Mechanics and Relativity: Numerical Methods Practical

Exercise 1

Consider the application of the Euler's and SOTA (Second Order Taylor Approximation) methods to the calculation of the solution of the equation for the harmonic oscillator:

$$m\ddot{x} = -k\dot{x}, \tag{1}$$

where $m = 1$, $\omega = (k/m)^{1/2} = 2\pi$, and with the initial conditions $x(0) = 1$, $\dot{x}(0) = 0$.

- Solve numerically this equation in the time interval $[0, 1]$ by the Euler's method using $\Delta t = 0.1$.
- Use the programme EULER to solve the same problem by the Euler's method, but with $\Delta t = 0.01$. Compare the results for the two values of Δt.
- Solve numerically the equation using the Second Order Taylor Approximation and the time step $\Delta t = 0.1$. Notice that in this case the applied force depends only on the position of the particle, so that it is convenient to calculate the velocity at each time step using the relation: $v_{i+1} = v_{i+1/2}(a_i + a_{i+1})\Delta t$. Compare the results with those obtained using the Euler's method.
- Analyze how the energy of the oscillator is conserved using these numerical methods. Make a critical analysis of the results.

A table will be filled up for each of the tests; one for EULER with $\Delta t = 0.1$, another one for EULER with $\Delta t = 0.01$, and another one for SOTA with $\Delta t = 0.1$. In order to find the accurate solutions, the code in Code 1 has been used; in order to find the EULER solutions, the code in Code 2 has been used; and in order to find the SOTA solutions, the code in Code 3 has been used.

The first thing that we can see very clearly is that the EULER method with $\Delta t = 0.1$ is not accurate at all. We can see this with the clear difference in the x values between the EULER numerical solution and the analytic accurate solution (**Table** 1). But this is especially obvious when we look at the values for the energy. Just after 10 iterations, the energy has changed from 19.7392J to 550.5000J (assuming SI units), which is almost thirty times the initial value. This means that we have increased the energy an average of around 53J per iteration. Energy is clearly not conserved in this solution, so it is not accurate at all. As a last proof of this fact, **Figure** 1 shows a graph of the position as a function of time given by the EULER method with $\Delta t = 0.1$, $x_0 = 1$, $v_0 = 0$, and $\omega = 2\pi$ over 20 iterations. It is clear that it does not correspond to a harmonic oscillatory motion.

On the other hand, when we decrease the time interval, we obtain a much closer solution to the real one. Even though the energy increases considerably, we must consider that we also have a lot more iterations

t	x (accurate sol.)	x (numerical sol.)	Velocity (num.sol.)	Energy (num.sol.)
0.0	1.000	1.00000	0.0000	19.7392
0.1	0.809	1.00000	-3.9478	27.5320
0.2	0.309	0.60522	-7.8957	38.4010
0.3	-0.309	-0.18435	-10.2850	53.5610
0.4	-0.809	-1.21290	-9.5572	74.7060
0.5	-1.000	-2.16860	-4.7690	104.2000
0.6	-0.809	-2.64550	3.7921	145.3400
0.7	-0.309	-2.26630	14.2360	202.7100
0.8	0.309	-0.84266	23.1830	282.7400
0.9	0.809	1.47560	26.5100	394.3600
1.0	1.000	4.12660	20.6840	550.5000

Table 1: Numerical Solution Using the EULER Method with $\Delta t = 0.1$

t	x (accurate sol.)	x (numerical sol.)	Velocity (num.sol.)	Energy (num.sol.)
0.0	1.000	1.0000	0.0000	19.817
0.1	0.809	0.8256	-3.7624	20.5325
0.2	0.309	0.3231	-6.2125	21.3576
0.3	-0.309	-0.3253	-6.3446	22.2159
0.4	-0.809	-0.8732	-4.0141	23.1087
0.5	-1.000	-1.1035	-0.0286	24.0374
0.6	-0.809	-0.9138	4.1282	25.0034
0.7	-0.309	-0.3610	6.8463	26.0082
0.8	0.309	0.3544	7.0106	27.0534
0.9	0.809	0.9607	4.4545	28.1406
1.0	1.000	1.2177	0.0631	29.2715

Table 2: Numerical Solution Using the EULER Method with $\Delta t = 0.01$

(due to the smaller time interval) than in the case where $\Delta t = 0.1$. Therefore, the average increase in energy per iteration is a lot lower (around 0.09J/iteration). **Figure 2** shows a graph of the position as a function of time given by the EULER method with $\Delta t = 0.01$, $x_0 = 1$, $v_0 = 0$, and $\omega = 2\pi$ over 200 iterations.

Finally, we can see that the SOTA method is much more accurate than any of the previous two methods, even with a time interval as large as 0.1. The values of x given by the SOTA numerical solution and those given by the analytical accurate solution are almost the same. Furthermore, the energy always stays around the same value, so in this method the energy is conserved much better. In **Figure 3**, we can see that the shape of $x(t)$ is much more *harmonic* than any of the previous two solutions. Especially notable is the fact that the amplitude of the motion is almost always the same, whereas in both EULER solutions the amplitude increased with the energy.

One big difference between the EULER and the SOTA methods is the way that energy is conserved (or the way that energy varies) through the different iterations. In both the EULER methods, we see a tendency of the energy to increase exponentially (**Figure 4**), just as the amplitude does. Thus, after just a few iterations the solution is no longer any good. On the other hand, the SOTA method gives an energy which oscillates between two values but does not increase (or decrease) outside those values (**Figure 5**). This is a huge advantage, because it means that this method has a much higher range of iterations for which it is valid (of course, keeping in mind the *controlled* fluctuations in the energy which show that the SOTA method is still not *perfect*). On the other hand, the EULER method ceases to be relevant after a set number of iterations, as the energy quickly increases beyond what can be used to accurately model any practical application.

t	x (accurate sol.)	x (numerical sol.)	Velocity (num.sol.)	Energy (num.sol.)
0.0	1.000	1.0000	0.0000	19.7392
0.1	0.809	0.8026	-3.5582	19.046
0.2	0.309	0.2884	-5.7117	17.953
0.3	-0.309	-0.3397	-5.6103	18.0159
0.4	-0.809	-0.8337	-3.2940	19.1451
0.5	-1.000	-0.9985	0.3227	19.7335
0.6	-0.809	-0.7692	3.8120	18.9436
0.7	-0.309	-0.2361	5.7964	17.8997
0.8	0.309	0.3901	5.4925	18.0875
0.9	0.809	0.8623	3.0202	19.2398
1.0	1.000	0.9941	-0.6444	19.7165

Table 3: Numerical Solution Using the SOTA Method with $\Delta t = 0.1$

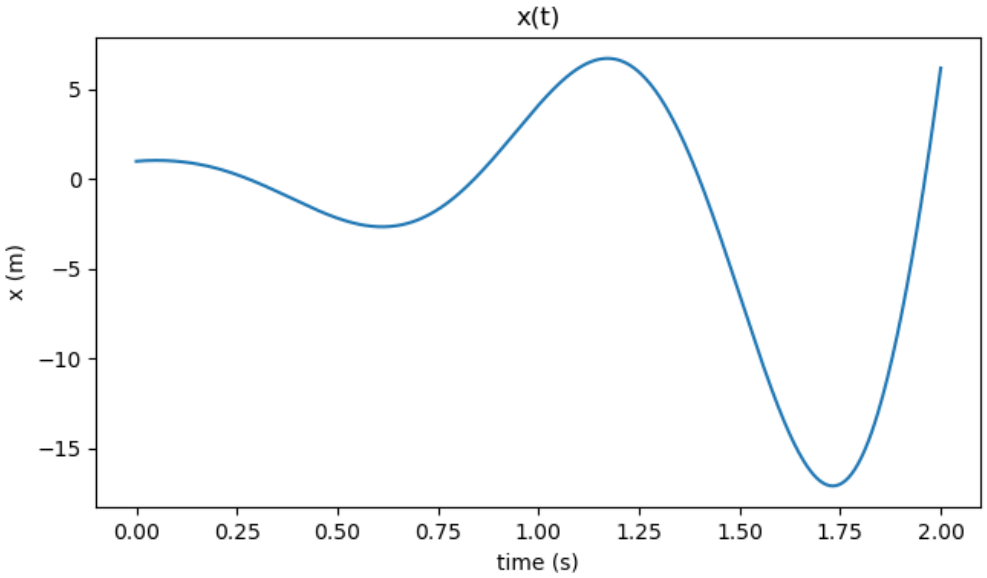

Figure 1: Graph of the position as a function of time given by the EULER method with $\Delta t = 0.1$, $x_0 = 1$, $v_0 = 0$, and $\omega = 2\pi$ over 20 iterations

There is another conclusion that we can make within the EULER method concerning the time interval width. It is clear from the data from both the graphs and the tables that the accuracy of the EULER method was much higher in the case with $\Delta t = 0.01$ than in the case with $\Delta t = 0.1$. We have not proven that this tendency extends forever (meaning that it is possible that for some $\Delta t < 0.01$ we may experiment another decrease in accuracy due to rounding errors), but we can safely say that, within some neighbourhood around 0.01 and 0.1, the smaller the time interval, the more accurate the results that we obtain with the EULER method.

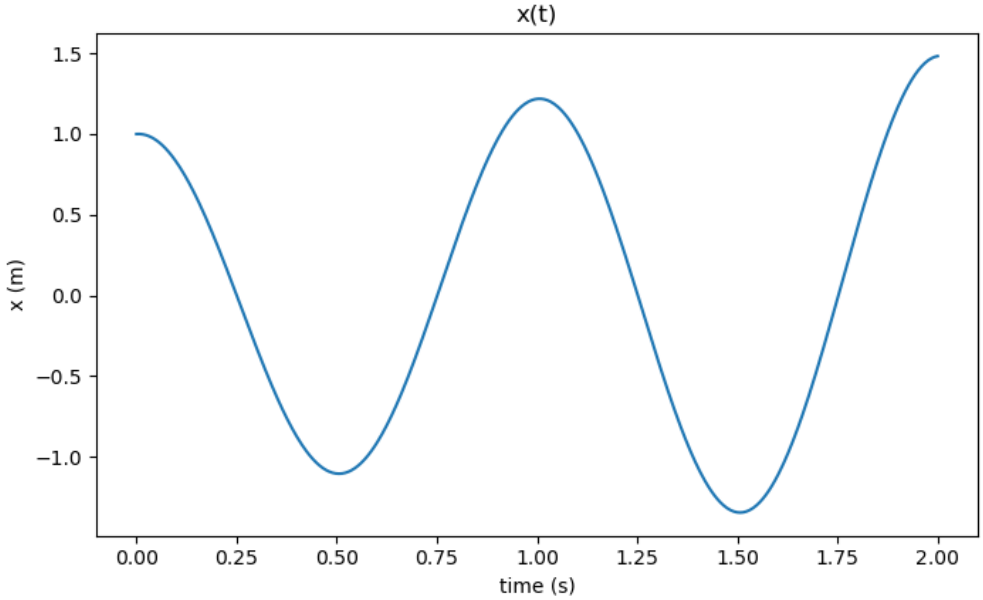

Figure 2: Graph of the position as a function of time given by the EULER method with $\Delta t = 0.01$, $x_0 = 1$, $v_0 = 0$, and $\omega = 2\pi$ over 200 iterations.

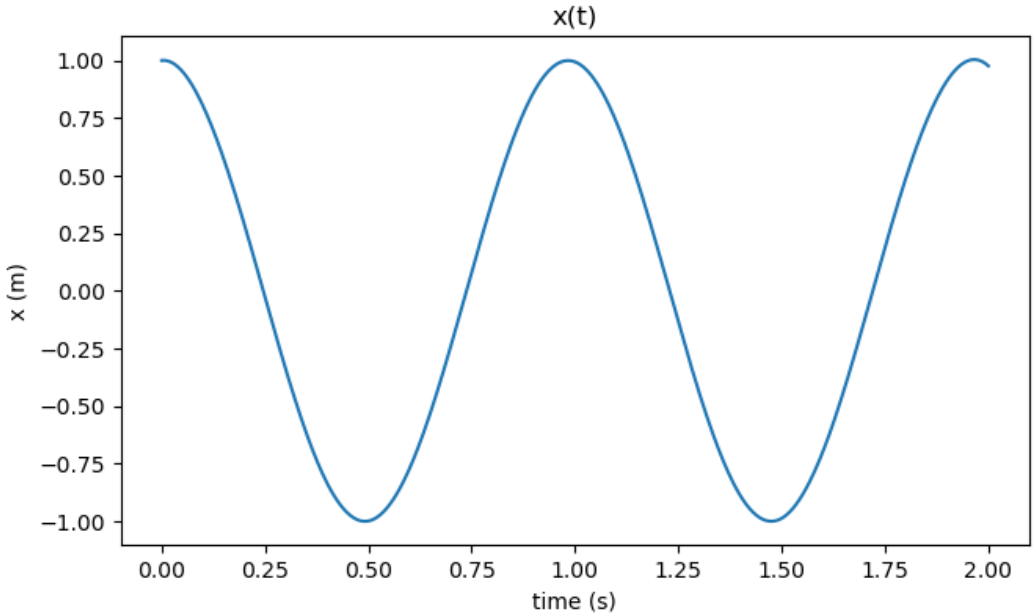

Figure 3: Graph of the position as a function of time given by the SOTA method with $\Delta t = 0.1$, $x_0 = 1$, $v_0 = 0$, and $\omega = 2\pi$ over 20 iterations.

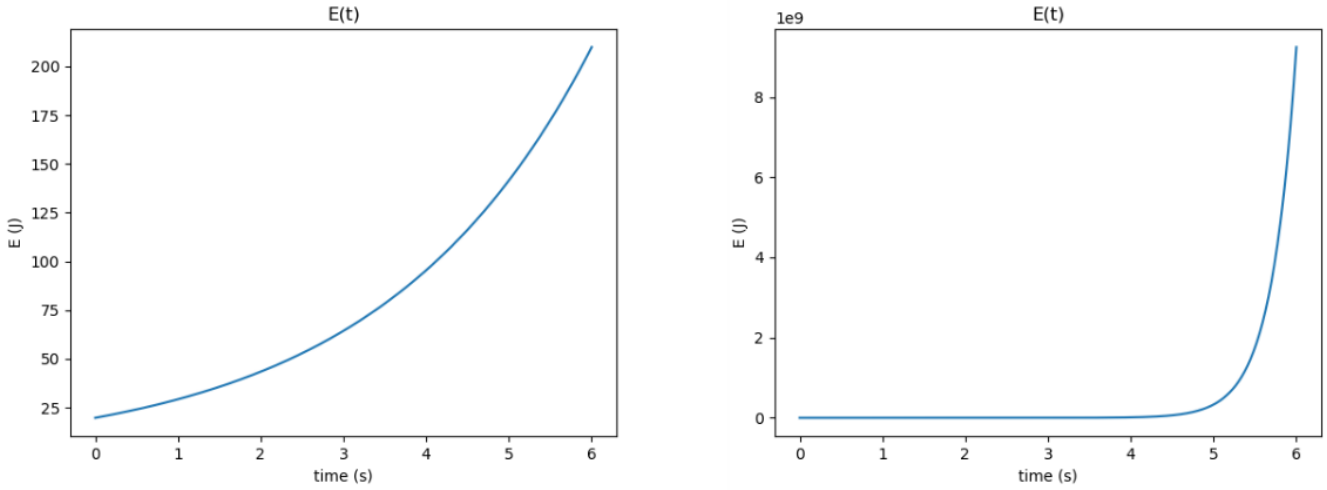

Figure 4: Graph of the energy as a function of time given by the EULER method with $\Delta t = 0.01$, $x_0 = 1$, $v_0 = 0$, and $\omega = 2\pi$ over 600 iterations (left) and with $\Delta t = 0.1$, $x_0 = 1$, $v_0 = 0$, and $\omega = 2\pi$ over 60 iterations (right).

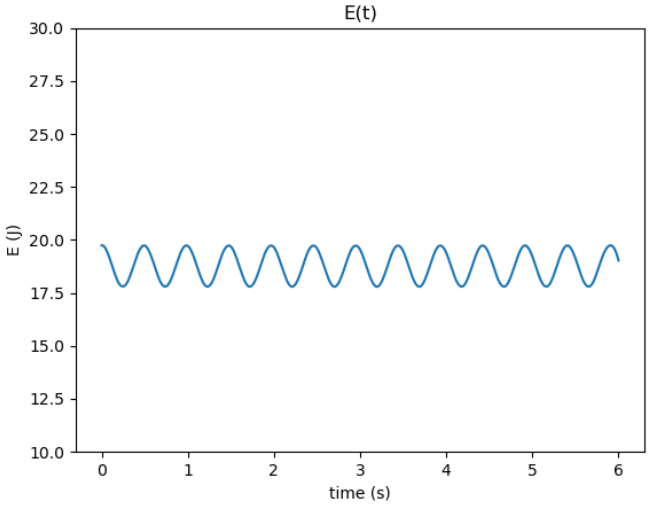

Figure 5: Graph of the energy as a function of time given by the SOTA method with $\Delta t = 0.1$, $x_0 = 1$, $v_0 = 0$, and $\omega = 2\pi$ over 60 iterations.

Exercise 2

Consider the potential $U(x) = A|x|^n$, where A is a constant and n is an integer number:

- Determine the period T of the oscillations: (i) as a function of the energy E; (ii) as a function of the amplitude a of the oscillations ($E = Aa^n$).

  Hint: write the result in terms of the *gamma function* ($\Gamma(x) = \int_0^\infty t^{x-1}e^{-x}dt$), using that:

  $$\int_0^1 \frac{dy}{\sqrt{1-y^n}} = \frac{\sqrt{\pi}}{n} \cdot \frac{\Gamma(\frac{1}{n})}{\Gamma(\frac{1}{2}+\frac{1}{n})}$$

  For what values of n is the period independent on the amplitude a?; what kind of motion do we have in such a case?

- Consider the case $n = 3$, $A = 1$. Write the equation of motion for $m = 1$. Solve this equation numerically using the programme SOTA, which uses the Second Order Taylor Approximation, for the following initial conditions:

  a) $\dot{x}_0 = 0$, $x_0 = 0.01$;
  b) $\dot{x}_0 = 0$, $x_0 = 0.1$;
  c) $\dot{x}_0 = 0$, $x_0 = 1.0$;
  d) $\dot{x}_0 = 0$, $x_0 = 10$;
  e) $\dot{x}_0 = 0$, $x_0 = 100$;

  Plot the period T as a function of the amplitude $a = x_0$. Discuss qualitatively the meaning of the resulting graphics. Plot $\log T$ versus $\log a$; make a least square fitting and compare the value of the slope with the results of the previous question.

From equation (9) from the guide, we know that:

$$T = \sqrt{2m}\int_{x_1}^{x_2} \frac{dx}{\sqrt{E-U(x)}}$$

then, as $U(X) = A|x|^n$:

$$T = \sqrt{2m}\int_{x_1}^{x_2} \frac{dx}{\sqrt{E-A|x|^n}} = \sqrt{\frac{2m}{E}}\int_{x_1}^{x_2} \frac{dx}{\sqrt{1-\frac{A}{E}|x|^n}}$$

We can compute x_1 and x_2 by solving the equation $E = U(x)$:

$$E = A|x|^n \Rightarrow \left(\frac{E}{A}\right)^{1/n} = |x| \Rightarrow x = \begin{cases} x_1 = -\left(\frac{E}{A}\right)^{1/n} \\ x_2 = \left(\frac{E}{A}\right)^{1/n} \end{cases}$$

As we have that $x_1 = -x_2$ and also the function is even, we can say that:

$$T = \sqrt{\frac{2m}{E}}\int_{x_1}^{x_2} \frac{dx}{\sqrt{1-\frac{A}{E}|x|^n}} = 2\sqrt{\frac{2m}{E}}\int_{0}^{x_2} \frac{dx}{\sqrt{1-\frac{A}{E}x^n}}$$

If we then use the change of variables $y = \left(\frac{A}{E}\right)^{1/n} x$, we have that $dx = \left(\frac{E}{A}\right)^{1/n} dy$. As for the integration intervals:

$$[0, x_2] \xrightarrow{y=\left(\frac{A}{E}\right)^{1/n} x} [y(0), y(x_2)] = [0, 1]$$

So, we obtain:

$$T = 2\sqrt{\frac{2m}{E}} \left(\frac{E}{A}\right)^{1/n} \int_0^1 \frac{dy}{\sqrt{1-y^n}} = 2\sqrt{\frac{2m}{E}} \left(\frac{E}{A}\right)^{1/n} \frac{\sqrt{\pi}}{n} \frac{\Gamma(1/n)}{\Gamma(1/2+1/n)}$$

And, finally:

$$\boxed{T_n = \frac{2}{n}\sqrt{\frac{2m\pi}{E}} \left(\frac{E}{A}\right)^{1/n} \frac{\Gamma(1/n)}{\Gamma(1/2+1/n)}}$$

Knowing that $E = Aa^n$, we have that $a = \left(\frac{E}{A}\right)^{1/n}$. Then, writing the previous expression as a function of a, we have that:

$$T_n = \frac{2a}{n}\sqrt{\frac{2m\pi}{Aa^n}} \frac{\Gamma(1/n)}{\Gamma(1/2+1/n)}$$

$$\boxed{T_n = \frac{2}{n}\sqrt{\frac{2m\pi}{A}} \frac{\Gamma(1/n)}{\Gamma(1/2+1/n)} a^{1-\frac{n}{2}}} \tag{2}$$

Therefore, the period T_n is independent of a when $1 - \frac{n}{2} = 0$, which means that:

$$\boxed{T_n \text{ is independent of } a \text{ when } n = 2}$$

In this case, $U(x) = A|x|^2 = Ax^2$, and we have a finite linear motion between the points $-a$ and a. A particle trapped inside the potential well (**Figure 6**) between those two values undergoes an oscillatory motion between $-a$ and a.

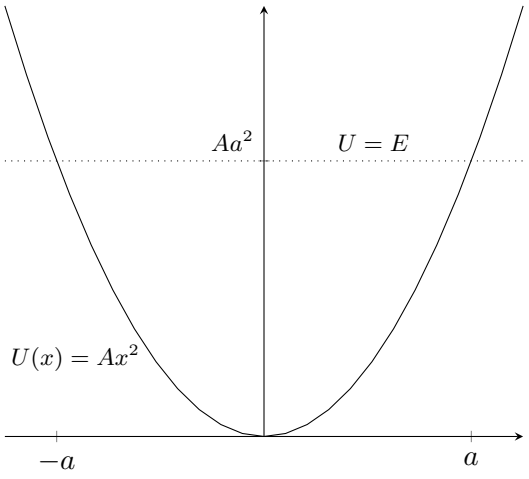

Figure 6: Graph of $U(x) = Ax^2$

If we take the case where $n = 3$, $m = 1$ and $A = 1$, we obtain a potential function defined as $U(x) = |x|^3$. Then, we can calculate the differential equation of motion:

$$U(x) = |x|^3 = \begin{cases} x^3, x \geq 0 \\ -x^3, x < 0 \end{cases} \quad \Rightarrow$$

$$\Rightarrow \frac{dU}{dx} = \left\{ \begin{matrix} 3x^2, x \geq 0 \\ -3x^2, x < 0 \end{matrix} \right\} = 3x|x|$$

Then, knowing that $F = m\ddot{x}$ and $F = -dU/dx$ (conservative forces):

$$\cancel{m}^{1}\ddot{x} = -\frac{dU}{dx} \Rightarrow \ddot{x} + 3x|x| = 0$$

Then, the equation of motion for a particle subject to this potential is:

$$\boxed{\ddot{x} + 3x|x| = 0}$$

If we solve this equation using the SOTA numerical method (Code 4) for the different initial conditions specified in the enunciation of the problem, we obtain the values shown on **Tables 6 to 10** (in **Appendix B**).

Furthermore, we can calculate the period in the different cases for each value of the amplitude by measuring the time between each maximum of the graph (n of peak-to-peak iterations $\times \Delta t$), as shown in **Table 4** (assuming SI units).

x_0 (m)	$\dot{x}_0$ (m/s)	n of peak-to-peak iterations	Δt (s)	T (s)
0.01	0	397	0.1	39.7
0.1	0	126	0.1	12.6
1.0	0	40	0.1	4
10	0	12.5	0.1	1.25
100	0	40	0.01	0.4

Table 4: Period of Motion of the Different Solutions to the equation $\ddot{x} + 3x|x| = 0$

The graph of **Figure 7** shows that the relation between the period T and the amplitude a of the motion is inverse exponential. As the amplitude increases, the period of the motion decreases exponentially.

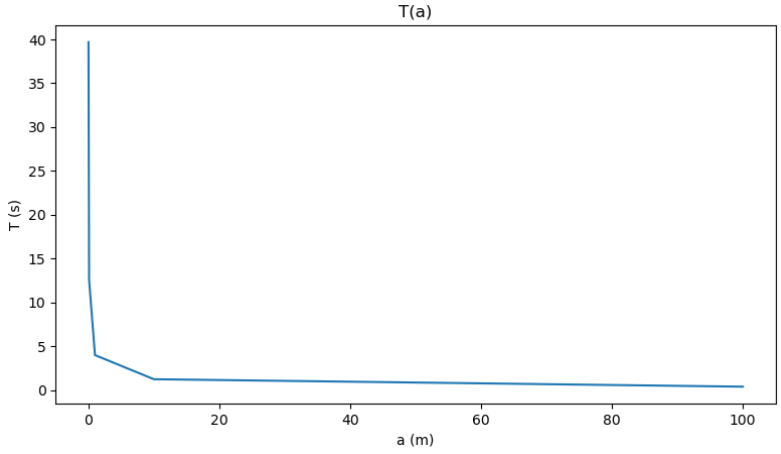

Figure 7: Graph of the period T as a function of the amplitude a.

In order to see this relation better, we can plot $\log T$ against $\log a$, as in **Figure 8** and calculate the fit parameters of the least squares fit of the line $\log T = m \cdot \log a + b$:

a	0.01	0.1	1	10	100
T	39.7	12.6	4	1.25	0.4
$\log a$	-4.6052	-2.3026	0	2.3026	4.6052
$\log T$	3.6814	2.5337	1.3863	0.2231	-0.9163

$$\sum x_i = \sum (\log a)_i = 0$$

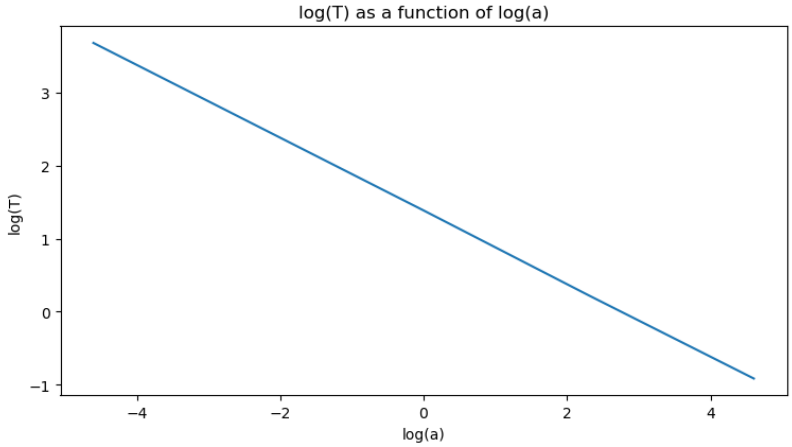

Figure 8: Graph of $\log T$ as a function of $\log a$.

$$\sum y_i = \sum (\log T)_i = 6.9082$$

$$\sum x_i y_i = \sum (\log a)_i \cdot (\log T)_i = -26.4932$$

$$\sum x_i^2 = \sum (\log a)_i^2 = 53.0190$$

$$n = 5$$

$$m = \frac{\sum x_i \sum y_i - n \sum x_i y_i}{\left(\sum x_i\right)^2 - n \sum x_i^2} = \frac{0 \cdot 6.9082 - 5 \cdot (-26.4932)}{0^2 - 5 \cdot 53.0190} = -0.49969 \approx -\frac{1}{2}$$

$$b = \frac{\sum x_i \sum x_i y_i - \sum y_i \sum x_i^2}{\left(\sum x_i\right)^2 - n \sum x_i^2} = \frac{0 \cdot (-26.4932) - 6.9082 \cdot 53.0190}{0^2 - 5 \cdot 53.0190} = 1.3816$$

Therefore, we have:

$$\log T = -\frac{1}{2} \cdot \log a + 1.3816 = \log \frac{1}{\sqrt{a}} + \log e^{1.3816} = \log \frac{e^{1.3816}}{\sqrt{a}} \Rightarrow T = \frac{e^{1.3816}}{\sqrt{a}} \Rightarrow$$

$$\Rightarrow \boxed{T = \frac{3.9813}{\sqrt{a}}}$$

If we calculate T using the result from (2), then we get:

$$T_3 = \frac{2}{3}\sqrt{\frac{2 \cdot 1 \cdot \pi}{1}} \frac{\Gamma(1/3)}{\Gamma(1/2 + 1/3)} a^{1-\frac{3}{2}} = \frac{2\sqrt{2\pi}}{3} \frac{\Gamma(1/3)}{\Gamma(5/6)} a^{-\frac{1}{2}} \xrightarrow[\Gamma(5/6)=1.12879]{\Gamma(1/3)=2.67894}$$

$$\longrightarrow T_3 = \frac{2\sqrt{2\pi}}{3} \frac{\Gamma(1/3)}{\Gamma(5/6)} a^{-\frac{1}{2}} \Rightarrow \boxed{T_3 = \frac{3.9660}{\sqrt{a}}}$$

Clearly, the values that we have obtained for T using both the numerical method and the analytic expression from (2) are very similar. We have yet again demonstrated with this fact that the SOTA numerical method gives very good results when approximating solutions.

Exercise 3B

The potential energy for the interaction between two molecules in a gas can be approximated by the *Lennard-Jones potential*,

$$U(x) = U_0 \left[\left(\frac{a}{x}\right)^{12} - 2\left(\frac{a}{x}\right)^{6} \right], \; x > 0 \tag{3}$$

where U_0, a are positve constants:

- Write the equation of motion.
- Plot $U(x)$ as a fuction of x.
- Identify the stable and unstable equilibrium points.
- Describe the different kinds of motion that can be found as a function of the energy E of the particle. Determine for the case E $>$ 0, assuming that at t = 0 the particle starts at $x(0)$ very large and positive, and with negative velocity v_0, at which value of x the velocity of the particle is maximum and obtain the value of this velocity as a function of v_0.
- If $a = 2.92 \cdot 10^{-10} m$, $U_0 = 2.08 \cdot 10^{-21} J$, find the period of the small oscillations about the stable equilibrium position for a small mass $m = 3.4 \cdot 10^{-27} kg$. Check this result solving numerically the equation of motion (programme SOTA) for several initial conditions with different degrees of proximity to the equilibrium point. Check that, if we separate sufficiently from the equilibrium point, the motion is not harmonic anymore and the value of the period separates from the one predicted for small oscillations.

We know that $F = m\ddot{x}$ and $F = -dU/dx$ (because we have conservative forces). Then:

$$\frac{dU}{dx} = U_0 \left[a^{12}x^{-12} - 2a^6x^{-6}\right]' = a^6 U_0 \left[-12a^6x^{-13} + 12x^{-7}\right] = 12a^6U_0 \left[x^{-7} - a^6x^{-13}\right]$$

Therefore:

$$m\ddot{x} = -12a^6U_0 \left[x^{-7} - a^6x^{-13}\right]$$

And we obtain the general equation of motion:

$$\boxed{m\ddot{x} = 12a^6U_0 \left[a^6x^{-13} - x^{-7}\right]}$$

Plotting $U(x)$ as a function of x, we obtain **Figure 9**. We can see that $U(x)$ has only one equilibrium point at $x = a$ (because $dU/dx = 0$ at $x = a$). It is a minimum, so it is also a stable equilibrium point, which means that we will obtain a simple harmonic motion around a small neighbourhood of it. There are no unstable equilibrium points for this potential, however $U(x)$ goes to zero as x goes to infinity (this is easy to show by taking the limit $\lim_{x\to\infty} U(x) = 0$ or simply by looking at the graph in **Figure 9**).

There are different kinds of motion that can be found in this potential depending on the value of the total energy E and also on the value of x. We will call point A that which corresponds to the minimum potential energy $U(a)$:

- If we have $0 < E$ and we approach point A from the left, then we will surpass it and carry on indefinitely to infinity, as we will never reach the potential barrier on that side. Eventually, most of the energy will be kinetic energy, and the particle will maintain the same speed indefinitely.

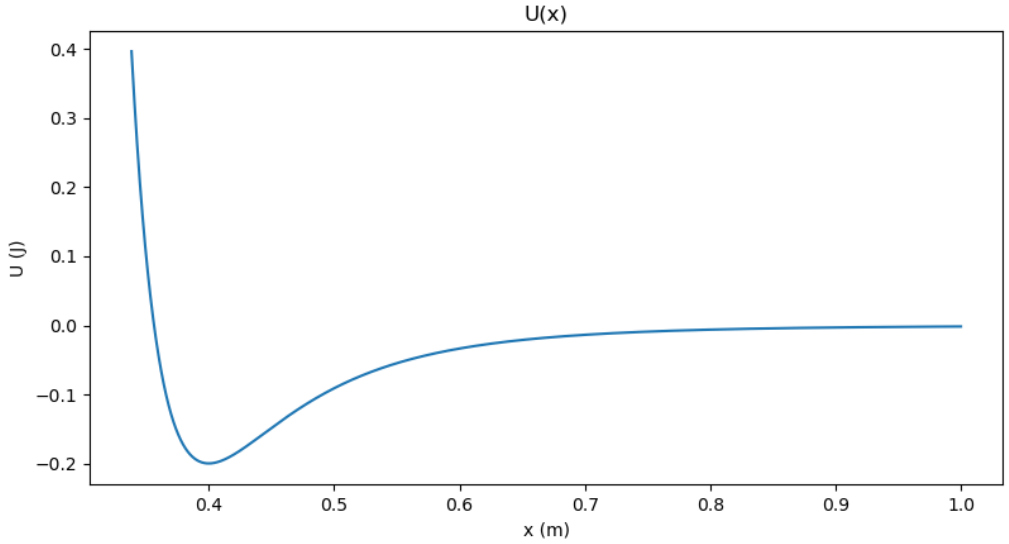

Figure 9: Graph of the potential function $U(x)$ as a function of x for $U_0 = 0.2$ and $a = 0.4$.

- If we have $0 < E$ and we approach point A from the right, then we will surpass it and start to climb the potential until we reach the potential barrier. Then, we will start to lose potential and move back towards point A. We now have the same situation as in the previous case.
- If we have $0 = E$ and we approach point A from the left, then we will surpass it and carry on indefinitely to infinity, as we will never reach the potential barrier on that side. The kinetic energy will be almost zero, although never exactly zero, and the particle will eventually come almost to a stop, although it will always have a slight bit of kinetic energy and it will always have some velocity $v > 0$.
- If we have $0 = E$ and we approach point A from the right, then we will surpass it and start to climb the potential until we reach the potential barrier. Then, we will start to lose potential and move back towards point A. We now have the same situation as in the previous case.
- If we have $U(a) < E < 0$ and we approach point A from the left (or right), we will surpass it but eventually reach a potential barrier and we will start to move back towards point A. As we pass point A again, we will continue until we reach the potential barrier on that side of the potential well, and again we will change directions back towards point A. This will go on indefinitely, and the result will be some kind of periodic motion, which will resemble a harmonic motion if we are in a close neighbourhood of $U(a)$.
- If we have $E = U(a)$, then we will remain in the same point indefinitely.
- E will never be smaller than $U(a)$, because the total energy can never be smaller than the potential energy.

When we work in a neighbourhood of the equilibrium points, we can approximate the potential using its Taylor Expansion:

$$U(x) \approx U(x_0) + \frac{1}{2}\frac{d^2U}{dx^2}(x_0) \cdot (x - x_0)^2$$

In order to calculate this Taylor expansion, we need to calculate:

$$U(x_0) = U(a) = U_0 \left[\left(\frac{a}{a}\right)^{12} - 2\left(\frac{a}{a}\right)^{6} \right] = -U_0$$

$$\frac{d^2U}{dx^2} = 12a^6U_0 \left[13a^6x^{-14} - 7x^{-8}\right]$$

$$\frac{d^2U}{dx^2}(x_0) = \frac{d^2U}{dx^2}(a) = 12a^6U_0\left[13a^6a^{-14} - 7a^{-8}\right] = 12a^{-2}U_0[13-7] = \frac{72U_0}{a^2}$$

Therefore,

$$U(x) \approx -U_0 + \frac{36U_0}{a^2}(x-a)^2$$

$$\frac{dU}{dx}(x) \approx \frac{72U_0}{a^2}(x-a) = k(x-a)$$

where $k = \frac{72U_0}{a^2}$. From this last expression we can clearly see that the motion is harmonic for small oscillations, and we can easily obtain the equation of motion of the system:

$$\left.\begin{aligned} F &= m\ddot{x} \\ F &= -\frac{dU}{dx} \end{aligned}\right\} \rightarrow m\ddot{x} = -\frac{dU}{dx} \Rightarrow m\ddot{x} = -\frac{72U_0}{a^2}(x-a)$$

And if we take $z = x - a$, then $\ddot{z} = \ddot{x}$ and we obtain the equation of motion:

$$\boxed{m\ddot{z} + kz = 0}$$

For the case where $a = 2.92 \cdot 10^{-10}m$, $U_0 = 2.08 \cdot 10^{-21}J$, and $m = 3.4 \cdot 10^{-27}kg$; we can calculate the period of small oscillations about the equilibrium position with the previously calculated equation of motion:

$$m\ddot{z} + kz = 0 \Rightarrow \ddot{z} + \frac{k}{m}z = 0 \Rightarrow \ddot{z} + \omega^2 z = 0$$

where $\omega^2 = k/m$. Then, as $\omega = 2\pi/T$ and $k = 72U_0/a^2$, we have:

$$T = \frac{2\pi}{\omega} = 2\pi\sqrt{\frac{m}{k}} = 2\pi a\sqrt{\frac{m}{72U_0}} = 2\pi \cdot 2.92 \cdot 10^{-10} \cdot \sqrt{\frac{3.4 \cdot 10^{-27}}{72 \cdot 2.08 \cdot 10^{-21}}} \Rightarrow$$

$$\Rightarrow \boxed{T = 2.7644 \cdot 10^{-13}s}$$

If we check this value with the *Matlab* executable that was provided for the SOTA method (with $\Delta t = 8.5 \cdot 10^{-16}$), we obtain the data shown in **Table 5**.

x_0 (m)	n of iterations per period	T (s)	Harmonic shape?
$2.93 \cdot 10^{-10}$	325	$2.76 \cdot 10^{-13}$	Yes
$2.94 \cdot 10^{-10}$	326	$2.771 \cdot 10^{-13}$	Yes
$2.95 \cdot 10^{-10}$	326	$2.771 \cdot 10^{-13}$	Yes
$2.96 \cdot 10^{-10}$	327	$2.7795 \cdot 10^{-13}$	Yes
$2.97 \cdot 10^{-10}$	327	$2.7795 \cdot 10^{-13}$	Yes
$2.99 \cdot 10^{-10}$	329	$2.7965 \cdot 10^{-13}$	Yes
$3.01 \cdot 10^{-10}$	331	$2.8135 \cdot 10^{-13}$	Yes
$3.03 \cdot 10^{-10}$	333	$2.8305 \cdot 10^{-13}$	Yes
$3.05 \cdot 10^{-10}$	336	$2.856 \cdot 10^{-13}$	Yes
$3.07 \cdot 10^{-10}$	340	$2.89 \cdot 10^{-13}$	Yes
$3.08 \cdot 10^{-10}$	341	$2.8985 \cdot 10^{-13}$	Yes
$3.09 \cdot 10^{-10}$	343	$2.9155 \cdot 10^{-13}$	Yes
$3.10 \cdot 10^{-10}$	346	$2.941 \cdot 10^{-13}$	No
$3.20 \cdot 10^{-10}$	369	$3.1365 \cdot 10^{-13}$	No
$3.30 \cdot 10^{-10}$	397	$3.3745 \cdot 10^{-13}$	No
$3.40 \cdot 10^{-10}$	431	$3.6635 \cdot 10^{-13}$	No

Table 5: Period for the different values of x_0

We can clearly see that, for values of x_0 close to a, the real period is very similar to the harmonic period. However, as we start to increase this difference, the values of T quickly start to diverge from it. Not only that, but the shape of the function $x(t)$ gradually starts to lose its sine-like harmonic shape. This is particularly obvious for the last four x_0 values in **Table 5**, which have been marked as "not harmonic" in the last column of the table. **Figure 10** and **Figure 11** clearly show the difference between the graphs of the first and last tests from **Table 5**.

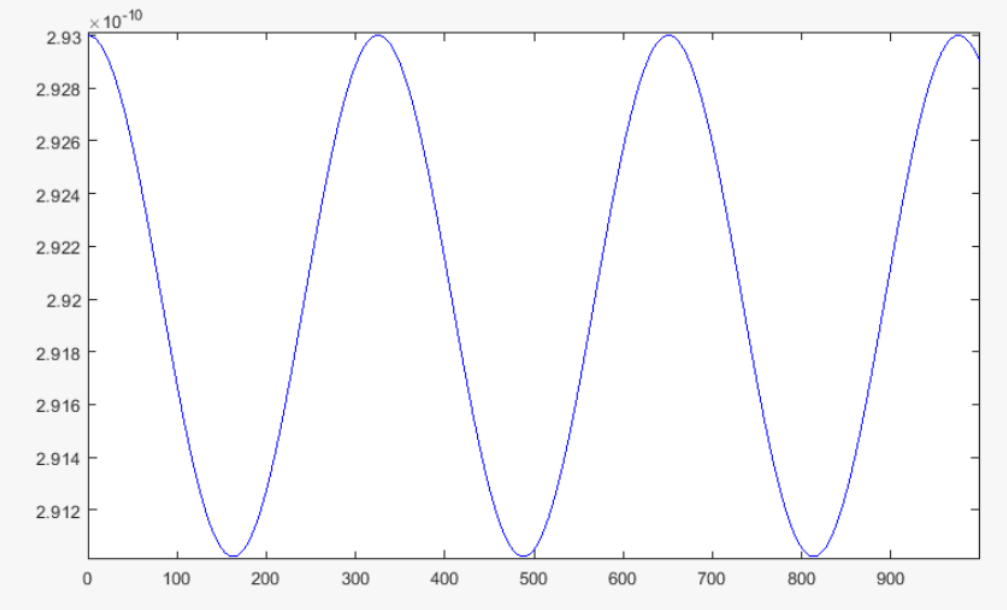

Figure 10: Graph of $x(t)$ for $x_0 = 2.93 \cdot 10^{-10}m$, with $\Delta t = 8.5 \cdot 10^{-16}s$ (generated with *Matlab*). The x axis shows the iteration number and the y axis shows the position in m.

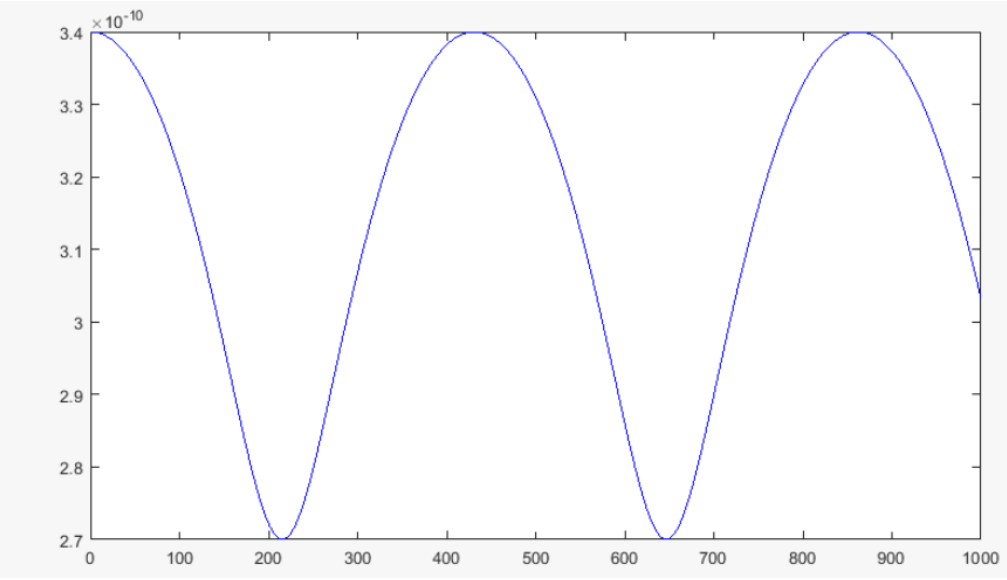

Figure 11: Graph of $x(t)$ for $x_0 = 3.40 \cdot 10^{-10}m$, with $\Delta t = 8.5 \cdot 10^{-16}s$ (generated with *Matlab*). The x axis shows the iteration number and the y axis shows the position in m.

Appendix A: Code

Code 1

Accurate Solution to Question 1

```
'''
The solution to the differential equation mx'' = -kx is of the form:

                        x = A sin(ωt + φ)

where A is the amplitude of the motion and ω = sqrt(k/m).
'''

from numpy import sin, pi, arctan, linspace, array
from colorama import Fore, Style
import matplotlib.pyplot as plt
from scipy.interpolate import make_interp_spline

def check(name, value):
    print(f"\t{Fore.RED}{name} set to {value}{Style.RESET_ALL}")
    return value

######################      Set Initial Parameters      ######################

time_interval = check("Time Interval", float(input("Time interval: ")))

x_0 = check("Initial x", float(input("Initial position in m: ")))

v_0 = check("Initial v", float(input("Initial velocity in m/s: ")))

w_string = input("Angular frequency in rad/s (write \"pi\" to indicate π): ")
w_list = w_string.split("pi")
w = 1

for number in w_list:                #get rational part of ω
    if number:
        w *= int(number)

w_string = str(w) + "π" * bool(len(w_list) - 1)
if len(w_list) - 1 > 1:
    w_string += f"^{len(w_list) - 1}"
w *= pi**(len(w_list) - 1)
check("ω", w_string)

n = check("Number of Iterations", int(input("Number of iterations: ")))

########################          Compute x(t)        ######################
'''
```

```
We know that x(t) = A sin(ωt + φ) and v(t) = Aω cos(ωt + φ). Then, from the
initial values:
                                  |  v_0 = Aω cos(φ)     (1)
                                 <
                                  |  x_0 = A sin(φ)      (2)

Dividing (2) by (1), we obtain: tan(φ) = ω·x_0/v_0. Then:

                                     φ = arctan(ω·x_0/v_0)
                                     A = x_0/sin(φ)
'''

t , x = 0, x_0

if v_0 == 0:
    A = x_0
    phase = pi/2

else:
    phase = arctan(w*x_0/v_0)
    if phase < 0:
        phase += 2*pi
    A = x_0/sin(phase)

values = []

for iteration in range(n):
    values.append((t, x))
    t += time_interval
    x = A*sin(w*t+phase)


##########################          Print x(t)        ######################

print("Time", "\t", "Position")

for point in values:
    print(round(point[0], 4), "\t", round(point[1], 4))


##########################          Plot x(t)         ######################

# x and y axis values
x = [point[0] for point in values]
y = [point[1] for point in values]

# make a curve to fit the data
x = array(x)
y = array(y)
X_Y_Spline = make_interp_spline(x, y)
X_ = linspace(x.min(), x.max(), 500)
Y_ = X_Y_Spline(X_)

# plot the points
plt.plot(X_, Y_)

# name axes and set title
plt.xlabel('time (s)')
```

```
plt.ylabel('x (m)')
plt.title(f'x(t) = {round(A, 3)} · sin({w_string}t + {round(phase, 3)})')

# show the plot
plt.show()
```

Code 2

EULER Algorithm for Question 1

```
from numpy import pi, array, linspace
from colorama import Fore, Style
import matplotlib.pyplot as plt
from scipy.interpolate import make_interp_spline

def check(name, value):
    print(f"\t{Fore.RED}{name} set to {value}{Style.RESET_ALL}")
    return value

#####################      Set Initial Parameters      ####################
time_interval = check("Time Interval", float(input("Time interval: ")))

x_0 = check("Initial x", float(input("Initial position in m: ")))

v_0 = check("Initial v", float(input("Initial velocity in m/s: ")))

w_string = input("Angular frequency in rad/s (write \"pi\" to indicate π): ")
w_list = w_string.split("pi")
w = 1

for number in w_list:               #get rational part of ω
    if number:
        w *= int(number)

w_string = str(w) + "π" * bool(len(w_list) - 1)
if len(w_list) - 1 > 1:
    w_string += f"^{len(w_list) - 1}"
w *= pi**(len(w_list) - 1)
check("ω", w_string)

n = check("Number of Iterations", int(input("Number of iterations: ")))

m = 1                               #set the mass
k = w**2*m                          #set the constant k
E = 1/2*k*x_0**2 + 1/2*m*v_0**2     #set initial energy
t = 0                               #initialize time variable
x_i, v_i = x_0, v_0                 #initialize pos. and vel. variables
a_i = -k*x_0 / m                    #set initial acceleration

values = [(t, x_0, v_0, a_i, E)]    #initialize list to store values

########################          Compute x(t)        ####################

for i in range(n):
    a_i = -k*x_i / m                        #set acceleration i
    x_i += v_i * time_interval              #set position i
```

```
    v_i += a_i * time_interval              #set velocity i
    E = 1/2*k*x_i**2 + 1/2*m*v_i**2         #set energy i
    t += time_interval                      #increment time
    values.append((t, x_i, v_i, a_i, E))    #save the values


#########################          Print x(t)        ######################


headers = ["Time", "Position", "Velocity", "Energy"]

for header in headers:
    print("{:<15}".format(header), end="")

print("")

for row in values:
    for column in [0, 1, 2, 4]:
        print("{:<15}".format(round(row[column], 4)), end="")
    print("")


#########################          Plot x(t)        ######################

# x and y axis values
x = [point[0] for point in values]
y = [point[1] for point in values]

# make a curve to fit the data
x = array(x)
y = array(y)
X_Y_Spline = make_interp_spline(x, y)
X_ = linspace(x.min(), x.max(), 500)
Y_ = X_Y_Spline(X_)

# plot the points
plt.plot(X_, Y_)

# name axes and set title
plt.xlabel('time (s)')
plt.ylabel('x (m)')
plt.title('x(t)')

# show the plot
plt.show()
```

Code 3

SOTA Algorithm for Question 1

```
from numpy import pi, array, linspace
from colorama import Fore, Style
import matplotlib.pyplot as plt
from scipy.interpolate import make_interp_spline

def check(name, value):
```

```
    print(f"\t{Fore.RED}{name} set to {value}{Style.RESET_ALL}")
    return value

#####################      Set Initial Parameters      ####################
time_interval = check("Time Interval", float(input("Time interval: ")))

x_0 = check("Initial x", float(input("Initial position in m: ")))

v_0 = check("Initial v", float(input("Initial velocity in m/s: ")))

w_string = input("Angular frequency in rad/s (write \"pi\" to indicate π): ")
w_list = w_string.split("pi")
w = 1

for number in w_list:                #get rational part of ω
    if number:
        w *= int(number)

w_string = str(w) + "π" * bool(len(w_list) - 1)
if len(w_list) - 1 > 1:
    w_string += f"^{len(w_list) - 1}"
w *= pi**(len(w_list) - 1)
check("ω", w_string)

n = check("Number of Iterations", int(input("Number of iterations: ")))

m = 1                                #set the mass
k = w**2*m                           #set the constant k
E = 1/2*k*x_0**2 + 1/2*m*v_0**2      #set initial energy
t = 0                                #initialize time variable
x_i, v_i = x_0, v_0                  #initialize pos. and vel. variables
a_i = -k*x_0 / m                     #set initial acceleration

values = [(t, x_0, v_0, a_i, E)]     #initialize list to store values

########################          Compute x(t)        ######################

for i in range(n):
    a_i = -k*x_i / m
    x_i += v_i * time_interval + 1/2 * a_i * time_interval**2
    a_i_plus_1 = -k*x_i / m
    v_i += 1/2 * (a_i + a_i_plus_1) * time_interval
    t += time_interval
    E = 1/2*k*x_i**2 + 1/2*m*v_i**2
    values.append((t, x_i, v_i, a_i, E))


########################          Print x(t)        ######################


headers = ["Time", "Position", "Velocity", "Energy"]

for header in headers:
    print("{:<15}".format(header), end="")

print("")

for row in values:
```

```
    for column in [0, 1, 2, 4]:
        print("{:<15}".format(round(row[column], 4)), end="")
    print("")


########################          Plot x(t)        ######################

# x and y axis values
x = [point[0] for point in values]
y = [point[1] for point in values]

# make a curve to fit the data
x = array(x)
y = array(y)
X_Y_Spline = make_interp_spline(x, y)
X_ = linspace(x.min(), x.max(), 500)
Y_ = X_Y_Spline(X_)

# plot the points
plt.plot(X_, Y_)

# name axes and set title
plt.xlabel('time (s)')
plt.ylabel('x (m)')
plt.title('x(t)')

# show the plot
plt.show()
```

Code 4

SOTA Algorithm for Question 2

```
from numpy import pi, array, linspace
from colorama import Fore, Style
import matplotlib.pyplot as plt
from scipy.interpolate import make_interp_spline

def check(name, value):
    print(f"\t{Fore.RED}{name} set to {value}{Style.RESET_ALL}")
    return value

####################      Set Initial Parameters      ####################
time_interval = check("Time Interval", float(input("Time interval: ")))

x_0 = check("Initial x", float(input("Initial position in m: ")))

v_0 = check("Initial v", float(input("Initial velocity in m/s: ")))

n = check("Number of Iterations", int(input("Number of iterations: ")))


m = 1                                   #set the mass
E = 1/2 * m * v_0**2 + abs(x_0)**3      #set initial energy
t = 0                                   #initialize time variable
x_i, v_i = x_0, v_0                     #initialize pos. and vel. variables
```

```
a_i = -1/m * 3 * x_i**2 * (x_i / abs(x_i))  #set initial acceleration

values = [(t, x_0, v_0, a_i, E)]    #initialize list to store values

########################          Compute x(t)        ######################


for i in range(n):
    a_i = -1/m * 3 * x_i**2 * (x_i / abs(x_i))
    x_i += v_i * time_interval + 1/2 * a_i * time_interval**2
    a_i_plus_1 = -1/m * 3 * x_i**2 * (x_i / abs(x_i))
    v_i += 1/2 * (a_i + a_i_plus_1) * time_interval
    t += time_interval
    E = 1/2 * m * v_i**2 + abs(x_i)**3
    values.append((t, x_i, v_i, a_i, E))


#########################          Print x(t)        ######################

headers = ["Time", "Position", "Velocity", "Energy"]

for header in headers:
    print("{:<15}".format(header), end="")

print("")

for row in values:
    for column in [0, 1, 2, 4]:
        print("{:<15}".format(round(row[column], 4)), end="")
    print("")


#########################          Plot x(t)        ######################

# x and y axis values
x = [point[0] for point in values]
y = [point[1] for point in values]

# make a curve to fit the data
x = array(x)
y = array(y)
X_Y_Spline = make_interp_spline(x, y)
X_ = linspace(x.min(), x.max(), 500)
Y_ = X_Y_Spline(X_)

# plot the points
plt.plot(X_, Y_)

# name axes and set title
plt.xlabel('time (s)')
plt.ylabel('x (m)')
plt.title('x(t)')

# show the plot
plt.show()
```

Appendix B: Tables

Time	Position	Velocity	Energy
0	0.01	0.0	$1 \cdot 10^{-6}$
0.1	$9.999 \cdot 10^{-3}$	$-3.00 \cdot 10^{-5}$	$1 \cdot 10^{-6}$
0.2	$9.994 \cdot 10^{-3}$	$-6.00 \cdot 10^{-5}$	$1 \cdot 10^{-6}$
0.3	$9.987 \cdot 10^{-3}$	$-8.99 \cdot 10^{-5}$	$1 \cdot 10^{-6}$
0.4	$9.976 \cdot 10^{-3}$	$-1.198 \cdot 10^{-4}$	$1 \cdot 10^{-6}$
0.5	$9.963 \cdot 10^{-3}$	$-1.496 \cdot 10^{-4}$	$1 \cdot 10^{-6}$
0.6	$9.946 \cdot 10^{-3}$	$-1.793 \cdot 10^{-4}$	$1 \cdot 10^{-6}$
0.7	$9.927 \cdot 10^{-3}$	$-2.090 \cdot 10^{-4}$	$1 \cdot 10^{-6}$
0.8	$9.904 \cdot 10^{-3}$	$-2.385 \cdot 10^{-4}$	$1 \cdot 10^{-6}$
0.9	$9.879 \cdot 10^{-3}$	$-2.678 \cdot 10^{-4}$	$1 \cdot 10^{-6}$
1.0	$9.851 \cdot 10^{-3}$	$-2.970 \cdot 10^{-4}$	$1 \cdot 10^{-6}$

Table 6: Numerical Solution Using the SOTA Method with $\Delta t = 0.1$, $x_0 = 0.01$ and $\dot{x}_0 = 0$

Time	Position	Velocity	Energy
0	0.1	0.0	0.001
0.1	$9.985 \cdot 10^{-2}$	$-2.996 \cdot 10^{-3}$	0.001
0.2	$9.940 \cdot 10^{-2}$	$-5.973 \cdot 10^{-3}$	0.001
0.3	$9.866 \cdot 10^{-2}$	$-8.915 \cdot 10^{-3}$	0.001
0.4	$9.762 \cdot 10^{-2}$	$-1.180 \cdot 10^{-2}$	0.001
0.5	$9.629 \cdot 10^{-2}$	$-1.462 \cdot 10^{-2}$	0.001
0.6	$9.469 \cdot 10^{-2}$	$-1.736 \cdot 10^{-2}$	0.001
0.7	$9.282 \cdot 10^{-2}$	$-2.000 \cdot 10^{-2}$	0.001
0.8	$9.069 \cdot 10^{-2}$	$-2.252 \cdot 10^{-2}$	0.001
0.9	$8.832 \cdot 10^{-2}$	$-2.493 \cdot 10^{-2}$	0.001
1.0	$8.571 \cdot 10^{-2}$	$-2.720 \cdot 10^{-2}$	0.001

Table 7: Numerical Solution Using the SOTA Method with $\Delta t = 0.1$, $x_0 = 0.1$ and $\dot{x}_0 = 0$

Time	Position	Velocity	Energy
0	1.0	0.0	1.0
0.1	0.985	−0.296	0.999
0.2	0.941	−0.574	0.998
0.3	0.870	−0.820	0.995
0.4	0.777	−1.024	0.993
0.5	0.665	−1.181	0.992
0.6	0.541	−1.294	0.992
0.7	0.407	−1.360	0.993
0.8	0.269	−1.396	0.994
0.9	0.128	−1.409	0.995
1.0	−0.013	−1.412	0.996

Table 8: Numerical Solution Using the SOTA Method with $\Delta t = 0.1$, $x_0 = 1.0$ and $\dot{x}_0 = 0$

Time	Position	Velocity	Energy
0	10.0	0.0	1000.0
0.1	8.5	−25.838	947.913
0.2	4.832	−40.178	919.988
0.3	0.464	−43.713	955.525
0.4	−3.91	−41.452	918.927
0.5	−7.826	−29.972	928.475
0.6	−9.905	−6.07	990.048
0.7	−9.04	20.903	957.24
0.8	−5.724	38.076	912.417
0.9	−1.425	43.295	940.114
1.0	2.935	42.307	920.232

Table 9: Numerical Solution Using the SOTA Method with $\Delta t = 0.1$, $x_0 = 10$ and $\dot{x}_0 = 0$

Time	Position	Velocity	Energy
0	100.0	0.0	1000000
0.1	−1.329	−1411.586	996290
0.2	−99.953	56.288	1000172
0.3	3.996	1411.529	996271
0.4	99.800	−112.672	1000348
0.5	−6.674	−1411.202	996043
0.6	−99.531	168.938	1000266
0.7	9.353	1410.499	995572
0.8	99.147	−224.734	999895
0.9	−12.023	−1409.482	995058
1.0	−98.658	279.830	999435

Table 10: Numerical Solution Using the SOTA Method with $\Delta t = 0.01$, $x_0 = 100$ and $\dot{x}_0 = 0$