# 7 — PHY 494: Homework assignment (28 points total)

Due Thursday, Feb 28, 2019, 11:59pm.

Submission is to your **private GitHub repository**.

Read the following instructions carefully. Ask if anything is unclear.

1. `cd` into your assignment repository (change *YourGitHubUsername* to your GitHub username) and run the update script `./scripts/update.sh` (replace *YourGitHubUsername* with your GitHub username):

   ```
   cd  assignments-2019-YourGitHubUsername
   bash ./scripts/update.sh
   ```

   It should create three subdirectories[1] `assignment_07/Submission`, `assignment_07/Grade`, and `assignment_07/Work`.

2. You can try out code in the `assignment_07/Work` directory but you don't have to use it if you don't want to. Your grade with comments will appear in `assignment_07/Grade`.

3. Create your solution in `assignment_07/Submission`. Use Git to `git add` files and `git commit` changes.

   You can create a PDF, a text file or Jupyter notebook inside the `assignment_07/Submission` directory as well as Python code (if required). **Name your files `hw07.pdf` or `hw07.txt` or `hw07.ipynb`**, depending on how you format your work. Files with code (if requested) should be named exactly as required in the assignment.

4. When you are ready to submit your solution, do a final `git status` to check that you haven't forgotten anything, commit any uncommited changes, and `git push` to your GitHub repository. Check on *your* GitHub repository web page[2] that your files were properly submitted.

[1] If the script fails, file an issue in the Issue Tracker for PHY494-assignments-skeleton and just create the directories manually.

[2] `https://github.com/ASU-CompMethodsPhysics-PHY494/assignments-2019-`*YourGitHubUsername*

You can push more updates up until the deadline. Changes after the deadline will not be taken into account for grading.

Homeworks must be legible and intelligible or may otherwise be returned ungraded with 0 points.

This assignment contains **bonus problems**. A bonus problem is optional. If you do it you get additional points that count towards this homework's total, although you can't get more than the maximum number of points. If you don't do it you can still get full points. Bonus problems and bonus points are indicated with an asterisk "*".

If you implement the functions as specified you can run the tests in the file `Submission/test_hw07.py` with `pytest`

```
cd Submission
pytest test_hw07.py
```

and all tests should pass. If you have errors, have a look at the output and try to figure out what is still not working. Having the tests pass is not a guarantee that you will get full points (but it is general a very good sign!).

## 7.1 The Lennard-Jones potential (8 points)

The *Lennard-Jones potential* is widely used to simulate the behavior of atoms and molecules. It models the interactions between two uncharged atoms as

$$V(r) = 4\epsilon \left[ \left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]. \tag{1}$$

where $r$ is the distance between two atoms at positions $\mathbf{r}_1$ and $\mathbf{r}_2$

$$r = \sqrt{\mathbf{r} \cdot \mathbf{r}} \quad \text{with} \quad \mathbf{r} := \mathbf{r}_2 - \mathbf{r}_1 \tag{2}$$

(and $\mathbf{r}$ is the distance vector). The LJ potential describes the atomic van der Waals interactions. They consist of an attractive term (the $-(\sigma/r)^6$ term) due to London dispersion forces (spontaneous fluctuations in the

electron clouds lead to attractive fluctuating dipoles), and a strongly repulsive term (the $+(\sigma/r)^{12}$), which crudely models the fact that atoms cannot come arbitrarily close (due to the quantum mechanical Pauli exclusion principle).

The LJ potential Eq. 1 has two parameters: $\epsilon$ is an energy and sets the energy scale and is related to the attractive strength whereas $\sigma$ is a length and sets the length scale and range of the interaction. Parameters for the noble gas **argon** in the liquid state are

| parameter | value | unit |
|---|---|---|
| $\epsilon$ | $1.654 \times 10^{-21}$ | J |
| $\sigma$ | $3.41 \times 10^{-10}$ | m |

For numerical convenience, we want to use units close to 1 so for all further numerical evaluations we use

| parameter | value | unit |
|---|---|---|
| $\epsilon/k_B$ | 119.8 | K |
| $\sigma$ | 0.341 | nm |

Note that we express $\epsilon$ as a temperature by dividing with the Boltzmann constant $k_B = 1.3806488 \times 10^{-23}\,\mathrm{J \cdot K^{-1}}$. Temperature is a direct measure of energy and in this case it is convenient to use Kelvin as a measure of energy instead of Joule. The length $\sigma$ is expressed in nano meters ($1\,\mathrm{nm} = 1 \times 10^{-9}\,\mathrm{m}$).

In the following, use Python (with `numpy`) for all numerical calculations and `matplotlib` for all plotting.

(a) Plot the Lennard-Jones potential Eq. 1 for argon in the range $0 < r \leqslant 4\sigma$ **[3 points]**

- Use a sufficiently large number of $r$ values for plotting so that all details of $V(r)$ are visible in the plot[3].

[3] For instance, `np.linspace(1e-15, 4*sigma, 100)`

- Set the display limits of the energy ($y$) axis to range from $-120$ K to $+100$ K.[4] Your plot should clearly show the smallest value of $V_{\mathrm{LJ}}$ and indicate the behavior for small and large $r$.
- Label your axes with the quantity plotted and the units.[5]

(b) The *force* along the direction of $\mathbf{r}$ is

$$F = -\frac{dV(r)}{dr}. \tag{3}$$

Calculate the Lennard-Jones force $F(r)$ *analytically*.[6]

Plot $F(r)$ over the same range of $r$ values as in (a). Adjust the force-axis limits so that you can clearly show the minimum in $F(r)$. **[5 points]**

(c) BONUS: Analytically calculate the general value $r_{\mathrm{min}}$ at which the Lennard-Jones potential Eq. 1 obtains a minimum and calculate its minimal value $V_{\mathrm{LJ}}(r_{\mathrm{min}}) = \min_r V_{\mathrm{LJ}}(r)$. First express your answers in terms of $\epsilon$ and $\sigma$ and then calculate the specific values for argon. **[bonus +2*]**

## 7.2 Molecular dynamics simulation of the Argon dimer (20 points)

At sufficiently low temperature, two atoms of the noble gas argon are weakly bound to each other and form a *dimer*. The interaction between two argon atoms with mass $m$ each can be approximated by the Lennard-Jones potential (Eq. 1).

[4] Generate the plot with `ax = plt.subplot(1, 1, 1)`, then plot with `ax.plot(...)` and set the limits with `ax.set_ylim(-120, 100)`.

[5] For example, for the $x$-axis you may use `ax.set_xlabel(r"distance $r$ (nm)")`.

[6] $F(r)$ is the force that one particle exerts on the other particle when they are at a distance $r$.

Simulate the dynamics of the weakly bound argon dimer $Ar_2$ by solving Newton's equations of motion for the interatomic distance $r$

$$\frac{d^2r}{dt^2} = \mu^{-1} F_{LJ}(r). \tag{4}$$

with the reduced mass $\mu = m/2$. (We can treat this as a 1-body, 1D problem because we can reduce the 2-body problem to an effective 1-body problem with the reduced mass

$$\mu = \frac{m_1 m_2}{m_1 + m_2} = \frac{m^2}{2m} = \frac{1}{2}m \tag{5}$$

and if we start with zero total angular momentum then the only motion is along the connecting vector and the problem can be fully described in in 1D).

Use as parameters for argon

- $\epsilon = 120\,\mathrm{K} \cdot k_B = 0.99774\,\mathrm{kJ} \cdot \mathrm{mol}^{-1}$ (use $\mathrm{kJ} \cdot \mathrm{mol}^{-1}$ as units, see below)
- $\sigma = 0.341\,\mathrm{nm}$
- $m = 39.948\,\mathrm{u}$ (in atomic mass units, $1\,\mathrm{u} = 1.660539040 \times 10^{-27}\,\mathrm{kg} = 1\,\mathrm{g} \cdot \mathrm{mol}^{-1}$)

When using these units for energy, length, and mass then for consistency the unit of time *must* be pico seconds (ps, $10^{-12}$ seconds).[7]

For the following problems you can use the provided module `integrators.py`, which was developed in the lesson on ODEs, without having to specifically note this. If you use the module, include it in your submission. You may also modify the module and include it as part of your solution.

(a) Use *Euler's method* to simulate the dynamics for $0 \leqslant t \leqslant 10\,\mathrm{ps}$ with a time step $\Delta t = 1 \times 10^{-4}\,\mathrm{ps}$. Use as initial conditions

[7] Using the atomic mass unit in conjunction with nm/ps as unit of velocity directly yields energies in kJ/mol, i.e. $\frac{1}{2}mv^2 = \frac{1}{2} \times 1\,\mathrm{u} \times (1\,\mathrm{nm} \cdot \mathrm{ps}^{-1})^2 = 1\,\mathrm{kJ} \cdot \mathrm{mol}^{-1}$.

- $r(t=0) = 0.38276$ (nm)
- $v(t=0) = 0.18$ (nm/ps).

Include all your functions in in a file `problem2.py` (which may itself import `integrators.py` if necessary).

(i) Create a function `F_LJ(r)` that returns the force (Eq. 3) at distance $r$ due to the Lennard-Jones potential Eq. 1 for Argon. The units of this problem should be used (nm, kJ/mol, kJ/(nm mol) for force). **[2 points]**

(ii) Create a function `dimer_md(r0, v0, t_max, dt, mass)` that takes the initial distance of the atoms $r(t=0)$ (in nm) and initial relative velocity $v(t=0)$ (in nm/ps) as well as the maximum time $t_{\max}$, the integration timestep $\Delta t$, and the mass as input and returns a tuple consisting of `(t, y)`, i.e., a numpy array of all the times $t$ and a numpy array consisting of an array of the ODE standard form of the dependent variables $\mathbf{y}(t) = (r(t), v(t))$, i.e., the positition and velocity at each timestep. **[10 points]**

(iii) Plot $r(t)$ and show the plot; include axes labels with units. **[2 points]**

(iv) Describe briefly what kind of motion the argon dimer undergoes. **[1 points]**

(b) Analyze the error in the total energy $E$ by computing the total energy from potential $U$ and kinetic energy $T_{\text{kin}}$.

(i) Create a function `U_LJ(r)` that calculates the value of the Lennard-Jones potential Eq. 1 for argon in the units of this problem (i.e., input nm, output kJ/mol). **[2 points]**

(ii) Plot (1) $E(t)$, $U(t)$, $T_{\text{kin}}(t)$ and (2) the decadic logarithm of the relative error in the total energy, i.e.,

$$\log_{10}\left|\frac{E(t)}{E(t=0)} - 1\right|.$$

Label the graph axes with appropriate units. **[2 points]**

(iii) Comment on your graph. **[1 points]**

(c) BONUS: Perform the MD in (a) with the *velocity Verlet* algorithm (create a function `dimer_md_vv()`) and perform the error analysis as in (b). **[bonus +4*]**