4 — PHY 494: Homework assignment (46 points total)

Due Saturday, Feb 13, 2016, 5pm.

Submission is now to your **private GitHub repository**. Follow the link provided to you by the instructor in order for the repository to be set up: It will have the name *ASU-CompMethodsPhysics-PHY494/assignments-2016*-YourGitHubUsername and will only be visible to you and the instructor/TA. Follow the instructions below to submit this (and all future) homework.

`git clone` your assignment repository and run the script `scripts/update.sh` (replace *YourGitHubUsername* with your GitHub username):

```
repo="assignments-2016-YourGitHubUsername.git"
git clone https://github.com/ASU-CompMethodsPhysics-PHY494/${repo}
cd ${repo}
bash ./scripts/update.sh
```

It should create three subdirectories[1] `assignment_04/Submission`, `assignment_04/Grade`, and `assignment_04/Work`. Commit and push a PDF, text file or Jupyter notebook inside the `assignment_04/Submission` directory and **name it `hw04.ipynb` (or `hw04.pdf`)**. 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 "*".

4.1 Discussion of the errors of finite difference operators (12 points)

In lesson 06 Differentiation you plotted the absolute error $|E_h(t)| = |D_h \cos t - \cos t$ for three different algorithms for the finite difference operator D_h with step size h (forward difference, central difference, and extended difference) and for three different values $t = 0.1, 1, 100$.

Complete all calculations and compare your plots to the ones shown at the end of the notebook 06_differentiation.ipynb. Discuss the following questions:

(a) Which algorithm produces the most accurate[2] result? Give order-of-magnitude estimates for each one, based on your data. **[3 points]**

(b) Describe the general shape and features of your graphs. Explain why you see increases and decreases in accuracy with varying h. **[4 points]**

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

[2] The *accuracy* is measured by the deviation from the exact result, i.e. the lower $|E|$ the more accurate. *Precision* measures how well a number is defined and is essentially the machine precision in our case.

(c) What is the best value of h in each case? How does the best value of h change with the algorithm? Explain the observed behavior in the light of the answer to your answer (b). **[5 points]**

4.2 The Lennard-Jones potential (22 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: ϵ is an energy and sets the energy scale and is related to the attractive strength whereas σ 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
ϵ	1.654×10^{-21}	J
σ	3.41×10^{-10}	m

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

parameter	value	unit
ϵ/k_B	119.8	K
σ	0.341	nm

Note that we express ϵ 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 σ 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].
- 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_{LJ} and indicate the behavior for small and large r.
- Label your axes with the quantity plotted and the units.[5]

(b) BONUS: Analytically decompose $V(r) = V_{\mathrm{repulsive}}(r) + V_{\mathrm{attractive}}(r)$ into a purely attractive and a purely repulsive component.[6] Plot these two components together in one graph. **[bonus +4*]**

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

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

Calculate the Lennard-Jones force $F(r)$ *analytically*.[7]

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]**

(d) BONUS: Analytically calculate the general value r_{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 ϵ and σ and then calculate the specific values for argon. **[bonus +4*]**

(e) Now calculate $F(r)$ *numerically* using

(i) the forward difference D_{fd} algorithm **[3 points]**

(ii) the central difference D_{cd} algorithm **[3 points]**

and plot the calculated forces in the same plot as the analytical solution (c).[8]

(f) BONUS: Time the calculation of the force for the analytical and the numerical derivative for the range of r-values `r = np.linspace(1e-15, 4*sigma, 100)` using `%timeit` in the Jupyter notebook and show in a table the total time taken and the ratio between the time for the analytical and the numerical calculation. Which calculation is preferrable? **[bonus +4*]**

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

[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] Come up with two functions $V_{\mathrm{repulsive}}(r)$ and $V_{\mathrm{attractive}}(r)$ that, when added, give the original potential. Hint: Carefully read the explanation of the Lennard-Jones potential above. . . .

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

[8] You need to choose appropriate h yourself.

(g) Determine the absolute error $\mathcal{E}_h(r) = |F_{\text{analytical}}(r) - F_{\text{calc}}(r)|$ and the relative error $\mathcal{E}_h(r)/|F_{\text{analytical}}(r)|$ and plot them each for five different values of $h = 0.1, 10^{-5}, 10^{-6}, 10^{-7}10^{-15}$ in a plot with a logarithmic y-axis for

(i) the forward difference algorithm **[3 points]**

(ii) the central difference algorithm **[3 points]**

Are there regions where the numerical derivative is particularly bad? Which algorithm is the best choice and what is the best h? **[2 points]**

4.3 Periodic boundary conditions (12 points)

In computer simulations of atoms we typically only simulate small numbers of atoms on the order of hundreds to millions of atoms (small compared to macroscopic numbers on the order of 10^{23}). Simulations take place in a virtual box or *unit cell*; the simplest boxes have a brick shape ("orthorhombic") with all angles 90° and lengths L_x, L_y and L_z. In order to reduce the effect of boundaries we use *periodic boundary conditions* ("PBC"). In this problem you should explore what PBC means in practice.

We use the function `minimage(x, box)` below to enforce PBC.

```
import numpy as np

def minimage(x, box):
    """Minimum image coordinates of x in orthorhombic box."""
    return x - box*np.rint(x/box)
```

For all calculations and plotting use Python/`matplotlib`.

(a) Assume a 1D "box", i.e. a particle with coordinate x is considered inside the box if $-\frac{L_x}{2} \leqslant x < \frac{L_x}{2}$.

(i) For $L_x = 2$, plot the value of `minimage(x, Lx)` for $-3.5 \leqslant x \leqslant 3.5$. **[2 points]**

(ii) Based on your plot, what does the `minimage()` function do to x? **[2 points]**

(iii) The NumPy `rint()` function rounds its argument (which can be an array) to the nearest integer.[9] Analyze the `minimage()` function and explain in words (and examples if necessary) how its algorithm works. **[3 points]**

(b) Now consider a 2D box, with lengths $L_x = 2$ and $L_y = 1$, i.e. `box = np.array([2., 1.])`.

(i) Apply `minimage()` to a list of positions of a particle,

```
positions = np.array([[0, 0], [1, 1], [-1, 0.5],
                      [3, -0.3], [1234, -456.1]],
                     dtype=np.float64)
```

[9] See http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.rint.html and play with `rint()` (and the individual terms in `x - box*np.rint(x/box)`) to understand step-by-step how the `minimage()` transforms its input.

and show the results. Where is the particle relative to the box? **[2 points]**

(ii) Using the same box, plot the following positions as black circles in the x-y plane[10]

```
randpos = np.random.random((100, 2)) * np.array(6) - 3
```

and plot the positions remapped with `minimage()` in the same plot as red crosses.[11] **[3 points]**

(c) BONUS: Visualize how `minimage()` transforms the trajectory of a projectile under gravity but with no air resistance,

$$\mathbf{r}(t) = \frac{1}{2}\mathbf{g}t^2 + \mathbf{v}_0 t + \mathbf{r}_0, \quad \mathbf{r}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} \tag{4}$$

with $\mathbf{g} = (0, -9.81\,\mathrm{m}\cdot\mathrm{s}^{-2})$ and initial conditions at $t = 0$ (velocity $\mathbf{v}_0 = (10\,\mathrm{m}\cdot\mathrm{s}^{-1}, 50\,\mathrm{m}\cdot\mathrm{s}^{-1})$ and position (in a valley) $\mathbf{r}_0 = (0, -100\,\mathrm{m})$).

You can calculate the positions with the Python function

```
def projectile(times, v0=np.array([10, 50]), x0=np.array([0, -100])):
    g = np.array([0, -9.81])
    t = times[:, np.newaxis]  # enable np broadcasting
    return 0.5*g*t**2 + v0*t + x0
```

(i) Plot the trajectory (x, y) for the times $0 \leqslant t \leqslant 20$ s (for 1000 time points).[12] **[bonus +2*]**

(ii) Into the same figure, plot the same trajectory under periodic boundary conditions inside a box with lengths $L_x = L_y = 200$ m. **[bonus +4*]**

[10] Use `plt.plot(x, y, color='black', marker='o', linestyle='None')`

[11] Use `plt.plot(x, y, color='red', marker='x', linestyle='None')`

[12] Ignore the fact that a real projectile would hit the ground somewhere... imagine that you are shooting into the Grand Canyon from a small neighboring valley.