Luis Vollmers | **Dr. Maria Reif** | **Prof. Martin Zacharias**

Molecular Dynamics (SS2025)
Exercise 1

Introduction to Molecular Dynamics

Report Tasks

1. Include plots containing E_{kin}, E_{pot} and E_{tot} for each of the 12 simulations you have to conduct: one correctly propagated simulation, one simulation that fails to conserve E_{tot} and one simulation that explodes.
2. Briefly explain your results and state what time step you would choose for the respective system in order to conserve energy but still simulate as much as possible. How can you explain the correlation between particle weight and suggestible time step?

The tasks-box is a summary of the tasks described in this sheet. Read through the sheet carefully for a complete description of the tasks, and when you need to make the report the tasks-box can be used as a reminder and a summary.

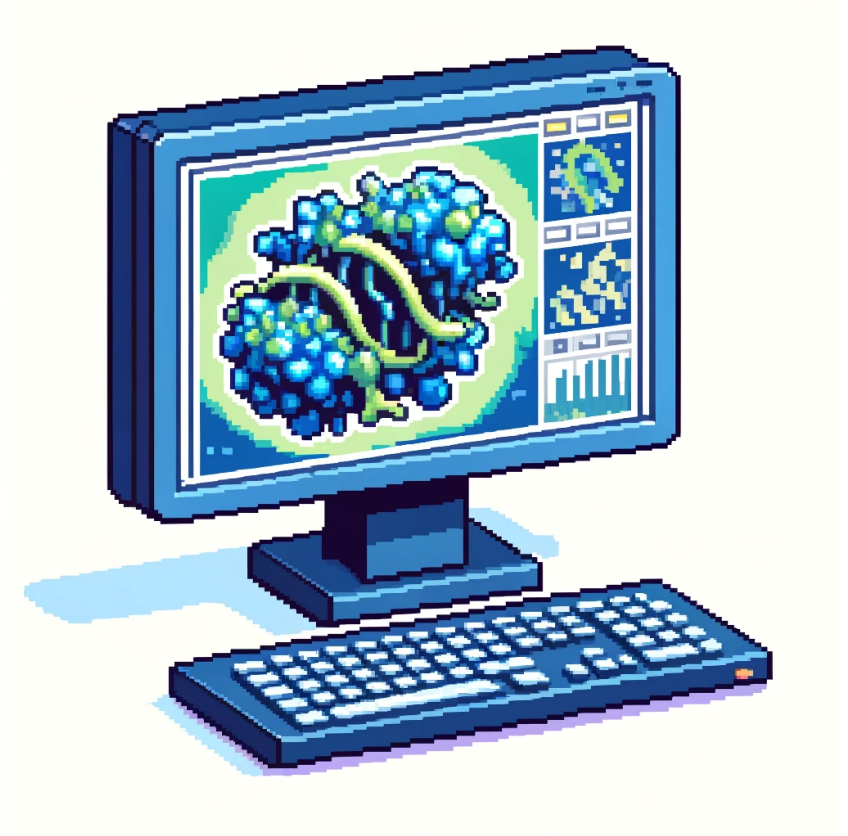


Figure 1: A computer screen showing an elusive protein structure to welcome the reader to computational biophysics. Art created with DALL-E.

Tutorial on Linux/BASH (optional)

GNU/Linux is a UNIX-type operating system. It is free software, thus it is free of charge and the source code can be downloaded and changed to fit your needs. Therefore, there are several different distributions. Many distributions offer a graphical user interface similar to Windows or MacOS. However, in the Linux world it is common to utilise a text-based interface (the shell), which is usually run in a terminal window. The shell reads and interprets user commands. After pressing enter, the commands are executed and the output, if any, is displayed on the screen (`STDOUT`). In this section a few features are introduced, offered by the Bourne-again shell (`Bash`), which is the most common shell in the Linux world. There are many excellent introductions for the use of the Unix/Linux operating system available online. Here are a few examples:

- On http://webminal.org/login/ you find a terminal emulator, i.e. a virtual terminal where you can try out Unix/Linux commands. It is very nice since it contains a tutorial section alongside the terminal. For this terminal emulator you need to register with an email address.
- On http://bellard.org/jslinux/ you can find another terminal emulator that does not require registration. Here, you can also try out typical commands, but there is no tutorial for help.
- The "UNIX Tutorial for Beginners" by Michael Stonebank at the University of Surrey can be done online (http://www.ee.surrey. ac.uk/Teaching/Unix)
- There is a "Cornell Virtual Workshop" (https://www.cac.cornell.edu/VW/Linux) which provides a more detailed introduction to Linux. This is probably too much for the moment but might be nice for future reference. As a side note, under "Topics" you can find more virtual workshops related to programming and computing including Matlab, Python, etc.

Basic Commands

Firstly, to use the command prompt the user needs to open a terminal window. On most Linux systems this can be done by using the keyboard short cut Strg + Alt + T. Some of the most useful commands are listed below. You should try to execute them in your terminal. Start with the easier ones, such as `ls`, `cd` and `echo`.

`awk`
: Whilst the other entries in this list are mostly programming tools, `awk` is a *Turing complete* programming language, that is specifically designed to alter files. It is quite hard to learn, but some utilities might be useful for this course.

`cat <file>`
: The content of `<file>` is printed to `STDOUT`. It can also be redirected into another file (`>`) or be used as an input for a follow-up command (`|`).

`cd <dir>`
: Change the directory to `<dir>`. Without an input option `cd` always changes to the home directory.

`chmod <octalnum> <file>`
: Modifies the user permissions of `<file>` according to `<octalnum>`. `chmod` is necessary to make files executable.

`cp <file> <filename>`
Copies `<file>` to another file called `<filename>`. Linux assumes you know what you are doing and overwrites any other file with the same name without question.

`echo <string>`
Takes `<string>` as an input text and prints it to `STDOUT`. It can also be redirected into another file (`>`) or be used as an input for a follow-up command (`|`).

`grep <pattern> <input>`
Search the `<input>` which can be a file or a text, for the `<pattern>` and print the matching lines to `STDOUT`. The pattern can be a so-called *regular expression* (`REGEX`) or a simple text string.

`history`
All the commands used, are saved to a file which can be viewed by typing this command. This is very useful, if you want to write your commands into an automatic script, or if you forget certain commands.

`htop`
Opens the interactive process viewer which compares to the windows task manager. CPU and memory usage aswell as all processes can be monitored. Also, it makes you feel like a hacker. Press q to exit back to the shell.

`locate <pattern>`
Locates files that match `<pattern>`. Pattern can either be a `REGEX` or any text string that is part of the file you are looking for.

`ls <path>`
List the contents of `<path>` to `STDOUT`. Note that `<path>` can also refer to any directory or a group of files. `ls` can also be prompted without `<path>` to list the current directory. `ls` offers a huge variety of options and is probably the most used command for Linux users.

`man <executable>`
Shows detailed information about the command given in `<executable>`, (the manual entry or man page). Tap q to leave the man page.

`mkdir <dirname>`
Create a directory called `<dirname>` in the current directory.

`mv <file> <filename>`
Like `cp`, but the original `<file>` is removed in the process. Use with caution. If `<filename>` corresponds to a directory the file is moved into that directory.

`pwd`
prints the current directory path to `STDOUT`.

`rm <file>`
This command removes `<file>` from the disk. Therefore, `<file>` is not moved to the bin like in windows, but deleted beyond recovery.

`rmdir <dir>`
Removes an empty directory `<dir>`.

`sed <option> <file>`
`sed` is a *turing complete* tool, that was originally used for text filtering and manipulation and offers diverse automatic applications in this field. Mostly, this command is used to delete, insert and substitute text in files exactly where the user wants it.

`sort <input>`
`<input>`, either a file or text string, is sorted alphabetically. Often, this command receives its input via the pipe `|`.

`tail <N> <file>`
The `<N>` last lines of `<file>` are printed to `STDOUT`. It can also be redirected into another file (`>`) or be used as an input for a follow-up command (`|`).

`vim <file>`
Opens a file for editing in the `VIM` editor. Its usage is accompanied by a steep learning curve.

`wc <input>`
This command means *word count*, thus it counts words in the given input. Furthermore, it counts lines, characters and bytes if given the appropriate options.

`whoami`
prints the current user to `STDOUT`.

Keep in mind, that all of these commands can change their behaviour when you specify certain options. For Linux commands, these options are usually specified by `-` followed by the `flag`, e.g. `ls` just lists the directory content. However, `ls -l` lists the contents in a more verbose way and gives information about permissions, disk usage and latest access. Another example would be `mv file file2` which renames `file` to file2 without communicating to the user. If `mv -v file file2` is specified on the other hand, the commands is more verbose and prints information to `STDOUT`. Most commands will tell you which options are available by specifying `--help`, e.g. `sort --help`.

Managing Files and Directories

Try to find out how to use the following commands, i.e. what inputs to use after each command in order to make them work:

```
# comments start with '#'; they will not be executed by bash
mkdir # create new, empty directory
touch # create new, empty file
cp # copy one or multiple files
cp -r # recursively copy one directory and its content to another one
rmdir # remove one empty directory
rm # remove one or multiple files
rm -r # remove one directory and its content
```

When in doubt, use `man` or specify `--help`. `rm` and `cp` aswell as most other linux commands allow for something called *globbing*. Basically, globbing means that instead of inputting a file name, you can use an expression resembling a variety of file names that match the so called `glob`, e.g. use the wildcard `*` to match any pattern in the file name. For instance `rm *.txt` will delete any file with the `.txt` extension. The question mark wildcard `?` is used to represent exactly one character, which can be any single character. Two question marks in succession would represent any two characters in succession, and three question marks in succession would represent any string consisting of three characters. For instance, `rm ???` will delete any file

whose name contains exactly 3 characters in total (including the extension). `rm ???.txt` will delete any file with the .txt extension and whose name contains exactly 3 characters. Both, `?` and `*` can be combined.
To rename a file you can use `mv`. The second input argument can either be a file, an absolute path or a relative path, which holds true for `cp` and `rm` aswell.

```
mv file1 file2 # If file2 exists it will be overwritten
mv file1 dir1 # moves file1 to a directory called dir1
mv dir1/file1 ../ # move file1 from directory1 to parent folder (../)
```

VIM/VI

One of the popular UNIX editors is `vi` , which stands for visual editor. `vi` is a full screen editor and has two main modes of operation:

- Command mode: commands to specify an action to be taken on the file.
- Insert mode: where text is inserted into the file.

In the Command mode, every character typed is a command that does something to the text file being edited. Take into account that both UNIX and vi are case-sensitive. This means that the action performed on a file when typing y will not be the same as when typing Y. To create a new file you simply need to type `vi <filename>`. If the file already exists you will see the first lines of it (you will see as many lines as your screen allows you). To save/exit the file you can type either of the following commands:

```
:q # quit vi without saving. This command will only work if you have not modified your file
:q! # quit vi without saving the last changes
:w # save (write) the file
:wq # save (write) the file and quit vi
:x # save and quit the file (exactly as :wq)
:w new_filename # save the current file with a different name
:wq new_filename # save the current file with a different name and exit. This command will only
    work if new_filename is not the name of any existing file in the current directory
:wq! new_filename # save the current file with a different name and exit if new_filename is the
    name of any existing file in the current directory the previous content of the file
    new_filename will be deleted.
```

To switch to the `insert mode` you need to type either `i`, `I`, `a`, `A`, `o` or `O`. See below for the meaning of each specific command. Once in the `insert mode` every character typed is added to the text in the file. To turn off the insert mode you need to press Esc.

```
i # insert text before cursor
I # insert text at beginning of current line
a # append text after cursor
A # append text to end of current line
o # insert text in a new line below current line
O # insert text in a new line above current line
```

Furthermore, there is a `replace mode` accessible by hitting shift + R. Switching between the `insert mode` and the `replace mode` is also possible by typing Insrt. There are more specific options to modify file contents via `VIM`. Only some of them are listed below.

```
r # replace single character under cursor
x # delete single character under cursor
Nx # delete N characters, beginning with current character
dw # delete single word, beginning with current character
C # change characters in the current line
```

```
D # delete remainder of the current line
Y # copy remainder of the current line
cw # change the current word with new text
cc # change entire current line
dd # delete entire current line
yy # copy entire current line
cNw # change the next N words, beginning with the current
cNc # change the next N lines, beginning with the current
dNw # delete the next N words, beginning with the current
dNd # delete the next N lines, beginning with the current
yNy # copy the next N lines, beginning with the current
p # paste the previously copied/deleted lines/words after the current line
```

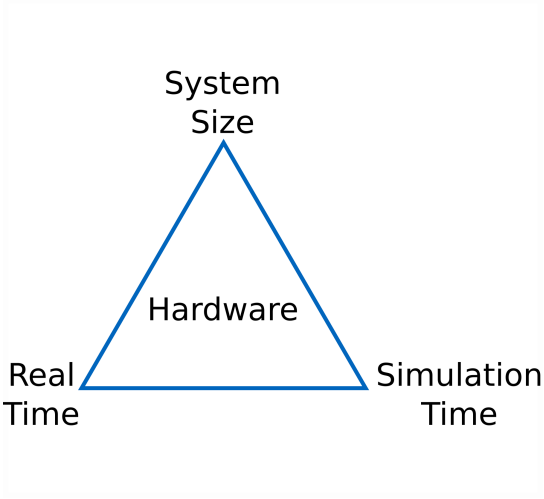
Undoubtedly, two of the most important commands are redo/undo which are available in VIM by typing u for undoing and Ctrl+r for redoing. At this point, VIM can be used just like any other text editor, however, it is much more versatile. It can search the text for text patterns and replace them with new text all by using convenient short cuts. Some of those short cuts are listed below. Furthermore, VIM has numerous plug-ins, since it is open source with a lively community, e.g. you could run latex by just using VIM.

```
/word # search forward for the pattern 'word' in text.
?word # search backward for the pattern word in text
n # move to following occurrence of search string
N # move to preceeding occurrence of search string
gg # go to the first line
G # go to the last line
:%s/pattern1/pattern2/g # replace word pattern1 by pattern2 throughout the whole file
```

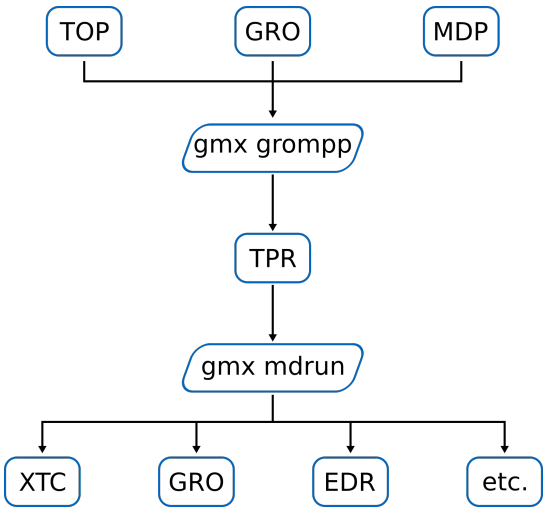
For a more hands on introduction to VIM you may use `vimtutor`. This programm gets installed simultaneously with VIM and can be opened by typing `vimtutor` in you command line. It contains seven interactive lessons, teaching the very basics of the program.

Introduction to GROMACS

GROMACS is an open-source software that is used to perform molecular dynamics (MD) simulations of biomolecules like proteins, DNA and lipids. This command-line program integrates Newton's equations of motion to generate a time series, also known as a trajectory, of a given atomic system. This tutorial aims to introduce the workflow and the fundamentals of GROMACS. More precisely, the subject of this course is the integration time step which is one of the most fundamental parameters to chose in an MD simulation. Its importance originates from the scarceness of computational ressources.
The computational ressources, hence the available hardware, determines the quality of three interdependent constraints, the most important one being the real time needed to complete the simulation. The other two contraints are the system size (number of equations) and simulation time (number of times said equations are solved). The size of the system usually depends on the scientific problem at hand, therefore the simulation time is the most flexible of the three constraints that can also be modulated by the integration time step (see figure 2a).



(a) For a given hardware, the user has to make a trade-off between real time, simulation time and the system size. To decrease the real time passing during the simulation, either the system size or the simulation time has to be sacrificed.



(b) The workflow in GROMACS uses the commands `gmx grompp` and `gmx mdrun` to generate diverse output files from three standardised input files.

Thus, the following question emerges: how large can the time step be, without destabilising the system? Small time steps are unable to capture biological processes within reasonable real time and too large time steps lead to truncation errors in the integration algorithm and violate energy conservation. (Furthermore, the Nyquist-Shannon-Abbast-theoreme should not be violated.)
Therefore, the aim of this exercise is to get accustomed to GROMACS and to understand benefits and problems of the underlying algorithm. Herein, four systems should be simulated with different time steps. Moreover, for each system the different time steps should generate a correct trajectory, one trajectory where the energy conservation is violated and one trajectory, with a time step unreasonably high, so that the simulation *explodes*.

GROMACS Workflow

Unlike many everyday programs GROMACS does not have a graphical user interface due to its manifold functionalities. Instead it is used by executing commands and tools stemming from its software package via the command line. E.g., a very simple command is `gmx -version` which outputs version information about your GROMACS package onto your screen (try it).
GROMACS requires three different files to run a simulation: TOP-, MDP- and GRO-file. Each file contributes different information to GROMACS. The GRO-file contributes the atomic positions and velocities, the TOP-file contributes the information about the atomic and molecular attributes (the topology) and the MDP-file is short for the molecular dynamics parameters, and determines which algorithms to use and how to tune them. The command `gmx grompp` takes these three files as input with their respective flags and outputs a TPR file that can be subsequently used to start the MD run via `gmx mdrun` (see figure 2b). In a nutshell, `gmx grompp` is the MD run preparation tool and `gmx mdrun` is the simulation engine. A variety of output files are generated by `gmx mdrun` which are explained in the GROMACS command line reference[1]. For today's tutorial, the EDR file outputted by `gmx mdrun` is most important. In this file the energetic information about the system is saved, which can be analysed via `gmx energy` to

[1] https://manual.gromacs.org/documentation/2018/onlinehelp/gmx-mdrun.html

extract specific information and plots.
Conclusively, the workflow for one system in this tutorial consists of the input file preparation (`grompp`), the actual simulations (`mdrun`) and the analysis of the energetic contributions (`energy`).

Breaking MD Simulations with GROMACS

Main subject of this tutorial are four different systems represented by the input files stored in the moodle repository. The coordinate file `intro.gro` represents a box of arbitrary diatomic particles which should be used for all simulations. The topologies `topol_*.top` differ with respect to the particle weight and the parameter file `intro.mdp` contains no value for the `dt` option, i.e. the simulation time step. This is deliberate, since the main task for the user is to tinker with this option (further elaborated later on in this section). Although a deep understanding of the input files is not necessary right now, it is recommended to take a moment to inspect them with `vim` and trying to make sense of their contents.

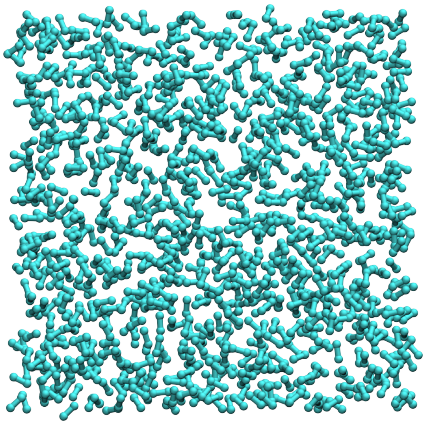


Figure 3: The particles visualised by `vmd intro.gro`. With representations set to *dynamic bonds* and *VDW*.

First and foremost, an energy minimisation using `em.mdp` is crucial prior every MD run. Since the coordinates in `intro.gro` are initialised more or less randomly, their positioning might disregard physical laws resulting in e.g. overlapping atoms. Thus, once the simulation introduces physics to these nonphysical states extremely high forces occur which lead to the explosion of the simulation box. Therefore, some steps of gradient descent along the system's potential energy are necessary to converge the simulation into a favourable energetic regime. The downloadable `em.mdp` file should be used for that before applying the user-adjusted `intro.mdp` file in the following simulation. Commands that conduct the energy minimisation and the MD simulation can look like the following lines of code. Their flags and options are described in closer detail in table 1.

```
gmx grompp -f em.mdp -p topol_2.top -c intro.gro -o em_2.tpr
gmx mdrun -v -deffnm em_2
gmx grompp -f intro_0.001.mdp -p topol_2.top -c em_2.gro -o intro_0.001_2.tpr -maxwarn 3
gmx mdrun -v -deffnm intro_0.001_2
gmx energy -f intro_0.001_2.edr -o intro_0.001_2.xvg
```

Table 1: Each flag given in the code above is explained in detail in this table using one example.

Flag (+ Option)	Meaning
`-f em.mdp`	Use the energy minimisation parameter file as `grompp` input.
`-p topol_2.top`	In this example use the system topology with an atomic weight of 2 Dalton.
`-c intro.gro`	Use the initial coordinates from moodle for the energy minimisation.
`-o em_2.tpr`	Output the file `em_2.tpr` for usage in the `gmx mdrun` command. The name is chosen to contain the parameter (em) and topology (2) information.
`-maxwarn 3`	Choosing an extremely large time step that might result in an exploding system does not make a lot of sense beyond teaching students the limits of the leapfrog integrator. Thus GROMACS recognises this misconduct and refuses to produce the necessary output, unless this flag is specified.
`-v`	Gives information about the time step and remaining simulation time if the simulation works. If the simulation does not work there is no timing information.
`-deffnm em_2`	all input and output names of the corresponding `gmx mdrun` should have the prefix `em_2` which must match the `-o` option of `gmx grompp`.
`-f intro_0.001_2.edr`	Instead of an MDP file for `gmx grompp`, `gmx energy` needs an EDR file for the `-f` flag. The prefix is the same as specified in the `-deffnm` option of `gmx mdrun`.

`gmx energy` is an interactive command, prompting the user to chose which energetic properties to collect information about. The interface is terminal-based and typically looks like shown in figure 4. Selecting the energetic properties is conducted by typing the corresponding number followed by hitting Enter twice, e.g. for the kinetic energy in figure 4, type 5 + Enter + Enter.

```
Select the terms you want from the following list by
selecting either (part of) the name or the number or a combination.
End your selection with an empty line or a zero.
-------------------------------------------------------------------
  1  Bond            2  LJ-(SR)         3  Coulomb-(SR)     4  Potential
  5  Kinetic-En.     6  Total-Energy     7  Temperature      8  Pressure
  9  Vir-XX         10  Vir-XY          11  Vir-XZ          12  Vir-YX
 13  Vir-YY         14  Vir-YZ          15  Vir-ZX          16  Vir-ZY
 17  Vir-ZZ         18  Pres-XX         19  Pres-XY         20  Pres-XZ
 21  Pres-YX        22  Pres-YY         23  Pres-YZ         24  Pres-ZX
 25  Pres-ZY        26  Pres-ZZ         27  #Surf*SurfTen   28  T-rest
```

Figure 4: User interaction after executing `gmx energy`. Multiple options can be selected at once by either typing the number or the whole name of the energetic term.

The user interaction can be skipped by pre-defining the terms of interest with `printf` and `|`.

This is called a pipe and can be used to chain different commands. `\n` is code for a newline and is equivalent to Enter for the user interaction.

```
printf "4\n5\n6\n" | gmx energy -f <...>.edr -o <...>.xvg
```

Finally, the task is to chose three different time steps, so that three different outcomes occur. The time step is one of the most fundamental parameters for MD simulations since it is pivotal for numerical integration algorithms in terms of energy conservation and time symmetry.
Firstly, a low time step should be selected, so that the system propagates correctly. The second case engulfs a simulation, where the energy conservation is violated, but the simulation finishes. In the last case, a large time step causes the simulation to crash due to very large truncation errors. In this case, `gmx mdrun` will halt and you have to abort the program with Ctrl + C. Eventually, 12 plots will be featured in the report. Three different time steps for each of the four particle weights (e.g. figure 5).

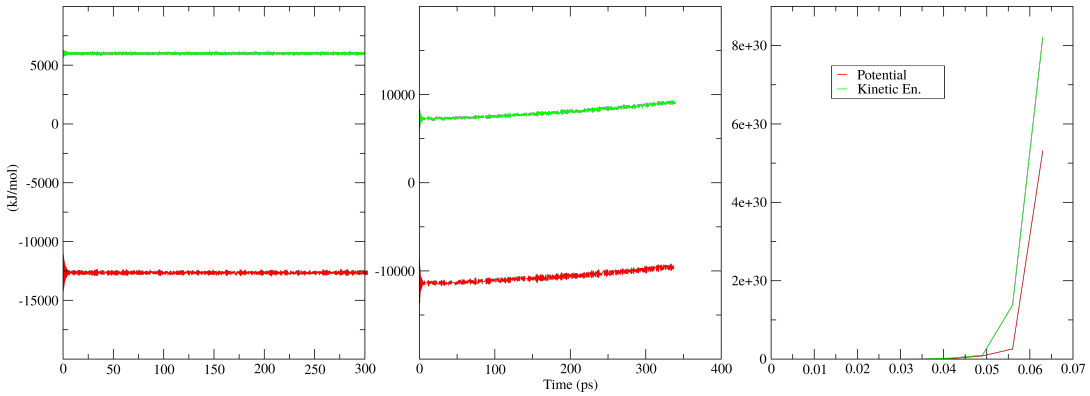


Figure 5: Example plots for one system. Left, the energy curves for the stable time step, then an unstable simulation, where the energy increases over time, and lastly, a simulation that crashes after 0.06 ps due to extremely high forces (*exploding*).

However easy it is to generate the two extremes, an intermediate time step that violates energy conservation is difficult to select. In the literature, a time step of 0.001 ps is recommended for regular MD simulations. Start with this time step and the lightest system. Then increase the time step in steps of 0.001 ps until the system either crashes, or violates energy conservation. If the system crashed, a time step in-between the current and the former needs to be tried (e.g. 0.0035 ps). Otherwise, if the intermediate case was achieved, the results can be saved and the next increment can be tried to achieve an exploding system. In order to check the energies, `xmgrace` or any other plotting utility can be used. Python, via `jupyter notebook` is the recommended option especially for figures to be featured in the reports. However, for having a quick look at the graphs, `xmgrace` suffices.

```
xmgrace -nxy output.xvg
```

Once, all three graphs are ready for the lightest system, the second-lightest system may be tried with the intermediate time step found for the lightest system. As before, the time step has to be incremented or decremented until all three cases are achieved. The same procedure applies for the remaining systems. For the report include the graphs and explain your results and apparent trends. Additionally, state what time step you would choose for the respective system.