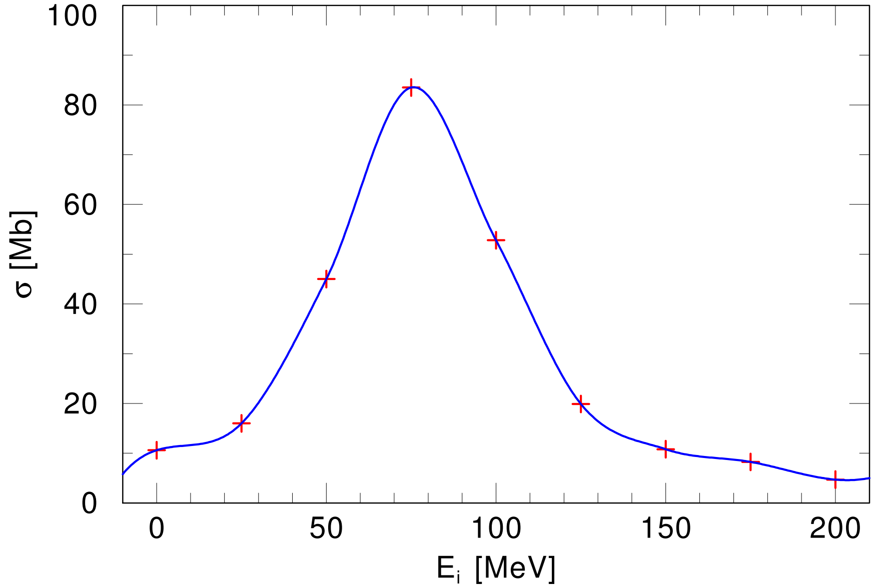
- hence with mapping $t = (x - x_i)/(x_{i+1} - x_i)$ on unit interval $[0, 1]$

$$p(t) = T\, M_\mathrm{h}\, C = (t^3\ \ t^2\ \ t) \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} y_i \\ y_{i+1} \\ m_i \\ m_{i+1} \end{pmatrix} \tag{402}$$

$$\begin{aligned} p(t) &= (2t^3 - 3t^2 + 1)y_i + (-2t^3 + 3t^2)y_{i+1} \\ &\quad + (t^3 - 2t^2 + t)m_i + (t^3 - t^2)m_{i+1} \end{aligned} \tag{403}$$

with the numericial 1st derivatives $m_i = \frac{1}{2}\left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}} + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right)$ and $m_{i+1} = \frac{1}{2}\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i} + \frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}}\right)$ and $m_1 = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$ and $m_n = 0$

Interpolating data XVIII



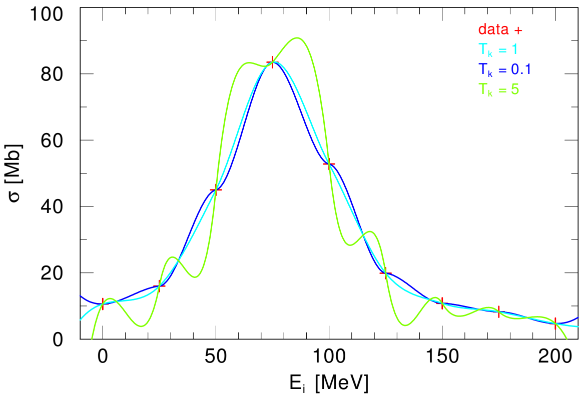
Cubic spline interpolation

Interpolating data XIX

Catmull-Rom splines

The "width" of the curve segment can be controlled by a parameter T_k according to (for $k = 2, \ldots, n-2$):

$$m_k = T_k \frac{y_{k+1} - y_{k-1}}{x_{k+1} - x_{k-1}} \tag{404}$$



Interpolating in 2D I

Simplest method on a rectilinear 2D grid: bilinear interpolation, i.e, linear interpolation in one direction, then again in another direction
$\rightarrow$ as for Neville's algorithm $2\times$ linear = quadratic order
If four f values are given as follows: $f_1 : Q_{11} = (x_1, y_1)$, $f_2 : Q_{12} = (x_1, y_2)$, $f_3 : Q_{21} = (x_2, y_1)$, $f_4 : Q_{22} = (x_2, y_2)$ then
1. linear interpolation in x-direction:

$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \tag{405}$$

$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \tag{406}$$

Interpolating in 2D II

2. linear interpolation in y-direction:

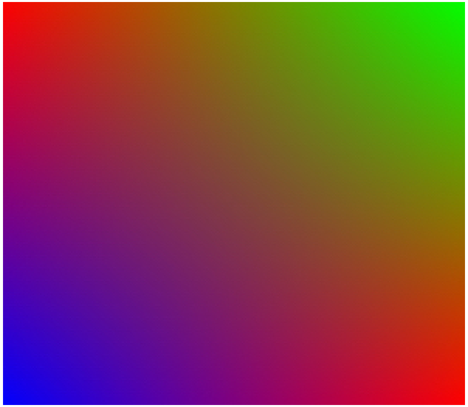
$$\begin{aligned}
f(x,y) &\approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2) \\
&= \frac{y_2 - y}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \right) \\
&+ \frac{y - y_1}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \right) \\
&= \frac{1}{(x_2 - x_1)(y_2 - y_1)} \left(f(Q_{11})(x_2 - x)(y_2 - y) \right. \\
&+ f(Q_{21})(x - x_1)(y_2 - y) + f(Q_{12})(x_2 - x)(y - y_1) \\
&\left. + f(Q_{22})(x - x_1)(y - y_1) \right) \qquad (407)
\end{aligned}$$

$\rightarrow$ same result as for 1. y-direction + 2. x direction

Interpolating in 2D III

So:

$$
\begin{aligned}
f(x,y) = {} & \frac{1}{(x_2 - x_1)(y_2 - y_1)} \\
& \cdot \left(f_1(x_2 - x)(y_2 - y)\right. \\
& + f_3(x - x_1)(y_2 - y) \\
& + f_2(x_2 - x)(y - y_1) \\
& \left. + f_4(x - x_1)(y - y_1)\right)
\end{aligned}
\tag{408}
$$



Example, here: rgb colors on corner points $f_{11} = b$, $f_{12} = f_{21} = r$, $f_{22} = g$

Interpolating in 2D IV

As the interpolation can also be written as:

$$f(x,y) = \sum_{i=0}^{1}\sum_{j=0}^{1} a_{ij}x^i y^j = a_{00} + a_{10}x + a_{01}y + a_{11}xy \tag{409}$$

$$a_{00} = f(0,0), \tag{410}$$

$$a_{10} = f(1,0) - f(0,0), \tag{411}$$

$$a_{01} = f(0,1) - f(0,0), \tag{412}$$

$$a_{11} = f(1,1) + f(0,0) - \big(f(1,0) + f(0,1)\big). \tag{413}$$

$\rightarrow$ interpolation only linear along lines of const. x or const. y, any other direction:
quadratic in position (but linear in f)

$\rightarrow$ other method: bicubic interpolation $f(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j$ with 16 coefficients

$\rightarrow$ extension to 3D: trilinear interpolation, tricubic interpolation (64 coefficients)

Numerical Integration and Differentiation

Numerical Integration I

(see also Landau et al. 2007)

Computing integrals

Often integrals have to be evaluated numerically. Examples:

- measured $dN(t)/dt$, the rate of some events, e.g., photons per unit time interval. Task: Determine the number of photons in the first second:

$$N(1) = \int_0^1 \frac{dN(t)}{dt} dt \tag{414}$$

- radiative rates in the statistical equations for non-LTE population numbers (stellar atmospheres, photoionized nebulae)

$$R_{\ell u} = \int \frac{4\pi}{h\nu} \sigma_{\ell u}(\nu) J_\nu d\nu \quad \text{where (in 1d): } J_\nu = \frac{1}{2} \int_{-1}^{1} I_\nu \, d(\cos\theta) \tag{415}$$

Numerical Integration II

Also, *analytical* integration sometimes difficult or impossible (e.g., elliptic integrals), but numerically straightforward. So, Riemann definition

$$\int_a^b f(x)dx = \lim_{h\to 0}\left[h\sum_{i=1}^{(b-a)/h} f(x_i)\right] \tag{416}$$

summing up areas of boxes of height $f(x)$ and width $h \to$ numerical quadrature

$$\int_a^b f(x)dx \approx \sum_{i=1}^{N} f(x_i)w_i \tag{417}$$

$\to$ problem: find appropriate sampling $f_i \equiv f(x_i)$, with *weights* w_i
generally: result improves with N

Numerical Integration III

some hints

- remove singularities before integration
- sometimes splitting of interval is helpful, e.g.,

$$\int_{-1}^{1} f(|x|)dx = \int_{-1}^{0} f(-x)dx + \int_{0}^{1} f(x)dx \tag{418}$$

- or transformation/substitution

$$\int_{0}^{1} x^{1/3}dx = \int_{y(0)=0^{1/3}}^{y(1)=1^{1/3}} y\,3y^2dy \qquad \left(y(x) = x^{1/3} \rightarrow\ dx = 3x^{2/3}dy = 3y^2dy\right) \tag{419}$$

Trapezoid rule I

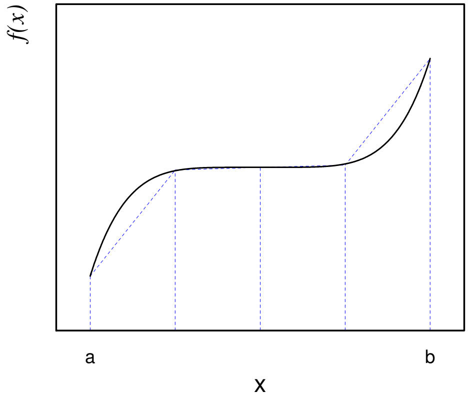
The Trapezoid rule

- uses values $f(x)$ at *evenly* spaced x_i $(i = 1, \ldots, N)$ with step size h on *integration region* $[a, b]$, including endpoints

- hence, $N - 1$ *intervals* of length h:

$$h = \frac{b-a}{N-1} \qquad x_i = a + (i-1)h \tag{420}$$

- so construct trapezoid on interval i of width $h \rightarrow f(x)$ approximated by straight line between $(a + i \cdot h, f_i)$ and $(a + (i+1) \cdot h, f_{i+1})$



Trapezoid rule II

with average height $(f_i + f_{i+1})/2$:

$$\int_{x_i}^{x_i+h} f(x)dx \simeq \frac{h(f_i + f_{i+1})}{2} = \frac{1}{2}hf_i + \frac{1}{2}hf_{i+1} \tag{421}$$

i.e. Eq. (417): $\int_a^b f(x)dx \approx \sum_{i=1}^N f(x_i)w_i$ for $N = 2$ and $w_i = \frac{1}{2}h$

- hence for full integration region $[a, b]$

$$\int_a^b f(x)dx \approx \frac{h}{2}f_1 + hf_2 + hf_3 + \ldots + hf_{N-1} + \frac{h}{2}f_N \tag{422}$$

i.e. $w_i = \{h/2, h, \ldots, h, h/2\}$

Simpson's rule I

Simpson's rule

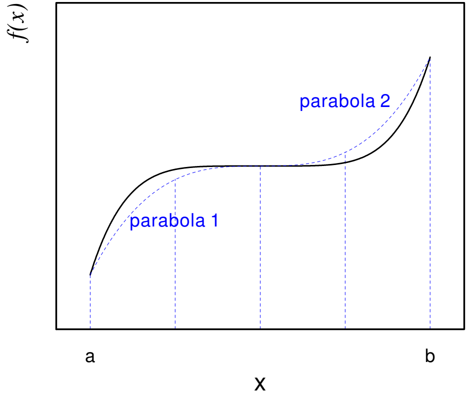
- similar to Trapezoid rule, but with _odd_ number of points N
- for each interval: $f(x)$ approximated by parabola

$$f(x) = \alpha x^2 + \beta x + \gamma \qquad (423)$$

hence area for each interval:

$$\int_{x_i}^{x_i+h} (\alpha x^2 + \beta x + \gamma) dx \qquad (424)$$

$\rightarrow$ like integrating the corresponding Taylor series up to *quadratic* term



Simpson's rule II

- need to determine α, β, γ for $f(x)$, so consider interval $[-1, 1]$

$$\int_{-1}^{1} (\alpha x^2 + \beta x + \gamma) dx = \frac{1}{3}\alpha x^3 + \frac{1}{2}\beta x^2 + \gamma x \Big|_{-1}^{+1} = \frac{2\alpha}{3} + 2\gamma \tag{425}$$

and $f(-1) = \alpha - \beta + \gamma$, $f(0) = \gamma$, $f(1) = \alpha + \beta + \gamma$, therefore:

$$\Rightarrow \alpha = \frac{f(1) + f(-1)}{2} - f(0), \quad \beta = \frac{f(1) - f(-1)}{2}, \quad \gamma = f(0) \tag{426}$$

so insert Eqn. (426) into Eq. (425)

$$\int_{-1}^{1} (\alpha x^2 + \beta x + \gamma) dx = \frac{2\alpha}{3} + 2\gamma = \frac{f(-1)}{3} + \frac{4f(0)}{3} + \frac{f(1)}{3} \tag{427}$$

Simpson's rule III

- or more general: use two neighboring intervals to evaluate $f(x)$ at three points for the parabola fit

$$\int_{x_i-h}^{x_i+h} f(x)dx = \int_{x_i-h}^{x_i} f(x)dx + \int_{x_i}^{x_i+h} f(x)dx \tag{428}$$

$$\simeq \frac{h}{3}f_{i-1} + \frac{4h}{3}f_i + \frac{h}{3}f_{i+1} \tag{429}$$

$\rightarrow$ pairs of intervals (hence: odd N)

- so for total integration region $[a, b]$

$$\int_a^b f(x)dx \approx \frac{h}{3}f_1 + \frac{4h}{3}f_2 + \frac{2h}{3}f_3 + \frac{4h}{3}f_4 + \ldots \frac{2h}{3}f_{N-2} + \frac{4h}{3}f_{N-1} + \frac{h}{3}f_N \tag{430}$$

with $w_i = \left\{\frac{h}{3}, \frac{4h}{3}, \frac{2h}{3}, \frac{4h}{3}, \ldots, \frac{4h}{3}, \frac{h}{3}\right\}$ $\rightarrow$ check: $\sum_{i=1}^{N} w_i \overset{!}{=} (N-1)h$

Integration error I

$\rightarrow$ numerical integration : use algorithm with least number of integration points for accurate answer
estimate error from Taylor expansion at midpoint of interval, e.g., for trapezoid rule $hf^{(2)}\frac{h^2}{12}$, $\times$ number of subintervals $N = [b-a]/h$:

$$E_{\text{trap}} = \mathcal{O}\left(\frac{[b-a]^3}{12\,N^2}\right) f^{(2)}, \qquad E_{\text{Simps}} = \mathcal{O}\left(\frac{[b-a]^5}{180\,N^4}\right) f^{(4)} \tag{431}$$

$$\epsilon_{\text{trap, Simps}} \simeq \frac{E_{\text{trap, Simps}}}{f} \tag{432}$$

Note that for Simpson's rule 3rd derivate cancels and $E \propto 1/N^4$
$\rightarrow$ Simpson's rule should converge faster

Integration error II

check: find N for minimum total error (usually for $\epsilon_{\text{ro}} \approx \epsilon_{\text{appr}}$):

$$\epsilon_{\text{tot}} = \epsilon_{\text{ro}} + \epsilon_{\text{approx}} \approx \sqrt{N}\epsilon_{\text{m}} + \epsilon_{\text{trap, Simps}} \tag{433}$$

$$\rightarrow \epsilon_{\text{ro}} \overset{!}{=} \epsilon_{\text{trap, Simps}} = \frac{E_{\text{trap, Simps}}}{f} \tag{434}$$

Assuming some scale:

$$\frac{f^{(n)}}{f} \approx 1 \qquad b - a = 1 \qquad \Rightarrow \quad h = \frac{1}{N} \tag{435}$$

Integration error III

For double precision ($\epsilon_\mathrm{m} \approx 10^{-15}$) and **trapezoid rule:**

$$\sqrt{N}\epsilon_\mathrm{m} \approx \frac{f^{(2)}(b-a)^3}{fN^2} = \frac{1}{N^2} \tag{436}$$

$$\Rightarrow N \approx \frac{1}{(\epsilon_\mathrm{m})^{2/5}} = \left(\frac{1}{10^{-15}}\right)^{2/5} = 10^6 \tag{437}$$

$$\Rightarrow \epsilon_\mathrm{ro} \approx \sqrt{N}\epsilon_\mathrm{m} = 10^{-12} \tag{438}$$

For double precision ($\epsilon_\mathrm{m} \approx 10^{-15}$) and **Simpson's rule:**

$$\sqrt{N}\epsilon_\mathrm{m} \approx \frac{f^{(4)}(b-a)^5}{fN^4} = \frac{1}{N^4} \tag{439}$$

$$\Rightarrow N \approx \frac{1}{(\epsilon_\mathrm{m})^{2/9}} = \left(\frac{1}{10^{-15}}\right)^{2/9} = 2154 \tag{440}$$

$$\Rightarrow \epsilon_\mathrm{ro} \approx \sqrt{N}\epsilon_\mathrm{m} = 5 \times 10^{-14} \tag{441}$$

Integration error IV

We conclude:

- Simpson's rule is better
- Simpson's rule gives errors close to ϵ_m (in general for higher order integration algorithms, e.g., RK4)
- best numerical approximation not for $N \to \infty$, but small $N \leq 1000$
- however, as $\epsilon_\mathrm{Simps} \sim f^{(4)} \to$ only for sufficiently smooth functions, i.e., for narrow peak-like functions trapezoidal rule might be more efficient

Gaussian quadrature I

So far: improvement by smart choice of weights w_i, but still equally spaced points x_i (= const. h) for integral evaluation (cf. Eq. (417)),
now: additional freedom of choosing x_i so that *order is twice* that of previous integration formulae (so-called Newton-Cotes formulae, see $\rightarrow$ interpolation) for *same number of nodes* N
$\rightarrow$ compute $N \times f(x_i)$.
$\rightarrow$ choose w_i and x_i such that integral is *exact* for

orthogonal polynomials $\times$ specific weight function $W(x)$

$$\int_a^b f(x)dx = \int_a^b W(x)g(x)dx \approx \int_a^b W(x)p_n(x)dx = \sum_{i=1}^{N} g(x_i)\, w_i = \sum_{i=1}^{N} \frac{f(x_i)}{W(x_i)} w_i \tag{442}$$

Note that the integration of the orthogonal polynomials is on $[-1;+1]$, hence a transformation of the variables is usually necessary, e.g., for $W(x) \equiv 1$:

$$\int_a^b g(x)dx \approx \frac{b-a}{2} \sum_{i=1}^{N} g\left(\frac{b-a}{2}x_i + \frac{a+b}{2}\right) w_i \tag{443}$$

Gaussian quadrature II

Example: Gauß-Chebyshev quadrature

The weight function is $W(x) = \frac{1}{\sqrt{1-x^2}}$, i.e, with $g(x) = f(x)\sqrt{1-x^2}$

$$\int_{-1}^{+1} f(x)dx = \int_{-1}^{+1} \frac{g(x)}{\sqrt{1-x^2}}dx \approx \int_{a}^{b} \frac{T_n(x)}{\sqrt{1-x^2}}dx = \sum_{i=1}^{N} g(x_i)w_i = \sum_{i=1}^{N} f(x_i)\sqrt{1-x_i^2}\, w_i \tag{444}$$

with analytic(!) $w_i = \frac{\pi}{N}$, and $x_i = \cos\left(\frac{2i-1}{2N}\pi\right)$ are the zeros of the associated Chebyshev polynomials of 1st kind $T_n(x)$, with $T_{n+1}(x) = 2xT_n(x) - T_{n-1}$, $T_0(x) = 1$, $T_1(x) = x$ and

$$\int_{-1}^{+1} T_n(x)\, W(x)\, T_m(x)dx = \delta_{nm} \tag{445}$$

And for the Chebyshev polynomials of 2nd kind $U_n(x)$ analogously:
$W(x) = \sqrt{1-x^2}$, $w_i = \frac{\pi}{N+1}\sin^2\left(\frac{i}{N+1}\pi\right)$, $x_i = \cos\left(\frac{i}{N+1}\pi\right)$

Gaussian quadrature III

Gauß-Chebyshev quadrature in C++ for some $f(x)$ on $[a; b]$

```
double gaussc (double const &a, double const &b, int const &N) {
   ...
 for ( i = 0 ; i < N ; ++i ) {
   x[i] = cos ( ((2. * (i+1) - 1.) * M_PI ) / (double(N) *2.) )  ;
   w[i] = M_PI / double(N) * (b-a) / 2. ; // transform weights [-1;1]->[a;b]
 }
 sum = 0. ;
 for (i = 0 ; i < N ; ++i) { // transform x in f(x), but not in sqrt()
   sum += f( x[i]*(b-a)/2. + (a+b)/2. ) * sqrt(1.-x[i]*x[i]) * w[i] ;
 }
 return sum  ;
}
```

→ note that this is maybe not optimum for some function $f(x)$, but should be rather used for functions of the form $g(x)/\sqrt{1-x^2}$

Gaussian quadrature IV

Most often: $W(x) \equiv 1 \rightarrow$ Gauß-Legendre quadrature with Legendre Polynomials $P_n(x)$, which are the solutions to Legendre's differential equation (a special case of the Sturm-Liouville differential equation) $\rightarrow$ Laplace equation in 3D for spherical coordinates $\rightarrow$ QM

$$\frac{d}{dx}\left[(1-x^2)\frac{dP_n(x)}{dx}\right] + n(n+1)P_n(x) = 0 \tag{446}$$

$$\rightarrow \quad P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left(x^2-1\right)^n \qquad \text{(Rodrigues' formula)} \tag{447}$$

so, $P_0(x) = 1$, $P_1(x) = x$, $P_2(x) = \frac{1}{2}(3x^2-1)$, $\ldots$
Then, the n weights (for the n points of the interval)

$$w_i = \frac{2}{(1-x_i^2)[P_n'(x_i)]^2} \tag{448}$$

where x_i are the n zeros of $P_n(x)$

Gaussian quadrature V

Table: Exact values for Gauß-Legendre integration for $n = 2, 3$

n	P_n	P'_n	x_i	w_i
2	$\frac{1}{2}(3x^2 - 1)$	$3x$	$\pm\frac{1}{\sqrt{3}}$	$1, 1$
3	$\frac{1}{2}(5x^3 - 3x)$	$\frac{1}{2}(15x^2 - 3)$	$0, \pm\sqrt{\frac{3}{5}}$	$\frac{8}{9}, \frac{5}{9}, \frac{5}{9}$

Alternatively, the n zeros of $P_n(x)$ can be computed, e.g., via Newton's method ($x_{k+1} = x_k - P(x_k)/P'(x_k)$), one may use the start approximation ($i = 1, \ldots, n$):

$$x_i \approx \cos\left(\frac{4i - 1}{4n + 2}\pi\right) \tag{449}$$

Then the values of $P_n(x)$ and $P'_n(x)$ for Newton's method can be obtained via recursion:

$$nP_n(x) = (2n - 1)xP_{n-1}(x) - (n - 1)P_{n-2}(x) \tag{450}$$

$$\rightarrow P_n(x) = [(2n - 1)xP_{n-1}(x) - (n - 1)P_{n-2}(x)]/n \tag{451}$$

$$(x^2 - 1)P'_n(x) = nxP_n(x) - nP_{n-1}(x) \tag{452}$$

$$\rightarrow P'_n(x) = (nxP_n(x) - nP_{n-1}(x))/(x^2 - 1) \tag{453}$$

Gaussian quadrature VI

Finally, the transformation from $t \in [-1; +1] \rightarrow x \in [a; b]$ can be done via the midpoint $\frac{a+b}{2}$

$$x_i = t_i \frac{b-a}{2} + \frac{a+b}{2} \tag{454}$$

$$w_{i,x} = w_{i,t} \frac{b-a}{2} \tag{455}$$

Gaussian quadrature VII

Alternatively, other mappings are possible, allowing for integration of improper integrals with the Gauß-Legendre quadrature

interval	midpoint	$x_i(t_i)$	$w_{i,x}$
$[a, b]$	$\frac{a+b}{2}$	$\frac{a+b}{2} + \frac{b-a}{2} t_i$	$\frac{b-a}{2} w_{i,t}$
$[0; \infty]$	a	$a\dfrac{1+t_i}{1-t_i}$	$\dfrac{2a}{(1-t_i)^2} w_{i,t}$
$[-\infty; +\infty]$	scale a	$a\dfrac{t_i}{1-t_i^2}$	$\dfrac{a(1+t_i^2)}{(1-t_i)^2} w_{i,t}$
$[b; +\infty]$	$a+2b$	$\dfrac{a+2b+at_i}{1-t_i}$	$\dfrac{2(b+a)}{(1-t_i)^2} w_{i,t}$
$[0; b]$	$ab/(b+a)$	$\dfrac{ab(1+t_i)}{b+a-(b-a)t_i}$	$\dfrac{2ab^2}{(b+a-(b-a)t_i)^2} w_{i,t}$

Gaussian quadrature VIII

Moreover, there exist other orthogonal polynomials useful for Gauß quadrature

interval	polynomials	$W(x)^\dagger$
$[-1;1]$	Legendre	1
$[-1;1]$	Chebyshev 1st kind	$\frac{1}{\sqrt{1-x^2}}$
$[-1;1]$	Chebyshev 2nd kind	$\sqrt{1-x^2}$
$(-1;1)$	Jacobi	$(1-t)^\alpha(1+x)^\beta, \quad \alpha,\beta > -1$
$[0;+\infty)$	Laguerre	e^{-x}
$[0;+\infty)$	Generalized Laguerre	$x^\alpha e^{-x}, \quad \alpha > -1$
$(-\infty;+\infty)$	Hermite	e^{-x^2}

†Remember that function $f(x)$ should be of form $g(x)W(x)$

Gaussian quadrature IX

In general, the Gauß quadrature is constructed from orthogonal polynomials $p_n(x)$ with

$$\int_a^b p_n(x)\, W(x)\, p_{n'}(x) dx = \langle p_n | p_{n'} \rangle = \mathcal{N}_n\, \delta_{nn'} \tag{456}$$

where $\mathcal{N}_n$ is a normalization constant. If we choose the roots x_i of $p_n(x) = 0$ and

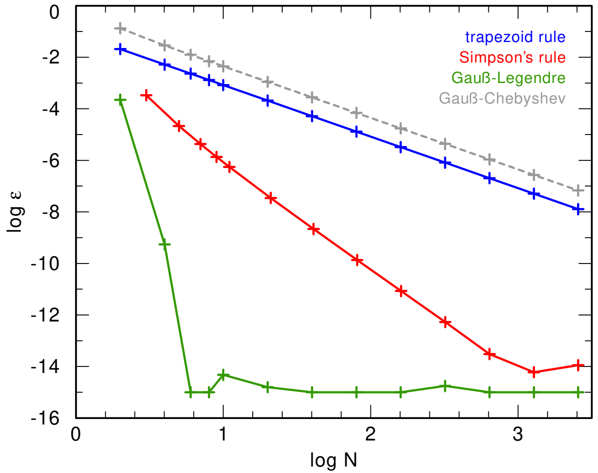
$$w_i = \frac{-a_n \mathcal{N}_n}{p'_n(x_i)\, p_{n+1}(x_i)} \tag{457}$$

with $i = 1, \ldots, n$, then the error in the quadrature is

$$\int_a^b f(x) dx - \sum_{i=1}^{n} g(x_i) w_i = \frac{\mathcal{N}_n}{A_n^2 (2n)!}\, g^{(2n)}(x_0) \tag{458}$$

where x_0 is some value in $[a, b]$, A_n a coefficient of the x^n term in the polynomial $p_n(x)$, $a_n = A_{n+1}/A_n$, e.g, for the Legendre polynomials $a_n = (2n+1)/(n+1)$ and $\mathcal{N}_n = 2/(2n+1)$.

Gaussian quadrature X



Numerical integration of $\exp(-x)$ on $[0,1]$ with different methods and number of integration points. Note that for Simpson's rule N must be odd.

Gauß-Legendre quadrature with $W(x) \equiv 1$ is superior to simple methods with fixed integration step width. Gauß-Chebyshev is not optimal, as $W(x) = \frac{1}{\sqrt{1-x^2}}$

Romberg integration I

Ideally: choose required accuracy $\epsilon \rightarrow$ know n for Gaussian quadrature (e.g, from Eq. (458)). Unfortunately, usually impossible. Therefore: increase n until ϵ small enough, recalculate all $f(x_i)$ for new degree $n \rightarrow$ disadvantage of Gaussian quadrature

Idea: trapezoid rule with subsequent calls with increasing n to refine until precision ϵ reached:

```
void trap (double const &a, double const &b, double &s, int const &n)
  ...
if (n == 1) s = 0.5 * (b-a) * (f(a)+f(b)) ;
else {
 it = pow(2,(n-2)) ;
 delx = (b-a) / double(it) ;
 x = a + 0.5 * delx ;
 sum = 0. ;
 for (int j=1 ; j <= it ; ++j) {
  sum += f(x) ; x += delx ; }
 s = 0.5 * (s + (b-a) * sum / double(it)) ;
}
```

Romberg integration II

For the trapezoid rule the approximation error (starting with $\frac{1}{n^2}$ has only even powers of $\frac{1}{n}$):

$$\int_{x_1}^{x_n} f(x)dx = h\left[\frac{1}{2}f_1 + f_2 \ldots f_{n-1} + \frac{1}{2}f_n\right] \tag{459}$$

$$- \frac{B_2 h^2}{2!}(f'_n - f'_1) - \ldots - \frac{B_{2k} h^{2k}}{(2k)!}(f_n^{(2k-1)} - f_1^{(2k-1)}) - \ldots \tag{460}$$

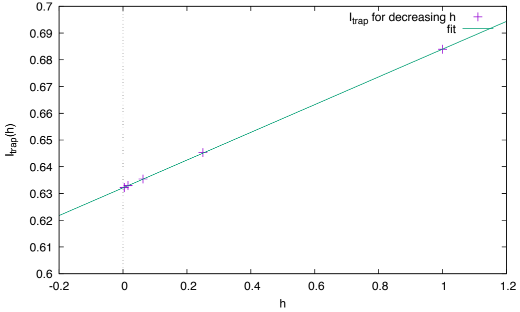
If compute Eq. (460) (without the error terms) for n and get s_n and once more with $2n$ and get s_{2n}, then leading error term in 2nd call is $1/4$ of error in 1st call, hence

$$s = \frac{4}{3}s_{2n} - \frac{1}{3}s_n \tag{461}$$

cancels leading error term, $1/n^4$ remains $\rightarrow$ recovers Simpson's rule

Romberg integration III

Often better: trapezoid rule for different N (or $h = \frac{b-a}{N}$) + extrapolation for $h \to 0$ (cf. Richardson extrapolation) $\to$ Romberg integration



1. calculate $I(h_k)$ for series h_k
2. extrapolate $(h_k^2, I(h_k))$ with polynomial in h^2

e.g., $\int_0^1 e^{-x}dx$

Note that polynomial $(a + bh^2)$ in h^2 is plotted, although h is used for the trapezoid rule $\to$ extrapolate polynomial in h^2

$\to$ trapezoid rule ideal: expansion in even powers of h (each refinement $\to$ 2 orders accuracy) and $I(h) = h(\frac{1}{2}f(a) + \sum_{j=1}^{N-1} f(x_j) + \frac{1}{2}f(b)) \to$ recycle already calculated nodes for $h/2$

Numerical differentiation I

Sometimes numerical derivative needed, e.g., for minimization algorithms, Newton method for root finding, so

$$f' = \frac{df(x)}{dx} := \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{462}$$

Problem: for $h \to 0 \to f(x+h) \approx f(x)$
$\to$ subtractive cancelation for numerator
& machine precision limit for denominator

often better (e.g., for large noise): analytic approximation of function (see, e.g., $\to$ interpolation) and its derivative

Numerical differentiation II

Forward difference

Taylor series with step size h

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f^{(3)}(x) + \ldots \tag{463}$$

$\rightarrow$ forward difference by solving Eq. (463) for f'

$$f'_{\mathrm{fd}}(x) := \frac{f(x+h) - f(x)}{h} \simeq f'(x) + \frac{h}{2}f''(x) + \ldots \tag{464}$$

approximate function by straight line through two points, error $\sim h$, e.g, consider $f(x) = a + bx^2$

$$f'_{\mathrm{fd}}(x) \approx \frac{f(x+h) - f(x)}{h} = 2bx + bh \quad \text{vs. exact } f' = 2bx \tag{465}$$

$\rightarrow$ only good for small $h \ll 2x$

Numerical differentiation III

Central difference

modify Eq. (462) by stepping forward $h/2$ and backward $h/2$

$$f'_{\text{cd}} := \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h} \tag{466}$$

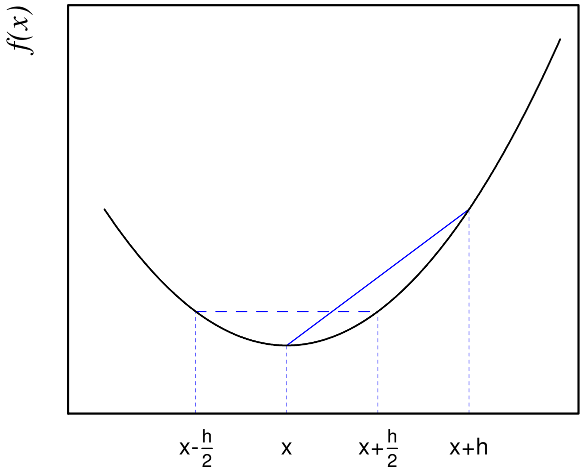
So, if we insert Taylor series for $f(x+\pm\frac{h}{2})$ in to Eq. (466)

$$f'_{\text{cd}} := \frac{\left[f(x) + \frac{h}{2}f'(x) + \frac{h^2}{8}f''(x) + \right] - [\ldots]}{h} \simeq f'(x) + \frac{1}{24}h^2 f^{(3)}(x) + \ldots \tag{467}$$

$\rightarrow$ all terms with odd power of h cancel $\rightarrow$ accuracy is of order h^2
if function well behaved, i.e., $f^{(3)}h^2/24 \ll f^{(2)}h/2$
$\rightarrow$ error for central difference method $\ll$ forward difference method, e.g., for $f(x) = a + bx^2$

$$f'_{\text{cd}}(x) \approx \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h} = 2bx \quad \text{vs. exact } f' = 2bx \tag{468}$$

Numerical differentiation IV



Forward difference (solid line) and central difference (dashed)
$\rightarrow$ central difference more accurate

Numerical differentiation V

Extrapolated difference

try to make also h^2 vanish by *algebraic exatrapolation*

$$f'_{\text{ed}}(x) \simeq \lim_{h\to 0} f'_{\text{cd}} \tag{469}$$

$\rightarrow$ need additional information for extrapolation by central difference with step size $h/2$:

$$f'_{\text{cd}}(x, h/2) = \frac{f(x+h/4) - f(x-h/4)}{h/2} \approx f'(x) + \frac{h^2 f^{(3)}(x)}{96} + \dots \tag{470}$$

We elminate linear and quadratic error term by forming

$$f'_{\text{ed}}(x) := \frac{4\frac{f(x+h/4)-f(x-h/4)}{h/2} - \frac{f(x+h/2)-f(x-h/2)}{h}}{3} \tag{471}$$

$$\approx f'(x) - \frac{h^4 f^{(5)}(x)}{4\cdot 16\cdot 120} + \dots \tag{472}$$

Numerical differentiation VI

for $h = 0.4$ and $f^{(5)} \simeq 1 \rightarrow$ approximation error close to ϵ_m. To minimize subtractive cancelation write Eq. (471) as

$$f'_\mathrm{ed}(x) = \frac{1}{3h}\left(8\left[f\left(x+\frac{h}{4}\right) - f\left(x-\frac{h}{4}\right)\right] - \left[f\left(x+\frac{h}{2}\right) - f\left(x-\frac{h}{2}\right)\right]\right) \tag{473}$$

Numerical differentiation VII

Error analysis

$\rightarrow$ usually decreasing h reduces approximation error but increases roundoff error (e.g., more calculation steps needed), moreover: subtractive cancelation. Hence, difference

$$f' \approx \frac{f(x+h) - f(x)}{h} \approx \frac{\epsilon_{\mathrm{m}}}{h} \approx \epsilon_{\mathrm{ro}} \tag{474}$$

and

$$\epsilon^{\mathrm{fd}}_{\mathrm{approx}} \approx \frac{f^{(2)}h}{2}, \qquad \epsilon^{\mathrm{cd}}_{\mathrm{approx}} \approx \frac{f^{(3)}h^2}{24} \tag{475}$$

Therefore $\epsilon_{\mathrm{ro}} \approx \epsilon_{\mathrm{approx}}$ for

Numerical differentiation VIII

$$\frac{\epsilon_\mathrm{m}}{h} \approx \epsilon_\mathrm{approx}^\mathrm{fd} = \frac{f^{(2)}h}{2}, \qquad \frac{\epsilon_\mathrm{m}}{h} \approx \epsilon_\mathrm{approx}^\mathrm{cd} = \frac{f^{(3)}h^2}{24} \tag{476}$$

$$\Rightarrow h_\mathrm{fd}^2 = \frac{2\epsilon_\mathrm{m}}{f^{(2)}} \qquad \Rightarrow h_\mathrm{cd}^3 = \frac{24\epsilon_\mathrm{m}}{f^{(3)}} \tag{477}$$

for $f' \approx f^{(2)} \approx f^{(3)} \simeq 1$ (e.g., $\exp(x)$, $\cos(x)$) and double precision ($\epsilon_\mathrm{m} \approx 10^{-15}$):

$$h_\mathrm{fd} \approx 4 \times 10^{-8} \qquad \& \qquad h_\mathrm{cd} \approx 3 \times 10^{-5} \tag{478}$$

$$\Rightarrow \epsilon_\mathrm{fd} \simeq \frac{\epsilon_\mathrm{m}}{h_\mathrm{cd}} \simeq 3 \times 10^{-8}, \qquad \Rightarrow \epsilon_\mathrm{cd} \simeq \frac{\epsilon_\mathrm{m}}{h_\mathrm{cd}} \simeq 3 \times 10^{-11} \tag{479}$$

$\rightarrow$ can choose 1000$\times$ larger h for *central difference* $\rightarrow$ error is 1000$\times$ smaller for *central difference*

Numerical differentiation IX

Second derivative

starting from first derivative with *central difference* method

$$f'(x) \simeq \frac{f(x+h/2) - f(x-h/2)}{h} \tag{480}$$

the 2nd derivative $f^{(2)}(x)$ is central difference from 1st derivative

$$f^{(2)}(x) \simeq \frac{f'(x+h/2) - f'(x-h/2)}{h}, \tag{481}$$

$$\simeq \frac{[f(x+h) - f(x)] - [f(x) - f(x-h)]}{h^2} \tag{482}$$

$$\simeq \frac{f(x+h) + f(x-h) - 2f(x)}{h^2} \tag{483}$$

$\rightarrow$ Eq. (482) better in terms of subtractive cancelation

Random numbers and Monte-Carlo methods

Motivation I

Many physical process can be described in two pictures:

- microscopic, individual, e.g., particle-particle interactions are considered realization usually with help of $\rightarrow$ Monte-Carlo (MC) methods
- macroscopic, only the *effective coaction* is described $\rightarrow$ usually analytical equations

Example: Thermodynamics

microscopic: motion of particles, e.g., $\overline{v^2} = \frac{1}{N}\sum_{i=1}^{N} v_i^2$
effective theory: thermodynamics (via statistical physics) averages particle quantities, e.g., $\frac{1}{2}m\overline{v^2} = \frac{3}{2}k_\mathrm{B}T$, so $\overline{v^2} \rightarrow T$

Monte-Carlo simulation

Computer algorithm based on a large number of repeated <u>random</u> experiments to obtain a representative sample of the possible configurations.

Motivation II

Example: Radiative transfer

- microscopic: interaction of photons with atoms/ions/molecules

  $\rightarrow$ `MOCASSIN` for Monte-Carlo simulation of photon propagation in gaseous nebulae
  $\rightarrow$ `MCRH` (Noebauer 2015) MC radiation hydrodynamics for stellar winds

  advantage: arbitrary geometries (e.g., torus) and density distributions (inhomogeneities) and processes; good for scattering (special non-LTE case)
  disadvantage: feedback on matter (often iteratively calculated) hard to implement because of MC noise

- macroscopic: radiative transfer equation (RTE) = effective theory, i.e. light (intensity I_ν) instead of single photons

  $\rightarrow$ `Cloudy` spectral synthesis code for astrophysical plasmas
  $\rightarrow$ `PoWR` for emergent spectra of stellar atmospheres

  advantage: feedback on matter (non-LTE) via iteration (boundary conditions, e.g., conservation of energy) $\rightarrow$ non-LTE population numbers
  disadvantage: hard to program (numerical stability); *consistent* only for some geometries, usually 1d, e.g., spherical symmetry

Random numbers I

For MC methods we need *good* and *many* random numbers. Usual base are uniformly distributed random numbers (= same probability for every event).
Humans are not a good source for random numbers:

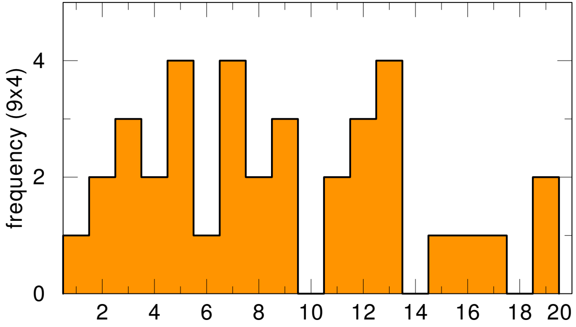


Figure: random numbers, created by colleagues $\rightarrow$ not uniformly distributed, too few

$\rightarrow$ direct, severe consequence: don't make up your own passwords!

Random numbers II

Other sources: rolling dices, tossing coins $\rightarrow$ low rate

most programming languages have a builtin random function, which gives pseudo-random numbers, e.g., in C/C++ integers (!) from `[0,RAND_MAX]`

```
#include <cstdlib>
  ...
 int i = rand () ;
```

- output of next random number of a sequence
- restart by `srand(i) ;`

To get uniformly distributed random numbers $\in [0;1]$:

```
 r = rand()/double(RAND_MAX) ;
```

Random numbers III

Definition

A result (a state) is random if it was not predictable.

Quality tests for random numbers:

- *uniform distribution:* random numbers should be fair
- *sequential tests:* for $n$tuple repetitions (usually only for $n = 2$ und $n = 3$)
- *run tests:* for monotonically increasing/decreasing sequences, and duration of stay in a certain interval
- and more . . .

$\rightarrow$ there is no sufficient criterion for randomness tests

Non-uniform distributions

Non-uniform random numbers I

- random number generators give *uniform* (pseudo) random numbers $\in [0, \mathtt{RAND\_MAX}]$ $\rightarrow r \in [0, 1]$ (from now on)
- we often need different distributions, e.g., normal (Gaussian) distributions or uniform distributions on an interval $x \in [a, b]$
- i.e., we need a transformation that maps r to x, so

Inverse transformation

$$x = P^{-1}(r) \tag{484}$$

Non-uniform random numbers II

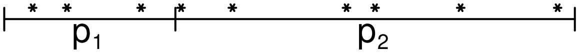
First, for the case of discrete numbers

- e.g., two events (1,2) with probabilities p_1 and p_2, such that

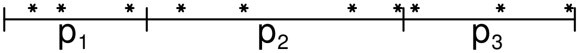
$$p_1 + p_2 = 1 \tag{485}$$

  How can we choose with help of r?

- obvious choice: for $r < p_1$ event 1, otherwise event 2



- for the case of 3 possible events with p_1, p_2, p_3: $r < p_1 \rightarrow$ event 1, $p_1 < r < p_1 + p_2 \rightarrow$ event 2, else event 3



- in general for n events, event i is selected if for r:

$$\sum_{j=0}^{i-1} p_j \leq r \leq \sum_{j=0}^{i} p_j \quad \text{where } p_0 \equiv 0 \tag{486}$$

Non-uniform random numbers III

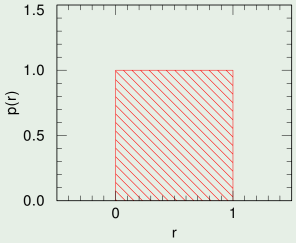
For continuous distributions:

- need the *probability density function* $p(x)$, where $p(x) \cdot dx$ is probability that x is in the interval $[x, x + dx]$
- moreover, $p(x)$ is normalized:

$$\int_{-\infty}^{+\infty} dx\, p(x) = 1 \tag{487}$$

Example: uniform distribution

$$p_{\mathrm{u}}(r) = \begin{cases} 1, & \text{if } 0 \leq r \leq 1 \\ 0, & \text{else} \end{cases} \tag{488}$$



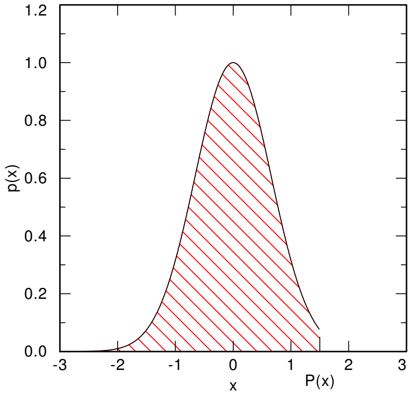
Non-uniform random numbers IV

- for the continuous case (continuum limit $i \to x$) in the Eqn. (486)

$$\sum_{j=0}^{i-1} p_j \le r \le \sum_{j=0}^{i} p_j \quad \text{where } p_0 \equiv 0$$

both sums are equal and become the integral:

$$P(x) = \int_{-\infty}^{x} p(x')\,dx' = r \qquad (489)$$



Non-uniform random numbers V

This corresponds to the cumulated distribution function

$$P(x) = \int_{-\infty}^{x} p(x')\,dx' \tag{490}$$

i.e. the probability to get a random number smaller or equal x. Geometrically: fraction of the area left of (smaller than) x. We state:

$$\begin{aligned} P(x) &= r & (491)\\ \Rightarrow x &= P^{-1}(r) & (492) \end{aligned}$$

i.e. exactly as r also $P(x)$ is uniformly distributed.
Therefore, the probability to find $P(x)$ in the interval $[P(x), P(x) + dP(x)]$ is $dP(x) = dr$ (Eq. 491).

Non-uniform random numbers VI

The relation between $dP(x)$ and dx is obtained by derivating Eq. (490) $\rightarrow$ Fundamental theorem of calculus:

$$\frac{dP(x)}{dx} = p(x) \tag{493}$$

for $0 \leq r \leq 1$ it is also:

$$dP(x) = p(x)\,dx = p_{\mathrm{u}}(r)\,dr \tag{494}$$

I.e., because of Eq. (491) $P(x) = r \rightarrow x$ is distributed according to $p(x)$

Non-uniform random numbers VII

To obtain such $p(x)$ distributed random numbers, one has to solve Eq. (492) $x = P^{-1}(r)$

Inverse transformation

1. Insert the required distribution $p(x)$ into:

$$r = P(x) = \int_{-\infty}^{x} p(x')\,dx' \tag{495}$$

2. solve for x, i.e. find

$$P^{-1}(r) = x \tag{496}$$

Not for all $p(x)$ are the corresponding conditions fulfilled (solvable integral and invertibility)

Non-uniform random numbers VIII

Example for inverse transformation

Let

$$p(x) = \begin{cases} a\,e^{-ax}, & \text{if } 0 \leq x \leq \infty \\ 0, & x < 0 \end{cases} \tag{497}$$

$$P(x) = \int_0^x a\,e^{-ax'}\,dx' = 1 - e^{-ax} = r \tag{498}$$

$$\Rightarrow x = -a^{-1} \ln(1-r) \tag{499}$$

and $(1-r)$ is exactly distributed as r, so:

$$x = P^{-1}(r) = -a^{-1} \ln r \tag{500}$$

The evaluation of ln on a computer is relatively time consuming
$\rightarrow$ inverse transformation not always the best method

Probability distributions in Physics I

Probability distributions are fundamental in, e.g., statistical mechanics and non-relativistic quantum mechanics:

- Boltzmann distribution: $p_i \propto \exp\left(-\frac{E_i}{k_\mathrm{B} T}\right)$ for some state i

  usually: discrete states (statistical mechanics), hence

$$p_i = \frac{N_i}{N} = \frac{\exp\left(-\frac{E_i}{k_\mathrm{B} T}\right)}{\sum_{j=1}^{m} \exp\left(-\frac{E_j}{k_\mathrm{B} T}\right)} \tag{501}$$

  for N_i particles in state i and a total number of N particles with m states
  but might be also continuous, e.g., barometric formula for molecule of mass m, height h above ground

$$\rho(h) \propto \exp\left(-\frac{m\,g\,h}{k_\mathrm{B}\,T}\right) \tag{502}$$

  $\rightarrow$ computer generated samples via Markov Chain Monte Carlo (MCMC), in particular
  $\rightarrow$ Metropolis algorithm (see below)

Probability distributions in Physics II

- Maxwell-Boltzmann distribution: continuous distribution of particle velocity in one direction (e.g., radial sightline) with $v_{\mathrm{th}} = \sqrt{\frac{2k_{\mathrm{B}}T}{m}}$

$$p(v_x)\,dv_x = \left(\frac{m}{2\pi k_{\mathrm{B}}T}\right)^{1/2} \exp\left(-\frac{mv_x^2}{2k_{\mathrm{B}}T}\right)\,dv_x = \frac{1}{v_{\mathrm{th}}\sqrt{\pi}} \exp\left(-\frac{v_x^2}{v_{\mathrm{th}}^2}\right) \tag{503}$$

Application: thermal Doppler broadening of spectral lines where $\Delta\nu_{\mathrm{th}} = \nu_0 \cdot v_{\mathrm{th}}/c$

Mean value $\langle v_x^2 \rangle = 2\int_0^\infty v_x^2\,p(v_x)dv_x = \frac{1}{2}v_{\mathrm{th}}^2 = \frac{k_{\mathrm{B}}T}{m} = v_{\mathrm{s}}^2 \rightarrow$ isothermal sound speed

$\rightarrow$ example for a "moment" of a distribution

For 3D, absolute value, speed v: $d^3v = dv_x\,dv_y\,dv_z = v^2\,dv\,d\Omega$ integration $\rightarrow 4\pi v^2 dv$ and $v^2 = v_x^2 + v_y^2 + v_z^2$:

$$p(v)\,dv = 4\pi \left(\frac{m}{2\pi k_{\mathrm{B}}T}\right)^{3/2} v^2 \exp\left(-\frac{mv^2}{2k_{\mathrm{B}}T}\right)\,dv \tag{504}$$

Hence, mean $\langle v^2 \rangle = \int_0^\infty v^2\,p(v)dv = \frac{3k_{\mathrm{B}}T}{m}$

$\rightarrow$ compare definition of T as measure of mean kinetic energy

Probability distributions in Physics III

- in non-relativistic QM (1d):
  the squared modulus of the wave function $|\psi(x,t)|^2$ gives probability of particle in "volume" dx around x at time $t \rightarrow p(x,t)dx = |\psi(x,t)|^2 dx$

  Physical quantities (observables) have corresponding operators, e.g., momentum $p_{\text{op}} \rightarrow -\imath\hbar\partial/\partial x$; expectation or average value of observable A:

$$\langle A \rangle = \int \psi^*(x,t)\, A_{\text{op}}\, \psi(x,t) dx \tag{505}$$

  And ψ evolves according to Schrödinger equation

$$\imath\hbar\frac{\partial\psi(x,t)}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\psi(x,t)}{\partial x^2} + V(x,t)\psi(x,t) \tag{506}$$

  $\rightarrow$ because of similarity to diffusion equation (with imaginary time), solutions to Eq. (506) can be found by random walk (see below)

Probability distributions in Physics IV

- the specific intensity $I_\nu(\vec{x}, t, \vec{n}, \nu) = \frac{d\,E}{d\vec{A}\cdot\vec{n}\,d\Omega\,d\nu\,dt}$ is a 7-dim distribution function on a 4-dim spacetime manifold $(\vec{x}, t)$, describing unpolarized radiation. Note: $I_\nu = n_{\text{phot}} ch\nu \geq 0$ (where n_{phot} is photons / volume / solid angle / frequency interval)

  Moments of the specific intensity (radiation field) = integrals over all directions, in 1d (plane parallel, spherical symmetry) over $\mu = \cos\theta$, n-th moment: $\frac{1}{2}\int_{-1}^{+1} \mu^n\, I_\nu(\mu) d\mu$

n	symbol	integral	type
0.	J_ν	$= \frac{1}{2}\int_{-1}^{+1} I_\nu(\mu) d\mu$	mean intensity, energy density $E_\nu = \frac{4\pi}{c} J_\nu$, $J_\nu \geq 0$
1.	H_ν	$= \frac{1}{2}\int_{-1}^{+1} \mu\, I_\nu(\mu) d\mu$	(Eddington-) flux, can be neg. (e.g. “inward” flux)
2.	K_ν	$= \frac{1}{2}\int_{-1}^{+1} \mu^2\, I_\nu(\mu) d\mu$	radiation pressure $K_\nu = \frac{c}{4\pi} P_\nu$
3.	N_ν	$= \frac{1}{2}\int_{-1}^{+1} \mu^3\, I_\nu(\mu) d\mu$	flux-like, i.e., can be negative

$\rightarrow$ usually: MC simulations of radiation field require large number of runs for individual photons to recover macroscopic quantities I, J, etc. correctly

Non-uniform distributions II

Box-Muller method I

non-uniform distribution:

- with help of the *inversion* method we can get non-uniform random numbers from uniform random numbers $\rightarrow$ condition: $P(x)$ invertable
- for the Gaussian normal distribution:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{507}$$

  $P(x)$ is not analytical representable (error function)

- idea: 2d-transformation where:

$$p(x,y)\,dx\,dy = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}\,dx\,dy \tag{508}$$

- change to polar coordinates:

$$r = \sqrt{x^2+y^2} \quad \theta = \tan^{-1}\frac{y}{x} \tag{509}$$

Box-Muller method II

- let $\rho = r^2/2$ and set $\sigma = 1$:

$$p(x, y)dx\,dy = p(\rho, \theta)\,d\rho\,d\theta = \frac{1}{2\pi}\,e^{-\rho}\,d\rho\,d\theta \tag{510}$$

- now generate random numbers ρ according to exponential distribution, so $\rho = -\ln u$ (u standard uniform distributed) and θ uniform distributed on $[0, 2\pi)$, then

$$x = \sqrt{-2\ln u}\cos\theta \quad \text{und} \quad y = \sqrt{-2\ln u}\sin\theta \tag{511}$$

  are each according to Eq. (507) with $\sigma = 1$ and $\mu = 0$ distributed because of

$$re^{\imath z} = \sqrt{-\ln u}e^{\imath 2\pi\theta} = \sqrt{-2\ln u}\,[\cos(2\pi\theta) + \imath\sin(2\pi\theta)] \tag{512}$$

Alternative: Rejection method (see below)

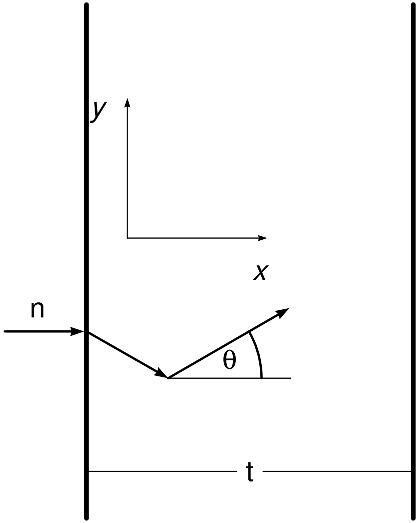
Example: Neutron transport

Neutron transport I

Application for non-uniform random numbers!

Transport of neutrons through matter – one of the first MC applications!

- consider a plate of thickness t
- plate is infinite in z and y direction, x-axis perpendicular to the plate
- at each point within the plate: probability p_c, that neutron gets absorbed (captured) and probability p_s that neutron is scattered
- after each scattering: find scattering angle θ in xy plane
- as motion in y, z direction irrelevant: azimuthal angle ϕ irrelevant



Neutron transport II

Determine scattering angle & scattering path length

1. Isotropic scattering:

$$p(\theta,\phi)\,d\theta\,d\phi = d\Omega/4\pi \tag{513}$$

$$\text{because of } d\Omega = \sin\theta\,d\theta\,d\phi \;: \tag{514}$$

$$p(\theta,\phi) = \frac{\sin\theta}{4\pi} \tag{515}$$

obtain $p(\theta)$ and $p(\phi)$ by integration over the complementary angle:

$$p(\theta) = \int_0^{2\pi} p(\theta,\phi)\,d\phi = 2\pi\,\frac{\sin\theta}{4\pi} = \frac{1}{2}\sin\theta \tag{516}$$

$$p(\phi) = \int_0^{\pi} p(\theta,\phi)\,d\theta = \frac{1}{4\pi}(-\cos\pi + \cos 0) = \frac{1}{2\pi} \tag{517}$$

I.e. $p(\theta,\phi) = p(\theta)p(\phi) \rightarrow$ independent variables

Neutron transport III

If random variable ϕ is wanted ($p(\phi) \equiv const.$):

$$\phi = 2\pi r \tag{518}$$

To get random θ according to Eq. (516) $\rightarrow$ inversion method:

$$\begin{aligned} r &= P(\theta) = \int_0^\theta \frac{1}{2} \sin x \, dx = -\frac{1}{2}(\cos\theta - \cos 0) && (519) \\ \cos\theta &= 1 - 2r && (520) \end{aligned}$$

I.e. $\cos\theta$ is uniformly distributed on $[-1; 1]$ and ϕ on $[0; 2\pi]$. Solving for θ possible, but unnecessary, as only $\cos\theta$ required for x component of the path $\rightarrow$
2. scattering path length:

$$x = \ell \cos\theta \tag{521}$$

where ℓ from $p(\ell) \sim e^{-\ell/\lambda}$ (see example for inversion method):

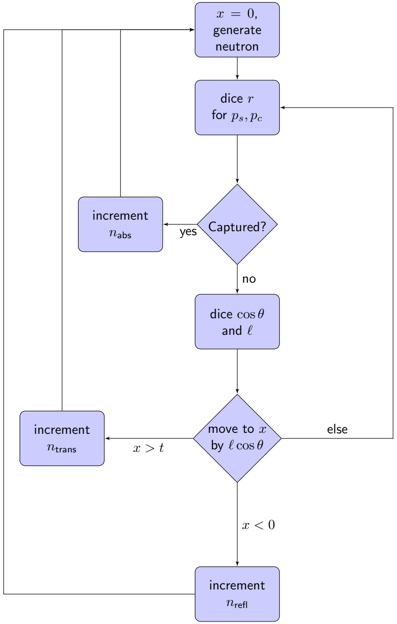
$$\ell = -\lambda \ln r \tag{522}$$

$\lambda \rightarrow$ mean free path (e.g., $\lambda = (\sigma n)^{-1}$)

Neutron transport IV

Algorithm, start at $x = 0$:

1. determine, if neutron is scattered or captured. If captured: increment number of absorbed neutrons, go to 5 step
2. scattering: "dice" $\cos\theta$ and ℓ, move to x position by $\ell\cos\theta$
3. if $x < 0$: increment number of reflected neutrons, if $x > t$: increment number of transmitted neutrones;
   go to 5
4. repeat step 1 - 3 until final result is achieved for all neutrons
5. repeat step 1 - 4 with more incident neutrons

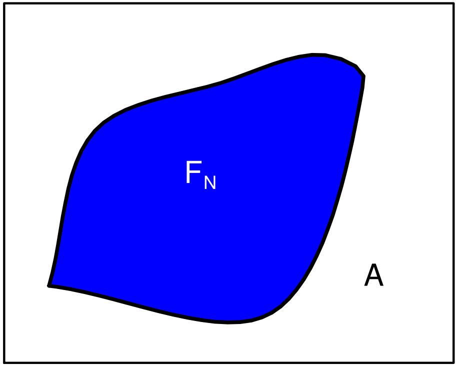


$x = 0$, generate neutron
dice r for p_s, p_c
increment n_{abs}
yes
Captured?
no
dice $\cos\theta$ and ℓ
increment n_{trans}
$x > t$
move to x by $\ell\cos\theta$
else
$x < 0$
increment n_{refl}

Monte-Carlo integration

MC integration I

Idea: Can the area of a pool (irregular!) be measured by throwing stones?



- pool with area F_n in a field with *known(!)* area A

MC integration II

- fraction of the *randomly* thrown stones which fall into the pool:

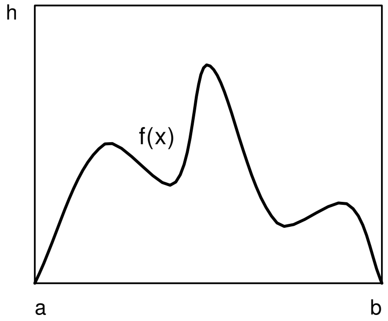
$$\frac{n_\mathrm{p}}{n} = \frac{F_n}{A} \tag{523}$$

  (n stones, n_p hit pool)

- determine F_n with help of the *hit-or-miss method*:

$$F_n = A\,\frac{n_\mathrm{p}}{n} \tag{524}$$

MC integration III



- choose rectangle of height h, width $(b-a)$, area $A = h \cdot (b-a)$, such that $f(x)$ within the rectangle
- generate n pairs of random variables x_i, y_i with $a \le x_i \le b$ and $0 \le y_i \le h$
- fraction n_t of the points, which fulfill $y_i \le f(x_i)$ gives estimate for area under $f(x)$ (integral)

MC integration IV

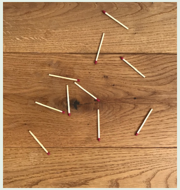
Excursus: Buffon's needle problem – determine π by throwing matches

Buffon's question (1773): What is the probability that a needle or a match of length ℓ will lie across a line between two strips on a floor made of parallel strips, each of same width t?
$\rightarrow x$ is distance from center of the needle to closest line, θ angle between needle and lines ($\theta < \frac{\pi}{2}$), hence the *uniform* probability density functions are

$$p(x) = \begin{cases} \frac{2}{t} & : \quad 0 \le x \le \frac{t}{2} \\ 0 & : \quad \text{elsewhere} \end{cases} \qquad p(\theta) = \begin{cases} \frac{2}{\pi} & : \quad 0 \le \theta \le \frac{\pi}{2} \\ 0 & : \quad \text{elsewhere} \end{cases}$$

x, θ independent $\rightarrow p(x,\theta) = \frac{4}{t\pi}$ with condition $x \le \frac{\ell}{2}\sin\theta$. If $\ell \le t$ (short needle):

$$P(\text{hit}) = \int_{\theta=0}^{\frac{\pi}{2}} \int_{x=0}^{\frac{\ell}{2}\sin\theta} \frac{4}{t\pi} dx d\theta = \frac{2\ell}{t\pi}$$



$\rightarrow$ count hits and misses and then:

$$\pi = \frac{2\ell}{t}\frac{1}{P(\text{hit})} = \frac{2\ell}{t}\frac{n_\text{hit} + n_\text{miss}}{n_\text{hit}}$$

MC integration V

Sample-mean method

- the integral

$$F(x) = \int_a^b f(x)\,dx \tag{525}$$

  is given in the interval $[a, b]$ by the mean $\langle f(x)\rangle$ (mean value theorem for integration)
- choose arbitrary x_i (instead of regular intervals) and calculate

$$F_n = (b-a)\langle f(x)\rangle = (b-a)\,\frac{1}{n}\sum_{i=1}^{n} f(x_i) \tag{526}$$

  where x_i are uniform random numbers in $[a, b]$

$$\left(\text{cf. rectangle rule} \quad F_n = \sum_{i=1}^{n} f(x_i)\Delta x \quad \text{with fixed } x_i, \Delta x = \frac{b-a}{n}\right) \tag{527}$$

Importance sampling I

Idea: improve MC integration by a better sampling $\rightarrow$ introduce a positive function $p(x)$ with

$$\int_a^b p(x)dx = 1 \tag{528}$$

and rewrite integral $\int_a^b f(x)dx$ as

$$F = \int_a^b \left[\frac{f(x)}{p(x)}\right] p(x)dx \tag{529}$$

this integral can be evaluated by *sampling according to* $p(x)$:

$$F_n = \frac{1}{n}\sum_{i=1}^{n} \frac{f(x)}{p(x)} \tag{530}$$

Note that for the *uniform case* $p(x) = 1/(b-a)$ $\rightarrow$ the *sample mean method* is recovered.
Now, try to minimize variance σ^2 of integrand $\frac{f(x)}{p(x)}$ by choosing $p(x) \approx f(x)$, especially for large $f(x)$

Importance sampling II

$\rightarrow$ slowly varying integrand $f(x)/p(x)$
$\rightarrow$ smaller variance σ^2

Example: Normal distribution

Evaluate integral $F = \int_a^b f(x)dx = \int_0^1 e^{-x^2}dx$ (error function) $\rightarrow F_n = \frac{1}{n}\sum_{i=1}^{n} \frac{e^{-x^2}}{p(x)}$

	$p(x) = 1$	$p(x) = Ae^{-x}$ [†]
x	$(b-a) * r + a$	$-\log(e^{-a} - \frac{r}{A})$
n	4×10^5	8×10^3
σ	0.0404	0.0031
$\sigma/\sqrt{n}$	6×10^{-5}	3×10^{-5}
total CPU time[††]	19 ms	0.8 ms
CPU time / trial	50 ns	100 ns

[†] A from normalization $A = (\exp(-a) - \exp(-b))^{-1}$, [††]CPU time on a Intel Core i7-4771 3.5 GHz

$\rightarrow$ the extra time needed per trial for getting x from uniform r is usually overcompensated by the smaller number of necessary trials for same $\sigma/\sqrt{n}$

Metropolis algorithm I

Similar: Metropolis algorithm (Metropolis, Rosenbluth, Rosenbluth, Teller & Teller 1953) useful for averages of the form

$$\langle f \rangle = \frac{\int p(x) f(x) dx}{\int p(x) dx} \quad \text{e.g.} \quad \langle f \rangle = \frac{\int e^{-\frac{E(x)}{k_B T}} f(x) dx}{\int e^{-\frac{E(x)}{k_B T}} dx}, \tag{531}$$

The Metropolis algorithm uses *random walk* (see below) of points $\{x_i\}$ (1D or higher) with asymptotic probability distribution approaching $p(x)$ for $n \gg 1$. Random walk from *transition probability* $T(x_i \to x_j)$, such that

$$p(x_i) T(x_i \to x_j) = p(x_j) T(x_j \to x_i) \qquad \text{(detailed balance)} \tag{532}$$

$$\text{e.g., choose } T(x_i \to x_j) = \min\left[1, \frac{p(x_j)}{p(x_i)}\right] \quad \text{(where, e.g., } p_j/p_i = \exp\left(-\frac{E_j - E_i}{k_B T}\right)\text{)} \tag{533}$$

Metropolis algorithm II

Metropolis algorithm

1. choose trial position $x_{\text{trial}} = x_i + \delta_i$ with random $\delta_i \in [-\delta, +\delta]$
2. calculate $w = p(x_{\text{trial}})/p(x_i)$ (might be: $w = \exp\left(-\frac{E(x_{\text{trial}})-E(x_i)}{k_B T}\right)$)
3. if $w \geq 1$, accept and $x_{i+1} = x_{\text{trial}}$ ($\rightarrow \Delta E \leq 0$)
4. if $w < 1$ ($\rightarrow \Delta E > 0$), generate random $r \in [0;1]$
5. if $r \leq w$, accept and $x_{i+1} = x_{\text{trial}}$ (and compute desired quantities, e.g. $f(x_{i+1})$)
6. if not, $x_{i+1} = x_i$

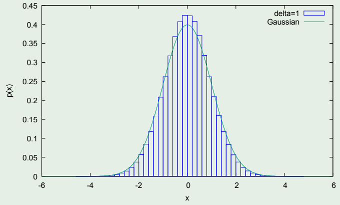
(finally: $\langle f \rangle = \frac{1}{n} \sum_{i=1}^{n} f(x_i)$)
problem: optimum choice of δ;

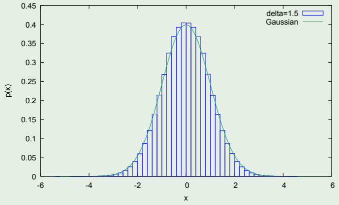
if too large, only small number of accepted trials $\rightarrow$ inefficient sampling
if too small, only slow sampling of $p(x)$.
Hence, rule of thumb: choose δ for which $\frac{1}{3} \ldots \frac{1}{2}$ trials accepted

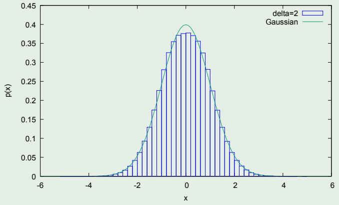
also: choose x_0 for which $p(x_0)$ is largest $\rightarrow$ faster approach of $\{x_i\}$ to $p(x)$

Metropolis algorithm III

Metropolis algorithm for Gaussian standard distribution







$\delta = 1.$
$f_{\text{accept}} = 0.72$
$\langle x \rangle = 0.0007533$
$\langle x^2 \rangle = 0.90306$

$\delta = 1.5$
$f_{\text{accept}} = 0.64$
$\langle x \rangle = -0.0000153$
$\langle x^2 \rangle = 0.935376$

$\delta = 2.$
$f_{\text{accept}} = 0.56$
$\langle x \rangle = 0.000200396$
$\langle x^2 \rangle = 0.988051$

$\langle x \rangle$ and $\langle x^2 \rangle$ computed from

```
xmean = xmean + xtrial ; xxmean = xxmean + xtrial * xtrial ;
...
xmean = xmean / naccept ; xxmean = xxmean / naccept ;
```

Metropolis algorithm IV

Typical applications for Metropolis algorithm: computation of integrals with weight functions $p(x) \sim e^{-x}$, e.g.,

$$\langle x \rangle = \frac{\int_0^\infty x e^{-x} dx}{\int_0^\infty e^{-x} dx} \tag{534}$$

$$\langle A \rangle = \frac{\int A(\vec{X}) e^{-U(\vec{X})/k_\mathrm{B} T} d\vec{X}}{\int e^{-U(\vec{X})/k_\mathrm{B} T} d\vec{X}} \tag{535}$$

where the latter is the average of a physical quantity A in a liquid system with good contact to a thermal bath, fixed number of particles (with $\vec{X} = (\vec{x}_1, \vec{x}_2, \ldots)$ of all particles) and volume $\rightarrow$ canonical ensemble, e.g.,

$$\left\langle \frac{m v_{ik}^2}{2} \right\rangle = \frac{1}{2} k_\mathrm{B} T \tag{536}$$

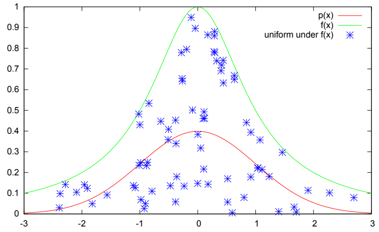
Rejection sampling (acceptance-rejection method)

Rejection sampling (acceptance-rejection method) I

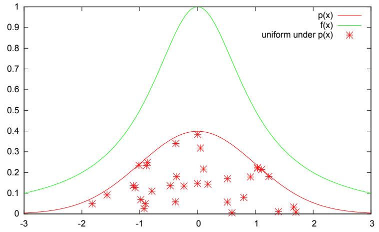
Problem: get random x for any $p(x)$, also if $P(r)^{-1}$ not (easily) computable

Idea:

- area under $p(x)$ in $[x, x + dx]$ is probability of getting x in that range
- if we can choose a random point in *two dimensions* with uniform probability in the area under $p(x)$, then x component of that *point* is distributed according to $p(x)$
- so, on same graph draw an $f(x)$ with $f(x) > p(x) \quad \forall x$
- if we can uniformly distribute points in the area under curve $f(x)$, then all points (x, y) with $y < p(x)$ are uniform under $p(x)$



$\Longrightarrow$



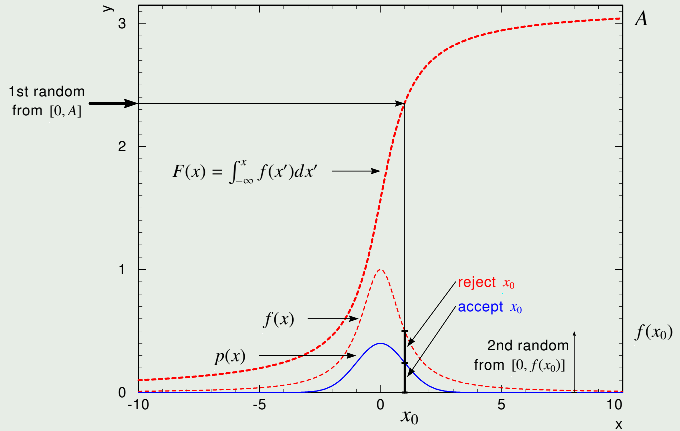
Rejection sampling (acceptance-rejection method) II

Creation of arbitrary probability distributions with help of rejection sampling (especially for compact intervals $[a, b]$):

- let $p(x)$ be the required distribution in $[a, b]$
- choose a $f(x)$ such that $p(x) < f(x)$ in $[a, b]$, e.g., $f(x) = c \cdot \max(p(x)) = \text{const.}$ where $c > 1$
- it is $A := \int_a^b f(x)dx$, i.e. A(x) must exist and must be invertible: $A(x) \rightarrow x(A)$
- generate *uniform* random number in $[0, A]$ and get the corresponding $x(A)$
- generate 2nd *uniform* random number y in $[0, f(x)]$, so x, y are uniformly distributed on A (area under $f(x)$)
- *accept* this point if $y < p(x)$, otherwise *reject* it

Rejection sampling (acceptance-rejection method) III

Example: normal distribution $p(x)$ sampled by $f(x) = (x^2+1)^{-1}$



$\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ (blue solid line) sampled with help of the function $\frac{1}{x^2+1}$ (red dashed) whose integral is arctan(x) (thick dashed red) and hence $F(x)^{-1} = \tan(x)$, see source code on page 441

Rejection sampling (acceptance-rejection method) IV

Requirements:

- $p(x)$ must be computable for every x in the intervall
- $f(x) > p(x) \rightarrow$ always possible, as $\int_{-\infty}^{+\infty} p(x)dx = 1$ (i.e. $A > 1$)
- to get x_0 for a chosen value in $[0, A]$ requires usually: $\int f(x)dx = F$ is analytically invertible, i.e. $F(x)^{-1}$ exists

$\rightarrow$ this is easy for a compact interval $[a, b]$, e.g., choose a $c > 1$ such
$F(x) = c \cdot \max(p(x)) \cdot (x - a) = k(x - a)$
$\rightarrow x = F/k - a$ for randomly chosen F in $[0, A]$, where $A = k \cdot (b - a)$

Rejection sampling (acceptance-rejection method) V

Example: acceptance-rejection for normal distribution (see p. 439)

```
double p(double x){ return exp(-0.5*x*x)/sqrt(2.*M_PI); }
double f(double x){ return 1./(x*x+1.); }
double  inv_int_f(double ax){ return tan(ax - M_PI /2.); }
 ...
for (int i = 0; i < nmax; ++i){
  // get random value between 0 and A:
  ax = A * double(rand())/double(RAND_MAX);
  // obtain the corresponding x value:
  x = inv_int_f(ax);
  // get random y value in interval [0,f(x)]:
  y = f(x) * double(rand())/double(RAND_MAX);
  // test for y =< p(x) for acceptance:
  if ( y <= p(x) ) { cout << x << endl ;}
}
```

Rejection sampling (acceptance-rejection method) VI

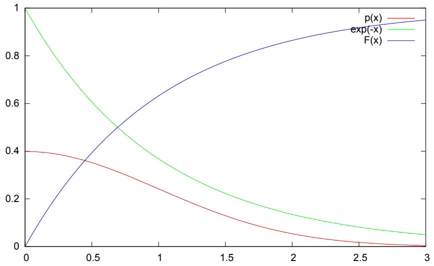
In our example:

- it is $p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ the standard normal distribution; normal distributions with $\sigma \neq 1, \mu \neq 0$ can be obtained by transformation
- the comparison function $f(x) = \frac{1}{x^2+1}$ is always $f(x) > p(x)$, moreover:
  - $F(x) = \int_{-\infty}^{x} f(x')dx' = \arctan(x) - \arctan(-\infty) = \arctan(x) - \left(-\frac{\pi}{2}\right)$
    $\rightarrow F(x) = \arctan(x) + \frac{\pi}{2}$
  - the total area A under $f(x)$ is $\int_{-\infty}^{+\infty} f(x')dx' = \arctan(+\infty) - \arctan(-\infty) = \pi$
  - the inverse $F(x)^{-1}$, which returns x for a given value $F \in [0, A]$ simply $x = \tan\left(F - \frac{\pi}{2}\right)$
  - efficiency of the acceptance is $N_{\text{accepted}}/\texttt{NMAX} = \int p(x)/\int f(x) = 1/\pi \approx 0.32$, i.e. efficiency can be increased by choosing $f(x) = \frac{1}{2}\frac{1}{x^2+1}$, then $x = \tan\left(2F - \frac{\pi}{2}\right) \rightarrow 63\%$ acceptance

Rejection sampling (acceptance-rejection method) VII

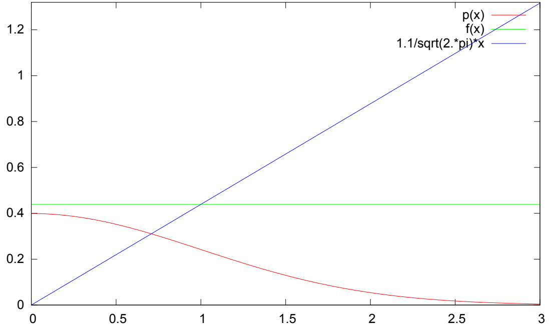
Alternative choice I: $f(x) = \exp(-x)$ only for $x \geq 0$, then

- the integral $F(x)$ is $\int_0^x = -\exp(-x) + 1$
- the total area $\int_0^\infty \exp(-x)dx = 1 > 0.5 = \int_0^\infty p(x)$
- the inverse is $x = -\log(-x + 1)$
- to obtain also negative $x \rightarrow$ add random sign $\pm$



Rejection sampling (acceptance-rejection method) VIII

Alternative choice II: $f(x) = 1.1 \cdot \max(p(x))$ in the compact interval $[0,3]$, then



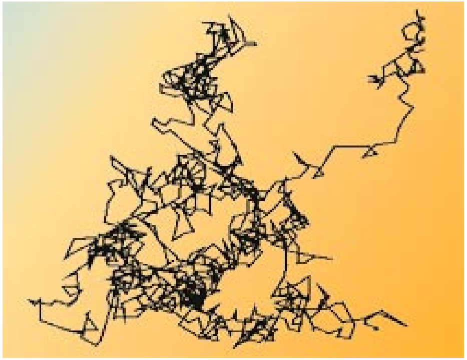
- it is $\max(p(x)) = \frac{1}{\sqrt{2\pi}}$ in $[0,3]$
  $\rightarrow f(x) = \frac{1.1}{\sqrt{2\pi}}$ in $[0,3]$
- hence $F(x)^{-1}$ is $x = \frac{F\sqrt{2\pi}}{1.1} - 0.$
- the total area A is $\frac{1.1}{\sqrt{2\pi}} \cdot (3-0)$

$\rightarrow$ clear: this choice (const. function) works only for compact intervals, otherwise A is infinite and $F(x)^{-1}$ does not exist

Random walk

Random walk I

Idea: Brownian motion, e.g., dust in water (lab course: determination of diffusion coefficient $D = \frac{\langle x^2 \rangle}{2t}$, with Fick's laws of diffusion: $j = -D\partial_x c$ and $\dot{c} = D\partial_x^2 c$)



frequent collisions between dust particles and water molecules
$\rightarrow$ frequent change of direction
$\rightarrow$ trajectory not predictable even for few collisions
$\rightarrow$ motion of dust particle into any direction with same probability

$\rightarrow$ Random walk

like "drunken sailor": N steps of equal length in arbitrary direction will lead to which distance from start point?

Random walk II

In one dimension:

- let's start at $x = 0$, each step with length ℓ
- for each step: probability p for step to the right and $q = 1 - p$ to the left (independent from previous step)
- displacement after N steps

$$x(N) = \sum_{i=1}^{N} s_i \quad \text{where } s_i = \pm\ell \quad \rightarrow x^2(N) = \left(\sum_{i=1}^{N} s_i\right)^2 \tag{537}$$

- for $p = q = 1/2 \rightarrow$ coin flipping
- for large N: $\langle x(N)\rangle = 0$ expected
- but for $\langle x^2(N)\rangle$? $\rightarrow$ rewrite Eq. (537)

$$x^2(N) = \sum_{i=1}^{N} s_i^2 + \sum_{i\neq j=1}^{N} s_i s_j \tag{538}$$

where (for $i \neq j$) $s_i s_j = \pm\ell^2$ with same probability, so: $\sum_{i\neq j}^{N} s_i s_j = 0$

Random walk III

- because of $s_i^2 = \ell^2 \to \sum_{i=1}^N s_i^2 = N\ell^2$:

$$\langle x^2(N)\rangle = \ell^2 N \tag{539}$$

- especially for constant time intervals of the random walk

$$\langle x^2(t)\rangle = \frac{\ell^2}{\Delta t} N\Delta t \quad \left(= \frac{\ell^2}{\Delta t} t\right) \tag{540}$$

- generally: if $p \neq 1/2$ and p for $+\ell$

$$\langle x(N)\rangle = (p-q)\ell N \tag{541}$$

$\rightarrow$ linear dependence on N

Random walk IV

Example: Diffusion of photons in the Sun

Simplification: constant density n, only elastic Thomson scattering (free e^-) with (frequency independent) cross section $\sigma_{\mathrm{Th}} = 6.652 \times 10^{-25}\,\mathrm{cm}^2$
mean free path length:

$$\ell = \frac{1}{n\sigma_{\mathrm{Th}}} = \left(\frac{\varrho}{m_{\mathrm{H}}}\sigma_{\mathrm{Th}}\right)^{-1} \tag{542}$$

one dimension $\rightarrow$ only $R = R_\odot$, total time $t = N\Delta t$

$$\Rightarrow t = 9 \times 10^{10}\,\mathrm{s} = 2900\,a \ll t_{\mathrm{KH}} (= 3 \times 10^7\,\mathrm{a})$$

Random walk V

Importance of the random walk model

many processes can be described by differential equation similar to diffusion equation (e.g., heat equation, Schrödinger equation with imaginary time)

$$\frac{\partial p(x,t)}{\partial t} = D \frac{\partial^2 p(x,t)}{\partial x^2} \tag{543}$$

with diffusion coefficient D and probability $p(x,t)dx$ to find particle at time t in $[x, dx]$
in 3 dimensions: $\partial^2/\partial x^2 \equiv \nabla^2$
Moments: mean value of a function $f(x)$

$$\langle f(x,t)\rangle = \int_{-\infty}^{+\infty} f(x,t)\, p(x,t) dx \tag{544}$$

$$\Rightarrow \quad \langle x(t)\rangle = \int_{-\infty}^{+\infty} x\, p(x,t) dx \tag{545}$$

Random walk VI

Compute integral in Eq. (545) $\rightarrow$ multiply Eq. (543) by x and integrate over x

$$\int_{-\infty}^{+\infty} x \frac{\partial p(x,t)}{\partial t} dx = D \int_{-\infty}^{+\infty} x \frac{\partial^2 p(x,t)}{\partial x^2} dx \tag{546}$$

left hand side

$$\int_{-\infty}^{+\infty} x \frac{\partial p(x,t)}{\partial t} dx = \frac{\partial}{\partial t} \int_{-\infty}^{+\infty} x\, p(x,t) dx = \frac{\partial}{\partial t} \langle x \rangle \tag{547}$$

right hand side via integration by parts ($\int g\, f\, dx = g\, F| - \int g' F\, dx$), note that $p(x = \pm\infty, t) = 0$, as well as all spatial derivatives ($\partial_x p(x = \pm\infty, t) = 0$):

$$D \int_{-\infty}^{+\infty} x \frac{\partial^2 p(x,t)}{\partial x^2} dx = Dx\, \frac{\partial p(x,t)}{\partial x}\bigg|_{x=-\infty}^{x=+\infty} - D \int_{-\infty}^{+\infty} 1 \cdot \frac{\partial p(x,t)}{\partial x} dx \tag{548}$$

$$= 0 \qquad - D\, p(x,t)|_{x=-\infty}^{x=+\infty} = 0 \tag{549}$$

$$\Rightarrow \frac{\partial}{\partial t} \langle x \rangle = 0 \tag{550}$$

I.e. $\langle x \rangle \equiv$ const. for all t. For $x(t=0) = 0 \rightarrow \langle x \rangle = 0$ for all t.

Random walk VII

Analogously for $\langle x^2(t) \rangle$: integration by parts twice

$$\frac{\partial}{\partial t}\langle x^2(t)\rangle = 0 + 0 + 2D\int_{-\infty}^{+\infty} p(x,t)dx = 2D \tag{551}$$

$$\rightarrow \langle x^2(t)\rangle = 2D\, t \tag{552}$$

compare with Eq. (540) $\langle x^2(t)\rangle = \frac{\ell^2}{\Delta t} N \Delta t = \frac{\ell^2}{\Delta t} t$
$\rightarrow$ random walk and diffusion equation have same time dependence (linear)
(with $2D = \frac{\ell^2}{\Delta t}$)

Random numbers

Pseudorandom numbers I

for scientific purposes

- fast method to generate huge number of “random numbers”
- sequence should be reproducible

$\rightarrow$ use deterministic algorithm to generate *pseudorandom* numbers

Linear congruential method

start with a *seed* x_0, use one-dimensional map

$$x_n = (a\,x_{n-1} + c) \mod m \tag{553}$$

- with integers: a (multiplier), c (increment), m (modulus)
- m largest possible integer from Eq. (553) $\rightarrow$ maximum possible period is m $\rightarrow$ obtain $r \in [0,1)$ by x_n/m
- real period depends on a, c, m, e.g.,
  $a = 3,\ c = 4,\ m = 32,\ x_0 = 1 \rightarrow 1, 7, 25, 15, 17, 23, 9, 31, 1, 7, 25, \ldots \rightarrow$ period is 8 not 32

Other sources of random numbers I

Better randomness can be obtained from physical processes:

- nuclear decay (*real* randomness!), e.g, $\rightarrow$ measure Δt (difficult to implement)
- image noise, thermal noise (Johnson-Nyquist noise), e.g., $\rightarrow$ darkened USB camera (simple), special expansion cards with a diode
- "activity noise" in Unix:

```
/dev/random
/dev/urandom
```

$\rightarrow$ random *bit* patterns from input/output streams (entropy pool) of the computer
`/dev/random` blocks, if entropy pool is exhausted (since Linux 2.6: 4096 bit, cf. `/proc/sys/kernel/random/poolsize`)
`urandom` uses pseudorandom numbers seeded with "real" random numbers

For readout of Unix random devices need to interpret random bits(!) as numbers

Other sources of random numbers II

Reading from `urandom`

E.g., by using `fstream` and `union`

```
ifstream fin("/dev/urandom/") ;
union {unsigned int num ;
       char buf[sizeof(unsigned int)]; } u ;
fin.read(u.buf, sizeof(u.buf)) ;
cout << u.num ;
```

→ `fstream` reads only `char`, `buf` and `num` are at the same address → read bits in as `char` output as `unsigned int`

Tests for random numbers I

quality check for uniformly distributed random numbers

- *equal distribution:* random numbers should be fair
- *entropy:* bits of information per byte of a sequence of random numbers (same as equal distribution)
- *serial tests:* for n-tuple repetitions (often only for $n = 2$, $n = 3$)
- *run test:* for monotonically increasing/decreasing sequences, also for length of stay for a distinct interval
- and more …

Be careful!

There is no necessary or sufficient test for the randomness of a finite sequence of numbers.

$\rightarrow$ can only check if it is "apparently" random

Correlation tests I

$\rightarrow$ testing for "clumping" of numbers

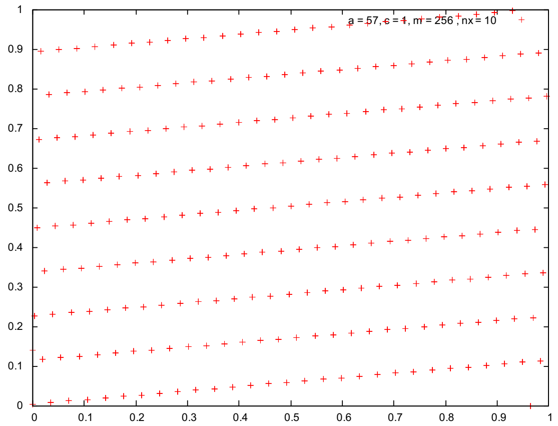
Test for doublets

- define a square lattice $L \times L$ and fill each cell at random:
- array $n(x, y)$ with discrete coordinates
- choose random $1 \leq x_i, y_i \leq L$ where x_i, y_i consecutive numbers of random number sequence
- fill cell $n(x_i, y_i)$ (e.g. set boolean to true)
- repeat procedure $t \cdot L^2$ times, t is MC time step
- $\rightarrow$ similar to nuclear decay, therefore expected:
  fraction of empty cells $\propto \exp(-t)$

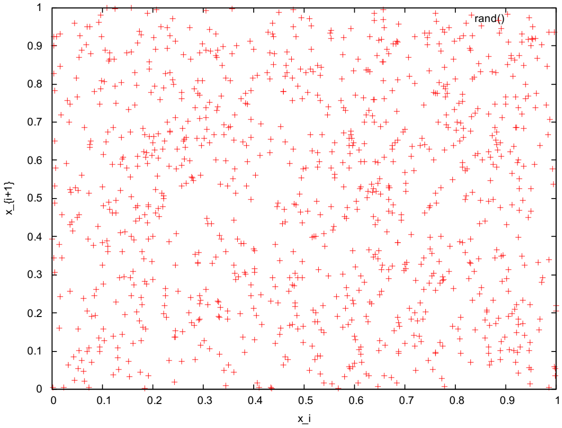
Correlation tests II

Simple correlation test

- just plot x_{i+1} over x_i $\rightarrow$ look for suspicious patterns



correlation plot for linear congruential method with bad parameters



same plot but for C++ `rand()` function

Confidence level I

Testing for randomness (also: numbers or detections)

$\rightarrow$ $\underline{\chi^2 \text{ test}}$

- let y_i the number of events in bin i and E_i the expectation value
- e.g., $N = 10^4$ random numbers, $M = 100$ bins $\rightarrow E_i = 100$ (numbers/bin)
- the χ^2 value (with y_i measured number of random numbers in bin i):

$$\chi^2 = \sum_{i=1}^{M} \frac{(y_i - E_i)^2}{E_i} \tag{554}$$

  measures the conformity of the measured and the expected distribution

- the individual terms in Eq. (554) should be ≤ 1, so for M terms $\chi^2 \leq M \rightarrow$ *reduced* χ^2 by deviding by M $\rightarrow$"minimum" red. $\chi^2 = 1$
- e.g., 5 independent runs (each $n = 10\,000$) yield $\chi^2 \approx 92, 124, 85, 91, 99 \rightarrow$ as expected for equal distribution,
  in general: χ^2 should be small (but $\chi^2 = 0$ is suspicious, e.g., here: N-periodicity in random numbers?)

Confidence level II

Confidence

- need a quantitative measure that shows normal distribution of the "error" $(y_i - E_i)$ (in particular, we test the hypothesis of uniform distribution) $\rightarrow$ chi-squared distribution

$$p(x,\nu) = \frac{1}{2^{\nu/2}\,\Gamma(\nu/2)} x^{(\nu-2)/2}\, e^{-x/2} \tag{555}$$

$$\text{where } \Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt \text{ and } \Gamma(z+1) = z! \tag{556}$$

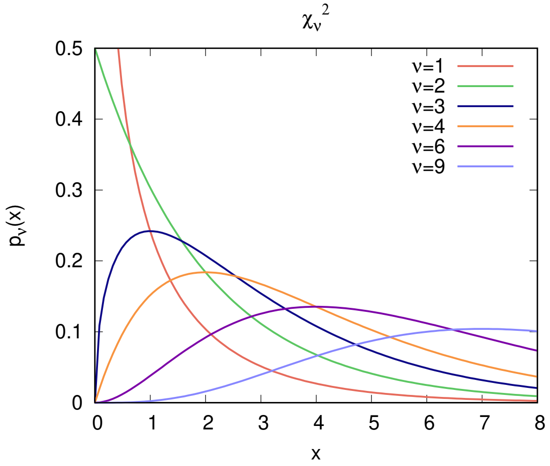
$\rightarrow$ cumulated χ^2 distribution $P(x,\nu)$:

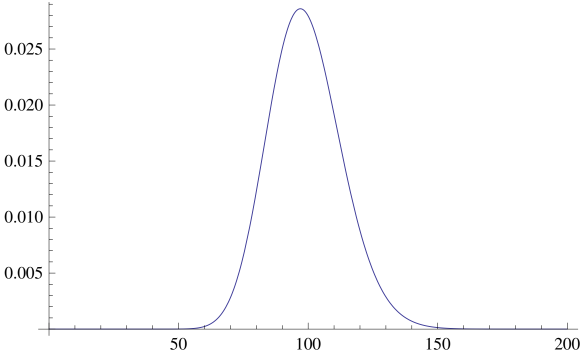
$$P(x,\nu) = \frac{1}{2^{\nu/2}\,\Gamma(\nu/2)} \int_0^x t^{(\nu-2)/2}\, e^{-t/2} dt \tag{557}$$

with ν degrees of freedom, here: $\nu = M - 1 = 99$, because of constraint $\sum_{i=1}^{M} E_i = N$

Confidence level III

- chi-square distribution





for $\nu > 30$ is $\sqrt{2x} - \sqrt{2\nu - 1}$ approximately normally distributed, for $\nu > 100$ is x approximately normally distributed with $E = \nu$ and and $\sigma = \sqrt{2\nu}$

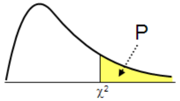
chi-square PDF for different degrees of freedom

ν

Confidence level IV

- function $Q(x,\nu) = 1 - P(x,\nu)$

  $\rightarrow$ probability that $\chi^2 > x$



- we want to check: How likely to get a χ^2 of, e.g., 124 (our largest measured χ^2)?
  $\rightarrow$ solve $Q(x,\nu) = q$ (probability $\chi^2 > x$ for given x,ν) for x, or look it up in tables for $\nu = M - 1 = 99$ (e.g., `https://www.medcalc.org/manual/chi-square-table.php`)

x	138.9	134.6	123.2	110.6	98
q	0.005	0.01	0.05	0.2	0.5

- for our case: 1 out of 5 runs (20%) had $y_2 = 124$, but $Q(x,\nu)$ implies for $x = 123$ only 5%, i.e., 1 out of 20 runs with $\chi^2 \geq 123$
- therefore: confidence level $< 95\%$, rather 80% (because of $q = 0.2$ for $x = 111$)
- try to increase confidence level: more runs $\rightarrow$ if still only 1 out 20 with $\chi^2 > 123$
  $\rightarrow$ confidence level at 95%

MC Error estimation

Numerical integration and error I

Numerical integration (exact or MC) gives approximation

$$\int_a^b f(x)dx = Q(f) + E(f) \tag{558}$$

$Q(f)$ so-called quadrature formula,
$E(f)$ error $\rightarrow$ unknown (obvious)

Aim: estimate magnitude of error

so far: error calculated from our knowledge of the exact result

Numerical integration and error II

- Obvious: for constant integrand f is $E = 0$, i.e. F_n is independent of n (and always the same)
- Idea: try to estimate the error with help of the *standard deviation* σ:

$$\sigma^2 = \langle f(x)^2 \rangle - \langle f(x) \rangle^2 \tag{559}$$

$$\langle f(x) \rangle = \frac{1}{n} \sum_{i=1}^{n} f(x_i) \tag{560}$$

$$\langle f(x)^2 \rangle = \frac{1}{n} \sum_{i=1}^{n} f(x_i)^2 \tag{561}$$

- if f constant $\rightarrow \sigma = 0$

Numerical integration and error III

- consider the example $f(x) = 4\sqrt{1-x^2}$ with $F = \int_0^1 f(x)dx = \pi$
  Calculate σ for different n (cf. Gould et al. 1996)

F_n	n	$E = \lvert F_n - \pi \rvert$	σ
3.271771	10^1	0.13017	0.78091
3.100276	10^2	0.04131	0.91441
3.173442	10^3	0.03185	0.85013
3.135863	10^4	0.00572	0.90317
3.142189	10^5	0.00059	0.89051
3.141798	10^6	0.00020	0.89236

- σ almost constant and much larger than E
- but: decrease of E from $n = 10^2$ to $n = 10^4$ by a factor of 10 $\rightarrow \sim 1/n^{1/2}$ (?)
- therefore: σ says how much f varies in $[a, b]$

Numerical integration and error IV

- idea: estimate E by several *runs* α for constant $n = 10^4$, each with result M_α:

M_α	α	$E = \lvert F - \pi \rvert$	$\lvert M_{\alpha+1} - M_\alpha \rvert$
3.14892	1	0.00735	0.00845
3.13255	2	0.00904	0.01637
3.14042	3	0.00117	0.00787
3.14600	4	0.00441	0.00558
3.15257	5	0.01098	0.00657
3.13972	6	0.00187	0.01285
3.13107	7	0.01052	0.00865
3.13585	8	0.00574	0.00478
3.13442	9	0.00717	0.00143
3.14047	10	0.00112	0.00605

- E varies, differences $\lvert M_\alpha - M_\beta \rvert_{\alpha \neq \beta}$ between results comparable with E, therefore:

Numerical integration and error V

- define standard deviation σ_m of the means:

$$\sigma_m^2 = \langle M^2\rangle - \langle M\rangle^2 \tag{562}$$

$$\langle M\rangle = \frac{1}{m}\sum_{\alpha=1}^{m} M_\alpha \quad \rightarrow \quad \langle M^2\rangle = \frac{1}{m}\sum_{\alpha=1}^{m} M_\alpha^2 \tag{563}$$

$$\tag{564}$$

- for the runs 1 till 10 one gets $\sigma_m = 0.006762 \rightarrow$ comparable with E
- exact: one run has the chance of 68% that M_α is in in the range $\pi \pm \sigma_m$
- however method not very usefull, as several runs are required
- actually for large n holds:

$$\sigma_m = \frac{\sigma}{\sqrt{n-1}} \approx \frac{\sigma}{\sqrt{n}} \tag{565}$$

e.g., for $n = 10^4$ is $\sigma_m = 0.90317/100 \approx 0.009$, i.e., consistent with our estimate $\sigma_m = 0.007$ and the error $E = 0.006$

How can we get σ without α runs?

Numerical integration and error VI

Hence, split one run, e.g., in $s = 10$ subsets k such that each contains $n/s = 1000$ trials and has result S_k

Then, with the mean $\langle S \rangle$ from the different runs is also

$$\sigma_s^2 = \langle S^2 \rangle - \langle S \rangle^2 \tag{566}$$

and

$$\sigma_m = \sigma_s / \sqrt{s} \tag{567}$$

Numerical integration and error VII

Derivation/proof:

- random variable x
- m runs with each n trials ($= m \times n$ trials in total)
- index α lables a run, i a single trial

result from one run (= measurement):

$$M_\alpha = \frac{1}{n} \sum_{i=1}^{n} x_{\alpha,i} \tag{568}$$

the arithmetic mean of all mn trials is:

$$\overline{M} = \frac{1}{m} \sum_{\alpha}^{m} M_\alpha = \frac{1}{nm} \sum_{\alpha=1}^{m} \sum_{i=1}^{n} x_{\alpha,i} \tag{569}$$

Numerical integration and error VIII

difference of a one run α and the total mean

$$e_\alpha = M_\alpha - \overline{M} \tag{570}$$

Hence the variance (standard deviation^2) can be written for the runs as:

$$\sigma_m^2 = \frac{1}{m}\sum_{\alpha=1}^{m}(M_\alpha - \overline{M})^2 = \frac{1}{m}\sum_{\alpha=1}^{m} e_\alpha^2 \tag{571}$$

Now finding the relation between σ_m and σ of the individual $m \times n$ trials. Difference between one trial and the the mean of one *run*:

$$d_{\alpha,i} = x_{\alpha,i} - \overline{M} \tag{572}$$

Therefore the variance for *all* $m \times n$ trials:

$$\sigma^2 = \frac{1}{mn}\sum_{\alpha=1}^{m}\sum_{i=1}^{n} d_{\alpha,i}^2 \tag{573}$$

Numerical integration and error IX

With help of Eq. (572) the Eq. (570) can be rewritten as:

$$e_\alpha = M_\alpha - \overline{M} = \frac{1}{n}\sum_{i=1}^{n}\left(x_{\alpha,i} - \overline{M}\right) = \frac{1}{n}\sum_{i=1}^{n} d_{\alpha,i} \tag{574}$$

Insert Eq. (574) into Eq. (571):

$$\sigma_m^2 = \frac{1}{m}\sum_{\alpha=1}^{m}\left(\frac{1}{n}\sum_{i=1}^{n} d_{\alpha,i}\right)\left(\frac{1}{n}\sum_{j=1}^{n} d_{\alpha,j}\right) \tag{575}$$

Numerical integration and error X

The products in Eq. (575) consist of terms $i = j$ and terms $i \neq j$. As the trials are independent of each other, for *large* n the differences $d_{\alpha,i}$ and $d_{\alpha,j}$ are on average as often negative as positive, i.e., the terms $i \neq j$ cancel out on average. What remains are the terms for $i = j$:

$$\sigma_m^2 = \frac{1}{mn^2} \sum_{\alpha=1}^{m} \sum_{i=1}^{n} d_{\alpha,i}^2 \tag{576}$$

By comparison with Eq. (573) for individual variance: $\sigma^2 = \frac{1}{mn} \sum_{\alpha=1}^{m} \sum_{i=1}^{n} d_{\alpha,i}^2$ one gets required variance of runs:

$$\sigma_m^2 = \frac{\sigma^2}{n} \quad \Rightarrow \quad \underline{\underline{\sigma_m = \frac{\sigma}{\sqrt{n}}}} \tag{577}$$

$\square$

$\rightarrow$ the standard deviation (= error estimate) scales with $\frac{1}{\sqrt{n}}$

Why Monte-Carlo (integration)?

Performance of integration techniques I

Already seen: for 1d integration, dependence of truncation error on number of intervals ($\sim$ samples)

method	$\sigma(N)$
rectangular rule	N^{-1}
trapezoid rule	N^{-2}
Simpson's rule	N^{-4}
MC sample-mean method	$N^{-1/2}$

$\rightarrow$ for 1d MC sample-mean inefficient integration method

Truncation error derived from Taylor series expansion of integrand $f(x)$:

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2}f''(x_i)(x - x_i)^2 + \ldots \tag{578}$$

$$\int_{x_i}^{x_{i+1}} f(x)dx = f(x_i)\Delta x + \frac{1}{2}f'(x_i)(\Delta x)^2 + \frac{1}{6}f''(x_i)(\Delta x)^3 + \ldots \tag{579}$$

Performance of integration techniques II

For the rectangular rule ($f(x_i)\Delta x$), error Δ_i in leading order for $[x_i, x_{i+1}]$ is

$$\Delta_i = \left[\int_{x_i}^{x_{i+1}} f(x)dx\right] - f(x_i)\Delta x \approx \frac{1}{2}f'(x_i)(\Delta x)^2 \tag{580}$$

$\rightarrow$ error per interval; as there are N intervals in total and $\Delta x = (b-a)/N$ $\rightarrow$ total error for rectangular rule $N\,\Delta_i \sim N\,(\Delta x)^2 \sim N(\frac{b-a}{N})^2 \sim N^{-1}$

Analogously for trapezoid rule, where we estimate $f(x_{i+1})$ by Eq. (578):

$$\Delta_i = \left[\int_{x_i}^{x_{i+1}} f(x)dx\right] - \frac{1}{2}[f(x_i) + f(x_{i+1})]\Delta x \tag{581}$$

$$= \left[f(x_i)\Delta x + \frac{1}{2}f'(x_i)(\Delta x)^2 + \frac{1}{6}f''(x_i)(\Delta x)^3 + \ldots\right] \tag{582}$$

$$- \frac{1}{2}\Delta x\left[f(x_i) + f(x_{i+1}) + f'(x_i)\Delta x + f'(x_{i+1})\Delta x + \frac{1}{2}f''(x_i)(\Delta x)^2 + \ldots\right] \tag{583}$$

$$\approx -\frac{1}{3}f''(x_i)(\Delta x)^3 \quad \rightarrow \quad \text{total error } \sim N^{-2} \tag{584}$$

Performance of integration techniques III

For Simpson's rule $f(x)$ is approximated as parabola on $[x_{i-1}, x_{i+1}] \rightarrow$ terms $\sim f''$ cancel, moreover because of symmetry terms $\sim f'''(\Delta x)^4$ cancel $\rightarrow$ error for interval $[x_i, x_{i+1}]$ is $\sim f^{(4)}(x_i)(\Delta x)^5$ and total error for $[a, b]$ is $\sim N^{-4}$

Integration error in 2d

extend previous estimates for rectangular rule in 2d, so for $f(x, y)$: integral $\rightarrow$ sum of volumes of parallelograms with cross section area $\Delta x \Delta y$ and height $f(x, y)$ at one corner
Taylor series expansion of $f(x, y)$

$$f(x,y) = f(x_i, y_i) + \frac{\partial f(x_i, y_i)}{\partial x}(x - x_i) + \frac{\partial f(x_i, y_i)}{\partial y}(y - y_i) + \ldots \tag{585}$$

$$\Delta_i = \left[\int\int f(x,y)dxdy\right] - f(x_i, y_i)\Delta x \Delta y \tag{586}$$

Performance of integration techniques IV

Now, substitute Taylor expansion Eq. (585) into error estimate Eq. (586), integrate each term
$\rightarrow$ term $\sim f$ cancels out
and $\int (x - x_i)dx = \frac{1}{2}(\Delta x)^2 \rightarrow \int dy$ gives another factor Δy; similar for $(y - y_i)$
As $O(\Delta y) = O(\Delta x)$, error for interval $[x_i, x_{i+1}]$ and $[y_i, y_{i+1}]$ is

$$\Delta_i \approx \frac{1}{2}[f'_x(x_i, y_i) + f'_y(x_i, y_i)](\Delta x)^3 \tag{587}$$

$\rightarrow$ error for one parallelogram $\sim (\Delta x)^3$, for N parallelograms $N \cdot (\Delta x)^3$
But in 2d: $N = A/(\Delta x)^2$
$\rightarrow$ total error $N(\Delta x)^3 = N\,A^{3/2}\,N^{-3/2} \sim N^{-1/2}$ (whereas in 1d: N^{-1})
Analogously for trapezoid rule in 2d: N^{-1}, for Simpson's rule in 2d: N^{-2}

In general: if in 1d integration error $\sim N^{-p}$
$\rightarrow$ integration error in d dimensions $\sim N^{-p/d}$ (curse of dimensionality)

In contrast: MC integration error $\sim N^{-1/2}$ independent of d $\rightarrow$ superior for large d
(think about integrals $\int_V \int_{V_p} f\, dp^3 dx^3$ in statistical mechanics)

How to integrate in higher dimensions I

Integrals of functions of more than 1 variable, over regions with $d > 1$, are difficult!

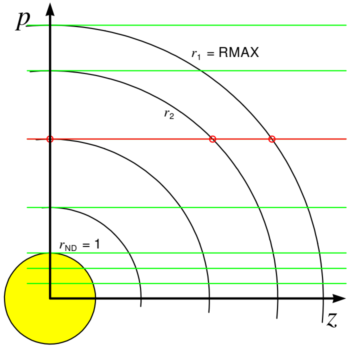
1. function evaluation: if n function calls required for some accuracy in 1d $\rightarrow \sim n^d$ samples needed for d dimensions (e.g., 30 calls in 1d vs. approx. 30 000 in 3d)
2. integration region in d dimensions defined by $d-1$ dimensional boundary $\rightarrow$ can be very complicated for $d > 1$ (e.g. not convex, not simply connected)

Ad 1.) $\rightarrow$ try to reduce integral to lower dimensions by exploiting symmetry of function and boundary and changing coordinates. E.g., spherically symmetric function over spherical region $\rightarrow$ in polar coordinates 1d integral

How to integrate in higher dimensions II

Example: PoWR code for expanding atmospheres

- non-LTE (i.e. $\vec{n}(\vec{J})$ from statistical equations + ALI $\rightarrow$ Newton's method) radiative transfer in wind (i.e. CMF RT with Mio. of frequency points K, coarsend $\vec{J}(\vec{n})$ for $\vec{n} \rightarrow K \approx 1000$) $\rightarrow$ iteratively solved
- assuming spherical symmetry with, e.g., $ND = 50$ depth-points, typically for each iteration $\approx 5\,\mathrm{s}$, in total ≈ 1000 iterations $\rightarrow \sim$ h
- in 3D: $2500 \times$ more "depthpoints" $\rightarrow$ each iteration now 3.5 h (!) $\rightarrow$ total $\frac{1}{2}$ a



For each depthoint: Solve $\boldsymbol{nP} = 0$ where $P_{ii} := -\sum_{j \neq i}^{n} P_{ij}$ with $n \sim 500$
+ radiative transfer (see sketch)

How to integrate in higher dimensions III

Ad 2.)

- if boundary complicated, integrand not strongly peaked in very small regions, relatively low accuracy required $\rightarrow$ MC integration! (see below)
- if boundary simple, smooth integrand, (+ high accuracy required) $\rightarrow$ repeated 1d integrals or multidimensional quadrature
- if integrand peaks in certain regions $\rightarrow$ split integral into several “smooth” regions (requires knowledge of behaviour of integrand)

How to integrate in higher dimensions IV

Repeated 1d integration

Let $d = 3$ with x, y, z and boundaries $[x_1, x_2]$, $[y_1(x), y_2(x)]$, $[z_1(x, y), z_2(x, y)] \rightarrow$ find x_1, x_2 and functions $y_1(x), y_2(x), z_1(x, y), z_2(x, y)$ such that

$$\int\int\int dx\, dy\, dz\, f(x, y, z) = \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} dy \int_{z_1(x,y)}^{z_2(x,y)} dz\, f(x, y, z) \tag{588}$$

Example: 2d integral over circle with radius R centered on $(0, 0)$

$$\int_{x_1=-R}^{x_2=+R} dx \int_{y_1(x)=-\sqrt{R-x^2}}^{y_2(x)=\sqrt{R-x^2}} dy\, f(x, y) \tag{589}$$

Note that Fubini's theorem for iterated integrals assumes that the integrand is absolutely integrable: $\int\int |f(x, y)| dx\, dy < +\infty$.

How to integrate in higher dimensions V

Innermost integration over z yields a function $G(x, y)$:

$$G(x, y) := \int_{z_1(x,y)}^{z_2(x,y)} f(x, y, z)\, dz \tag{590}$$

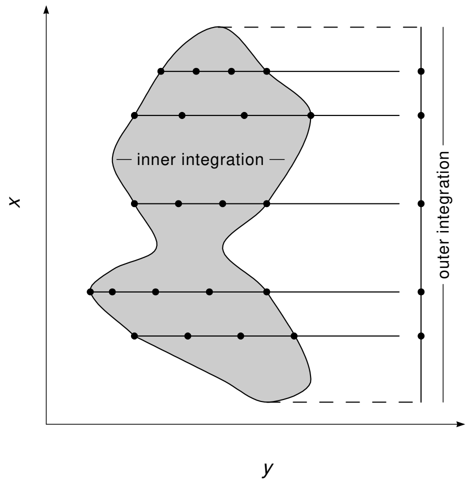
then intgration over y yields $H(x)$:

$$H(x) := \int_{y_1(x)}^{y_2(x)} G(x, y)\, dy \tag{591}$$

finally the overall integral I is

$$I = \int_{x_1}^{x_2} H(x)\, dx \tag{592}$$

How to integrate in higher dimensions VI



instead of using fixed Cartesian mesh of points, better evaluate function at suitable x locations (along y-axis), while inner integration (over y) chooses suitable y values;
$\rightarrow$ inner integration call (over y) many more times than outer integration (over x)

Implementation of Eq. (590)-(592) requires 3 separate copies of some 1d integration routine, so one for each x, y, z integration or recursive calls of the same routine

Example: Fortran sniplet for 3d iterated integration

```
!     identical copies quadx, quady, quadz
!     of 1d-integration routine;
!     user provides func(x,y,z), y1(x),
!     y2(x), z1(x,y), z2(x,y) as in Eq.(588)

      SUBROUTINE quad3d(x1, x2, ss)
      REAL ss, x1, x2, h
      CALL quadx(h, x1, x2, ss)
      RETURN
      END

      FUNCTION f(zz)
      REAL f, zz, func, x, y, z
      COMMON /xyz/ x, y, z
      z = zz
      f = func(x, y, z)
      RETURN
      END

      FUNCTION g(yy)
      REAL g, yy, f, z1, z2, x, y, z
      COMMON /xyz/ x, y, z
      REAL ss
      y = yy
      CALL quadz(f, z1(x,y), z2(x,y), ss)
      g = ss
      RETURN
      END

      FUNCTION h(xx)
      REAL h, xx, g, y1, y2, x, y, z
      COMMON /xyz/ x, y, z
      REAL ss
      x = xx
      CALL quady(g, y1(x), y2(x), ss)
      h = ss
      RETURN
      END
```

MC integration in higher dimensions I

Example: Mass and center of mass of cut torus

Section of a torus with radius R and cross section radius r

$$z^2 + (\sqrt{x^2 + y^2} - R)^2 \leq r \tag{593}$$

section defined by

$$x \geq a \qquad y \geq b \tag{594}$$

Need to evaluate following integrals

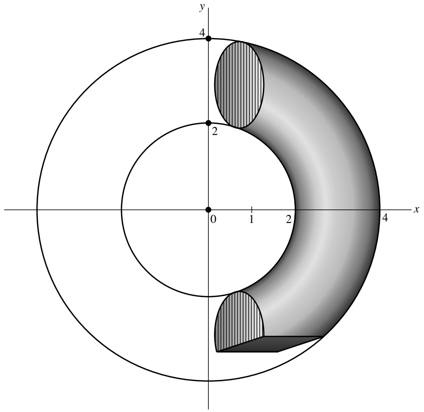
$$M = \int \rho \, dx \, dy \, dz \quad M_x = \int x\rho \, dx \, dy \, dz \tag{595}$$

$$M_y = \int y\rho \, dx \, dy \, dz \quad M_z = \int z\rho \, dx \, dy \, dz \tag{596}$$

i.e., x-coordinate of center of mass is $x = M_x / M$ and so on

MC integration in higher dimensions II

MC integration for a torus (centered on origin, outer radius = 4, inner radius = 2) section, where $x \leq 1$ and $y \leq -3$, i.e., bounds given by intersection of two planes. Integration limits cannot be easily given in analytically closed form



from Press et al. (2007)

Choose region that encloses torus section, e.g, rectangular box with $1 \leq x \leq 4$, $-3 \leq y \leq 4$, and $-1 \leq z \leq 1$, hence total volume of box is $V = 3 * 7 * 2$

MC integration in higher dimensions III

Example: C/C++ sniplet for MC integration of torus section

```
int N = 1000 ; // sample points
double V = 3. * 7. * 2. ; // sample volume
double den = 1. ; // density rho
double sw = 0., varw = 0. ; // mass and variance
double swx = 0., varx = 0. ; // x-coordinate and var. for center of mass
 ...
for (i = 0 ; i < N ; ++i) {
x =  1. + 3. * rand()/double(RAND_MAX) ; // cut of torus
y = -3. + 7. * rand()/double(RAND_MAX) ; // cut of torus
z = -1. + 2. * rand()/double(RAND_MAX) ;
 if ( pow(z*z + (sqrt(x*x + y*y) -3. ), 2.) <= 1. ) {
  sw  = sw + den ; varw = varw + den*den ;
  swx = swx + x * den ; varx = varx + (x*den)*(x*den) ;
   ...
 } }
w = V * sw / N ; // mass of torus
x = V * swx / N ; // x-coordinate
dw = V * sqrt((varw / N - (sw/N)*(sw/N)) / N) ; // error estimate mass
dx = V * sqrt((varx / N - (swx/N)*(swx/N)) / N) ; // error estimate x-coordinate
 ...
```

MC integration in higher dimensions IV

Conclusions about advantage of MC integration

1. MC integration error decreases independent of dimension with $\sim N^{-1/2}$ $\rightarrow$ superior for integrals with many integration variables (e.g., phase space integrals, QM)
2. MC integration easy to implement for any geometry $\rightarrow$ superior for 3d models without simple symmetry (e.g., spherical symmetry)

Techniques of MC parallelization

Neutron transport with packets I

So far: single neutron n^0
Improvement/speed up: consider "neutron packets", i.e. we follow an ensemble of neutrons (which advances with random $\ell, \cos\theta$ as before)
$\rightarrow$ determine *fraction* of the scattered and captured neutrons

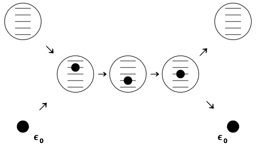
1. scattering:	fraction of scattered n^0: p_s, fraction of absorbed n^0: p_c
2. scattering:	fraction of scattered n^0: p_s^2, fraction of absorbed n^0: $p_c p_s$
*m*th scattering	fraction of scattered n^0: p_s^m, fraction of absorbed n^0: $p_c p_s^{m-1}$

so, after *m*th scattering:
$\rightarrow$ total fraction of captured neutrons:
$f_c = p_c + p_c p_s + p_c p_s^2 + \ldots + p_c p_s^{m-1}$
$\rightarrow$ total fraction of scattered neutrons:
$f_s = p_s^m$
$\rightarrow$ if position $x < 0$: add f_s to f_refl
$\rightarrow$ if position $x > t$: add f_s to f_trans
$\rightarrow$ Note: requires normalization of the fractions afterwards

Neutron transport with packets II

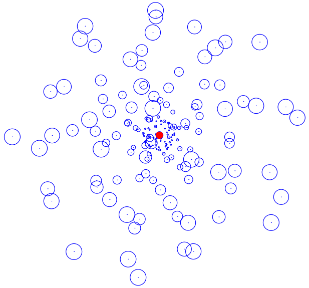
$\rightarrow$ see: Lucy (2002): "Monte Carlo transition probabilities"

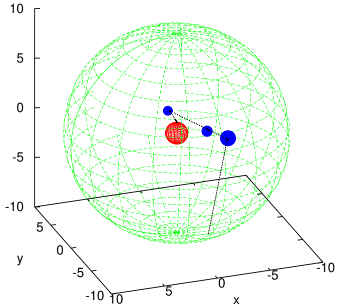
- instead of individual photons, use energy packets of photons of same frequency ν ($\epsilon(\nu) = nh\nu$), packets always have same energy ϵ_0 $\rightarrow$ different n
- elastic scattering (e.g., Thomson, resonance): $\nu_e = \nu_a$
- absorption leads to re-emission following: $\epsilon(\nu_e) = \epsilon(\nu_a)$, no packet (= energy) lost or created $\rightarrow$ divergence-free radiation field
- macro-atoms with discrete internal states, activation via r-packet (radiative) of appropriate CMF frequency or k-packet (kinetic); active macro-atom performs internal transitions and gets inactive by emission of r- or k-packet

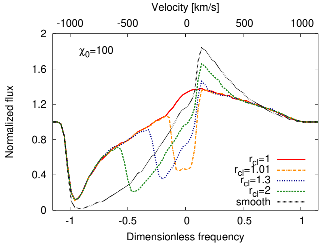


Neutron transport with packets III

→ see Šurlan et al. (2012): "Three-dimensional radiative transfer in clumped hot star winds. I. Influence of clumping on the resonance line formation"







2-D projection of an example of a realization of a stochastic 3-D wind model. The path of one particular photon inside a realization of our clumped wind. The effect of variation of the onset of the clumping $r_{\rm cl}$ on a resoance line

Parallelization

Parallelization

Many runs in MC simulations required for reliable conclusions ($\sigma \sim \frac{1}{\sqrt{N}}$)

Often: Result of one run (e.g., path of a neutron through a plate) *independent* from other runs

$\rightarrow$ Idea: acceleration by parallelization
Problem: concurrent access to memory resources, i.e. variables (e.g., n_s, f_refl)
Solution: special libraries that enable multithreading (e.g., `OpenMP`) or multiple processes (e.g., `MPI`) for one program

$\rightarrow$ insert: pipelining, vectorization, parallelization

CPU Performance

What influences the performance of a CPU (= runtime of your code)?

- architecture/design: out-of-order execution (all x86 except for Intel Atom), pipelining (stages), vectorization units (width)
- cache sizes (kB . . . MB) and location: L1 cache for each core, L3 for processor
- clock rate ($\sim$GHz): only within a processor family usable for comparison due to different number of instruction per clock (IPC) of design, even more complicated because of variable clock rates (base, peak) to exploit TDP (thermal design power)
  $\rightarrow$ impact on single-thread performance
- number of cores (1 . . .): $\rightarrow$ impact on multi-thread performance

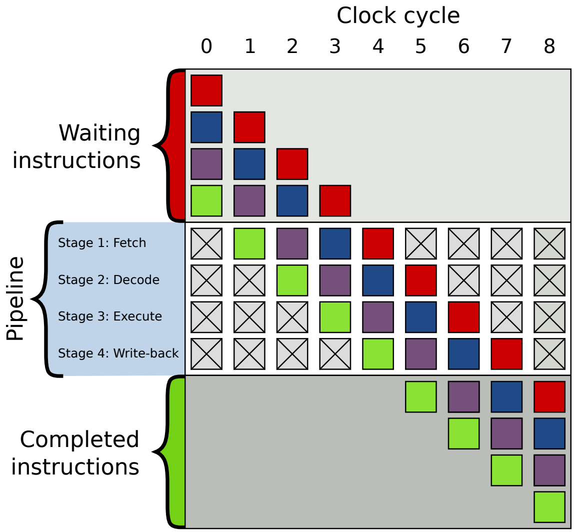
Pipelining I

splitting machine instruction into a sequence

independent execution of instructions, each consisting of

- instruction fetching (IF)
- instruction decoding (ID) + register fetch
- execution (EX)
- write back (WB)

operations of instructions are processed at the same time → quasi parallel execution, higher throughput



By en:User:Cburnett - Own workThis vector image was created with Inkscape., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1499754

Pipelining II

NetBurst disaster

Pentium 4 (2000-2008) developed to achieve $>$ 4GHz (goal: 10 GHz) clockrate by several techniques, i.a., *long* pipeline:

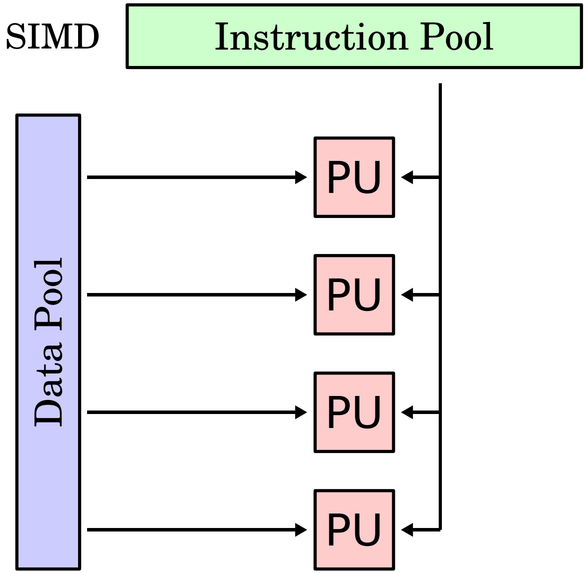
- 20 stages (Pentium III: 10) up to 31 stages (Prescott core)
- smaller number of instructions per clock (IPC) (!)
- increased branch misprediction (also only 10%, improved by 33% for Pentium III)
- larger penalty for misprediction

$\rightarrow$ compensated by higher clock rate

higher clock rate $\rightarrow$ higher power dissipation, especially for 65 (Presler, Pentium D), 90 (Prescott) up to 180 nm (Williamette) structures

$\rightarrow$ power barriere at 3.8 GHz (Prescott)

SSE and AVX I



- SSE - Streaming SIMD Extensions (formerly: ISSE - Internet SSE)
- SIMD - Single Instruction Multiple Data ($\rightarrow$ cf. Multivec, AMD3Dnow!), introduced with Pentium III (Katamai, Feb. 1999)

SSE and AVX II

- enables vectorization of instructions (not to be confused with pipelining or parallelization), often new, complex machine instructions required,
  e.g., `PANDN` $\rightarrow$ bitwise `NOT` + `AND` on *packed integers*
- comprises 70 different instructions, e.g., `ADDPS` – add packed single-precision floats (two "vectors" each with 4 32 bit) into a 128 bit register
- works with 128 bit registers (3Dnow! only 64 bit), but first execution units (before *Core* architecture) only with 64 bit
- AVX - Advanced Vector Extensions with 256 bit registers, theoretically doubled speed!
  since Sandy Bridge (Intel Core 2nd generation, e.g., i7-2600 K) and Bulldozer (AMD)
  $\rightarrow$ AVX-512 with 512 bit registers in Skylake (6th generation, e.g., Core i7-6700); AMD Zen 4
  Note: AVX-512 instructions may reduce the clockrate on Intel CPUs (heat limit)

SSE and AVX III

- supported by all common compilers, e.g.,
  ```
  ifort -sse4.2
  ifort -axcode COMMON-AVX512
  g++ -msse4.1
  g++ -mavx512f
  ```
- very easy (automatic) and efficient optimization, e.g., for unrolled loops → vectorization

Caution!

Different precisions for SSE-`doubles` (e.g., 64 bit) and FPU-`doubles` (80 bit), especially for buffering, so results of `doubles`, e.g.,

```
xx = pow(x,2) ;
sqrt( xx - x*x) ;
```

usually not predictable

Multi-cores

Mulit-cores

- originally one core per processor, sometimes several processors per machine/board (supercomputer)
- many units, e.g., arithmetic logic unit (ALU), register, already multiply existing in one processor
- first multi-core processors: IBM POWER4 (2001);
  desktop $\rightarrow$ Smithfield (2005), e.g., Pentium D
- Hyper-threading (HT): introduced in Intel Pentium 4 $\rightarrow$ for better workload of the computing units by simulation of another, logical processor core (compare: AMD Bulldozer design with modules)
- today: up to 64 cores for desktop (AMD Zen: Ryzen Threadripper 5995WX, TDP 280 W) or 96 for servers (e.g., AMD EPYC 9654, TDP 360 W – even 2 CPUs per board)
  + Hyperthreading
- arms race of cores instead of clock rate (NetBurst disaster)

Multi-cores and compilers

Acceleration by parallelization

- parallelization done, e.g., by multithreading (from *thread*)
  for shared memory (RAM on one “node”, usually on one mainboard)
- “The free lunch is over” $\rightarrow$ no simple acceleration more of *single-thread* programs by pure increase of clock rate (exceptions: Turbo Boost, Turbo Core, in some ways larger caches may help)
- multithreading supported by, e.g., `OpenMP` (shared memory), see below
- different from: multiprocessing parallelization via `MPI` (Message Passing Interface)
  $\rightarrow$ distributed computing (cf. Co-array Fortran) but can be combined: MPI + OpenMP;
  usually: MPI more complicated (and slower) than OpenMP $\rightarrow$ trend for “larger nodes”

GPGPU

General-purpose computing on graphics processing units $\rightarrow$ further development of graphic cards

- e.g., Nvidia (Tesla, Fermi); AMD (Radeon Instinct)
  $\rightarrow$ *Frontier* (USA, 1st since June 2022 in Top500) with 9 472 nodes (each with AMD-EPYC-7A53 64core CPU + 4 GPU MI250X x2) reaches 1.1 ExaFLOPS (for comparison: 24 core desktop CPU $\approx$ 8 TeraFLOPS $\rightarrow 7 \times 10^{-6}$ of *Frontier*)
- so-called shaders $\rightarrow$ highly specialized ALUs, often only with single precision (opposite concept: Intel's Larrabee)
- programming (not only graphics) via CUDA (Nvidia) or OpenCL (more general)
- OpenCL $\rightarrow$ parallel programming for arbitrary systems, also NUMA (non-uniform memory access), but very abstract and complex concept and also complicated C-syntax
- CUDA support, e.g., by PGI Fortran compiler $\rightarrow$ simple acceleration without code modifications

OpenMP

OpenMP - Intro and Syntax I

OpenMP - Open Multi-Processing

- for shared-memory systems (e.g., multi core) per node
- directly available in `g++`, `gfortran`, and Intel compilers
- insertion of so-called OpenMP (pragma) directives :

Example: `for` loop

C++

```
#include <omp.h>
 ...
#pragma omp parallel for
for (int i = 1 ; i <= n ; ++i)
{ ... }
```

Fortran

```
      USE omp_lib ! ifort declarations
!$OMP PARALLEL DO
      DO i = 1, n
       ....
      ENDDO
!$OMP END PARALLEL DO
```

instructs parallel execution of the `for` loop, i.e., there are copies of the loop (different iterations) which run in parallel
$\rightarrow$ only the labeled section runs in parallel

OpenMP - Intro and Syntax II

$\rightarrow$ pragma directives are syntactically seen comments, i.e., invisible for compilers without OpenMP support

- realization during runtime by *threads*
- number of used threads can be set, e.g., by environment variable

```
export OMP_NUM_THREADS=4  # bash
setenv OMP_NUM_THREADS 4  # tcsh
```

$\rightarrow$ obvious: per core only one thread can run at the same time (but: Intel's hyper-threading, AMD's Bulldozer design) $\rightarrow$ in HPC often reasonable:

number of threads = number of physical CPU cores

Caution!

Distributing and joining of threads produces some overhead in CPU / computing time (e.g., copying data) and is therefore only efficient for complex tasks within each thread. Otherwise multithreading can slow down program execution.

OpenMP - Intro and Syntax III

Including the OpenMP library:

C++

```
#ifdef _OPENMP
#include <omp.h>
#endif
```

Fortran

```
!     only needed for declaration of
!     OMP functions etc. with ifort:
!$    use omp_lib
```

→ instructions between `#ifdef _OPENMP` and `#endif` (Fortran: following `!$`) are only executed if compiler invokes OpenMP

Compile with

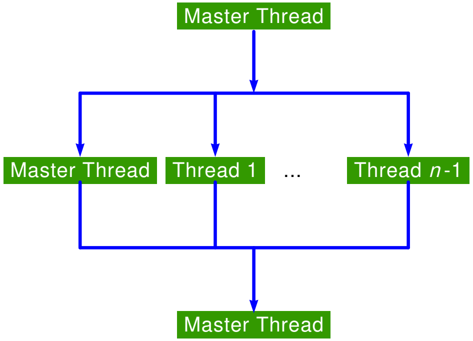
```
g++        -fopenmp
icpx       -qopenmp   (deprecated: -openmp)

gfortran   -fopenmp
ifort      -qopenmp   (deprecated: -openmp)
```

OMP functions

Useful: functions specific for OpenMP, e.g., for number of available CPU cores, generated (maximum) number of threads, and current number of threads:

```
omp_get_num_procs()   // number of (logical) processor cores
omp_get_max_threads() // max. number of (automatic) generated threads
omp_get_num_threads() // number of current threads
omp_get_thread_num()  // number of the current thread
```

Join-fork model:
thread that executes `parallel` directive becomes master of thread group with ID$= 0$



OMP – Access to variables: `shared` and `private` I

Very important: organization of the accessibility of the involved data, i.e. assign attributes `shared` or `private` to thread variables

`shared`

→ default for variables declared outside the parallel section
data are visible in all threads and can be modified (concurrent access)

```
int sum = 0 ;
#omp pragma parallel for
for (int k = kmax ; k > 0 ; --k) {
     sum += k ; // sum is implicitly shared
```

```
      NSUM  = 0
!$OMP PARALLEL DO
      DO K = KMAX, 1, -1
         NSUM = NSUM + K   ! NSUM is implicitly shared
```

OMP – Access to variables: shared and private II

in contrast to:

private

each thread has its own copy of the data, which are invisible for other threads, especially from outside of the parallel section.

Loop iteration variables are private by default and should be declared in the loop header for clarity:

```
#omp pragma parallel for
for (int k = kmax ; k > 0 ; --k)  // k is implicitly private

!$OMP PARALLEL DO
      DO K = KMAX, 1, -1           ! K is implicitly private
```

Moreover, there are further so-called `data clauses`, e.g., `firstprivate` (initialization before the parallel section), `lastprivate` (last completed thread determines the value of the variable after the parallel section) and many more . . .

→ This is the complicated part of OpenMP!

OMP – Access to variables: shared and private III

Example private

C++:

```
int j, m = 4 ;
#pragma omp parallel for private (j)
for (int i = 0 ; i < max ; i++) {
  j = i + m ;
    ... ;
}
```

Fortran:

```
      INTEGER :: j, m
!$OMP PARALLEL DO PRIVATE (j)
      DO i = 0, max
        j = i + m
        ...
      ENDDO
!$OMP END PARALLEL DO
```

→ loop variable `i` and explicitly private variable `j` as “local” copies in each thread
→ variable `m` implicitly `shared` (be careful in Fortran because of implicit declarations within, e.g. loops)

OMP – `critical` and `reduction` I

General form of OpenMP directive for parallelization:

`#pragma omp parallel`

→ `parallel` section also possible without a loop, section is executed per thread (in C/C++: `{ }` block required for multiple commands):

C++:

```
#pragma omp parallel
{
 cout << "Hi!" ;
 cout << endl ;
}
```

Fortran:

```
!$OMP PARALLEL
       print *, "Hi!"
!$OMP END PARALLEL
```

OMP – critical and reduction II

#pragma omp critical

→ within a parallel section
is executed by each thread, but never at the same time (avoiding race conditions for shared resources)

C++:

```
#pragma omp critical
{
  WDrawPoint(myworld, x, y, c) ;
}
```

Fortran:

```
!$OMP CRITICAL
      CALL PGDRAW (x, y)
!$OMP END CRITICAL
```

OMP – critical and reduction III

Example: critical access to an array

C++:

```
#pragma omp parallel for private (j)
for (int i = 0 ; i < nymax ; ++i) {
  for (j = 0 ; j < nxmax ; ++j ) {
    ...
    #pragma omp critical
    subset[i][j] = result ;
  }
}
```

Fortran:

```
!$OMP PARALLEL DO private (j)
      DO i = 0, nymax - 1
        DO j = 0, nxmax - 1
          ...
!$OMP CRITICAL
          subset(i,j) = result
        ENDDO
      ENDDO
```

→ `critical` forces threads to queue, hence slows down execution, better: if possible, use `reduction` clause:

OMP – critical and reduction IV

`#pragma omp parallel reduction (`*`operator: list of variables`*`)`

The `reduction` clause defines corresponding (scalar) variables in a parallel section.

Example: summing up with `reduction`

C++:

```
#pragma omp parallel for \
 private(x) reduction(+:sum_this)
for (int i = 1; i <= nmax ; i++) {
  x = 0.01 / (i + 0.5) ;
  sum_this += x ;
}
```

Fortran:

```
!$OMP PARALLEL DO PRIVATE(x)
!$    > REDUCTION(+:sum_this)
      DO i = 1, nmax
        x = 0.01 / (i + 0.5)
        sum_this = sum_this + x
      ENDDO
```

There are a number of allowed operators for reduction, e.g.:

operator	meaning	data type	neutral element / initial value
`+`,`-`	sum	`int`, `float`	0
`*`	product	`int`, `float`	1
`&`	bitwise and	`int`	all bits 1

Syntax II

- Heads up! OpenMP needs clear syntax for loop parallelization:

```
for (int i = 0 ; i < n ; i++)
```

make sure that your loop has *canonical loop form*, especially the loop iteration variable (here: `i`) is integer as well as variables used for comparison (here: `n`). OpenMP is very picky and might otherwise (e.g., if n is `float`) stop compilation:

```
error: invalid controlling predicate.
```

- Note that `omp parallel for` / `OMP PARALLEL DO` is the contracted form of

C++:

```
#pragma omp parallel
{
 #pragma omp for
 for ( ... ) {
  ...
 }
}
```

Fortran:

```
!$OMP PARALLEL
!$OMP DO
      ...
!$OMP END DO
!$OMP END PARALLEL
```

OMP – Scheduling I

`schedule(runtime)`

Examples:

`#pragma omp parallel for schedule (runtime)`

→ way of distributing the `parallel` section to threads is defined at runtime, e.g., by (bash)

`export OMP_SCHEDULE "dynamic,1"`

→ each thread gets a *chunk* of size 1 (e.g., one iteration) as soon as it is ready

`export OMP_SCHEDULE "static"`

→ the parallel section (e.g., loop iterations) is divided by the number of threads (e.g., 4)
and each thread gets a chunk of the same size

→ static is the default

OMP – Performance and infos

Useful for performance measurement:

`omp_get_wtime()` // $\rightarrow$ returns the so-called *wall clock time* (not the cpu time)

`omp_get_thread_num()` // $\rightarrow$ returns the number of the current thread

Weblinks:

`http://www.openmp.org/`

especially the documentation of the specifications:

`http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf`

Matrices and Linear Algebra

Matrices in physics I

Methods to solve matrix problems (e.g., inversion) useful for ODEs and PDEs, e.g., eigenvalue problem or radiative transfer with Feautrier scheme

Example: Vibrational spectrum of a molecule 1

n degrees of vibrational freedom $\rightarrow$ potential energy

$$U(q_1, q_2, \ldots, q_n) \simeq \frac{1}{2} \sum_{j,k}^{n} A_{jk} q_j q_k \tag{597}$$

in generalized coordinates around equilibrium state up to 2nd order term, coupling/potential parameter A_{jk} (e.g., spring constant).
Kinetic energy with generalized mass M_{jk}

$$T(\dot{q}_1, \dot{q}_2, \ldots, \dot{q}_n) \simeq \frac{1}{2} \sum_{j,k}^{n} M_{jk} \dot{q}_j \dot{q}_k \tag{598}$$

Matrices in physics II

Example: Vibrational spectrum of a molecule 2

Apply Lagrange equation of 2nd kind

$$\frac{\partial \mathcal{L}}{\partial q_j} - \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_j} = 0 \qquad \text{with } \mathcal{L} = T - U \tag{599}$$

$$\text{Hence, equations of motion, for } k = 1, \ldots, n\text{:} \quad \sum_{j=1}^{n} (A_{jk} q_j + M_{jk} \ddot{q}_j) = 0 \tag{600}$$

Assume an oscillatory motion $q_j = x_j\, e^{\imath\omega t} \rightarrow \frac{d^2}{dt^2}(x_j\, e^{\imath\omega t}) = -x_j \omega^2\, e^{\imath\omega t}$

$$\rightarrow \sum_{j=1}^{n} (A_{jk} - M_{jk}\omega^2) x_j = 0 \quad \text{or with } k = 1, \ldots, n \;: \quad \boldsymbol{A}\boldsymbol{x} = \omega^2 \boldsymbol{M}\boldsymbol{x} \tag{601}$$

set of linear homogenous equations. Nontrivial solution $\rightarrow$ determinant of coefficient matrix $\overset{!}{=} 0$
$\rightarrow \omega_k = \sqrt{\lambda_k}$ $(k = 1, \ldots, n)$ from equation

$$\det(\boldsymbol{A} - \lambda \boldsymbol{M}) = 0 \tag{602}$$

Matrix operations I

Matrix $\boldsymbol{A}$ with elements A_{ij} and $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n \to m \times n$ matrix.

$$\begin{array}{cc} & n \text{ columns} \rightarrow \\ \begin{array}{c} m \\ \text{rows} \\ \downarrow \\ \end{array} & \begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1n} \\ A_{21} & \ldots & & \\ \ldots & & & \\ A_{m1} & & & A_{mn} \end{pmatrix} \end{array}$$

If $m = n \to$ <u>square matrix</u>

Remember: Computer stores array in memory sequentially (1d), for C/C++ stored by <u>rows</u> (last index runs first)

$$A_{11}, A_{12}, \ldots, A_{1n}, A_{21}, \ldots, A_{mn} \tag{603}$$

whereas for Fortran stored by <u>column</u> (first index runs first):

$$A_{11}, A_{21}, \ldots, A_{m1}, A_{12}, \ldots, A_{mn} \tag{604}$$

Matrix operations II

Variable array $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$: $n \times 1$ matrix. Hence set of linear equations for $i = 1, 2, \ldots, n$, where x_i is unknown:

$$A_{i1}\,x_1 + A_{i2}\,x_2 + \ldots + A_{in}\,x_n = b_i \tag{605}$$

with coefficients A_{ij} and constants b_i, so express Eq. (605) in matrix form

$$\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b} \tag{606}$$

with $\boldsymbol{A}\,\boldsymbol{x}$ from standard matrix multiplication for $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$, i.e.

$$C_{ij} = \sum_k A_{ik}\,B_{kj} \tag{607}$$

(number of columns of $\boldsymbol{A} \overset{!}{=}$ number of rows of $\boldsymbol{B}$)

Matrix operations III

Example: Population numbers from statistical equilibrium (non-LTE)

"inflow" to level n_j (from all other levels) balanced by "outflow" from level n_j (to all other levels)

$$\sum_{\substack{i=1\\i\neq j}}^{N} n_i P_{ij} = \sum_{\substack{i=1\\i\neq j}}^{N} n_j P_{ji} \quad \forall\, j = 1, \ldots, N \tag{608}$$

$$\boldsymbol{n}\,\boldsymbol{P} = 0 \quad \text{with } P_{ii} := -\sum_{j\neq i} P_{ij} \tag{609}$$

Remember definitions: Inverse of a matrix $\boldsymbol{A}$ is $\boldsymbol{A}^{-1}$:

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I} \tag{610}$$

with $I_{ij} = \delta_{ij}$.
The transpose of a matrix $\boldsymbol{A}^T$ is with column and row indices of $\boldsymbol{A}$ interchanged

$$A_{ij}^T = A_{ji} \tag{611}$$

Matrix operations IV

Trace of $\boldsymbol{A}$ ($\mathrm{Tr}\,\boldsymbol{A}$) is summation of diagonal elements of $\boldsymbol{A}$

$$\mathrm{Tr}\,\boldsymbol{A} = \sum_{i=1}^{n} A_{ii} \tag{612}$$

The determinant of square matrix $\boldsymbol{A}$

$$\det(\boldsymbol{A}) = \sum_{i=1}^{n} (-1)^{i+j} A_{ij} \det(\boldsymbol{R}_{ij}) \tag{613}$$

where $\boldsymbol{R}_{ij}$ is residual matrix of $\boldsymbol{A}$ with ith row and jth column removed ($\rightarrow$ recursive computation)

$$\text{e.g.,}\ \det \begin{pmatrix} A_{11} & A12 \\ A_{21} & A_{22} \end{pmatrix} = A_{11}A_{22} - A_{12}A_{21} \tag{614}$$

Matrix operations V

Important properties of the determinant:

- Determinant of a 1×1 matrix = element itself.
- Determinant of a triangular matrix (lower or upper) is the product of diagonal elements: $\det(\boldsymbol{A}) = \prod_{i=1}^{n} A_{ii}$
- $\det(\boldsymbol{B}\boldsymbol{A}) = \det(\boldsymbol{B}) \cdot \det(\boldsymbol{A})$ (if both $n \times n$)
- $\det(\boldsymbol{A}^{-1}) = \frac{1}{\det(\boldsymbol{A})} \rightarrow$ integer entries for $\boldsymbol{A}$ and $\boldsymbol{A}^{-1} \Leftrightarrow \det(\boldsymbol{A}) = \pm 1$
- $\det(\boldsymbol{A}^T) = \det(\boldsymbol{A})$
- The determinant is an *n-linear function* of the n columns (rows). It is moreover an *alternating form*. Together with $\det(\boldsymbol{A}^T) = \det(\boldsymbol{A})$, this means:
  Interchanging any pair of columns or rows of a matrix multiplies its determinant by -1.

Inverse of $\boldsymbol{A}$ via (Cramer's rule)

$$A_{ij}^{-1} = (-1)^{i+j} \frac{\det(\boldsymbol{R}_{ij})}{\det(\boldsymbol{A})} \tag{615}$$

$\rightarrow$ if $\boldsymbol{A}^{-1}$ exists or $\det(\boldsymbol{A}) \neq 0 \rightarrow$ nonsingular matrix, singular otherwise ().

Matrix operations VI

Examples for singular / non-singular (=regular) matrices:

- the matrix

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} \tag{616}$$

  is non-singular, its determinant is $\det(\boldsymbol{A}) = -1$ and its inverse is

$$\boldsymbol{A}^{-1} = \begin{pmatrix} -3 & 2 \\ 2 & -1 \end{pmatrix} \tag{617}$$

Matrix operations VII

- the matrix

$$\boldsymbol{B} = \begin{pmatrix} 1 & 2 \\ 0 & 0 \end{pmatrix} \tag{618}$$

  is singular, its determinant is $\det(\boldsymbol{A}) = 0$ and there exists no inverse

$$\boldsymbol{B} \cdot \boldsymbol{M} = \begin{pmatrix} 1 & 2 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1a + 2c & 1b + 2d \\ 0 & 0 \end{pmatrix} \neq \boldsymbol{I} \tag{619}$$

- the matrix

$$\boldsymbol{C} = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} \tag{620}$$

  is singular, its determinant is $\det(\boldsymbol{A}) = 0$, as two of its lines are linearly dependent

Matrix operations VIII

Moreover, it can be useful to perform the following transformations, represented by a matrix multiplications : $\boldsymbol{A}' = \boldsymbol{M}\boldsymbol{A}$

1. interchanging two rows i and j, elements: $M_{ij} = 1; M_{ji} = 1; M_{kk} = 1$ for $k \neq i, j$ other elements $= 0 \rightarrow \det(\boldsymbol{M}\,\boldsymbol{A}) = -\det(\boldsymbol{A})$
2. multiply one row by λ: $M_{kk} = 1$ for $k \neq i; M_{ii} = \lambda \neq 0$, all other elements $= 0$ $\rightarrow \det(\boldsymbol{M}\,\boldsymbol{A}) = \det(\boldsymbol{M})\det(\boldsymbol{A}) = \lambda\det(\boldsymbol{A})$
3. add a row (or column) to another row (or column) multiplied by a factor λ: $M_{ii} = 1,\ M_{ij} = \lambda,\ M_{kl} = 0.$ This can be also be written as

$$A'_{ij} = A_{ij} + \lambda A_{kj} \qquad \text{for } j = 1, 2, \ldots, n \tag{621}$$

   and i and k are row indices, which can be the same. The determinant is preserved $\det(\boldsymbol{A}') = \det(\boldsymbol{A})$.

$\rightarrow$ see below for Gaussian elimination and matrix decomposition

Eigenvalue problems I

The matrix eigenvalue problem is for a given matrix $\boldsymbol{A}$

$$\boldsymbol{A}\,\boldsymbol{x} = \lambda\boldsymbol{x} \tag{622}$$

with eigenvector $\boldsymbol{x}$ and corresponding eigenvalue λ of the matrix.
Also for the example of the vibrating molecules:

$$\boldsymbol{A}\boldsymbol{x} = \omega^2\boldsymbol{M}\boldsymbol{x} \quad \Big| \quad \boldsymbol{B} := \boldsymbol{M}^{-1}\boldsymbol{A} \tag{623}$$

$$\rightarrow \; \boldsymbol{B}\boldsymbol{x} = \omega^2\boldsymbol{x} \tag{624}$$

Eigenvalue problems II

$\rightarrow$ Matrix eigenvalue problem = linear equation set problem
$\rightarrow$ e.g., iterative solution

$$\boldsymbol{A}\,\boldsymbol{x}_{n+1} = \lambda_n \boldsymbol{x}_n \tag{625}$$

Moreover, the eigenvalues are preserved under a similarity transformation with a non-singular matrix $\boldsymbol{S}$

$$\boldsymbol{B} = \boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S} \tag{626}$$

$$\rightarrow\ \boldsymbol{B}\,\boldsymbol{y} = \lambda \boldsymbol{y} \quad \Leftrightarrow \quad \boldsymbol{A}\,\boldsymbol{x} = \lambda \boldsymbol{x} \qquad \text{for } \boldsymbol{x} = \boldsymbol{S}\boldsymbol{y} \tag{627}$$

$$\rightarrow\ \det(\boldsymbol{B}) = \det(\boldsymbol{A}) = \prod_{i=1}^{n} \lambda_i \tag{628}$$

$\rightarrow$ computation of eigenvalues & eigenvectors usually complicated ...

Systems of linear equations – Direct methods I

The general problem:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{629}$$

where matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$ given and vector $\boldsymbol{x}$ unknown.

Straightforward solutions:

- Cramer's rule:

$$x_i = \frac{\det(\boldsymbol{A}_i)}{\det(\boldsymbol{A})} \tag{630}$$

  where in $\boldsymbol{A}_i$ the i-th column is replaced by $\boldsymbol{b}$
  $\rightarrow$ for a system of n equations: need to compute $n+1$ determinants, each of order n (see above), i.e., compute $n!$ terms each with $(n-1)$ multiplications
  $\rightarrow (n+1) \times n! \times (n-1)$ multiplications,
  e.g., for $n = 20 \rightarrow 10^{21}$ multiplications and for a computer with, e.g., 10 TFLOPS
  $\rightarrow t \approx 3\,\mathrm{a}$ only for multiplications (also note large accumulation of roundoff error)

Systems of linear equations – Direct methods II

- find the inverse $\boldsymbol{A}^{-1}$

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} \tag{631}$$

$\rightarrow$ also time-consuming and instable, e.g., ($n = 1$, float)

$$7x = 21 \tag{632}$$
$$x = \frac{21}{7} = 3 \quad \text{(direct division)} \tag{633}$$
$$x = (7^{-1})(21) \quad \text{(compute inverse)} \tag{634}$$
$$= (.142857)(21) = 2.999997 \quad \text{(less accurate)} \tag{635}$$

computation of the inverse, e.g., via Cramer's rule (see above) or

Systems of linear equations – Direct methods III

with *Gauß-Jordan elimination* (see below) for system $\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}$:

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} \hat{a}_{11} & \dots & \hat{a}_{1n} \\ \vdots & & \vdots \\ \hat{a}_{n1} & \dots & \hat{a}_{nn} \end{pmatrix} = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix} \tag{636}$$

hence, the j-th column of the inverse $\hat{a}_j = (\hat{a}_{1j}, \hat{a}_{2j}, \dots, \hat{a}_{nj})^T$ is solution of the system of linear equations

$$A \cdot \hat{a}_j = e_j \tag{637}$$

These equations are solved simultaneously by extending matrix $\boldsymbol{A}$ with $\boldsymbol{I}$:

$$(\,A\,|\,I\,) = \left(\begin{array}{ccc|ccc} a_{11} & \dots & a_{1n} & 1 & & 0 \\ \vdots & & \vdots & & \ddots & \\ a_{n1} & \dots & a_{nn} & 0 & & 1 \end{array}\right) \tag{638}$$

Systems of linear equations – Direct methods IV

$\rightarrow$ elementary row operations $\rightarrow$ matrix $\boldsymbol{A}$ into upper triangular form (forward elimination)

$$(\, D \,|\, B \,) = \left(\begin{array}{ccc|ccc} * & \dots & * & * & \dots & * \\ & \ddots & \vdots & \vdots & & \vdots \\ 0 & & * & * & \dots & * \end{array} \right) \tag{639}$$

$\rightarrow$ if no zeros on diagonal $\rightarrow$ invertible, bring into diagonal form:

$$(\, I \,|\, A^{-1} \,) = \left(\begin{array}{ccc|ccc} 1 & & 0 & \hat{a}_{11} & \dots & \hat{a}_{1n} \\ & \ddots & & \vdots & & \vdots \\ 0 & & 1 & \hat{a}_{n1} & \dots & \hat{a}_{nn} \end{array} \right) \tag{640}$$

or compute inverse with *characteristic polynomial*:

$$A^{-1} = \frac{-1}{\det(A)} \left(\alpha_1 I_n + \alpha_2 A + \ldots + \alpha_n A^{n-1} \right) \tag{641}$$

where the coefficients of the chracteristical polynomial of $\boldsymbol{A}$ can be obtained from $\chi(t) = \det(t\boldsymbol{I} - \boldsymbol{A}) = \alpha_0 + \alpha_1 \cdot t^1 + \ldots + \alpha_n \cdot t^n$

Gaussian elimination I

Matrix problems can be easily solved for an upper (lower) triangular matrix, for which elements below (above) the diagonal = 0,

$$\begin{pmatrix} R_{11} & R_{12} & \dots & R_{1n} \\ 0 & R_{22} & \dots & R_{2n} \\ & & \ddots & \\ 0 & 0 & \dots & R_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_n \end{pmatrix} \tag{642}$$

via backward (forward) substitution, i.e. starting with $x_n = c_n/R_{nn}$ and

$$x_i = \frac{c_i - \sum_{j=i+1}^{n} R_{ij}x_j}{R_{ii}} \qquad \text{for } i = n-1, \dots, 1 \tag{643}$$

$\rightarrow$ need algorithms for transformation into triangular form

Gaussian elimination II

Gaussian elimination

1. *Forward elimination:* Transform linear equation set $\boldsymbol{Ax} = \boldsymbol{b}$ by a sequence of matrix operations j from original matrix $\boldsymbol{A} = \boldsymbol{A}^{(0)}$ to $\boldsymbol{A}^{(j)}$, hence after $n-1$ steps for a $n \times n$ matrix

$$\boldsymbol{A}^{(n-1)}\boldsymbol{x} = \boldsymbol{b}^{(n-1)} \tag{644}$$

where $A_{ij}^{(n-1)} = 0$ for $i > j$:

1. multiply 1st equation (1st row $\boldsymbol{A}$ and $b_1^{(0)}$) by $-A_{i1}^{(0)}/A_{11}^{(0)}$ and add to ith equation (row) for $i > 1 \rightarrow$ 1st element of every row except 1st row eliminated $\rightarrow \boldsymbol{A}^{(1)}$
2. multiply 2nd equation by $-A_{i2}^{(1)}/A_{22}^{(1)}$ and add to ith equation for $i > 2 \rightarrow$ 2nd element of every row except 1st & 2nd row eliminated $\rightarrow \boldsymbol{A}^{(2)}$
3. ...
4. upper triangular matrix $\boldsymbol{A}^{(n-1)}$

2. *backward substitution* according to Eq. (643)

Gaussian elimination III

ad 1.: all diagonal elements A_{jj} are used in denominators $-A_{ij}^{(j-1)}/{A_{jj}}^{(j-1)}$
$\rightarrow$ problems if diagonal elements $= 0$ or ≈ 0
Solution: pivoting (from french pivot=center of rotation) $\rightarrow$ interchange rows/columns to put always largest (absolut value) element on diagonal
full pivoting: interchange columns and rows, need to keep track of order ...
partial pivoting: only search for pivot in remaining elements of the current column (swap rows only)
$\rightarrow$ partial pivoting usually good compromise between speed and accuracy
$\rightarrow$ use index to record order of pivot elements instead of physically interchanging
$\rightarrow$ *rescaling*: rescale all elements from a row by its largest element before comparing to find pivot (reduces rounding errors)

Gaussian elimination IV

Example: Gaussian elimination in Fortran - code sniplet

```fortran
! partial pivot. Gaussian elimin.
DIMENSION A(N,N),INDX(N),C(N)
DO I = 1, N
 INDX(I) = I ! init. index
 C1 = 0.0
 DO J = 1, N ! rescale coeff.
  C1 = AMAX1(C1,ABS(A(I,J)))
 ENDDO
 C(I) = C1
ENDDO

DO J = 1, N-1 ! search pivots
 PI1 = 0.0
 DO I = J, N
  PI = ABS(A(INDX(I),J)) / C(INDX(I))
  IF (PI.GT.PI1) THEN
   PI1 = PI
   K   = I
  ENDIF
 ENDDO
 ITMP = INDX(J)
 INDX(J) = INDX(K)
 INDX(K) = ITMP
 DO I = J + 1, N ! elimin. subdiagonal
  PJ = A(INDX(I),J) / A(INDX(J),J)
  A(INDX(I),J) = PJ
  DO L = J + 1, N
   A(INDX(I),L) = A(INDX(I),L) - &
   PJ * A(INDX(J),L)
  ENDDO
 ENDDO
ENDDO
```

Gaussian elimination V

Example: Gaussian elimination by hand I

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ 6 \end{pmatrix} \tag{645}$$

1.) eliminate x_1 from row 2 & 3 $\rightarrow$ add $3/10 = 0.3\times$ 1st row to 2nd row & add $-5/10 = -0.5\times$ 1st row to 3rd row:

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6.1 \\ 2.5 \end{pmatrix} \tag{646}$$

2.) eliminate x_2 from row 3 $\rightarrow$ a) pivoting: interchange row 2 & 3 so that coefficient of x_2 in row 2 is largest (because of roundoff errors $\rightarrow$ only for computers necessary)

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.1 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 2.5 \\ 6.1 \end{pmatrix} \tag{647}$$

Gaussian elimination VI

Example: Gaussian elimination by hand II

2.b) now add $0.1/2.5 = 0.04\times$ 2nd row to 3rd row:

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 2.5 \\ 6.2 \end{pmatrix} \tag{648}$$

Finally: backward substitution, starting with last row:

$$6.2\, x_3 = 6.2 \rightarrow x_3 = 1 \tag{649}$$

$$2.5\, x_2 + 5 \cdot 1 = 2.5 \rightarrow x_2 = -1 \tag{650}$$

$$10\, x_1 + (-7) \cdot (-1) + 0 = 7 \quad \rightarrow x_1 = 0 \tag{651}$$

This can be also expressed in matrix notation: Let

$$\boldsymbol{M}_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0.3 & 1 & 0 \\ -0.5 & 0 & 1 \end{pmatrix} \rightarrow \boldsymbol{M}_1 \boldsymbol{A} = \begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{pmatrix}, \quad \boldsymbol{M}_1 \boldsymbol{b} = \begin{pmatrix} 7 \\ 6.1 \\ 2.5 \end{pmatrix} \tag{652}$$

Gaussian elimination VII

Example: Gaussian elimination by hand III

Let then

$$\boldsymbol{P}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \quad \boldsymbol{M}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0.04 & 1 \end{pmatrix} \tag{653}$$

$$\rightarrow \boldsymbol{M}_2\boldsymbol{P}_2\boldsymbol{M}_1\boldsymbol{A} = \begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix} = \boldsymbol{U}, \quad \boldsymbol{M}_2\boldsymbol{P}_2\boldsymbol{M}_1\boldsymbol{b} = \begin{pmatrix} 7 \\ 2.5 \\ 6.2 \end{pmatrix} = \boldsymbol{c} \tag{654}$$

Hence $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{c}$, with upper triangular matrix $\boldsymbol{U}$.
The matrices $\boldsymbol{P}_k, k = 1, \dots, n-1$ are the permutations matrices, inferred from the identity matrix $\boldsymbol{I}$ by interchanging rows in same way as for $\boldsymbol{A}$ in the kth step, and $\boldsymbol{M}_k$ is multiplication matrix, inferred from identy matrix by inserting mulitpliers used in kth step below diagonal in kth column $\rightarrow \boldsymbol{M}_k$ are lower triangular matrices

$$\boldsymbol{M} := \boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1}\dots\boldsymbol{M}_1\boldsymbol{P}_1 \tag{655}$$

$$\boldsymbol{U} = \boldsymbol{M}\boldsymbol{A} \quad \text{(“triangular decomposition” of } \boldsymbol{A}\text{)} \tag{656}$$

LU decomposition I

More general approach: decompose *nonsingular* matrix $\boldsymbol{A}$ into two triangular matrices

$$\boldsymbol{A} = \boldsymbol{L}\,\boldsymbol{U} \tag{657}$$

with lower (left) triangular matrix $\boldsymbol{L}$ and upper (right) triangular matrix $\boldsymbol{U}$ (or $\boldsymbol{R}$), hence

$$\begin{aligned} \boldsymbol{Ax} &= \boldsymbol{LUx} = \boldsymbol{b} && (658)\\ \rightarrow \text{ first, solve 1. } \boldsymbol{Ly} &= \boldsymbol{b} \rightarrow \ \boldsymbol{y} && (659)\\ \text{then 2. } \boldsymbol{Ux} &= \boldsymbol{y} \rightarrow \ \boldsymbol{x} && (660) \end{aligned}$$

i.e. once $\boldsymbol{A} = \boldsymbol{L}\,\boldsymbol{U}$ obtained $\rightarrow$ easy to solve for *any* b.
More general case: re-order matrix $\boldsymbol{A}$ by, e.g., row-permutations (partial pivoting):

$$\begin{aligned} \boldsymbol{PA} &= \boldsymbol{LU}, \ \text{then} && (661)\\ \boldsymbol{LUx} &= \boldsymbol{Pb} && (662)\\ 1.\ \boldsymbol{Ly} &= \boldsymbol{Pb} \rightarrow \ \boldsymbol{y} && (663)\\ 2.\ \boldsymbol{Ux} &= \boldsymbol{y} \rightarrow \ \boldsymbol{x} && (664) \end{aligned}$$

LU decomposition II

e.g. $\rightarrow$ Crout's method

start with $L_{i1} = A_{i1}$ and $U_{1j} = A_{1j}/A_{11}$, then recursively:

$$L_{ij} = A_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj} \tag{665}$$

$$U_{ij} = \frac{1}{L_{ii}} \left(A_{ij} - \sum_{k=1}^{i-1} L_{ik} U_{kj} \right) \tag{666}$$

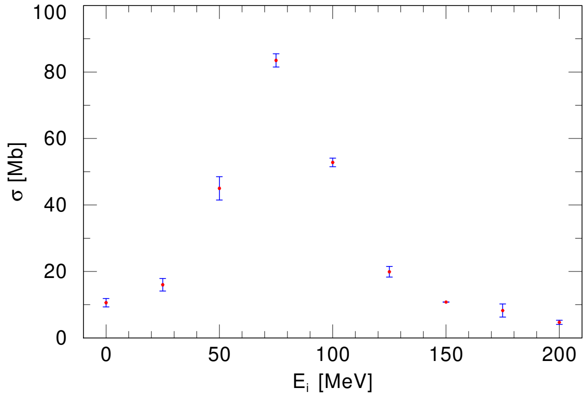
Usually no need to implement by yourself, instead use libraries, e.g., `LINPACK`:

- `DGEFA` performs LU decomposition by Gaussian elimination
- `DGESL` uses that decomposition to solve the given system of linear equations
- `DGEDI` uses decomposition to compute inverse of a matrix

Application: Interpolating data I

Remember following measurement of a cross section

E_i [MeV]	0	25	50	75	100	125	150	175	200
$\sigma(E_i)$ [Mb]	10.6	16.0	45.0	83.5	52.8	19.9	10.8	8.25	4.7
$\sigma_{\sigma(E_i)}$ [Mb]	1.26	1.9	3.5	2.0	1.3	1.6	0.04	1.96	0.61



The cross section can be described by Breit-Wigner formula

$$f(E) = \frac{f_r}{(E - E_r)^2 + \Gamma^2/4} \qquad (667)$$

Application: Interpolating data II

Interpolation problem

We want to determine $\sigma(E)$ for values of E which lie between measured values of E

By

- numerical interpolation (assumption of data representation by polynomial in E):
  $\rightarrow$ see previous lectures
  $\rightarrow$ ignores errors in measurement (noise)
- fitting parameters of an underlying model, e.g., Breit-Wigner with f_r, E_r, Γ, (taking errors into account), i.e., minimizing χ^2
- Fourier analysis (next semester lecture)

Least square fitting I

Already seen for linear regression:
We have N_D data points

$$(x_i, y_i \pm \sigma_i) \quad i = 1, \ldots, N_D \tag{668}$$

and a function $y = g(x)$ (=model) with parameters $\{a_m\}$; fit function to data, such that $\chi^2 = min$:

$$\chi^2 := \sum_{i=1}^{N_D} \left(\frac{y_i - g(x_i; \{a_m\})}{\sigma_i} \right)^2 \tag{669}$$

i.e. for M_P parameters $\{a_m, m = 1 \ldots M_P\}$

$$\frac{\partial \chi^2}{\partial a_m} \overset{!}{=} 0 \Rightarrow \sum_{i=1}^{N_D} \frac{[y_i - g(x_i)]}{\sigma_i^2} \frac{\partial g(x_i)}{\partial a_m} = 0 \quad (m = 1, \ldots, M_P) \tag{670}$$

$\rightarrow$ solve M_P equations, usually nonlinear in a_m

Least square fitting II

goodness of fit, assumptions

- deviations to model only due to random errors
- Gaussion distribution of errors

$\rightarrow$ then, fit is good when $\chi^2 \approx N_D - M_P$ (degrees of freedom)

- if $\chi^2 \ll N_D - M_P \rightarrow$ probably too many parameters or errors σ_i to large (fitting random scatter)
- if $\chi^2 \gg N_D - M_P \rightarrow$ model not good or underestimated errors or non-random errors

$\rightarrow$ for linear fit see above

Least square fitting III

Non-linear fit

remember Breit-Wigner resonance formula Eq. (667)

$$f(E) = \frac{f_\mathrm{r}}{(E - E_\mathrm{r})^2 + \Gamma^2/4} \tag{671}$$

$\rightarrow$ determine f_r, E_r, Γ
$\rightarrow$ nonlinear equations in the parameters

$$a_1 = f_\mathrm{r} \qquad a_2 = E_\mathrm{r} \qquad a_3 = \Gamma^2/4 \tag{672}$$

$$\Rightarrow g(x) = \frac{a_1}{(x-a_2)^2 + a_3} \tag{673}$$

$$\frac{\partial g}{\partial a_1} = \frac{1}{(x-a_2)^2 + a_3}, \quad \frac{\partial g}{\partial a_2} = \frac{-2a_1(x-a_2)}{[(x-a_2)^2 + a_3]^2}, \quad \frac{\partial g}{\partial a_3} = \frac{-a_1}{[(x-a_2)^2 + a_3]^2} \tag{674}$$

Least square fitting IV

Insert into Eq. (670):

$$\sum_{i=1}^{9} \frac{y_i - g(x_i, a)}{(x_i - a_2)^2 + a_3} = 0 \qquad \sum_{i=1}^{9} \frac{[y_i - g(x_i, a)](x_i - a_2)}{[(x_i - a_2)^2 + a_3]^2} = 0$$
$$\sum_{i=1}^{9} \frac{y_i - g(x_i, a)}{[(x_i - a_2)^2 + a_3]^2} = 0 \tag{675}$$

$\rightarrow$ *three* nonlinear equations for unknown a_1, a_2, a_3, i.e. cannot be solved by linear algebra but can be solved with help of Newton-Raphson method, i.e. find the roots for the equations above

$$f_i(a_1, \ldots, a_M) = 0 \qquad i = 1, \ldots, M \tag{676}$$

Least square fitting V

So

$$f_1(a_1, a_2, a_3) = \sum_{i=1}^{9} \frac{y_i - g(x_i, a)}{(x_i - a_2)^2 + a_3} = 0 \tag{677}$$

$$f_2(a_1, a_2, a_3) = \sum_{i=1}^{9} \frac{[y_i - g(x_i, a)](x_i - a_2)}{[(x_i - a_2)^2 + a_3]^2} = 0 \tag{678}$$

$$f_3(a_1, a_2, a_3) = \sum_{i=1}^{9} \frac{y_i - g(x_i, a)}{[(x_i - a_2)^2 + a_3]^2} = 0 \tag{679}$$

with intial guesses for a_1, a_2, a_3.

Least square fitting VI

Newton-Raphson method for a system of nonlinear equations
Remember for 1dim Newton-Raphson method, correction for Δx:

$$f(x_0) + f'(x_0) \cdot \Delta x \overset{!}{=} 0 \tag{680}$$

$$\Delta x = -\frac{f(x_0)}{f'(x_0)} \tag{681}$$

For our system of equations $f_i(a_1, \ldots, a_M) = 0$, we assume that for our approximation (intial guess) $\{a_i\}$ corrections $\{\Delta x_i\}$ exist so that

$$f_i(a_1 + \Delta a_1, a_2 + \Delta a_2, a_3 + \Delta a_3) = 0 \qquad i = 1, 2, 3 \tag{682}$$

$\rightarrow$ linear approximation (two terms of Taylor series):

$$f_i(a_1 + \Delta a_1, \ldots) \simeq f_i(a_1, a_2, a_3) + \sum_{j=1}^{3} \frac{\partial f_i}{\partial a_j} \Delta a_j = 0 \qquad i = 1, 2, 3 \tag{683}$$

$\rightarrow$ set of 3 linear equations in 3 unknowns

Least square fitting VII

as explicit equations:

$$f_1 + \partial f_1/\partial a_1 \Delta a_1 + \partial f_1/\partial a_2 \Delta a_2 + \partial f_1/\partial a_3 \Delta a_3 = 0 \quad (684)$$

$$f_2 + \partial f_2/\partial a_1 \Delta a_1 + \partial f_2/\partial a_2 \Delta a_2 + \partial f_2/\partial a_3 \Delta a_3 = 0 \quad (685)$$

$$f_3 + \partial f_3/\partial a_1 \Delta a_1 + \partial f_3/\partial a_2 \Delta a_2 + \partial f_3/\partial a_3 \Delta a_3 = 0 \quad (686)$$

Or as single matrix equation:

$$\begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} + \begin{pmatrix} \partial f_1/\partial a_1 & \partial f_1/\partial a_2 & \partial f_1/\partial a_3 \\ \partial f_2/\partial a_1 & \partial f_2/\partial a_2 & \partial f_2/\partial a_3 \\ \partial f_3/\partial a_1 & \partial f_3/\partial a_2 & \partial f_3/\partial a_3 \end{pmatrix} \begin{pmatrix} \Delta a_1 \\ \Delta a_2 \\ \Delta a_3 \end{pmatrix} = 0 \quad (687)$$

Or in matrix notation

$$\mathrm{f} + \mathrm{F}' \boldsymbol{\Delta a} = 0 \Rightarrow \mathrm{F}' \boldsymbol{\Delta a} = -\boldsymbol{f} \quad (688)$$

Where we want to solve for $\boldsymbol{\Delta a}$ (the corrections)
Matrix $\boldsymbol{F}'$ sometimes written as $\boldsymbol{J}$ is called the *Jacobian* matrix (with entries $f'_{ij} = \partial f_i/\partial a_j$).

Least square fitting VIII

Equation $\boldsymbol{F}'\Delta a = -\boldsymbol{f}$ corresponds to standard form $\boldsymbol{Ax} = \boldsymbol{b}$ for systems of linear equations. Formally solution obtained by multiplying with inverse of $\boldsymbol{F}'$

$$\boldsymbol{\Delta a} = -\boldsymbol{F}'^{-1}\boldsymbol{f} \tag{689}$$

$\rightarrow$ inverse must exist for unique solution
$\rightarrow$ same form as for 1d Newton-Raphson: $\Delta x = -(1/f')f$
$\rightarrow$ iterate as for 1d Newton-Raphson till $\boldsymbol{f} \approx 0$

compute derivatives for the system numerically

$$f'_{ij} = \frac{\partial f_i}{\partial a_j} \simeq \frac{f_i(a_j + \Delta a_j) - f_i(a_j)}{\Delta a_j} \tag{690}$$

with Δa_j sufficiently small, e.g., 1% of a

Least square fitting IX

Nonlinear fit with Newton-Raphson

In our nonlinear fit problem the Newton step

$$\mathsf{F}' \, \boldsymbol{\Delta a} = -\boldsymbol{f} \tag{691}$$

can be solved for $\boldsymbol{\Delta a}$ with help of `DGEFA` and `DGESL` (see p. 546):

```
CALL DGEFA(FPRIME, NDIM, NDIM, IPVT, INFO)
IF (INFO .NE. 0) STOP 'JACOBIAN MATRIX WITH 0 ON DIAGONAL'
CALL DGESL(FPRIME, NDIM, NDIM, IPVT, F)
```

where the solution $\boldsymbol{\Delta a}$ is written to vector `F`

Insertion: data analysis

Recursive mean I

arithmetic mean

$$\langle x \rangle = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{692}$$

Problem: calculation of the mean for measured data

- Eq. (692) must be evaluated again for every new data point
- for $n \gg 1$ and at the same time $x_i \ll 1$ numerical inaccuracy for strict use of Eq. (692) because of → saturation in x_i

→ hence: definition of the recursive mean:

$$\langle x \rangle_i = \frac{i-1}{i} \langle x \rangle_{i-1} + \frac{1}{i} x_i \tag{693}$$

Recursive mean II

proof:

$$i-1 : \langle x \rangle_{i-1} = \frac{x_1 + \ldots + x_{i-1}}{i-1} \quad (694)$$

$$i : \langle x \rangle_i = \frac{x_1 + \ldots + x_{i-1} + x_i}{i} \quad (695)$$

$$= \frac{(i-1)\frac{x_1 + \ldots x_{i-1}}{i-1} + x_i}{i} \quad (696)$$

$$= \frac{i-1}{i}\langle x \rangle_{i-1} + \frac{x_i}{i} \quad (697)$$

q.e.d.

Recursive mean III

analogously: **recursive variance**

$$\sigma_i^2 = \frac{i-1}{i}\sigma_{i-1}^2 + \frac{1}{i-1}(x_i - \langle x\rangle_i)^2 \tag{698}$$

proof similar as for recursive mean

correction of a single value:

$$\begin{aligned}\langle x\rangle_\mathrm{new} &= \langle x\rangle_\mathrm{old} + \frac{x_\mathrm{new} - x_\mathrm{old}}{n} \qquad (699)\\ \sigma_\mathrm{new}^2 &= \sigma_\mathrm{old}^2 + \frac{x_\mathrm{new}^2 - x_\mathrm{old}^2}{n} - \frac{x_\mathrm{new} - x_\mathrm{old}}{n}\left(\langle x\rangle_\mathrm{old} + \frac{x_\mathrm{new} - x_\mathrm{old}}{n}\right)\end{aligned}$$

proof for the correction of the mean:

$$\begin{aligned}\langle x\rangle_\mathrm{new} &= \frac{1}{n}\left(\sum_{i=1}^{n} x_i - x_\mathrm{old} + x_\mathrm{new}\right) = \qquad (700)\\ &= \langle x\rangle_\mathrm{old} + \frac{x_\mathrm{new} - x_\mathrm{old}}{n} \qquad \text{q.e.d.}\end{aligned}$$

Linear regression I

We already know

Straight line fit without errors

$$\begin{aligned} y &= b \cdot x + a && (701)\\ \text{with slope} \quad b &= \frac{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \langle x\rangle)(y_i - \langle y\rangle)}{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \langle x\rangle)^2} && (702)\\ \text{and} \quad a &= \langle y\rangle - b \cdot \langle x\rangle && (703) \end{aligned}$$

quality of fit $y = a + bx$ measured by χ^2:

$$\chi^2(a,b) = \sum_{i=1}^{n}\left(\frac{y_i - a - bx_i}{\sigma_i}\right)^2 \qquad (704)$$

with error σ_i in measuring of y_i (x_i exact)

Linear regression II

Best fit for χ^2 minimum, hence (see also *Numerical Recipes*)

$$0 \overset{!}{=} \frac{\partial \chi^2}{\partial a} = -2 \sum_{i=1}^{n} \frac{y_i - a - bx_i}{\sigma_i^2} \tag{705}$$

$$0 \overset{!}{=} \frac{\partial \chi^2}{\partial b} = -2 \sum_{i=1}^{n} \frac{x_i(y_i - a - bx_i)}{\sigma_i^2} \tag{706}$$

can be rewritten as system of equations:

$$a \sum_{i=1}^{n} \frac{1}{\sigma_i^2} + b \sum_{i=1}^{n} \frac{x_i}{\sigma_i^2} = \sum_{i=1}^{n} \frac{y_i}{\sigma_i^2} \tag{707}$$

$$a \sum_{i=1}^{n} \frac{x_i}{\sigma_i^2} + b \sum_{i=1}^{n} \frac{x_i^2}{\sigma_i^2} = \sum_{i=1}^{n} \frac{x_i y_i}{\sigma_i^2} \tag{708}$$

$$\tag{709}$$

Linear regression III

solution for the system:

$$a = \frac{\sum_{i=1}^{n} \frac{x_i^2}{\sigma_i^2} \sum_{i=1}^{n} \frac{y_i}{\sigma_i^2} - \sum_{i=1}^{n} \frac{x_i}{\sigma_i^2} \sum_{i=1}^{n} \frac{x_i y_i}{\sigma_i^2}}{\sum_{i=1}^{n} \frac{1}{\sigma_i^2} \sum_{i=1}^{n} \frac{x_i^2}{\sigma_i^2} - \left(\sum_{i=1}^{n} \frac{x_i}{\sigma_i^2}\right)^2} \tag{710}$$

$$b = \frac{\sum_{i=1}^{n} \frac{1}{\sigma_i^2} \sum_{i=1}^{n} \frac{x_i y_i}{\sigma_i^2} - \sum_{i=1}^{n} \frac{x_i}{\sigma_i^2} \sum_{i=1}^{n} \frac{y_i}{\sigma_i^2}}{\sum_{i=1}^{n} \frac{1}{\sigma_i^2} \sum_{i=1}^{n} \frac{x_i^2}{\sigma_i^2} - \left(\sum_{i=1}^{n} \frac{x_i}{\sigma_i^2}\right)^2} \tag{711}$$

Linear regression IV

errors in a and b from error propagation for a quantity f:

$$\sigma_f^2 = \sum_{i=1}^{n} \sigma_i^2 \left(\frac{\partial f}{\partial y_i}\right)^2 \tag{712}$$

where
$$\frac{\partial a}{\partial y_i} = \frac{\sum_{i=1}^{n} \frac{x_i^2}{\sigma_i^2} - x_i \sum_{i=1}^{n} \frac{x_i}{\sigma_i^2}}{\sigma_i^2 \left(\sum_{i=1}^{n} \frac{1}{\sigma_i^2} \sum_{i=1}^{n} \frac{x_i^2}{\sigma_i^2} - \left(\sum_{i=1}^{n} \frac{x_i}{\sigma_i^2}\right)^2\right)} \tag{713}$$

$$\frac{\partial b}{\partial y_i} = \frac{x_i \sum_{i=1}^{n} \frac{1}{\sigma_i^2} - \sum_{i=1}^{n} \frac{x_i}{\sigma_i^2}}{\sigma_i^2 \left(\sum_{i=1}^{n} \frac{1}{\sigma_i^2} \sum_{i=1}^{n} \frac{x_i^2}{\sigma_i^2} - \left(\sum_{i=1}^{n} \frac{x_i}{\sigma_i^2}\right)^2\right)} \tag{714}$$

Linear regression V

adding up according to Eq. (712)

$$\sigma_a^2 = \frac{\sum_{i=1}^n \frac{x_i^2}{\sigma_i^2}}{\sum_{i=1}^n \frac{1}{\sigma_i^2} \sum_{i=1}^n \frac{x_i^2}{\sigma_i^2} - \left(\sum_{i=1}^n \frac{x_i}{\sigma_i^2}\right)^2} \tag{715}$$

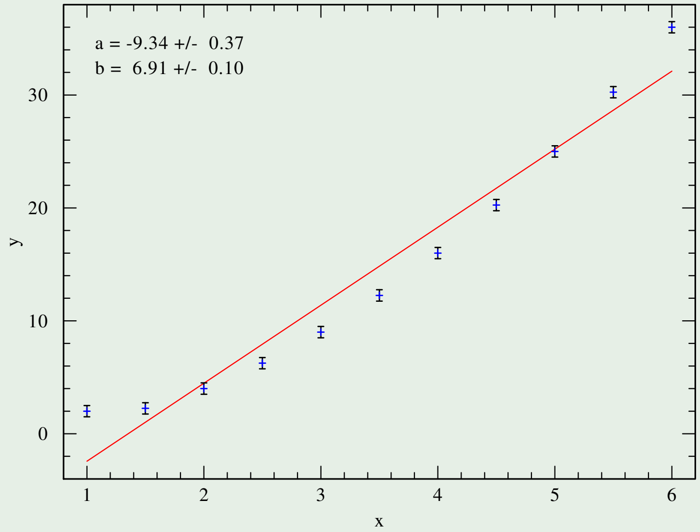
$$\sigma_b^2 = \frac{\sum_{i=1}^n \frac{1}{\sigma_i^2}}{\sum_{i=1}^n \frac{1}{\sigma_i^2} \sum_{i=1}^n \frac{x_i^2}{\sigma_i^2} - \left(\sum_{i=1}^n \frac{x_i}{\sigma_i^2}\right)^2} \tag{716}$$

Caution!

This (purely formal) error may drastically underestimate the real error in a, b!

Linear regression VI

Example: bad fit but small error



$\rightarrow$ small formal error, as error in the measurements is small

but:
$\rightarrow$ model does not fit to data

Linear regression VII

Our original case: errors σ_i not available.

Then: Set $\sigma_i = 1$ in equations for a, b and multiply factor $\sqrt{\frac{\chi^2}{n-2}}$ to the formal errors

$$\sigma_a^2 = \frac{\sum_{i=1}^n x_i^2}{n\sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i\right)^2}\sqrt{\frac{\chi^2}{n-2}} \quad (717)$$

$$\sigma_b^2 = \frac{n}{n\sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i\right)^2}\sqrt{\frac{\chi^2}{n-2}} \quad (718)$$

$$\text{where} \quad \chi^2 = \sum_{i=1}^n (y_i - a - bx_i)^2 \quad (719)$$

Bootstrapping I

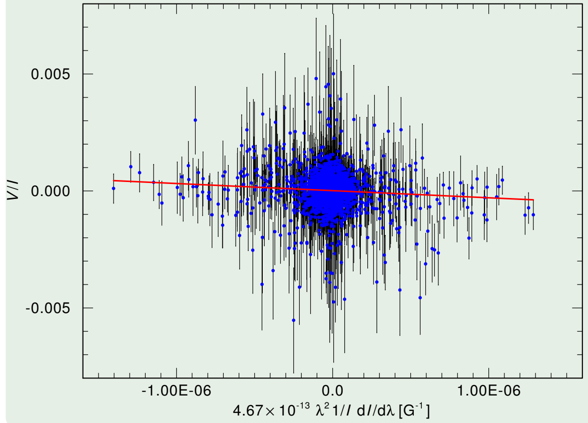
Estimating the errors in fit variables (e.g., the slope b)
Methods:

1. formal error from errors in measuring in y_i $\rightarrow$ without consideration of the fit quality χ^2
2. error from χ^2 without consideration of the errors in measuring y_i

$\rightarrow$ usually results in an underestimation of σ_b

Bootstrapping II

Example: measuring the magnetic field from polarization



Stokes I : intensity

Stokes $V = I_\mathrm{R} - I_\mathrm{L}$ (so: right-hand circularly polarized – left-hand circularly polarized)

$\rightarrow V/I$: fraction of polarized light

$\rightarrow \lambda^2/I \; dI/d\lambda$: Zeeman shift

Idea: for broad spectral lines (Balmer lines in WD, WR emission lines) Zeeman splitting not directly detectable because of Doppler shifts. Therefore: measure "line displacement" per pixel per line together with V/I.

Bootstrapping III

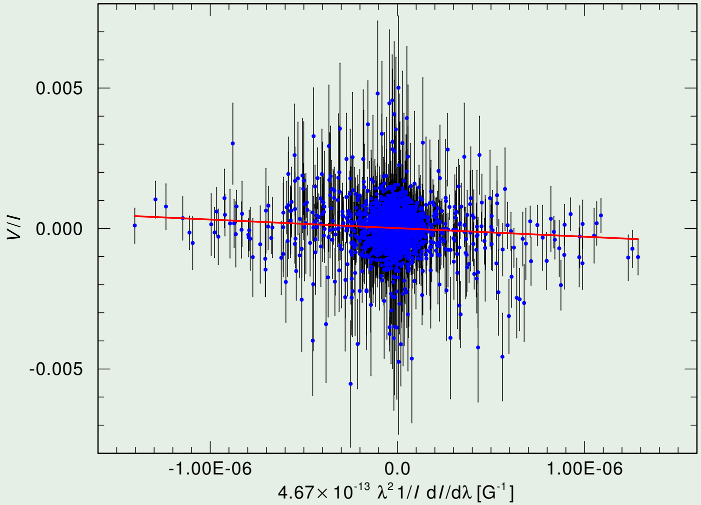
No B-field $\rightarrow$ no correlation. Otherwise, slope gives longitudinal $\langle B_z \rangle$

$$\frac{V}{I} = -\frac{g_{\text{eff}} e \lambda^2}{4\pi m_e c^2} \frac{1}{I} \frac{dI}{d\lambda} \langle B_z \rangle \qquad (720)$$

with average effective Landé factor

Bootstrapping IV

Example: measuring the magnetic field from polarization



$\rightarrow$ slope dominated by only few data points?

Bootstrapping V

Problem: the distribution of b is usually not known
Idea: construct a distribution with help of *Bootstrapping*

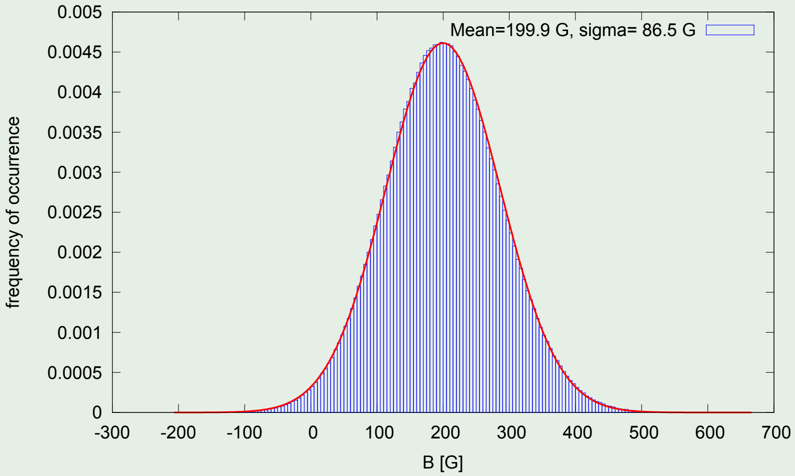
Construction of a Bootstrapping distribution

random sample j by random drawing of n data (x_i, y_i) from the complete set of n data with *repetition* and determination of b_j. Repeating m-times this procedure, where $m \gg n$.
In each random sample are only $\sim 1/e \approx 37\,\%$ of the original data because of *repetitions*.
$\rightarrow$ result: sample of m measured quantity b_j.

Then, determination of the expectation value, variance, etc. for the obtained bootstrapping sample, e.g.,

Bootstrapping VI

Example: magnetic field B_z from polarization



$\rightarrow$ interpretation: $\mu > 2\sigma \rightarrow$ marginal detection (5% proability that actually $B_z = 0$)

Literature I

Aarseth, S. J. 2003, Gravitational N-Body Simulations (Johns Hopkins University Press)

Barnes, J. & Hut, P. 1986, Nature, 324, 446

Barnes, J. E. & Hut, P. 1989, ApJS, 70, 389

Beeman, D. 1976, Journal of Computational Physics, 20, 130

Bestenlehner, J. M. 2020, MNRAS, 493, 3938

Bodenheimer, P., Laughlin, G. P., Rózyczka, M., & Yorke, H. W., eds. 2007, Numerical Methods in Astrophysics: An Introduction (CRC Press)

Bonnor, W. B. 1956, MNRAS, 116, 351

Cox, J. P. & Giuli, R. T. 1968, Principles of stellar structure

Cromer, A. 1981, American Journal of Physics, 49, 455

Ebert, R. 1955, ZAp, 37, 217

Emden, R. 1907, Gaskugeln (B. Teubner, Berlin)

Feautrier, P. 1964, SAO Special Report, 167, 80

Literature II

Gould, H., Tobochnik, J., & Wolfgang, C. 1996, An Introduction to Computer Simulation Methods: Applications to Physical Systems (2nd Edition) (USA: Addison-Wesley Longman Publishing Co., Inc.)

Hansen, C. J. & Kawaler, S. D. 1994, Stellar Interiors. Physical Principles, Structure, and Evolution. (Spinger), 84

Landau, R. H., Páez, M. J., & Bordeianu, C. C., eds. 2007, Computational Physics (Wiley-VCH)

Lane, H. J. 1870, American Journal of Science, 50, 57

Lucy, L. B. 2002, A&A, 384, 725

Pang, T. 1997, Introduction to Computational Physics (USA: Cambridge University Press)

Pooley, D., Lewin, W. H. G., Anderson, S. F., et al. 2003, ApJ, 591, L131

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. 2007, Numerical Recipes 3rd Edition: The Art of Scientific Computing, 3rd edn. (Cambridge University Press)

Schofield, P. 1973, Computer Physics Communications, 5, 17

Schuster, A. 1905, ApJ, 21, 1

Literature III

Srivastava, S. 1962, ApJ, 136, 680

Stoer, J. & Bulirsch, R., eds. 2005, Numerische Mathematik 2 (Springer)

Szebehely, V. & Peters, C. F. 1967, AJ, 72, 876

Toomre, A. & Toomre, J. 1972, ApJ, 178, 623

Šurlan, B., Hamann, W. R., Kubát, J., Oskinova, L. M., & Feldmeier, A. 2012, A&A, 541, A37