Computational Physics Exercise 5: Molecular Dynamics Simulations

Fynn Janssen 411556[1] and Tamilarasan Ketheeswaran 411069[2]

April 2023

Contents

[1] fynn.janssen@rwth-aachen.de
[2] tamilarasan.ketheswaran@rwth-aachen.de

1 Introduction

In this exercise, we use multiple integration algorithms, such as the Euler-, Euler-Cromer-, and Velocity-Verlet, to solve differential equations. In tasks 1-3, we use 4 different methods to solve a second-order differential equation, compare the results and discuss why some methods perform better. The studied system is a one-dimensional harmonic oscillator. Finally, in task 4, we discuss a system of coupled oscillators and solve this system with the Velocity-Verlet algorithm for different discretizations, number of particles, and initial conditions.

The code is written in the programming language `python`, and the packages `numpy`[1] as well as `matplotlib` are used. Furthermore, all constants, such as mass or spring constant are equal to one.

2 Task 1: The Euler Algorithm

2.1 Simulation Model and Method

We examine the one-dimensional harmonic oscillator with the Euler method. The system is described by Newton's equation of motion

$$m\ddot{x}(t) = F(x(t)) = -kx(t) \;\Leftrightarrow\; \ddot{x}(t) = -x(t), \tag{1}$$

where we use $m = k = 1$, as already mentioned. At first, we rewrite this second-order differential equation as two coupled first-order differential equations.

$$\dot{x}(t) = v(t) \tag{2}$$
$$\dot{v}(t) = F(x(t)) = -x(t) \tag{3}$$

Now, to integrate this equation numerically we can apply the Euler method, which describes the differential equations as

$$\begin{aligned} x(t+\Delta t) &= x(t) + \Delta t \cdot v(t) + \mathcal{O}\left(\Delta t^2\right) \\ v(t+\Delta t) &= v(t) + \Delta t \cdot F(x(t)) + \mathcal{O}\left(\Delta t^2\right). \end{aligned} \tag{4}$$

Here, Δt is a discrete time-step that has to be chosen appropriately to obtain reasonable results. Since eq.(4) is a first-order Taylor expansion, this method has a local error of order $\mathcal{O}\left(\Delta t^2\right)$. However, the global error of order $\mathcal{O}\left(\Delta t\right)$. Furthermore, the Euler method is not symplectic and not stable, which we expect to find in our results. To implement these equations in Python we do the following replacements

$$t \longrightarrow \texttt{t0+i}\Delta\texttt{t}, \tag{5}$$
$$X(t+\Delta t) \longrightarrow \texttt{X}_{\texttt{i+1}}, \tag{6}$$
$$X(t) \longrightarrow \texttt{X}_{\texttt{i}}, \tag{7}$$

where t_0 is a chosen initial time, t is the time at the i^{th} step, and $X(t)$ is an arbitrary function.[2] Therefore, we obtain the following recursive equations

$$\begin{aligned} \texttt{x}_{\texttt{i+1}} &= \texttt{x}_{\texttt{i}} + \texttt{v}_{\texttt{i}}\Delta\texttt{t} \\ \texttt{v}_{\texttt{i+1}} &= \texttt{v}_{\texttt{i}} - \texttt{x}_{\texttt{i}}\Delta\texttt{t}, \end{aligned} \tag{8}$$

where we already substituted the force $\texttt{F}_{\texttt{i}}\texttt{=-x}_{\texttt{i}}$.[3] Finally, with given initial conditions $\texttt{x}_0$ and $\texttt{v}_0$, eq.(8) can be solved iteratively in a for loop as follows:

[1] abbreviated as `np`

[2] In our case $X(t)$ represents either $x(t)$, $v(t)$ or the force $F(x(t))$.

[3] Note that these equations do not state that this is exact, they simply represent the implementation in the Code.

```python
import numpy as np

x = np.zeros(M)
v = np.zeros(M)
x[0] = x0
v[0] = v0
for i in range(M-1):
    v[i+1] = v[i] + F(x[i])*dt
    x[i+1] = x[i] + v[i]*dt
```

This is a small code snippet to demonstrate how we generate the solution $x(t)$ as an array. We first initialize the arrays `x` and `v` as a `numpy` array with M zeros and set the initial conditions. Here, M is the number of entries in the position, and velocity arrays. Note, that we iterate over $M-1$ steps since we calculate `x[i+1]`.

Before presenting and discussing the result, we introduce the analytical solution, given by

$$x(t) = x_0 + v_0 \sin(t - t_0), \tag{9}$$

such that $x(t_0) = x_0$ and $v(t_0) = v_0$.

2.2 Results

In this subsection, we present the results for the initial conditions $t_0 = 0$, $x_0 = 0$, and $v_0 = 1$. Furthermore, we investigate the discretizations $\Delta t = 0.1, 0.01, 0.001$ and $M = 10000/\Delta t + 1$.[4] Thus, we solve the system for the discrete time interval $t \in \{0, \Delta t, 2\Delta t, ..., 10000\}$. In fig.(1), we present the entire solution on the left and a small clipping on the right, to see the oscillations. Every plot contains the theoretical prediction $x_{theory}(t) = \sin(t)$ represented by a dashed black line.

2.3 Discussion

In fig.(1) we can see that the Euler algorithm strongly deviates from the theoretical predictions. We find that for increasing time the deviations also increase. As already mentioned, the Euler method has a local error of order $\mathcal{O}\left(\Delta t^2\right)$ and a global error of order $\mathcal{O}\left(\Delta t\right)$. Especially, the Euler method is not symplectic and, therefore, unstable.

From a physical point of view, we can understand the increase in amplitude over time by investigating the energy. To this end, we calculate the change in energy after one step

$$\Delta E_i = E_{i+1} - E_i. \tag{10}$$

The energy at the i^{th} step is given by

$$E_i = E_{kin,i} + V_i = \frac{v_i^2}{2} + \frac{x_i^2}{2}, \tag{11}$$

where we already used that $m = k = 1$. To calculate E_i, we use eq.(8), which yields

$$\begin{aligned} E_{i+1} &= \frac{v_{i+1}^2}{2} + \frac{x_{i+1}^2}{2} \\ &= \frac{1}{2}\left((v_i - x_i\Delta t)^2 + (x_i + v_i\Delta t)^2\right) \\ &= \frac{1}{2}\left(v_i^2 + x_i^2\Delta t^2 + x_i^2 + v_i^2\Delta t^2\right) \\ &= E_i\left(1 + \Delta t^2\right). \end{aligned} \tag{12}$$

[4] Note that M refers to the number of entries of the time, position, and velocity arrays. This is not to be confused with the number of iterations in the `for loop` in the algorithm. Further, the $+1$ in M is needed to account for the first value $t = 0$.

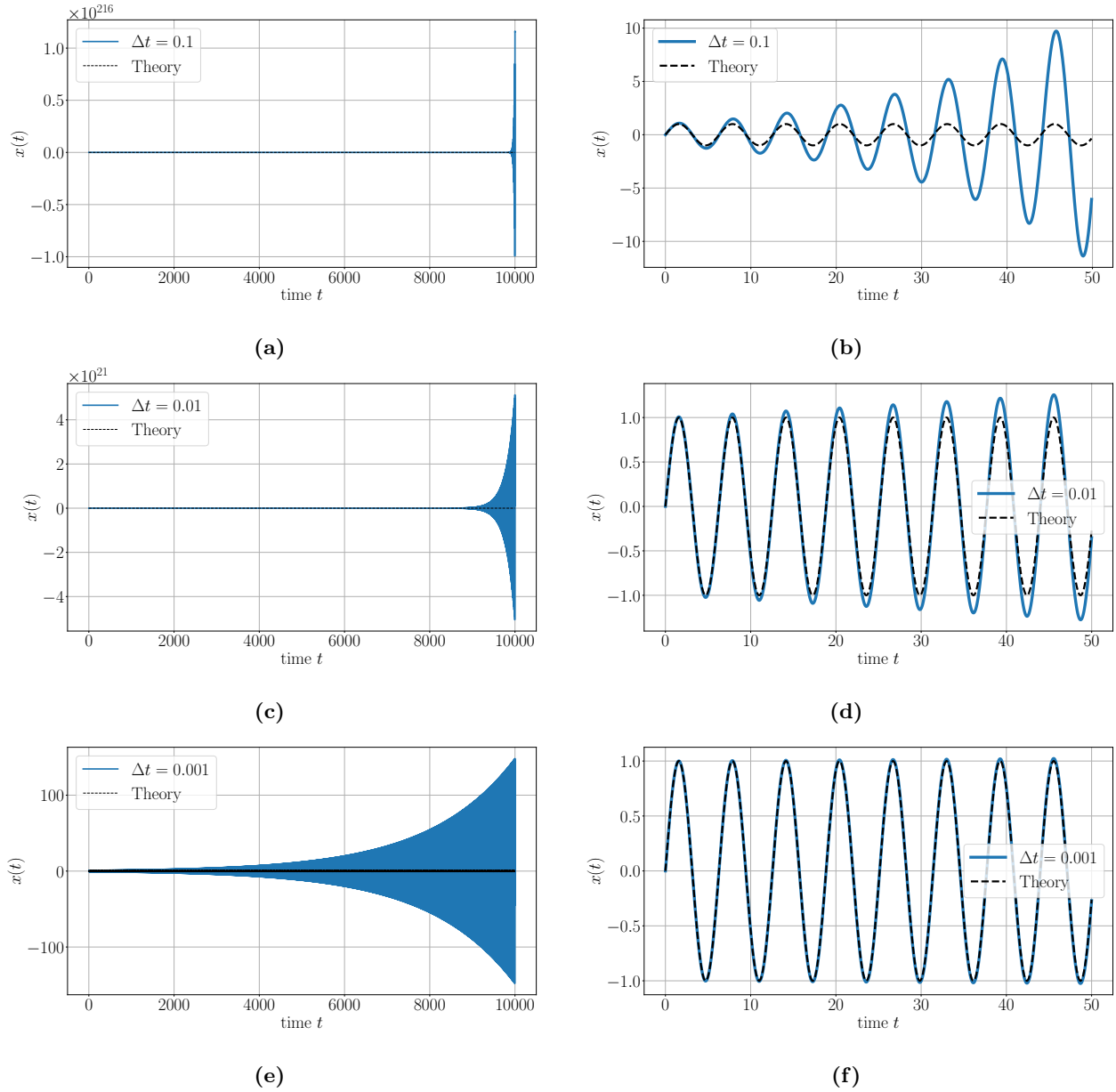


Figure 1: Numerical solutions of the harmonic oscillator for initial conditions $t_0 = 0$, $x_0 = 0$, as well as $v_0 = 1$, and the number of discrete time steps $M = 10000/\Delta t + 1$. The discretizations are (a) and (b) $\Delta t = 0.1$, (c) and (d) $\Delta t = 0.01$, as well as (e) and (f) $\Delta t = 0.001$. Further, the plots on the right-hand side ((b), (d), (f)) only show the solutions for the times $t \in \{0, \Delta t, ..., 50\}$.

This recursive series has the exact solution

$$E_i = E_0 \cdot \left(1 + \Delta t^2\right)^i, \tag{13}$$

such that the energy difference after the i^{th} step is given by

$$\Delta E_i = E_i \Delta t^2 = E_0 \cdot \left(1 + \Delta t^2\right)^i \cdot \Delta t^2 > 0. \tag{14}$$

Therefore, after each step, the energy of the system increases which leads to an increasing amplitude. Here, E_0 is the expected energy, coinciding with the numerical solution for $i = 0$. This shows that the Euler method is not symplectic and not stable.

We can conclude that the Euler algorithm is not useful for this application due to the large global error and the increase in energy over time. Thus, systems with large simulation times t have increasingly large errors.

3 Task 2: The Euler-Cromer Algorithm

3.1 Simulation Model and Method

In the second task, we discuss two different implementations of the Euler-Cromer algorithm, which is an extension of the Euler algorithm discussed in task 1. In order to compare the results of this task with the results of task 1, we discuss the same differential equation describing a one-dimensional harmonic oscillator, with $m = k = 1$.

The two implementations of the Euler-Cromer algorithm are given by

$$\textbf{(a)}$$
$$\begin{aligned} v\left((i+1)\Delta t\right) &= v\left(i\Delta t\right) + F\left(i\Delta t\right)\Delta t \\ x\left((i+1)\Delta t\right) &= x\left(i\Delta t\right) + v\left((i+1)\Delta t\right)\Delta t \end{aligned} \tag{15}$$

and

$$\textbf{(b)}$$
$$\begin{aligned} x\left((i+1)\Delta t\right) &= x\left(i\Delta t\right) + v\left(i\Delta t\right)\Delta t \\ v\left((i+1)\Delta t\right) &= v\left(i\Delta t\right) + F\left((i+1)\Delta t\right)\Delta t, \end{aligned} \tag{16}$$

where the differences compared to the Euler algorithm are highlighted by colors. As in task 1, the force is given by $F(x(i\Delta t)) = -x(i\Delta t)$. Similar to the Euler algorithm, the local error is order $\mathcal{O}\left(\Delta t^2\right)$ and the global error is order $\mathcal{O}\left(\Delta t\right)$. However, other than the Euler method, this algorithm is symplectic and, therefore, stable.

Now, we can rewrite this as a recursive series and substitute the force, similar to eq.(8).

$$\textbf{(a)}$$
$$\begin{aligned} \texttt{v}_{\texttt{i+1}} &= \texttt{v}_\texttt{i} - \texttt{x}_\texttt{i}\Delta\texttt{t} \\ \texttt{x}_{\texttt{i+1}} &= \texttt{x}_\texttt{i} + \texttt{v}_{\texttt{i+1}}\Delta\texttt{t}, \end{aligned} \tag{17}$$

and

$$\textbf{(b)}$$
$$\begin{aligned} \texttt{x}_{\texttt{i+1}} &= \texttt{x}_\texttt{i} + \texttt{v}_\texttt{i}\Delta\texttt{t} \\ \texttt{v}_{\texttt{i+1}} &= \texttt{v}_\texttt{i} - \texttt{x}_{\texttt{i+1}}\Delta\texttt{t}. \end{aligned} \tag{18}$$

This is implemented in the same way as the Euler method discussed in task 1, where we loop over $M - 1$ iterations to obtain the arrays for `x` and `v`. Finally, with the resulting arrays we can calculate the energy according to eq.(11).

3.2 Results

Now, we present the results for the two implementations described by eqs. (17) and (18). We choose a time step of $\Delta t = 0.01$ and $M = 10/\Delta t + 1 = 1000 + 1$ steps. Thus, the time array contains the entries $t \in \{0, \Delta t, 2\Delta t, ..., 10\}$. In fig.(2), the results for implementation (a) (compare eq.(15)) are shown on the left-hand side and the results for implementation (b) (compare eq.(16)) on the right-hand side. In the first row of fig.(2) (a) and (b), the results for the position $x(t)$, velocity $v(t)$, and energy $E(t)$ are presented. Furthermore, these plots include the theoretical prediction for the position $x_{theory}(t) = \sin(t)$, represented by the black dashed line. In the second, we present the difference between the calculated energy $E(t)$ and the theoretical prediction $E_{theory} = 1/2$.

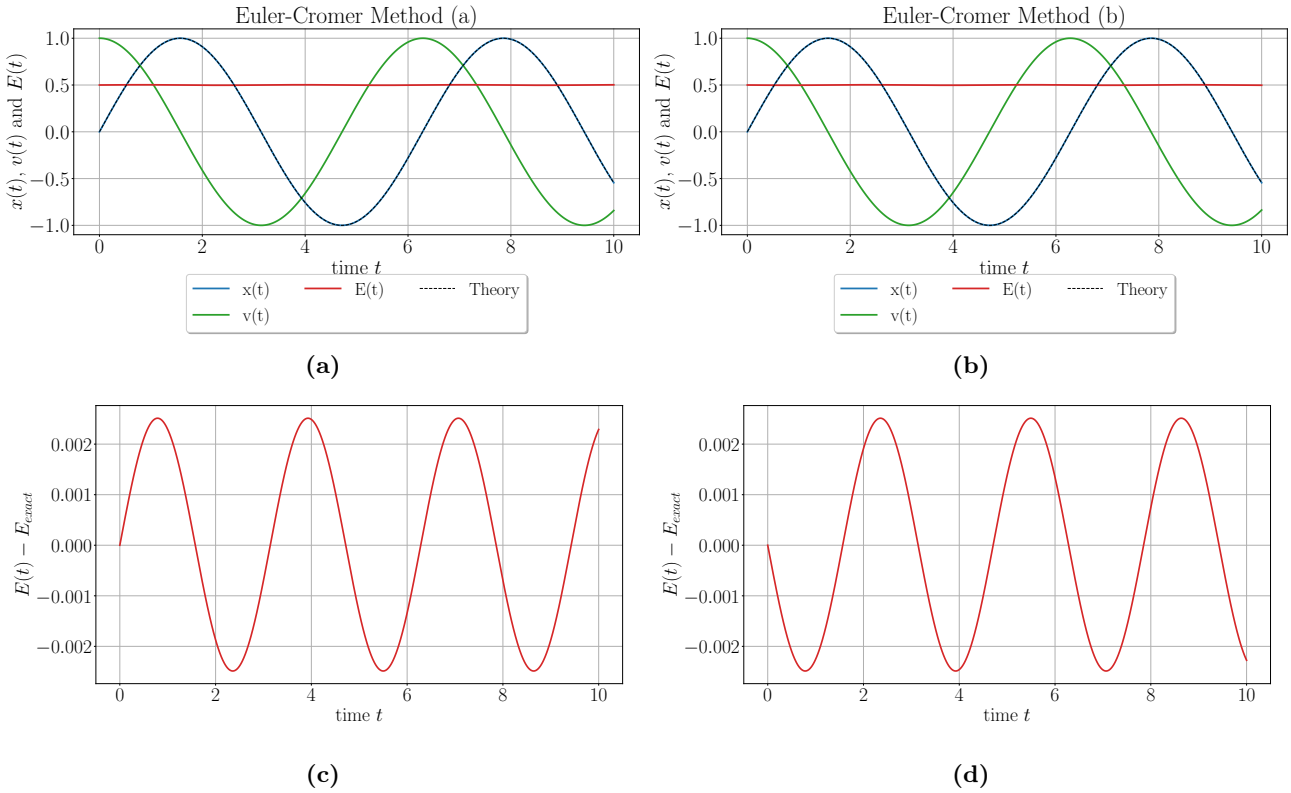
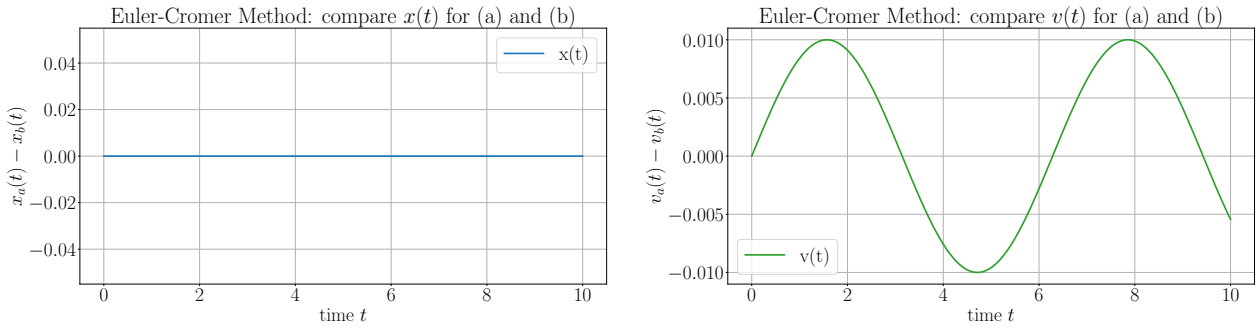


Figure 2: Numerical solutions of the harmonic oscillator by using two implementations of the Euler-Cromer algorithm (see eqs. (17) and (18)) for initial conditions $t_0 = 0$, $x_0 = 0$, as well as $v_0 = 1$, and the number of iterations $M - 1 = 10/\Delta t = 1000$. Hence, each array contains M values. figures (a) and (c) show the results for method (a) while plots (b) and (d) for method (b). Figures (a) and (b) contain the numerical solutions for the position $x(t)$, the velocity $v(t)$, and energy $E(t)$, as well as the theoretical prediction for the position $x_{theory} = \sin(t)$. Figures (c) and (d) contain the deviations of the energy from the theory $E(t) - 1/2$.

Furthermore, to compare the two different implementations of the Euler-Cromer method, the differences between the resulting position and velocity arrays are presented in fig.(3). Here, fig.(3a) shows the difference in the position, while fig.(3b) shows the difference in the velocity arrays.



(a) **(b)**

Figure 3: Comparison of the resulting position arrays (a) and velocity arrays (b) for the two different implementations of the Euler-Cromer method.

Additionally, we present results for a duration time of $t = 1000$ and the same time step as before $\Delta t = 0.01$. Thus, we have $N = 1000/\Delta t + 1 = 100000 + 1$ time steps. These results are presented in fig.(4). The purpose of these plots is to investigate whether the algorithm is stable or not, or more precisely, to study the amplitude of $x(t)$ and $E(t) - E_{theory}$ for larger times t.

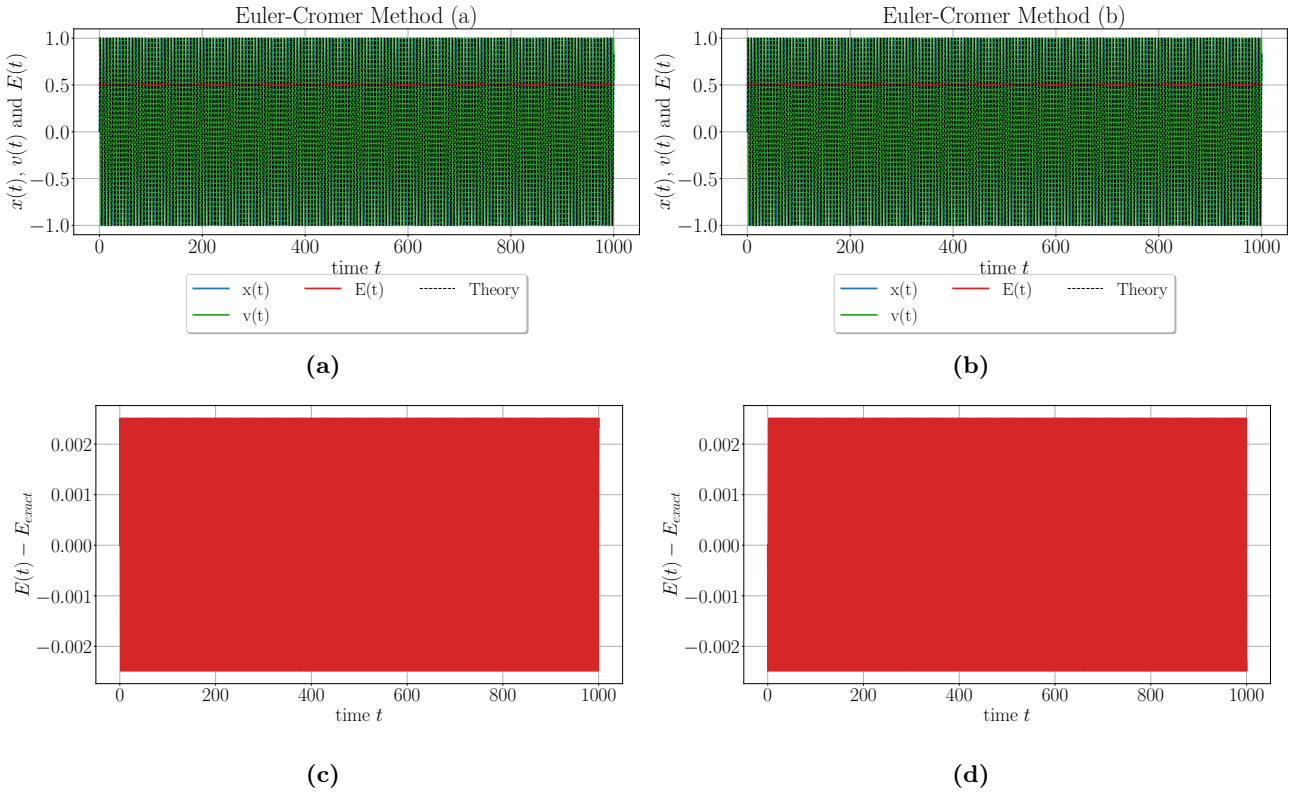


Figure 4: Numerical solutions of the harmonic oscillator by using two implementations of the Euler-Cromer algorithm (see eqs. (17) and (18)) for initial conditions $t_0 = 0$, $x_0 = 0$, as well as $v_0 = 1$, and the number of iterations $M - 1 = 1000/\Delta t = 100000$. Figs. (a) and (c) show the results for implementation (a) while plots (b) and (d) for implementation (b). Figures (a) and (b) contain the numerical solutions for the position $x(t)$, the velocity $v(t)$, and energy $E(t)$, as well as the theoretical prediction for the position $x_{theory} = \sin(t)$. Figures (c) and (d) the deviations from the theory for the energy $E(t) - 1/2$.

3.3 Discussion

First of all, we compare the results for the different simulation times $t = 10$ and $t = 1000$ presented in figs. (2) and (4), respectively. In figs. (2a) and (2b), we can see that the simulation result seems to be in good approximation with the theoretical prediction. Neither the position nor velocity seem to increase over time. Further, the energy is not exactly constant, but oscillates around the theoretical prediction and, also, does not seem to increase, as we can see in figs. (2c) and (2d). The stability of this method can be seen in fig.(4), in which we have a stable amplitude for a long duration ($t = 1000$) and the deviation of the energy does not increase. This is significantly different from the Euler algorithm. Based on these results we conclude that the Euler-Cromer algorithm is stable.

Furthermore, we can describe the energy of the system by calculating $\Delta E_i = E_{i+1} - E_i$, as we did for the Euler method.

$$
\begin{aligned}
&\textbf{(a)}\\
E_{i+1} &= \frac{1}{2}\left(x_{i+1}^2 + v_{i+1}^2\right)\\
&= \frac{1}{2}\left[(x_i + v_{i+1}\Delta t)^2 + (v_i - x_i\Delta t)^2\right]\\
&= \frac{1}{2}\left(x_i^2 + 2x_i v_{i+1}\Delta t + v_{i+1}^2\Delta t^2 + v_i^2 - 2x_i v_i\Delta t + x_i^2\Delta t^2\right)\\
&= E_i + x_i\Delta t\left(v_{i+1} - v_i\right) + \frac{1}{2}\Delta t^2\left(v_{i+1}^2 + x_i^2\right),\\
\text{now we use } v_{i+1} - v_i &= -x_i\Delta t\\
E_{i+1} &= E_i + \frac{\Delta t^2}{2}\left(v_{i+1}^2 - x_i^2\right)\\
\Rightarrow \Delta E_i &= \frac{\Delta t^2}{2}\left(v_i^2 - x_i^2 - 2x_i v_i\Delta t + \Delta t^2 x_i^2\right)
\end{aligned}
\tag{19}
$$

and

$$
\begin{aligned}
&\textbf{(b)}\\
E_{i+1} &= \frac{1}{2}\left(x_{i+1}^2 + v_{i+1}^2\right)\\
&= \frac{1}{2}\left[(x_i + v_i\Delta t)^2 + (v_i - x_{i+1}\Delta t)^2\right]\\
&= \frac{1}{2}\left(x_i^2 + 2x_i v_i\Delta t + v_i^2\Delta t^2 + v_i^2 - 2x_{i+1} v_i\Delta t + x_{i+1}^2\Delta t^2\right)\\
&= E_i - v_i\Delta t\left(x_{i+1} - x_i\right) + \frac{1}{2}\Delta t^2\left(v_i^2 + x_{i+1}^2\right),\\
\text{use } x_{i+1} - x_i &= v_i\Delta t\\
E_{i+1} &= E_i + \frac{\Delta t^2}{2}\left(x_{i+1}^2 - v_i^2\right)\\
\Rightarrow \Delta E_i &= \frac{\Delta t^2}{2}\left(x_i^2 - v_i^2 + 2x_i v_i\Delta t + \Delta t^2 v_i^2\right)
\end{aligned}
\tag{20}
$$

The resulting ΔE_i are very similar for the different implementations, which becomes more clear if we write it in the following way.

$$
\begin{aligned}
\textbf{(a)}\ \Delta E_i &= \frac{\Delta t^2}{2}\left(v_i^2 - x_i^2 - 2x_i v_i\Delta t\right) + \frac{\Delta t^4}{2}x_i^2\\
\textbf{(b)}\ \Delta E_i &= -\frac{\Delta t^2}{2}\left(v_i^2 - x_i^2 - 2x_i v_i\Delta t\right) + \frac{\Delta t^4}{2}v_i^2
\end{aligned}
\tag{21}
$$

Here we can see that the first term is identical up to a minus sign and in the second term we have a x_i^2 for implementation (a) and a v_i^2 for implementation (b). Since the local error in $x(t)$ and $v(t)$ of this algorithm is of order $\mathcal{O}(\Delta t^2)$, we can neglect the terms of order $\mathcal{O}(\Delta t^3)$ and higher in ΔE_i. This implies that ΔE_i leads to the same results for both implementations up to a minus sign and errors of order $\mathcal{O}(\Delta t^2)$ in x_i and v_i. Furthermore, the time average of x_i^2 and v_i^2 are approximately equal[5] and, therefore, the time average of ΔE_i vanishes. Furthermore, if we neglect the terms of order $\mathcal{O}(\Delta t^3)$ and higher, ΔE_i is identical for both implementations up to a minus sign. This corresponds to a phase shift of π as we can see in figs. (2c) and (2d). In summary, we can conclude that the energy oscillates around the expected value but does not increase. Hence, the oscillating solutions $x(t)$ of both methods also remain stable and do not diverge as in the Euler algorithm.

At last, we compare the different implementations directly by discussing fig.(3). We find that the position arrays $x(t)$ are identical while the velocity arrays $v(t)$ deviate from each other. This is due to the fact that we chose the initial conditions $v_0 = 1$ and especially $x_0 = 0$. In the implementation (a) (see eq.(17)), we first calculate v_{i+1}. Thus, since $x_0 = 0$ we find $v_1 = v_0$, which is then used to determine $x_1 = v_1\Delta t = v_0\Delta t$ in x_{i+1}. In implementation (b) (see eq.(18)) we first calculate x_{i+1}. Here, $x_1 = v_0\Delta t$, which is exactly the same as in implementation (a). Then, in (b) we continue with v_{i+1}. In summary, $x(t)$ is only identical for both implementations due to the initial condition $x_0 = 0$. In general, the implementations are not exactly identical but have the same properties such as a local [global] error of order $\mathcal{O}(\Delta t^2)$ $[\mathcal{O}(\Delta t)]$, both are not time-reversible, as well as, symplectic and, therefore, stable.

Overall, the Euler-Cromer algorithm yields better results than the Euler-algorithm because it is symplectic and stable. However, the downside of the Euler-Cromer algorithm is, that it has a local [global] error of order $\mathcal{O}(\Delta t^2)$ $[\mathcal{O}(\Delta t)]$ and it is not time-reversible. Thus, this algorithm performs well, but there are better algorithms, which are also simple and fast.

[5] up to numerical uncertainties, as mentioned

4 Task 3: The Velocity-Verlet Algorithm

4.1 Simulation Model and Method

In the third exercise, one was ought to solve the harmonic oscillator using the velocity Verlet algorithm. The results obtained should then be compared to the algorithm used in task 2. The time evolution of the position x and velocity v from t to $t + \Delta t$ is according to the script given by

$$\begin{aligned} x(t+\Delta t) &= x(t) + v(t)\Delta t + \frac{1}{2}a(t)(\Delta t)^2 + \mathcal{O}\left(\Delta t^3\right) \\ v(t+\Delta t) &= v(t) + \frac{1}{2}\left[a(t) + a(t+\Delta t)\right]\Delta t + \mathcal{O}\left(\Delta t^3\right), \end{aligned} \tag{22}$$

where $a(t) = -x(t)$, the same acceleration as defined in eq.(1).

Similar to tasks 1 and 2, to implement this in Python we can write this as a recursive series

$$\begin{aligned} \texttt{x}_{\texttt{i+1}} &= \texttt{x}_\texttt{i} + \texttt{v}_\texttt{i}\Delta\texttt{t} - \frac{1}{2}\texttt{x}_\texttt{i}\Delta\texttt{t}^2 \\ \texttt{v}_{\texttt{i+1}} &= \texttt{v}_\texttt{i} - \frac{1}{2}\left(\texttt{x}_\texttt{i} + \texttt{x}_{\texttt{i+1}}\right)\Delta\texttt{t}. \end{aligned} \tag{23}$$

Here we already substituted the acceleration. With `np.zeros(M)`, we create an array consisting of only zeros of the length M. This is done twice, once for the position `x` and once for the velocity `v`. In this array we will save the time evolution of the system. Each element of the array with index i corresponds to the time $t = i\Delta t$ passed. Hence the time difference between to consequent elements is equal to Δt. We impose initial conditions to the system by `x[0]` $= x_0$ and `v[0]` $= v_0$. The algorithm itself is implemented using one `python-function` which calculates the $x((i+1)\Delta t)$ and $v((i+1)\Delta t)$ according to eq.(22).[6] This function is called iteratively by a `for-loop` `M-1` times, such that with each step in the loop Δt has passed. A small code snippet of the `python-code` then looks like this:

```
def Velocity_Verlet(x,v):
        xdt= x + v*dt       + 1/2*a(x)*dt**2
        vdt= v + 1/2*(a(x)  + a(xdt))*dt
        return xdt,vdt

for i in range(M-1):
    x[i+1],v[i+1] = Velocity_Verlet(x[i],v[i])
```

Here `xdt` and `vdt` correspond to $x((i+1)\Delta t)$ and $v((i+1)\Delta t)$, respectively.

Fruther, we calculate the energy of the system at each time according to eq.(11).
The results we obtain for the position are compared to the expectation given in eq.(9). The array containg the time t is initialized using `t = np.linspace(0,(M-1)*`Δt`,M)`. This yields us an array where with each step the time Δt elapses - `M`-times.

4.2 Results

In the following, we present the results of the implementation described by eq.(22) for the initial conditions $x_0 = 0$, $v_0 = 1$ and $t_0 = 0$. Furthermore, we use the time steps $\Delta t = 0.1(0.01)$ and $M = 10/\Delta t + 1 = 101(1001)$. Note, that we need 101(1001) entries and not 100(1000), because we take 100(1000) steps and thus need 101(1001) entries in the array - containing also the initial condition. Hence, we consider the times $t \in \{0, \Delta t, ..., 10\}$.

[6] it is important to calculate the $x((i+1)\Delta t)$ before $v((i+1)\Delta t)$, as for the calculation of latter one needs the result of $x((i+1)\Delta t)$

In fig.(5), the results for $\Delta t = 0.1$ with $M = 101$ are shown on the left-hand side, and the results for $t = 0.01$ with $M = 1001$ on the right-hand side. In the first row of fig.(5) (a) and (b), the results for the position $x(t)$, velocity $v(t)$, and energy $E(t)$ are presented. Furthermore, these plots include the theoretical prediction for the position $x_{theory}(t) = \sin(t)$, represented by the black dashed line. In the second row, we present the difference between the calculated energy $E(t)$ and the theoretical prediction $E_{theory} = 1/2$.

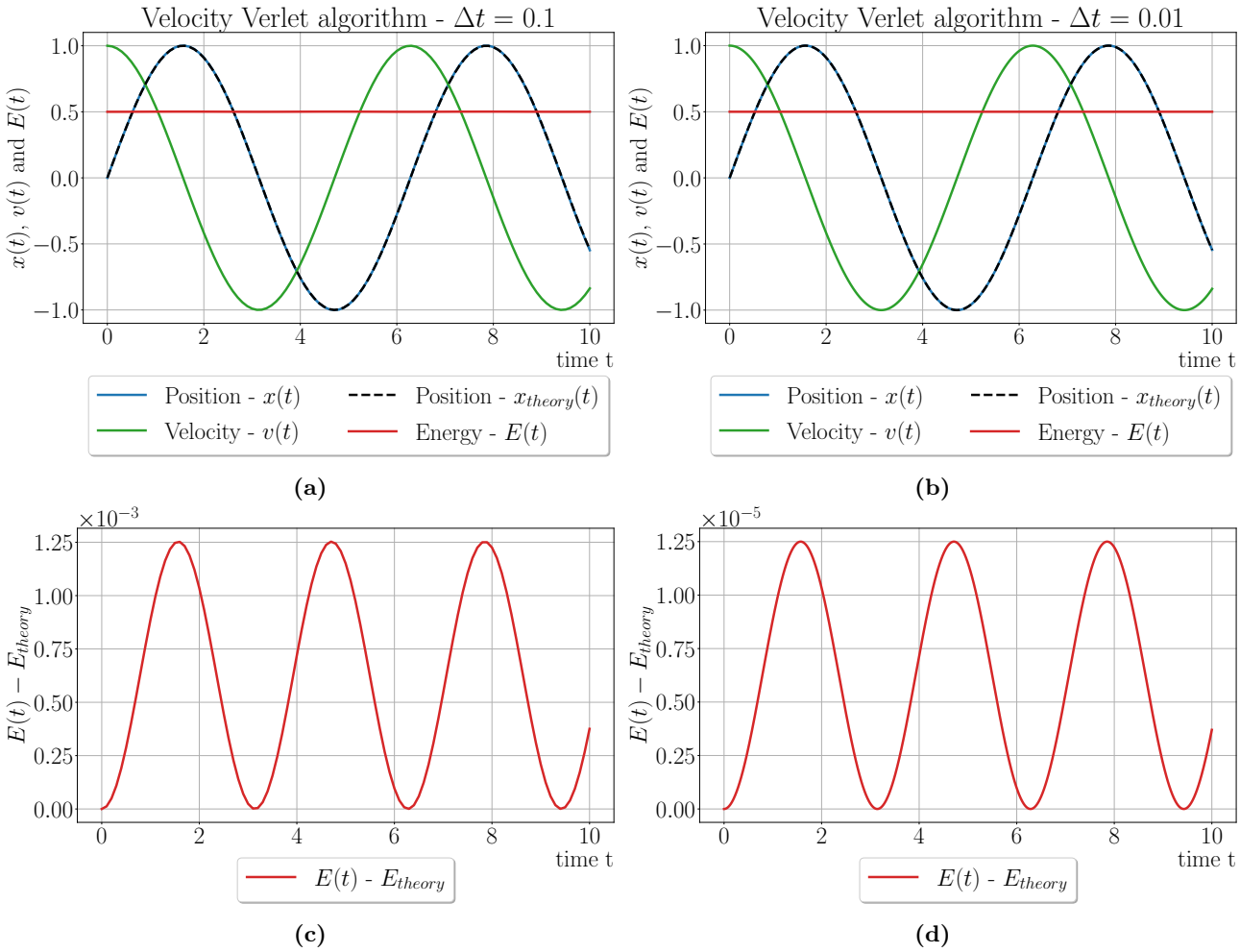


Figure 5: Numerical solutions of the harmonic oscillator by using the velocity Verlet algorithm (see eq.(22)) for initial conditions $t_0 = 0$, $x_0 = 0$, as well as $v_0 = 1$, and the number of iterations $M - 1 = 100(1000)$. Figures (a), (c) show the results for $\Delta t = 0.1$ while plots (b), (d) for $\Delta t = 0.01$. Figures (a) and (b) contain the numerical solutions for the position $x(t)$, the velocity $v(t)$, and energy $E(t)$, as well as the theoretical prediction for the position $x_{theory} = \sin(t)$. Figures (c) and (d) contain the deviation of the energy from the theory $E(t) - E_{theory} = E(t) - 1/2$.

4.3 Discussion

In the results presented in fig.(5)((a) and (b)), we can see that the position $x(t)$ is in very good agreement with the theory $x_{theory}(t) = \sin(t)$ for both discretizations $\Delta t = 0.1$ and $\Delta t = 0.01$. Also, the velocity in both cases is just a shifted function compared to the positions. This makes sense as in theory the velocity v is just the first derivative of the position x, and hence must be a cosine function. Also, in both cases from plots (a) and (b), the energy seems constant for both discretizations. As the expectation for the energy is given by $E = \frac{1}{2}$, we subtracted this value from our calculation for the energy and presented it in plots (c) and (d). Here we can see that the difference between simulation and theory oscillates around a non-zero value.

The minima of this oscillation coincide with 0. However, on average the calculated energy is greater than the theoretical prediction by a small amount.

This implies that the total energy on average is not given by $\frac{1}{2}$. There is an offset to the expectation value depending on the height of the energy amplitude. Interesting, however, is the fact that the minima of the difference are zero.

Thus, the energy of the system has a lower boundary given by the theoretical expectation.

The deviation magnitude depends on the discretization Δt. For $\Delta t = 0.1$ the deviation has a magnitude of 10^{-3} and for $\Delta t = 0.01$ of 10^{-5}. Hence, for decreasing Δt the results of the algorithm deviate less from the theoretical prediction for energy, which was expected. But in any case, the overall magnitude of the energy does not change apart from fluctuations, which is a direct consequence of the velocity Verlet algorithm being symplectic. As the fluctuation does not change with time the algorithm is also stable.

Comparing this method to the Euler-Cromer algorithm, we can see that both methods yield good results for position and velocity. However, there is a difference in the calculated energy. While the Euler-Cromer gives us an energy that equals the expectation if averaged over time, the velocity Verlet algorithm yields us a result that is equal to the expectation plus the offset discussed before. But in both cases, we have an algorithm that is symplectic and stable. However, the velocity-Verlet algorithm has some advantages. It has a local [global] error of order $\mathcal{O}(\Delta t^3)$ $[\mathcal{O}(\Delta t^2)]$ and it is time-reversible, which is both not the case for the Euler-Cromer algorithm. Due to these properties and the still fast computation time, this algorithm is most widely used in molecular dynamics.

5 Task 4: System of Coupled Oscillators

5.1 Simulation Model and Method

In the fourth and final task, one was ought to implement the velocity Verlet algorithm to solve the equation of motion of N coupled oscillators. The Hamiltonian for the coupled system is given by

$$H = K + V = \frac{1}{2}\sum_{n=1}^{N} v_n^2 + \frac{1}{2}\sum_{n=1}^{N-1}(x_n - x_{n+1})^2 \tag{24}$$

where the forces acting on particle n are given by

$$\begin{aligned} F_n &= -\frac{\partial V}{\partial x_n} = -(2x_n - x_{n-1} - x_{n+1}), \quad 1 < n < N, \\ F_1 &= -\frac{\partial V}{\partial x_1} = -(x_1 - x_2), \\ F_N &= -\frac{\partial V}{\partial x_N} = -(x_N - x_{N-1}). \end{aligned} \tag{25}$$

The first and last oscillators in this system are only coupled one-sided, while all the others are coupled on two sides.

As `python` starts indexing the arrays at 0, there is a shift of 1 in the code for each oscillator. The position and velocity of each particle will be saved into two separate `numpy-arrays`. We initialize the array with `np.zeros((N,M))`, which returns us an array of shape `(N,M)` containing zeros. The `N` depicts the number of oscillators N, while the `M` is the length of the time interval we consider. Hence, we do $M-1$ iterations where each iteration corresponds to an elapse of time Δt. The initial conditions can be implemented by imposing the condition to the first entry of the corresponding `python` array `v` and `x`. The forces defined in eq.(25), can be implemented by a `python-function` with input values `(n,i)`, where `n` indexes the oscillator and `i` the iteration. Hence given the oscillator at time t

$$t \longrightarrow \texttt{i}\Delta\texttt{t},$$

the forces are given by

$$F_n(x_n(t)) \longrightarrow \texttt{Fn(n,i)},$$

with $n = \texttt{n+1}$[7] and

```
def Fn(n,i):
    if n == 0:
        return -(x[0,i]-x[1,i])
    elif n == (N-1):
        return -(x[N-1,i]-x[N-2,i])
    else:
        return -(2*x[n,i] - x[n-1,i] - x[n+1,i])
```

The three different cases for the forces, depending on the oscillator, are distinguished using `if-statements`.

[7] This refers to the fact that the indexing of the particles in the Hamiltonian start at n=1 and in Python we start at `n=0`.

Next, we want to implement the velocity Verlet integration algorithm. With the introduction of a Liouville operator L, in the lecture the following formula for the time evolution of $x_n(t)$ and $v_n(t)$ was derived[8].

$$\begin{pmatrix} x_n(t+\Delta t) \\ v_n(t+\Delta t) \end{pmatrix} \approx \begin{pmatrix} x_n(t) + \Delta t \cdot (v_n(t) + \Delta t \cdot F_n(x_n(t))/2) \\ v_n(t) + \frac{\Delta t}{2} \cdot (F_n(x_n(t)) + F_n(x_n(t+\Delta t))) \end{pmatrix} \tag{26}$$

In eq.(26), on can see that in order to calculate $x_n(t+\Delta t)$ one does need all the position $x_n(t)$ and velocity $v_n(t)$ at time t. For the calculation of $v_n(t+\Delta t)$, however, we need the $x_n(t)$, $v_n(t)$ as well as the position $x_n(t+\Delta t)$ at the time $t+\Delta t$.[9] Eq. (26), can be transferred into `python` in the following way

```
1 def Velocity_Verlet(x,v,i,N):
2         for k in range(N):
3             x[k,i+1]= x[k,i]+dt*(v[k,i]+dt/2*Fn(k,i))
4         for k in range(N):
5             v[k,i+1]= v[k,i]+dt/2*(Fn(k,i) + Fn(k,i+1))
6         return x,v
7
8 for i in range(M-1):
9     x,v = Velocity_Verlet(x,v,i,N)
```

The first `for-loop` in line 2 of the code snippet, loops through the oscillators to calculate the $x_n(t+\Delta t)$. The indexing is done by `x[k,i]`, where `k` indexes the oscillator and `i` the iteration, or in other words the time t. `i+1` then corresponds to $t+\Delta t$. The second `for-loop` in line 4 iterates over the velocities. This whole function is called repeatedly by a `for-loop` in line 8, but every time the index `i` is incremented by 1, to indicate that another Δt has elapsed. As long as the index `i` is smaller than `M-1`, this process will be repeated. The array containg the time t is initialized using `t = np.linspace(0,(M-1)*`Δt`,M)`, similar to the exercises before.

5.2 Results and Discussion

In the following, the results for the coupled system of harmonic oscillators of the different initial conditions are presented. We chose $N = 4, 16, 128$, $\Delta t = 0.1(0.01)$ and $M = 101(1001)$.

[8] in the lecture the derivation was done for $r_{n,\alpha}(t)$, $p_{n,\alpha}(t)$. With renaming r into x, and with $m = 1$, the momentum p equals the velocity v. α depicts dimensions of the system, having a 1D oscillator we don't have to differentiate between.

[9] One could implement the code in such a manner that we do another calculation of $x_n(t+\Delta t)$, but doing the same calculation again is not the efficient way

5.2.1 Results of the Coupled Oscillators - Initial Configuration (1)

The initial configuration (1) is given by

$$v_1(0), ..., v_N(0) = 0$$
$$x_1(0), ..., x_N(0) = 0 \qquad \text{except } x_{N/2}(0) = 1.$$

In the following the graphs only contain the solutions for the position $x_k(t)$ for the first and last oscillator, the oscillator with the non-zero initial condition and its immediate neighbors. The results for all the oscillators are given in the appendix.

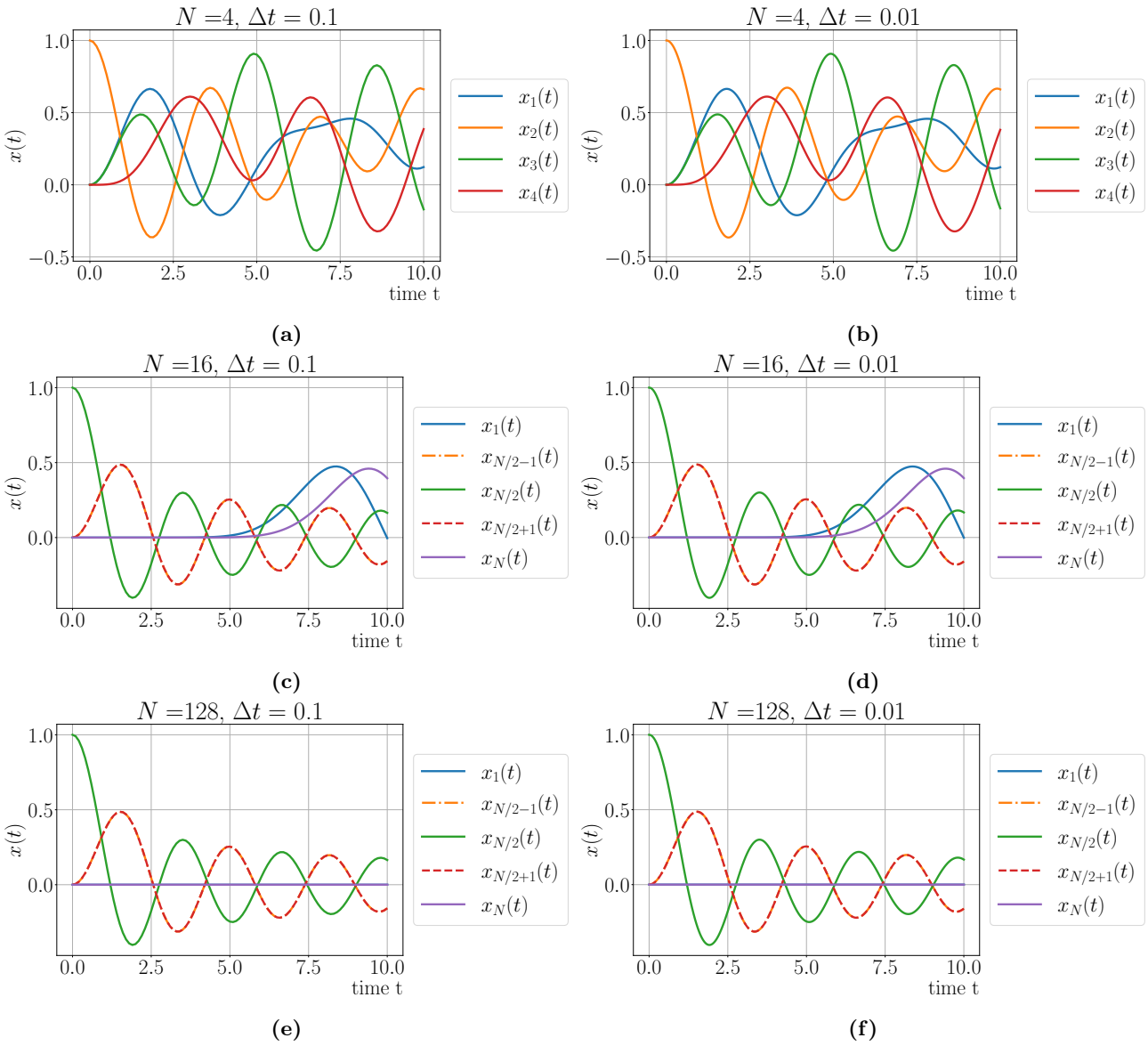


Figure 6: Numerical solutions of the coupled system of harmonic oscillators by using the velocity Verlet algorithm (see (eq.(26))) for initial configuration (1), and the number of iterations $M - 1 = 100(1000)$. The figures in the first column show the results for $\Delta t = 0.1$ in the second for $\Delta t = 0.01$. Figures (a) and (b) are for $N = 4$, (c) and (d) for $N = 16$, and (e) and (f) for $N = 128$. The graphs only contain the solutions for the position $x_k(t)$ for the first and last oscillator, the oscillator with the non-zero initial condition, and its immediate neighbors.

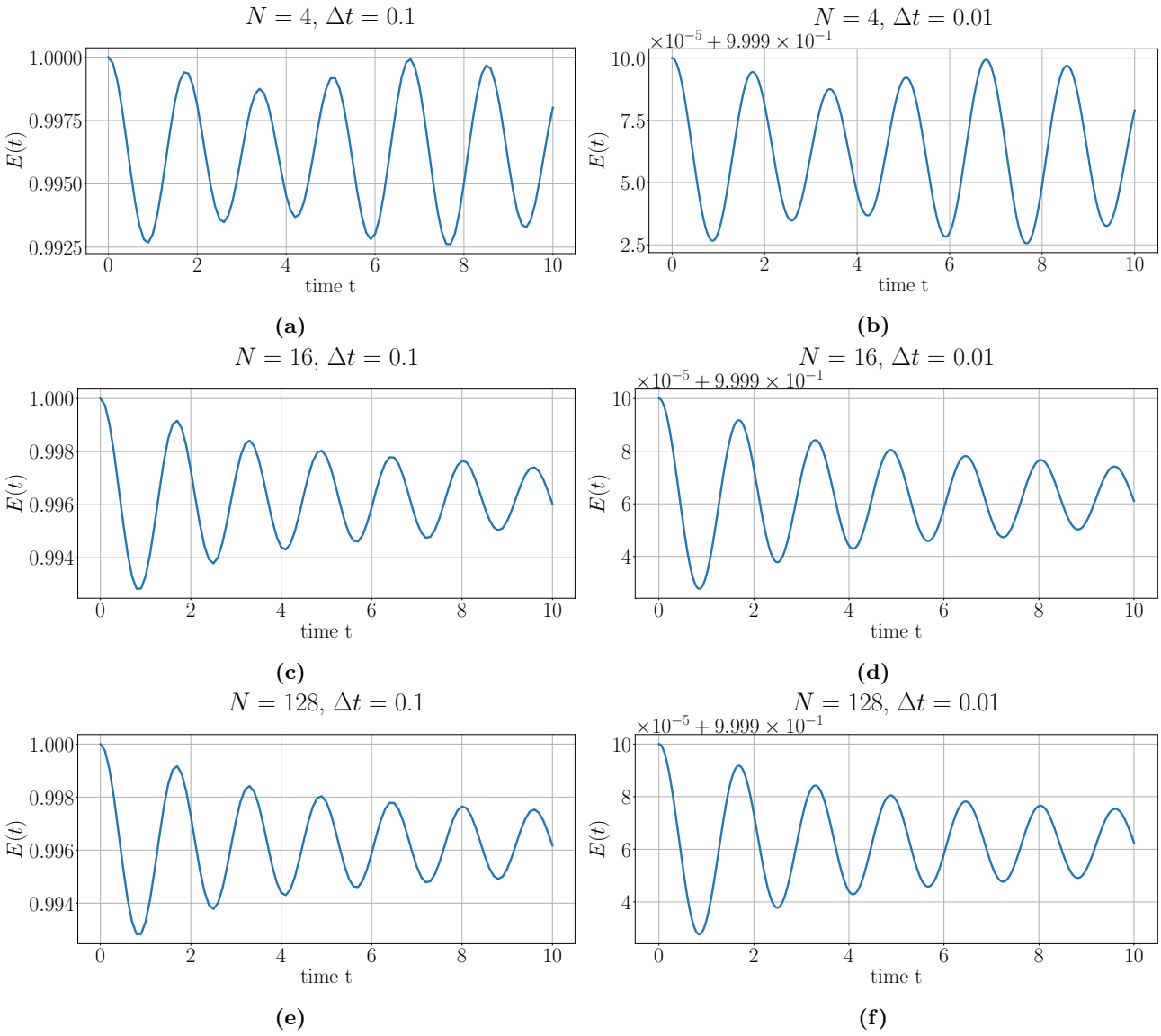


Figure 7: Numerical solutions of the total energy, given by the Hamiltonian, of the coupled system of harmonic oscillators by using the velocity Verlet algorithm (see (eq.(26))) for initial configuration (1), and the number of iterations $M - 1 = 100(1000)$. Figures in the first column show the results for $\Delta t = 0.1$ in the second for $\Delta t = 0.01$. Figures (a) and (b) are for $N = 4$, (c) and (d) for $N = 16$ and (e) and (f) for $N = 128$.

5.2.2 Discussion of the Coupled Oscillators - Initial Configuration (1)

In fig.(6), the results for the initial configuration (1) are given. First, we can see that the results for the different discretizations of Δt do not affect the positions too much. In the case of the system of $N = 4$ oscillators, it is difficult to interpret the result apart from the fact that with time all the oscillators start to oscillate. For $N = 16$ and 128 oscillators, we can see systematics that can be understood physically. At first only the oscillator $x_{N/2}$ is deflected as this was our initial condition. The moment the system is left free to oscillate, we can see that the immediate neighbors of the oscillators ($N/2 + 1$ and $N/2 - 1$) start to move from their initial position. With time all the oscillators start to move from their initial position, but the further they are from the oscillator $N/2$, the longer it takes. This makes sense, as we are considering a system of oscillators, where each one is only connected to their immediate neighbor. Hence, it takes time until the oscillators, further away from $N/2$, are influenced. In the case of the $N = 128$, the first and last oscillators seem to be not moving, because they are

not influenced yet. While for the calculated position array x the result does not change significantly for the different Δt, for the energy of the total system presented in fig.(7), we have a different result. Comparing the energies for different Δt and N, we can see in fig.(7), that the time evolution of the total energy follows a similar shape. For all the N the energy starts at one and oscillates below this value. The variation of the Δt gives us the same oscillation periods, but the amplitude of oscillation is of much less magnitude. This implies that with smaller Δt, the energy at all times gets closer to 1. In (c), (d), (e), and (f) it seems like the energy gets dampened out with time. But in (a) and (b), it seems like the amplitude itself is oscillating too, which gives us reason to believe that at a later time this will also happen for the systems of $N = 16$ and 128. Important to note is the fact that the average of the energy does not change with time. This shows that the simulation we used is symplectic. Comparing this to the previous task, it might be that the average of the total energy over time is again offset from the expectation.

5.2.3 Results of the Coupled Oscillators - Initial Configuration (2), $j = 1$

The initial configuration (2) with $j = 1$ is given by

$$v_1(0), ..., v_N(0) = 0$$
$$x_k(0) = \sin \frac{\pi jk}{N+1} = \sin \frac{\pi k}{N+1} \qquad k = 1, ..., N.$$

Due to the symmetry of the initial conditions, we present the results for the first two, the last two as well as the oscillators in the middle $N/2-1$, $N/2$, $N/2+1$, and $N/2+2$.[10]

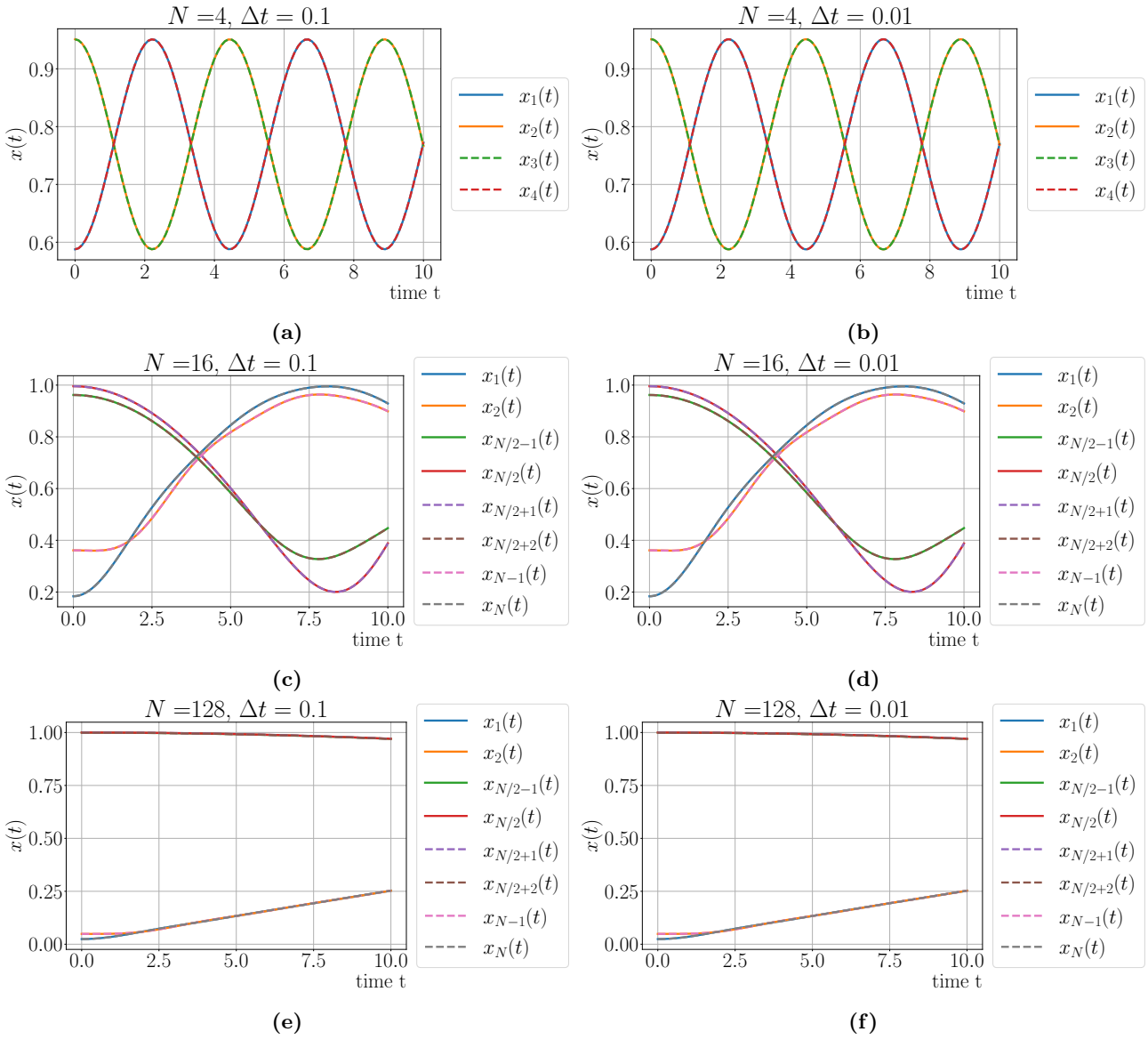


Figure 8: Numerical solutions of positions of the coupled system of harmonic oscillators by using the velocity Verlet algorithm (see (eq.(26))) for initial configuration (2) with $j = 1$, and the number of iterations $M - 1 = 100(1000)$. Figures in the first column show the results for $\Delta t = 0.1$ in the second for $\Delta t = 0.01$. Figures (a) and (b) are for $N = 4$, (c) and (d) for $N = 16$ and (e) and (f) for $N = 128$. The graphs only contains the solutions of the position $x_k(t)$ for a selection of particles.

[10] same color depicts oscillators with same position evolution

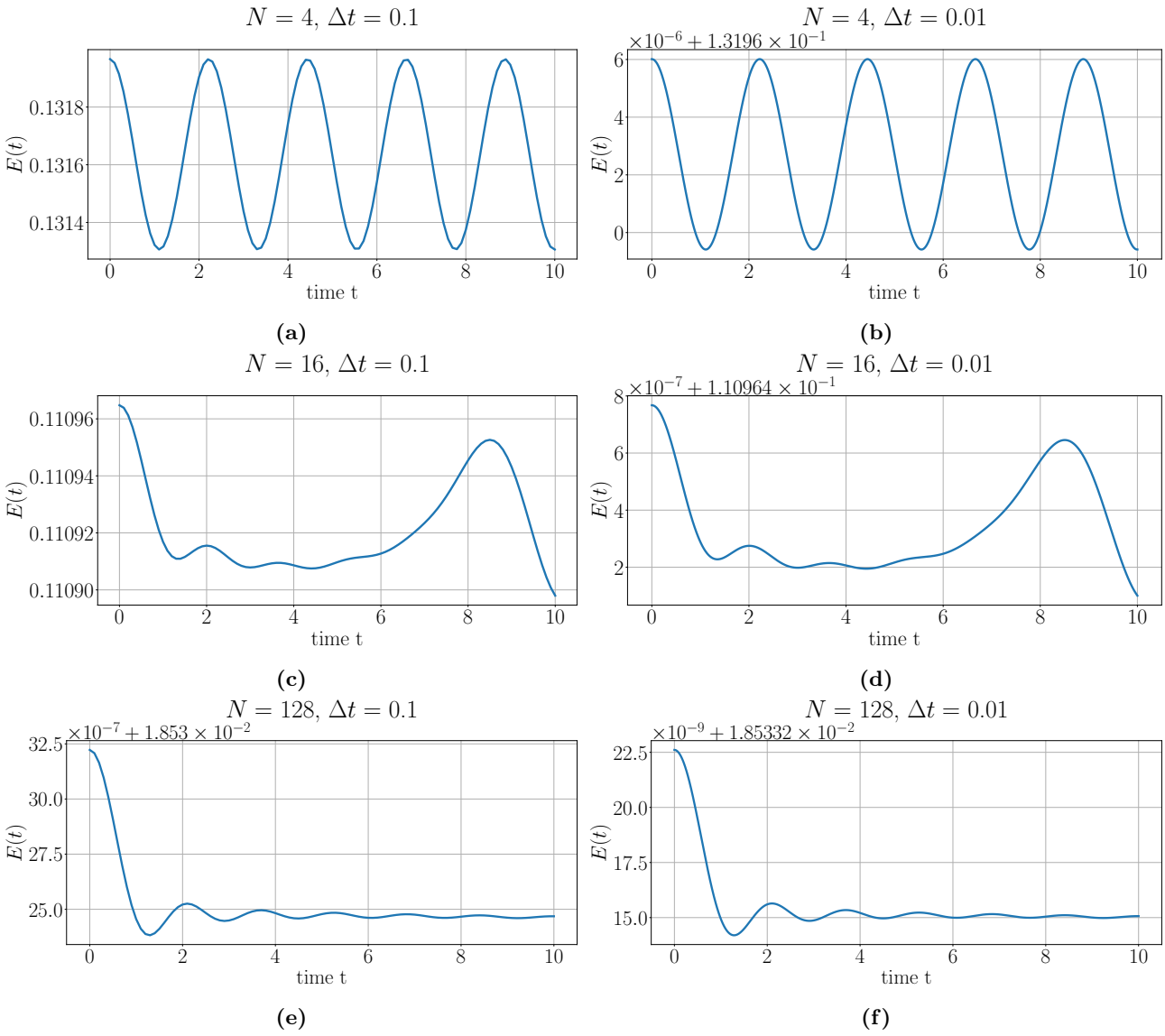


Figure 9: Numerical solutions of the total energy, given by the Hamiltonian, of the coupled system of harmonic oscillators by using the velocity Verlet algorithm (see (eq.(26))) for initial configuration (2) with $j = 1$, and the number of iterations $M - 1 = 100(1000)$. Figures in the first column show the results for $\Delta t = 0.1$ in the second for $\Delta t = 0.01$. Figures (a) and (b) are for $N = 4$, (c) and (d) for $N = 16$, and (e) and (f) for $N = 128$.

5.2.4 Discussion of the Coupled Oscillators - Initial Configuration (2), $j = 1$

To understand the results for the given initial configuration, one has to understand what the initial condition is. We sketch the position of each oscillator for $N = 16$, the idea behind this is the same for all numbers of particles N.

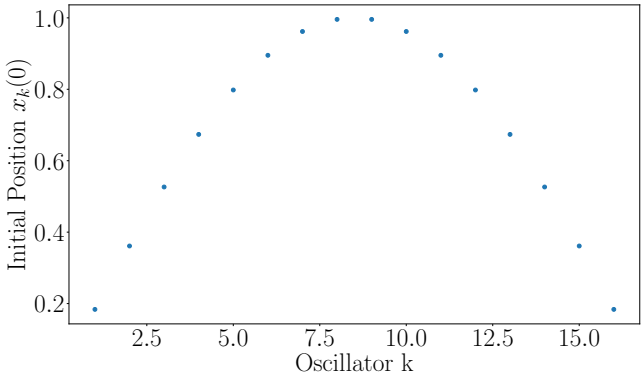


Figure 10: Initial position of each oscillator for $N = 16$

In the figure above, we can see that the initial position of each oscillator is distributed sinusoidal. Therefore, the first and the last, the second and the penultimate,..., $N/2 - 1$ and $N/2 + 2$, and $N/2$ and $N/2 + 1$ have the same initial condition and can be seen as a pair. This symmetry in the initial condition leads to the fact that in fig.(8), the position of these pairs of oscillators oscillate in the same way. This can be understood physically, as the whole system is built symmetrical and therefore this symmetry will prevail over time. It is important to mention that the results of the position arrays do not change significantly for different Δt. In the appendix, one can see that this symmetry is for all the oscillators and not only for the ones drawn in fig.(8). For the energies, presented in fig.(9), we see that similar to the previous case, changing the Δt, does not affect the overall shape of the energy fluctuation but only the amplitudes. For $N = 4$, it seems like the energy oscillates around one value. But it is difficult to say if this is really the case, as for $N = 16$ and $N = 128$, this is not the case. But in any case, the overall magnitude of the energy does not change apart from fluctuations, which is a direct consequence of the algorithm we used being symplectic.

5.2.5 Results of the Coupled Oscillators - Initial Configuration (2), $j = N/2$

The initial configuration (2) with $j = N/2$ is given by

$$v_1(0), ..., v_N(0) = 0$$
$$x_k(0) = \sin\frac{\pi jk}{(N+1)} = \sin\frac{\pi Nk}{2(N+1)} \qquad k = 1, ..., N$$

Similar to the second initial conditions with $j = 1$, we present the results for the first two, the last two as well as the oscillators in the middle $N/2-1$, $N/2$, $N/2+1$, and $N/2+2$.[11]

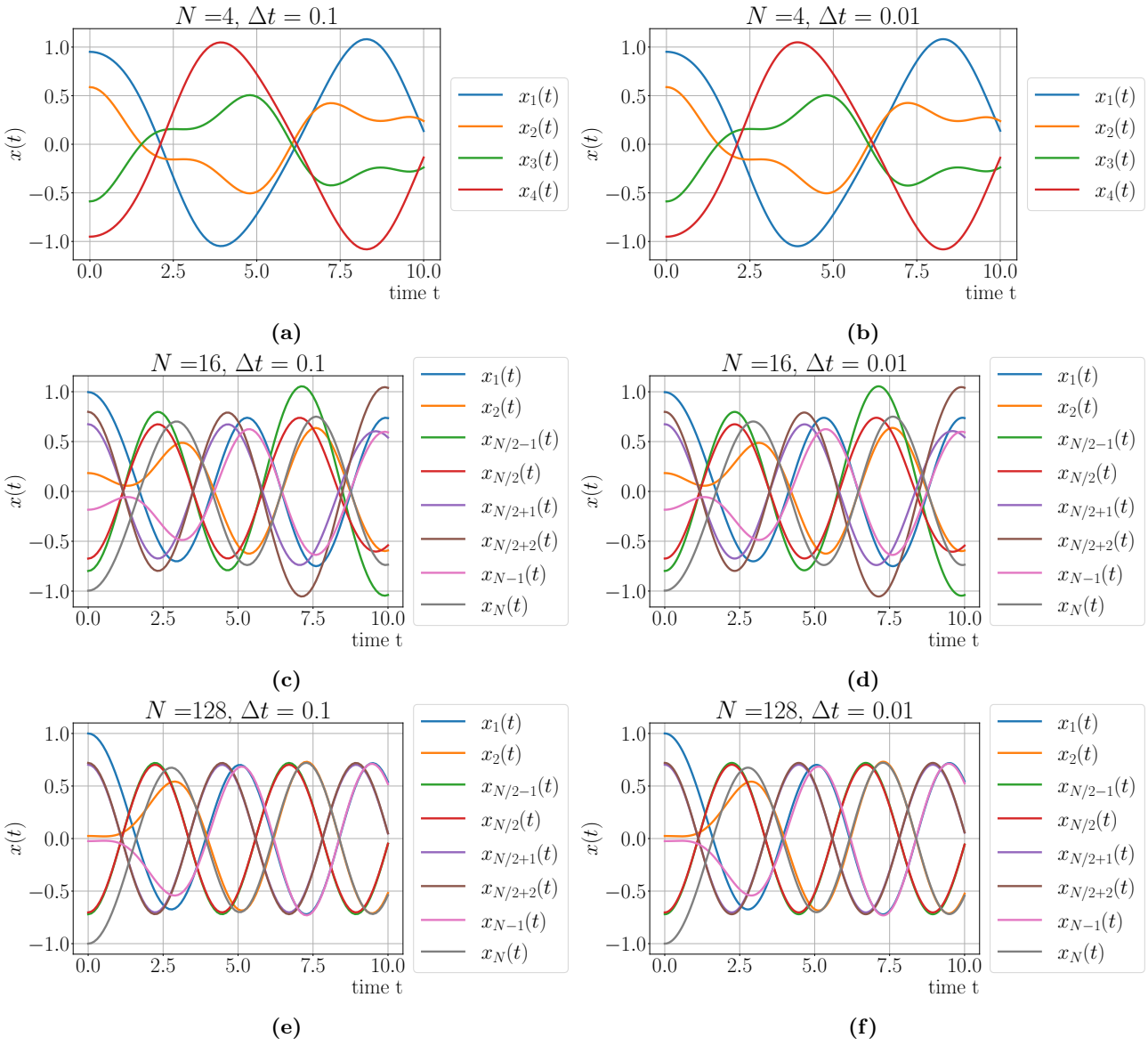


Figure 11: Numerical solutions of the coupled system of harmonic oscillators by using the velocity Verlet algorithm (see (eq.(26))) for initial configuration (2) with $j = N/2$, and the number of iterations $M - 1 = 100(1000)$. Figures in the first column show the results for $\Delta t = 0.1$ in the second for $\Delta t = 0.01$. Figures (a) and (b) are for $N = 4$, (c) and (d) for $N = 16$ and (e) and (f) for $N = 128$. The graphs only contains the solutions of the position $x_k(t)$ for a selection of particles.

[11] same color depicts oscillators with same position evolution up to an minus sign

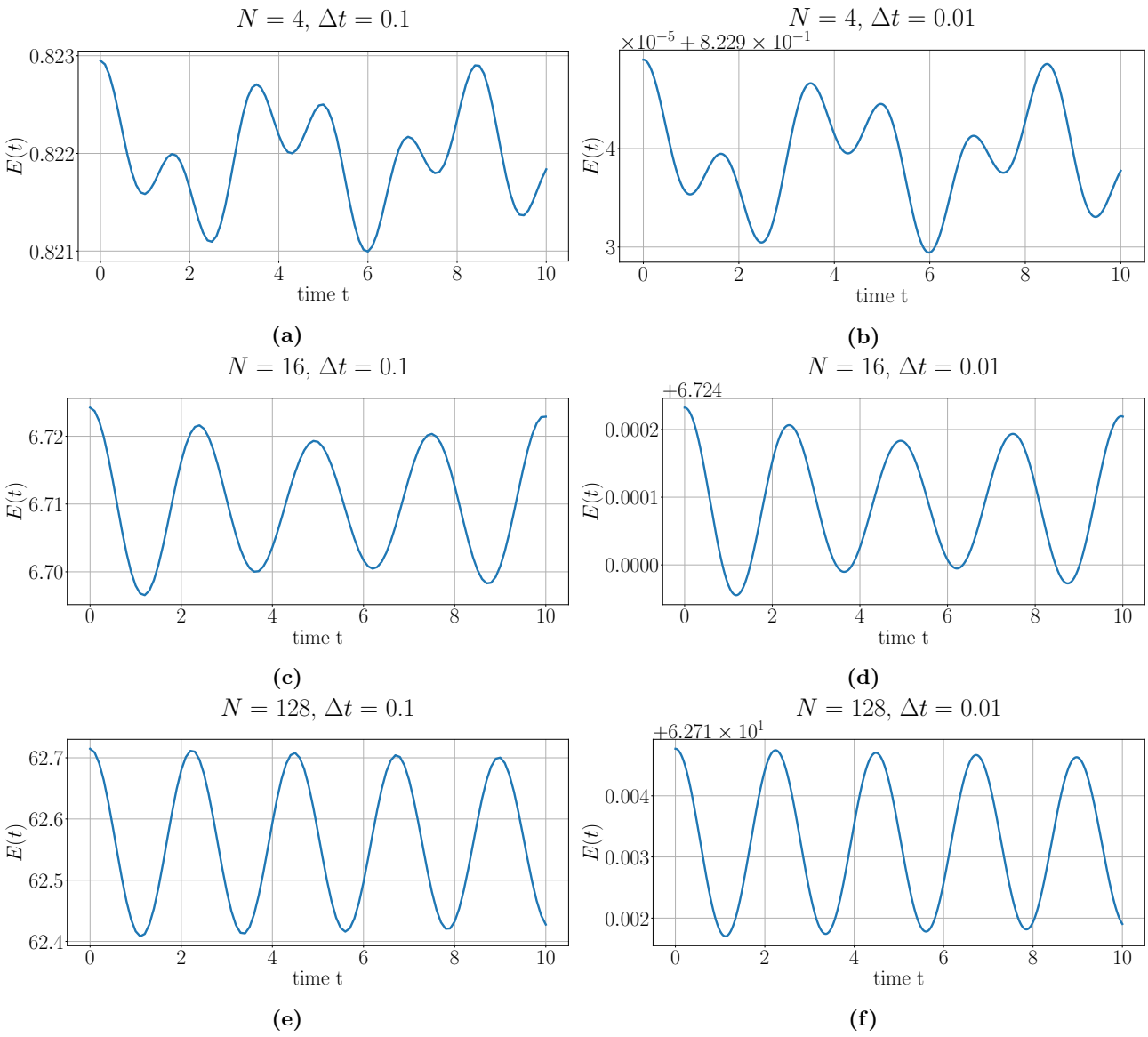


Figure 12: Numerical solutions of the total energy, given by the Hamiltonian, of the coupled system of harmonic oscillators by using the velocity Verlet algorithm (see (eq.(26))) for initial configuration (2) with $j = N/2$, and the number of iterations $M - 1 = 100(1000)$. Figures in the first column show the results for $\Delta t = 0.1$ in the second for $\Delta t = 0.01$. Figures (a) and (b) are for $N = 4$, (c) and (d) for $N = 16$ and (e) and (f) for $N = 128$.

5.2.6 Discussion of the Coupled Oscillators - Initial Configuration (2), $j = N/2$

Again, to understand the results for the given initial configuration, one has to understand what the initial condition is. We sketch the position of each oscillator for $N = 16$, the idea behind this is the same for all Ns.

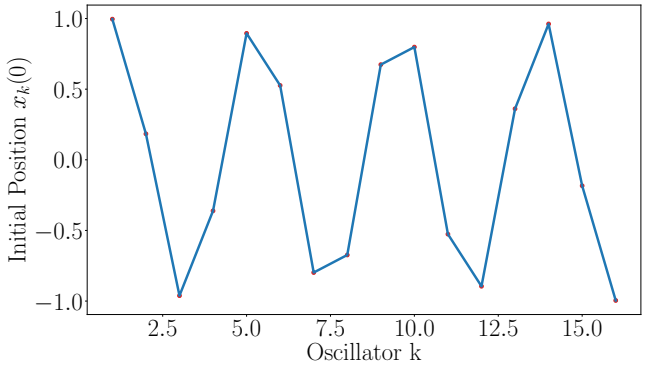


Figure 13: Initial position of each oscillator for $N = 16$. In red the positions, the line in blue is to make the symmetry more obvious

In the figure above, the symmetry of the initial position is less obvious than in the previous case. This time, we have an anti-symmetric distribution around the oscillator in the middle. For any oscillator $k = (N/2 + 1) - i$ being at $x_k(0)$, the corresponding oscillator $k' = N/2 + i$ is at $x_{k'}(0) = -x_k(0)$. Therefore, the first and the last, the second and the penultimate,..., $N/2-1$ and $N/2+2$, and $N/2$ and $N/2+1$ have anti-symmetric initial condition and can be interpreted as a pair. This anti-symmetry in the initial condition leads to the fact that in fig.(11), the position of these pairs of oscillators oscillate mirror-inverted around the position. This is very obvious in the figures of fig.(11), where mirroring the first $N/2$ oscillators around the time axis, yields the positions of the second half of oscillators. This can be understood physically, as the whole system is built anti-symmetrical, and therefore this anti-symmetry will prevail over time. It is important to mention that the calculated position arrays do not change significantly for different Δt. In the appendix, one can see that this anti-symmetry is for all the oscillators and not only for the ones drawn in fig.(8). For the energies, presented in fig.(9), we see that similar to the previous case, changing the Δt, does not affect the overall shape of the energy fluctuation but only the amplitudes. For $N = 4$, the oscillation is less pure, indicating the superposition of different oscillation frequencies. For $N = 16$ and $N = 128$, it is difficult to say if there is also a superposition of frequencies, as for $N = 16$ it looks like a modulated oscillation, while for $N = 128$ a slight damping is visible. But in any case, the overall magnitude of the energy does not change apart from fluctuations.

5.2.7 Energy evolution for greater times

In the last section, we made statements about the algorithm being symplectic and stable. As one could argue, in the time window we analyzed this effect could have been too small to see, we present the energies for $N = 16$ and 128 with $\Delta t = 0.1$ for $M = 10001$.

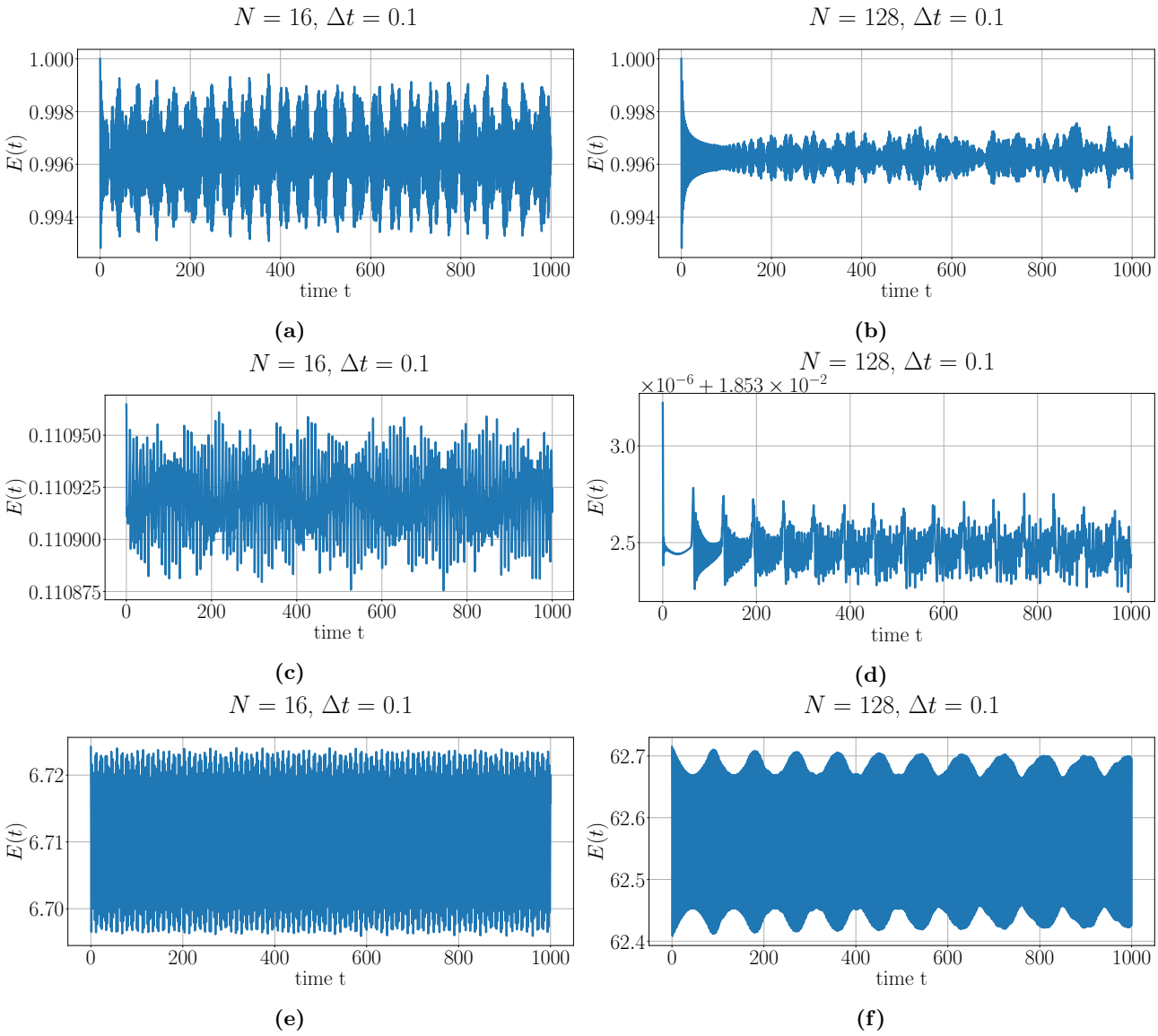


Figure 14: Numerical solutions of the total energy of the coupled system of harmonic oscillators by using the velocity Verlet algorithm (see (eq.(26))). The first row is for the initial configuration (1), the second for (2) with $j = 1$ and the third for (2) with $j = N/2$

In the figure above we can see that with time the energies average over a constant value of the energy. This shows that the algorithm is stable as well as symplectic.

5.3 Discussion

In this chapter, we have solved a coupled system of harmonic oscillators for different initial conditions. We have seen the effects the different initial conditions can have on the time evolution of the positions of each oscillation. Furthermore, we have investigated the time evolution of the system if there is an (anti-)symmetric initial condition given and have shown how this symmetry itself does not change

over time. Additionally, we have seen that, while the total energy of the system is not constant, the fluctuations get smaller with smaller Δt. Also, the fluctuation itself is of a smaller magnitude than the total energy itself, around which the values fluctuate. This behavior is a direct indicator that the algorithm we have used, namely the velocity Verlet algorithm is symplectic, and therefore, stable.

Appendix A Task 1

```python
import numpy as np
import matplotlib.pyplot as plt

# For fancy plots :)
plt.rcParams["text.usetex"] = True
plt.rcParams["font.family"] = "times new roman"
plt.rcParams["font.size"]   = "35"

def setup(toggle_title, title, xlabel, ylabel):
    plt.figure(figsize=(16, 9))
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    if toggle_title:
        plt.title(title, fontsize=40)
    plt.xticks()
    plt.yticks()
    plt.locator_params(nbins=6)
    plt.grid()
    plt.tight_layout()

# Define Parameters
k = 1
m = 1

# Force function
def F(x):
    return -k*x

# energy function
def E(x, v):
    return m*v**2/2 + k*x**2/2

# Euler algorithm function that returns
# postion x and veloctiy v arrays
def solve_Euler(dt, M, x0, v0):
    x = np.zeros(M)
    v = np.zeros(M)

    # set initial conditions
    x[0] = x0
    v[0] = v0

    # itereate over the recursive series discussed in the report
    for i in range(M-1):
        v[i+1] = v[i] + F(x[i])*dt
        x[i+1] = x[i] + v[i]*dt
    return x, v

# define initial conditions
x0  = 0
v0  = 1

# Generate plots for exercise 1
def plot_ex1():
    # Loop over different discretizations
    dt_arr  = [0.1, 0.01, 0.001]
    for dt in dt_arr:
        clip = int(50/dt)

        M       = int(10000/dt) + 1
```

```
        tmax     = (M-1)*dt
        t        = np.linspace(0, tmax, M)
        x, v     = solve_Euler(dt, M, x0, v0)

        setup(False, r"Euler Method: $\Delta t={}$".format(dt), "time $t$", "$x(t)$")
        plt.plot(t, x, linewidth = 3, label=rf"$\Delta t={dt}$")
        plt.plot(t, np.sin(t), color="black", ls="--", label="Theory")
        plt.legend()

        setup(False, r"Euler Method: $\Delta t={}$, Clipping".format(dt), "time $t$", "$x(t)$")
        plt.plot(t[:clip], x[:clip], linewidth = 6, label=rf"$\Delta t={dt}$")
        plt.plot(t[:clip], np.sin(t[:clip]), linewidth = 4, color="black", ls="--", label="Theory")
        plt.legend()

# create plots
plot_ex1()
```

Appendix B Task 2

```
import numpy as np
import matplotlib.pyplot as plt

# For fancy plots :)
plt.rcParams["text.usetex"] = True
plt.rcParams["font.family"] = "times new roman"
plt.rcParams["font.size"]   = "35"

def setup(toggle_title, title, xlabel, ylabel):
    plt.figure(figsize=(16, 9))
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    if toggle_title:
        plt.title(title, fontsize=40)
    plt.xticks()
    plt.yticks()
    plt.locator_params(nbins=6)
    plt.grid()
    plt.tight_layout()

# Define Parameters
k = 1
m = 1

# Force function
def F(x):
    return -k*x

# energy function
def E(x, v):
    return m*v**2/2 + k*x**2/2


# Euler-Cromer (implementation a) algorithm function that returns
# postion x and veloctiy v arrays
def solve_Euler_Cromer_a(dt, M, x0, v0):
    x = np.zeros(M)
    v = np.zeros(M)

```

```
    # set initial conditions
    x[0] = x0
    v[0] = v0

    # itereate over the recursive series discussed in the report
    for i in range(M-1):
        v[i+1] = v[i] + F(x[i])*dt
        x[i+1] = x[i] + v[i+1]*dt
    return x, v

# Euler-Cromer (implementation b) algorithm function that returns
# postion x and veloctiy v arrays
def solve_Euler_Cromer_b(dt, M, x0, v0):
    x = np.zeros(M)
    v = np.zeros(M)

    # set initial conditions
    x[0] = x0
    v[0] = v0

    # itereate over the recursive series discussed in the report
    for i in range(M-1):
        x[i+1] = x[i] + v[i]*dt
        v[i+1] = v[i] + F(x[i+1])*dt
    return x, v

# define initial conditions
x0  = 0
v0  = 1

# Generate Plots for exercise 2:
# For both implementations and different end times tmax
def plot_ex2(tmax, implementation):
    dt  = 0.01
    M   = int(tmax/dt) + 1
    t   = np.linspace(0, tmax, M)

    if implementation == "a":
        x, v = solve_Euler_Cromer_a(dt, M, x0, v0)
    else:
        x, v = solve_Euler_Cromer_b(dt, M, x0, v0)

    setup(True, f"Euler-Cromer Method ({implementation})", "time $t$", r"$x(t)$, $v(t
)$ and $E(t)$")
    plt.plot(t, x, linewidth = 3, color="tab:blue", label="x(t)")
    plt.plot(t, v, linewidth = 3, color="tab:green", label="v(t)")
    plt.plot(t, E(x, v), linewidth = 3, color="tab:red", label="E(t)")
    plt.plot(t, np.sin(t), ls="--", color="black", label="Theory")
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), fancybox=True, shadow
=True, ncol=3, fontsize=30)
    plt.tight_layout()


    setup(False, "X", "time $t$", r"$E(t) - E_{exact}$")
    plt.plot(t, E(x, v)-1/2, linewidth = 3, color="tab:red")


# Generate Plots that compare the results for
# both implementations
def plot_ex2_ab_compare():
    dt  = 0.01
    tmax  = 10
```

```
    M    = int(tmax/dt) + 1
    t    = np.linspace(0, tmax, M)

    xa, va = solve_Euler_Cromer_a(dt, M, x0, v0)
    xb, vb = solve_Euler_Cromer_b(dt, M, x0, v0)

    setup(True, "Euler-Cromer Method: compare $x(t)$ for (a) and (b) ", "time $t$", r"$x_a(t)-x_b(t)$")
    plt.plot(t, xa-xb, linewidth = 3, color="tab:blue", label="x(t)")
    plt.legend()

    setup(True, "Euler-Cromer Method: compare $v(t)$ for (a) and (b)", "time $t$", r"$v_a(t)-v_b(t)$")
    plt.plot(t, va-vb, linewidth = 3, color="tab:green", label="v(t)")
    plt.legend()

# create Plots
plot_ex2(10, "a")
plot_ex2(1000, "a")
plot_ex2(10, "b")
plot_ex2(1000, "b")

plot_ex2_ab_compare()
```

Appendix C Task 3

C.1 Code for Task 3

```
import numpy as np
import matplotlib.pyplot as plt

import os
os.environ["PATH"] += os.pathsep + '/Library/TeX/texbin'
#some plotting arguments to make the plot prettier
plt.rcParams["text.usetex"] = True
plt.rcParams["font.family"] = "times new roman"
plt.rcParams["font.size"] = "40"
plt.rcParams['lines.linewidth'] = 4
plt.rcParams["font.weight"] = "bold"


Deltat = np.array([0.1,0.01]) # the delta t we use

#function solving the equation for given delta t = dt
def Exercise3(dt):
    #function defining the acceleration
    def a(x):
        return -x
    #length of array depends on the delta t
    if dt == 0.1:
        M = 101
    elif dt == 0.01:
        M = 1001
    #initializing the arrays for position x and velocity v
    x = np.zeros(M)
    v = np.zeros(M)
    #initial condition
    v[0] =1

    #velocity verlet algorithm, xdt = x(t+dt) and vdt = v(t+dt)
```

```
    def Velocity_Verlet(x,v):
        xdt= x+ v*dt+ 1/2 *a(x)*dt**2
        vdt= v+1/2*(a(x) + a(xdt))*dt
        return xdt,vdt

    #iterating over all the indexes in the array
    for i in range(M-1):
        x[i+1],v[i+1] = Velocity_Verlet(x[i],v[i])


    #calculate the energy
    Et = v**2/2 + x**2/2
    #defining the time array
    t = np.linspace(0,(M-1)*dt,M)

    #below only code needed for the plots
    plt.figure(figsize= [16,9])
    plt.title(r"Velocity Verlet algorithm - $\Delta t = $ "+str(dt))
    plt.plot(t,x, label = r"Position - $x(t)$")
    plt.xlabel("time t", loc="right")
    plt.ylabel(r"$x(t)$, $v(t)$ and $E(t)$")
    plt.plot(t,v,label = r"Velocity - $v(t)$", color = "tab:green")
    plt.plot(t,np.sin(t), linestyle='dashed', label = r"Position - $x_{theory}(t)$",
    color ="black")
    plt.plot(t,Et, label = r"Energy - $E(t)$", color = "tab:red")
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.14),fancybox=True, shadow=
    True, ncol=2)
    plt.grid()
    plt.savefig("Velocity-Verlet_x_v_"+str(dt)+".pdf",bbox_inches='tight')
    plt.show()
    plt.clf()
    plt.figure(figsize= [16,9])
    plt.plot(t,(Et - 1/2), label = r"$E(t)$ - $E_{theory}$", color = "tab:red")
    plt.xlabel("time t", loc="right")
    plt.ylabel(r"$E(t) - E_{theory}$")
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.08),fancybox=True, shadow=
    True, ncol=3)
    plt.ticklabel_format(axis='y', scilimits=[-3, 3])
    plt.grid()
    plt.savefig("Velocity-Verlet_E_"+str(dt)+".pdf",bbox_inches='tight')
    plt.show()
    plt.clf()
    return 0

#executing the task for both delta ts
Exercise3(Deltat[0])
Exercise3(Deltat[1])
```

Appendix D Task 4

D.1 Code for Task 4

```
import numpy as np
import matplotlib.pyplot as plt
import os
os.environ["PATH"] += os.pathsep + '/Library/TeX/texbin'
#plotting arguments to make plot prettier
plt.rcParams["text.usetex"] = True
plt.rcParams["font.family"] = "times new roman"
plt.rcParams["font.size"] = "40"
plt.rcParams['lines.linewidth'] = 4
plt.rcParams["font.weight"] = "bold"

pi      = np.pi

M = [100,1000]      #number of steps
N = [4,16,128]      #number of oscillators
dt = [0.1, 0.01]    #delta t


def Exercise4(N,dt,M,config,plot, z ="selected"):
    #depending on the initial condition the arrays are initialized differently
    def initial(config = "a"):
        x = np.zeros((N,M))
        v = np.zeros((N,M))
        if config == "a":           #first condition
            x[N//2 -1,0] = 1
        elif config == "b":         #second condtion with j = 1
            j = 1
            x[:,0] = np.sin(pi*j*(np.arange(N)+1)/(N+1))
        elif config =="c":          #second condtion with j = N/2
            j = N/2
            x[:,0] = np.sin(pi*N/2*(np.arange(N)+1)/(N+1))
        return x,v

    x,v = initial(str(config))  #initialize arrays for configuration

    #function that yields the forces acting oscillator depending on oscillator
    def Fn(n,i):
        if n == 0:
            return -(x[0,i]-x[1,i])
        elif n == (N-1):
            return -(x[N-1,i]-x[N-2,i])
        else:
            return -(2*x[n,i] - x[n-1,i] - x[n+1,i])

    #algorithm that calculates position and velocity
    def Velocity_Verlet(x,v,i,N):
        #iterate over all the oscillators
        for k in range(N):
            x[k,i+1]= x[k,i]+dt*(v[k,i]+dt/2*Fn(k,i))   #x[k,i+1] = x_k (t+dt)
        for k in range(N):
            v[k,i+1]= v[k,i]+dt/2*(Fn(k,i) + Fn(k,i+1)) #v[v,i+1] = v_k (t+dt)
        return x,v

    #call function iteratively
    for i in range(M-1):
        x,v = Velocity_Verlet(x,v,i,N)
    #energy of system
    def E(x, v):
```

```
        KE = 1/2*np.sum(v**2, axis=0)
        PE = 1/2* np.sum(np.diff(x, axis = 0)**2, axis = 0)
        return KE+PE

    #calculates velocity and position for the arguments given to the function
    X,V = Velocity_Verlet(x,v,0,N)
    #time array
    t = np.linspace(0,dt*(M-1),M)

    #code that chooses which functions are plotted
    if plot =="x":
        plt.figure(figsize= [16,9])
        plt.grid()
        if z == "all":
            for k in range(N):
                plt.plot(t,X[k])

        elif N ==4 and config != "c":
            plt.plot(t,X[0], label = r"$x_{1}(t)$")
            plt.plot(t,X[1], label = r"$x_{2}(t)$")
            plt.plot(t,X[2], label = r"$x_{3}(t)$", ls = "--")
            plt.plot(t,X[3], label = r"$x_{4}(t)$",  ls = "--")
        elif config == "a":
            if N!=4:
                plt.plot(t, X[0], label =r"$x_{1}(t)$")
                plt.plot(t, X[N//2 -2], label =r"$x_{N/2-1}(t)$", ls = "-.")
                plt.plot(t, X[N//2 -1], label =r"$x_{N/2}(t)$") #oscillator which is
    extendend
                plt.plot(t, X[N//2 ], label =r"$x_{N/2+1}(t)$", ls = "--")
                plt.plot(t, X[-1], label =r"$x_{N}(t)$")

        elif config == "b":
            if N!=4:
                plt.plot(t, X[0], label =r"$x_{1}(t)$", ls = "-")
                plt.plot(t, X[1], label =r"$x_{2}(t)$", ls = "-")
                plt.plot(t, X[N//2 -2], label =r"$x_{N/2-1}(t)$", ls = "-")
                plt.plot(t, X[N//2 -1], label =r"$x_{N/2}(t)$")
                plt.plot(t, X[N//2 ], label =r"$x_{N/2+1}(t)$", ls = "--")
                plt.plot(t, X[N//2 +1], label =r"$x_{N/2+2}(t)$", ls = "--")
                plt.plot(t, X[-2], label =r"$x_{N-1}(t)$",ls = "--")
                plt.plot(t, X[-1], label =r"$x_{N}(t)$",ls = "--")
        elif config == "c":
            if N ==4:
                plt.plot(t,X[0], label = r"$x_{1}(t)$")
                plt.plot(t,X[1], label = r"$x_{2}(t)$")
                plt.plot(t,X[2], label = r"$x_{3}(t)$")
                plt.plot(t,X[3], label = r"$x_{4}(t)$")
            elif N!=4:
                plt.plot(t, X[0], label =r"$x_{1}(t)$")
                plt.plot(t, X[1], label =r"$x_{2}(t)$")
                plt.plot(t, X[N//2 -2], label =r"$x_{N/2-1}(t)$")
                plt.plot(t, X[N//2 -1], label =r"$x_{N/2}(t)$")
                plt.plot(t, X[N//2 ], label =r"$x_{N/2+1}(t)$")
                plt.plot(t, X[N//2 +1], label =r"$x_{N/2+2}(t)$")
                plt.plot(t, X[-2], label =r"$x_{N-1}(t)$")
                plt.plot(t, X[-1], label =r"$x_{N}(t)$")
        plt.title(r"$N =$"+str(N)+", $\Delta t = $ "+str(dt))
        plt.xlabel("time t", loc="right")

        plt.ylabel(r"$x(t)$")
        plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
        plt.tight_layout()
```

```
        plt.savefig("images/"+str(N)+"_"+str(dt)+"_"+config+".pdf",bbox_inches='tight')
        plt.show()
    elif plot =="E":
        plt.figure(figsize= [16,9])
        plt.grid()
        plt.plot(t, E(X,V))
        plt.xlabel("time t")
        plt.ylabel(r"$E(t)$")
        plt.title(r" $N = $ "+str(N)+", $\Delta t = $ "+str(dt), y=1.1)
        plt.tight_layout()
        plt.savefig("images/"+str(N)+"_E_"+str(dt)+"_"+config+".pdf",bbox_inches='tight')
        plt.show()

#vectorize function to call function with arrays
Exercise4vec = np.vectorize(Exercise4, excluded= "dt,M,N")


#calclulate the function for each parameter
for n in N:
    for it in range(len(dt)):
        #execute code with M+1 because we do M steps
        Exercise4vec(n,dt[it],M[it]+1, ["a","b", "c"],plot = "E", z = "all")
        Exercise4vec(n,dt[it],M[it]+1, ["a","b", "c"],plot = "x")
        Exercise4vec(n,dt[it],M[it]+1, ["a","b", "c"],plot = "E")
```

D.2 Results of the Coupled Oscillators - Initial Configuration (1)

The initial configuration (1) is given by

$$v_1(0), ..., v_N(0) = 0$$
$$x_1(0), ..., x_N(0) = 0 \qquad \text{except } x_{N/2}(0) = 1$$

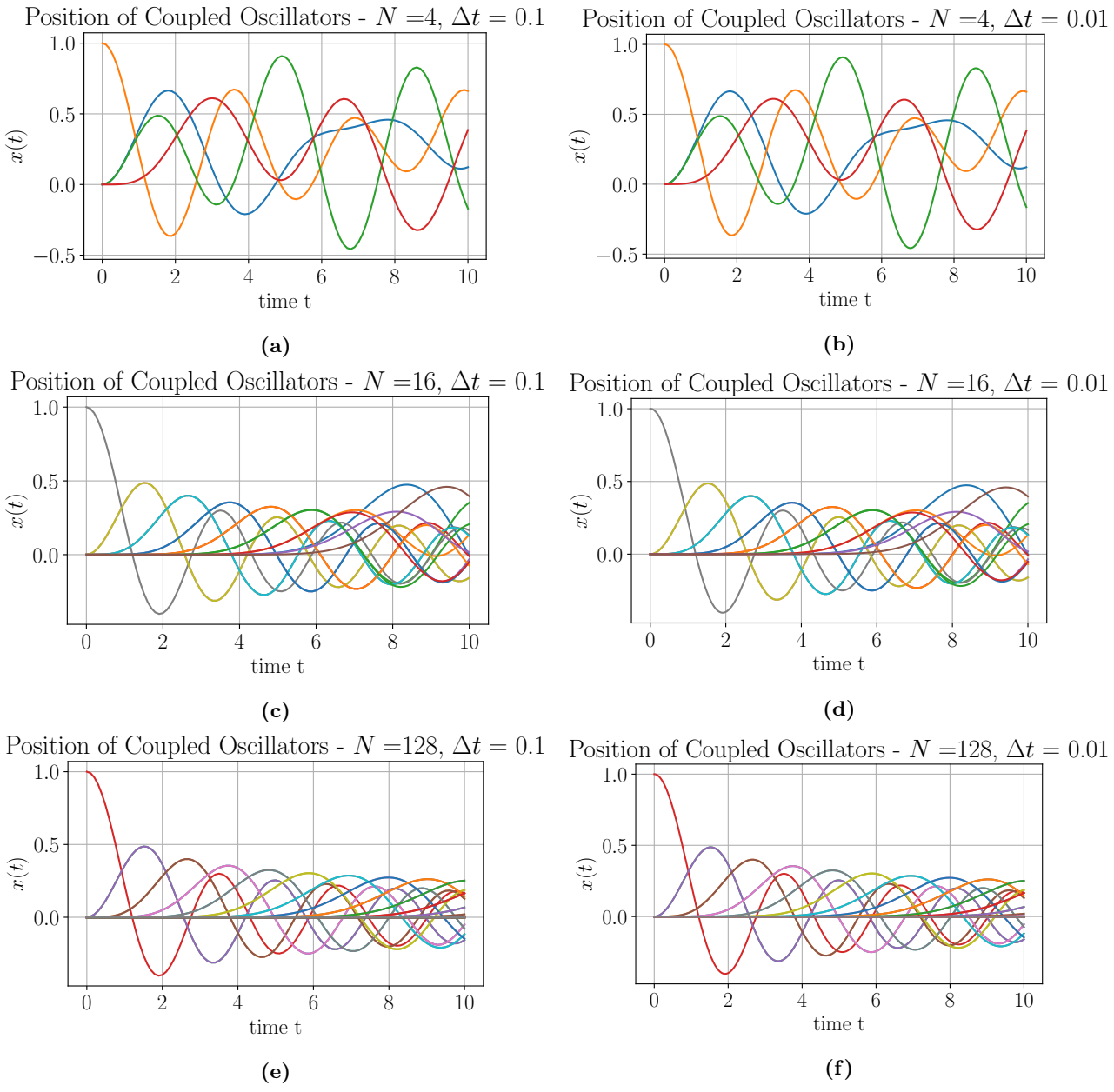


Figure 15: Numerical solutions of the coupled system of harmonic oscillator by using the velocity Verlet algorithm (see (eq.(26))) for initial configuration (1), and the number of iterations $M - 1 = 100(1000)$. Figures in the first column show the results for $\Delta t = 0.1$ in the second for $\Delta t = 0.01$. Figures (a) and (b) are for $N = 4$, (c) and (d) for $N = 16$ and (e) and (f) for $N = 128$. The graphs contain the solutions for the position $x_k(t)$ for all the oscillators

D.3 Results of the Coupled Oscillators - Initial Configuration (2), $j = 1$

The initial configuration (2) with $j = 1$ is given by

$$v_1(0), ..., v_N(0) = 0$$
$$x_k(0) = \sin \frac{\pi k}{N+1} \qquad k = 1, ..., N$$

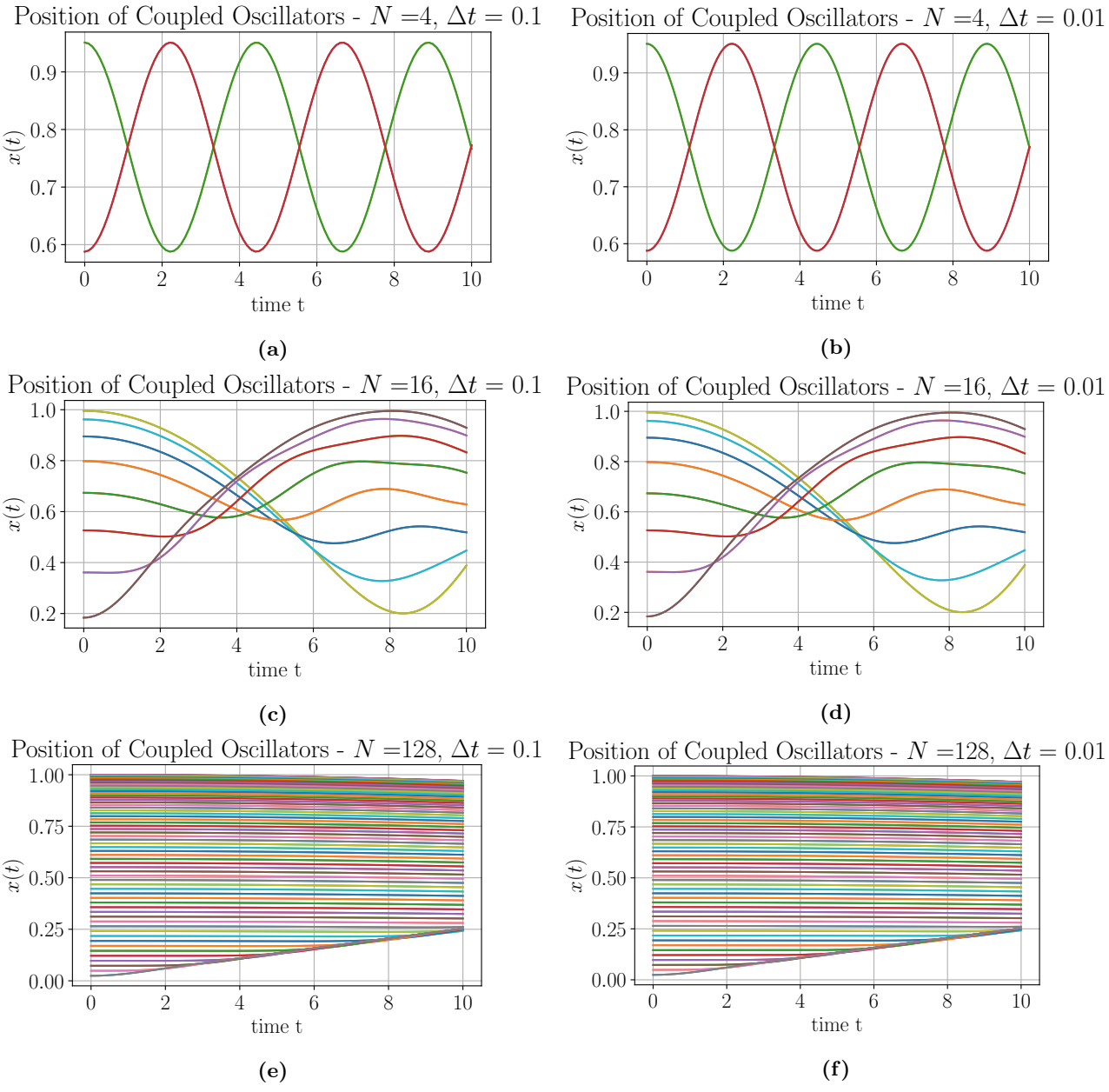


Figure 16: Numerical solutions of the coupled system of harmonic oscillator by using the velocity Verlet algorithm (see (eq.(26))) for initial configuration (2) with $j = 1$, and the number of iterations $M-1 = 100(1000)$. Figures in the first column show the results for $\Delta t = 0.1$ in the second for $\Delta t = 0.01$. Figures (a) and (b) are for $N = 4$, (c) and (d) for $N = 16$ and (e) and (f) for $N = 128$. The graphs contain the solutions for the position $x_k(t)$ for all the oscillatorss

D.4 Results of the Coupled Oscillators - Initial Configuration (2), $j = N/2$

The initial configuration (2) with $j = N/2$ is given by

$$v_1(0), ..., v_N(0) = 0$$
$$x_k(0) = \sin \frac{\pi N \cdot k}{2(N+1)} \qquad k = 1, ..., N$$

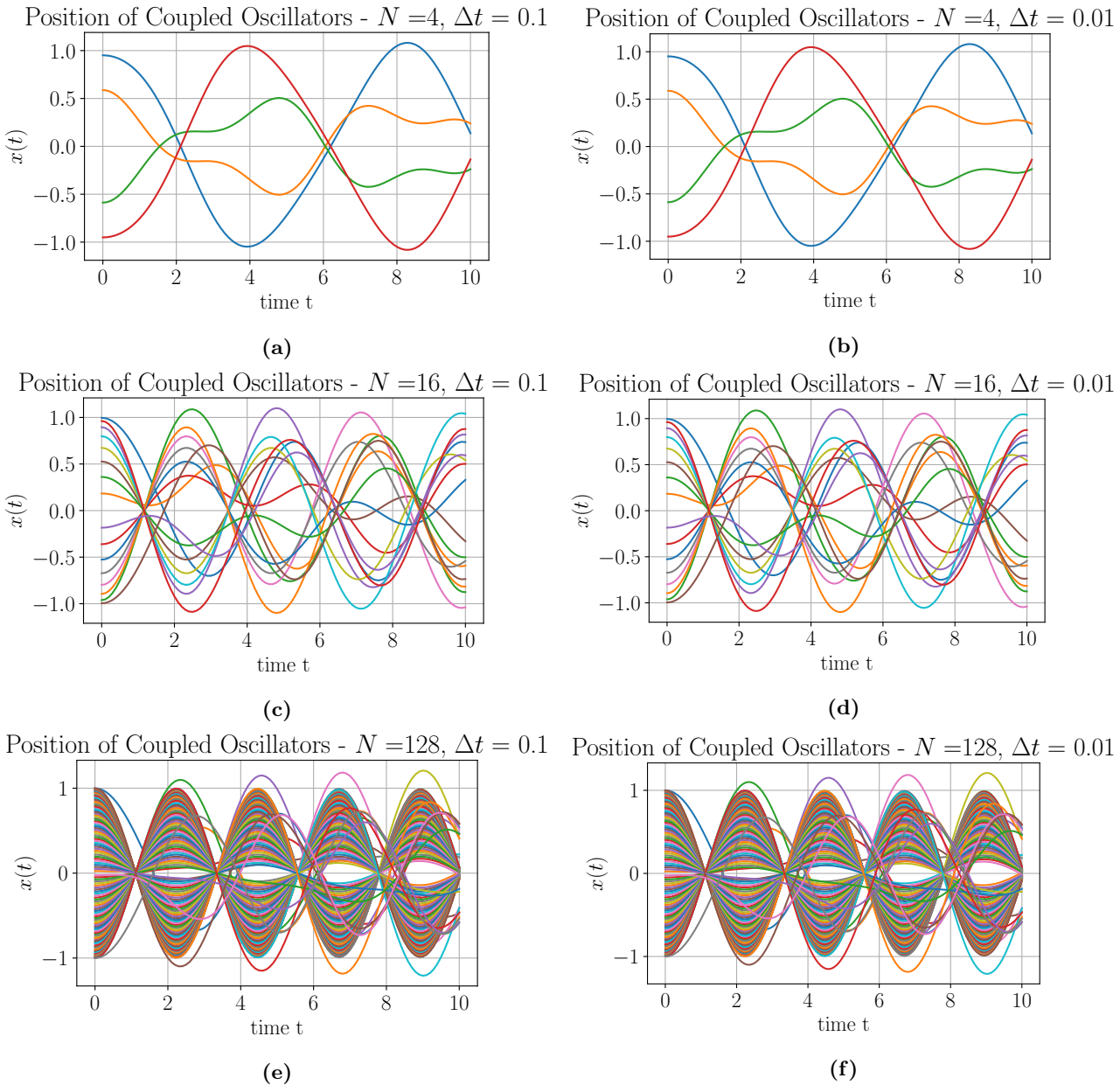


Figure 17: Numerical solutions of the coupled system of harmonic oscillator by using the velocity Verlet algorithm (see (eq.(26))) for initial configuration (2) with $j = N/2$, and the number of iterations $M - 1 = 100(1000)$. Figures in the first column show the results for $\Delta t = 0.1$ in the second for $\Delta t = 0.01$. Figures (a) and (b) are for $N = 4$, (c) and (d) for $N = 16$ and (e) and (f) for $N = 128$. The graphs contain the solutions for the position $x_k(t)$ for all the oscillators