# Computational Physics Homework Report

Fateme RamezanZade-99100693 Problem Set 9

## 1. Dependence of error to stepsize in Euler algorithm

### 1.1 Functions Description

f(x,t):

This function is dedicated to the function responsible for the variable's change. In this problem we have:

$$\frac{dx}{dt} = -\frac{x}{RC}$$

Euler_step(x_0,h,T_f):

This function performs Euler algorithm from initial value (x_0) with our desired stepsize (h) until final time (T_f). firstly, the interval (0,T_f) is divided to parts with h length, and the number of parts is stored as N to be used as the number of required steps. Then the Euler algorithm is performed in a loop and x amounts are stored in the x_arr array:

$$x_{n+1} = x_n + f(x_n)h$$

In order to be able to see the cutoff error, we need to store our variables as float16. The function lastly returns x_arr and the array of time intervals.

analetic_solve(x_0,t):

This function returns the value of analytic solution at given time (t):

$$x = x_0 e^{-\frac{t}{RC}}$$

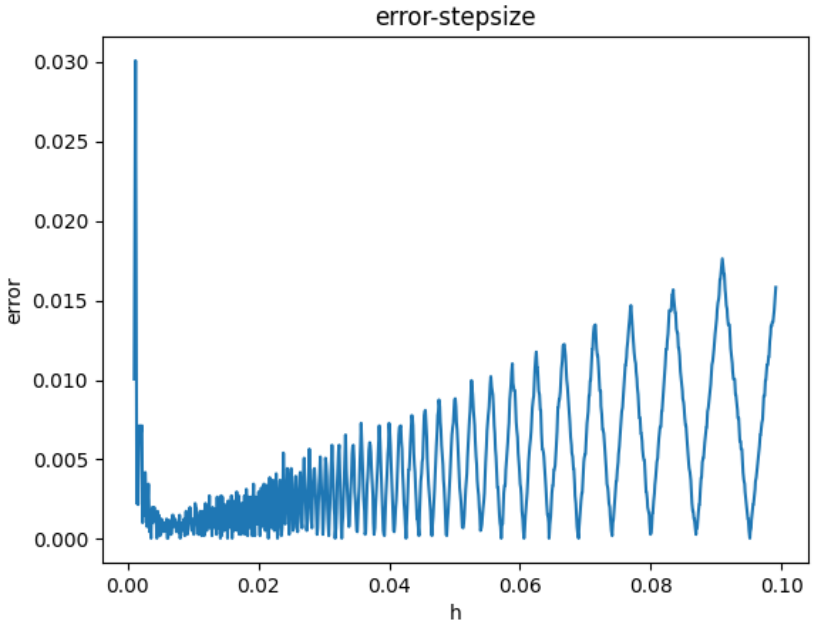
### 1.2 Main Code

Firstly, the values of x_0 and final time are defined. Then the interval (0.001,0.1) is divided to parts with 0.001 length and the values are sorted in an array with datatype float16. Now for each element in this array, the absolute value of the difference between analytical solution and numerical solution is calculated and stored as float16 in an array called error_r. lastly, each element of this array is plotted with corresponding h.

### 1.3 Results

Initial values: x_0=1 and T_f=1

We know that the order of Euler algorithm error is $\mathcal{O}(h)$. That's why the error gets larger as we increase h. on the other hand, for very small h values the cutoff error shows up, leading to even more error in our calculation. We can see that the best value of h is around 0.004.



## 2. Instability of Algorithms

### 2.1 Functions Description

The functions f(x,t) and analetic_solve(x_0,t) are explained before.

numeric_step(x_0,h,T_f):

this function works the same as the euler_step function in the previous part, except that it updates the values of x using this algorithm:

(It is worth mentioning that I calculated the amount of $x_{-1}$ using the Euler method at first.)

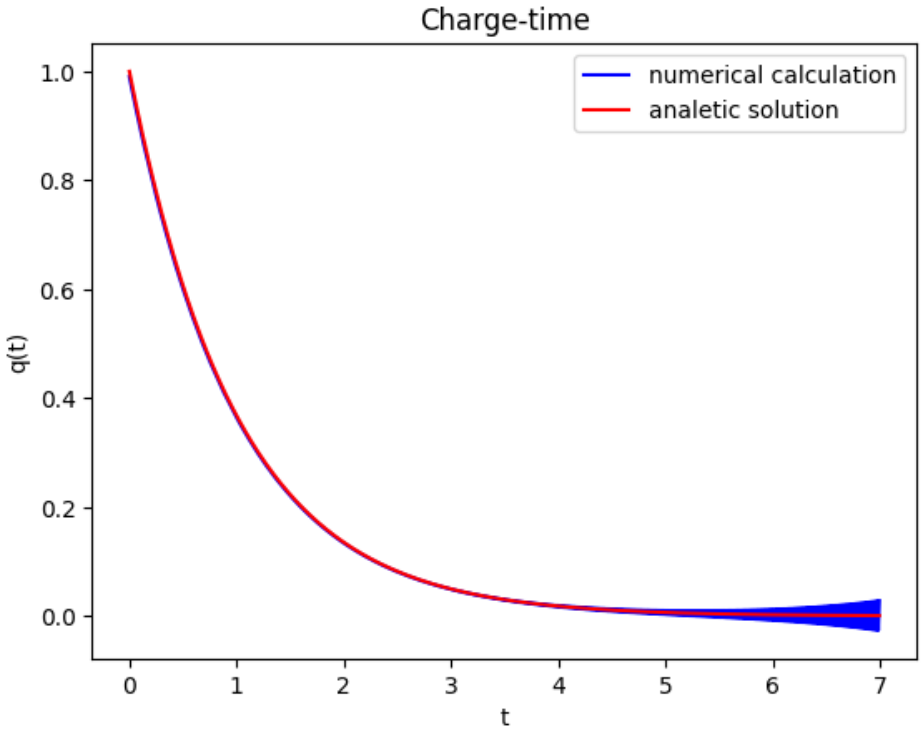
$$x_{n+1} = x_{n-1} + 2f(x_n)h$$

### 2.2 Main Code

Firstly, the values of final time, initial x and stepsize are defined. Then the numeric_step function is performed and both the results of this and analetic_solve function are plotted in a graph versus time.

### 2.3 Results

Initial values: x_0=1 , T_f=7 and h=0.01




As expected, we can see that our numerical calculation works well until a certain time, and shows growing error after that. As expected in the class, this algorithm has a term that grows by increasing the number of steps, which leads to an oscillating error that can be seen in the graph.

## 3. Comparing different algorithms for harmonic oscillator problem

### 3.1 Functions Description

E(x,v):

This function returns the value of total energy for given x and v:

$$E = \frac{1}{2}x^2 + \frac{1}{2}v^2$$

analetic_solve(x_0,t):
This function returns the value of analytic solution at given time (t):

$$x = x_0 \cos t + v_0 \sin t \quad , \quad v = v_0 \cos t - x_0 \sin t$$

The rest of the functions are different algorithms for numerical calculation. They work the same way as the euler_step function in the first part, unless they update and return both x and v. They also calculate the value of energy using the function E at each step and return the amounts stored in an array. For each algorithm, the ways that the parameters are updated are as follows ($f(x) = -x$)):
eulerK_step(x_0,v_0,h,T_f):

$$x_{n+1} = x_n + v_{n+1}h \quad , \quad v_{n+1} = v_n + f(x_n)h$$

euler_step(x_0,v_0,h,T_f):

$$x_{n+1} = x_n + v_n h \quad , \quad v_{n+1} = v_n + f(x_n)h$$

frog(x_0,v_0,h,T_f):

$$x_{n+1} = x_n + v_{n+1/2}h \quad , \quad v_{n+3/2} = v_{n+1/2} + f(x_{n+1})h$$

(In this algorithm, I calculated the value of $v_{1/2}$ using Euler method at first)
vele(x_0,v_0,h,T_f):

$$x_{n+1} = x_n + v_n h + \frac{1}{2}f(x_n)h^2 \quad , \quad v_{n+1} = v_n + \frac{h}{2}\big(f(x_n) + f(x_{n+1})\big)$$

beeman(x_0,v_0,h,T_f):

$$x_{n+1} = x_n + v_n h + \frac{1}{6}\big(4f(x_n) - f(x_{n-1})\big)h^2 \quad , \quad v_{n+1} = v_n + \frac{h}{6}(2f(x_{n+1}) + 5f(x_n) - f(x_{n-1}))$$

(In this algorithm, I calculated the value of $x_{-1}$ using Euler method at first)

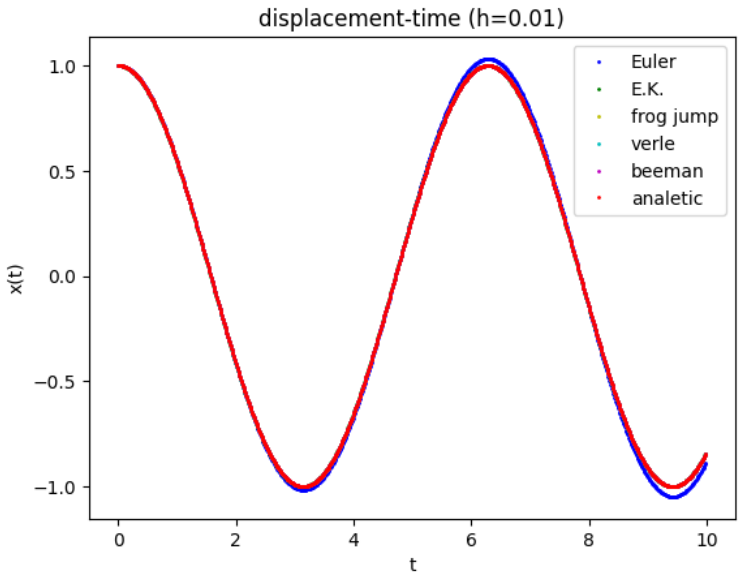
### 3.2 Main Code

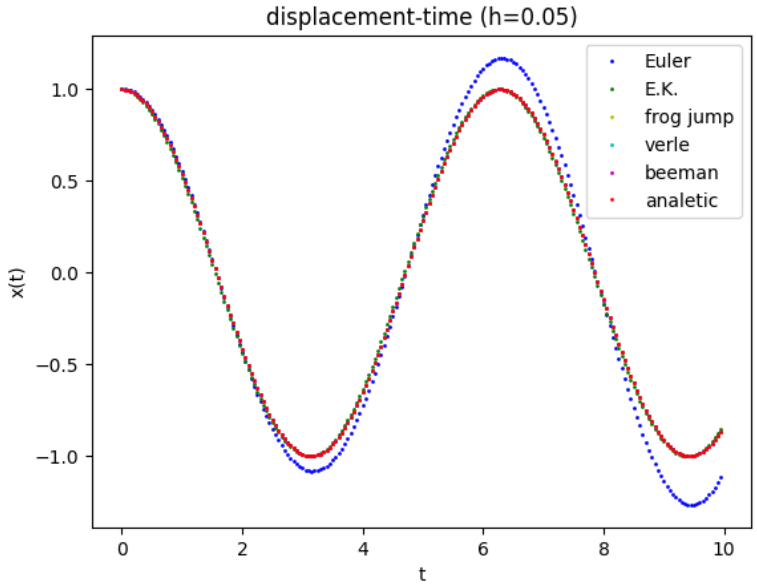
Firstly, the initial values of x and v and the final time are defined. Then a list of different stepsizes is created. For each stepsize, the numerical calculation functions are performed and the graphs of x over t, v over x (phase diagram) and E over t are plotted.

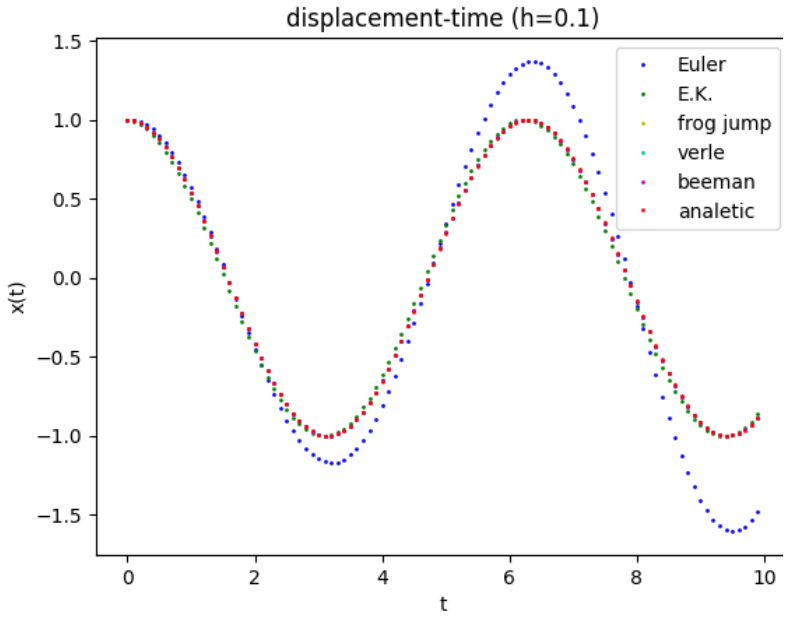
### 3.3 Results

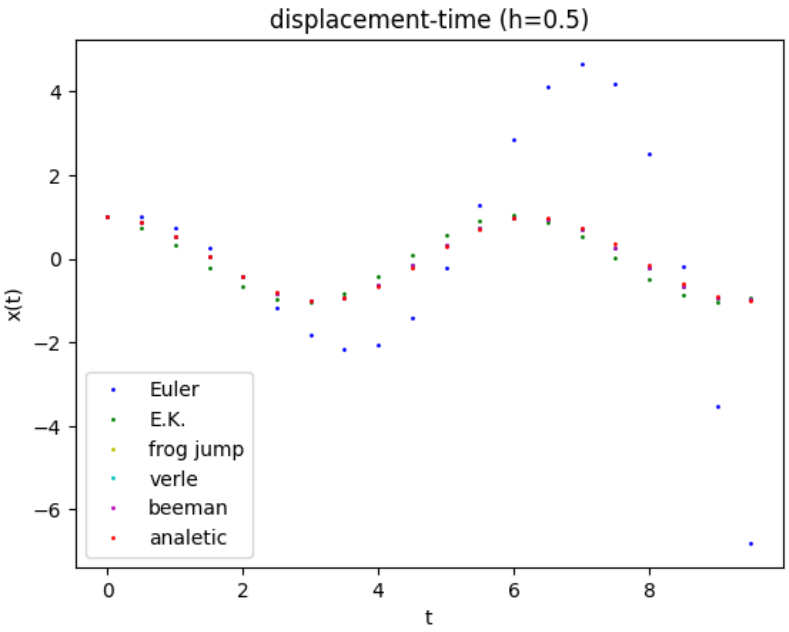
(x_0=1 , v_0=0 , T_f=10)

#### 3.3.1 Displacement over time:

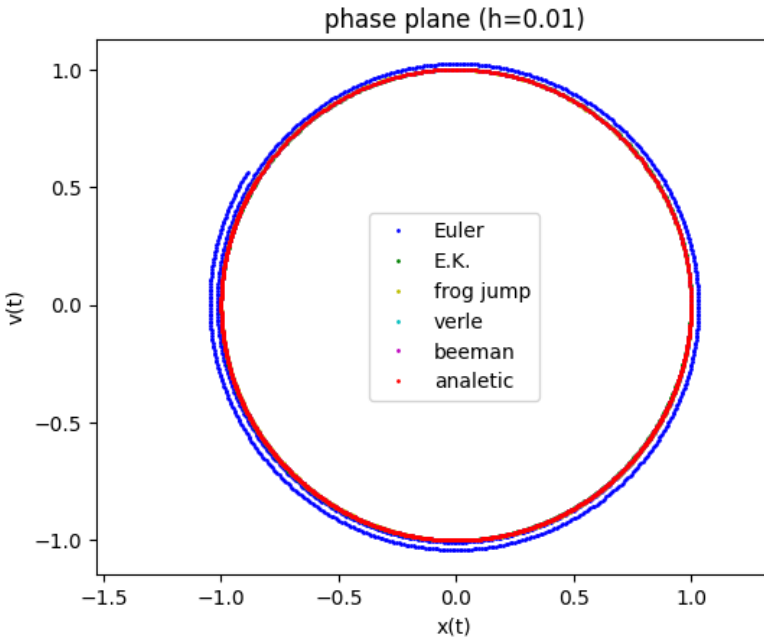


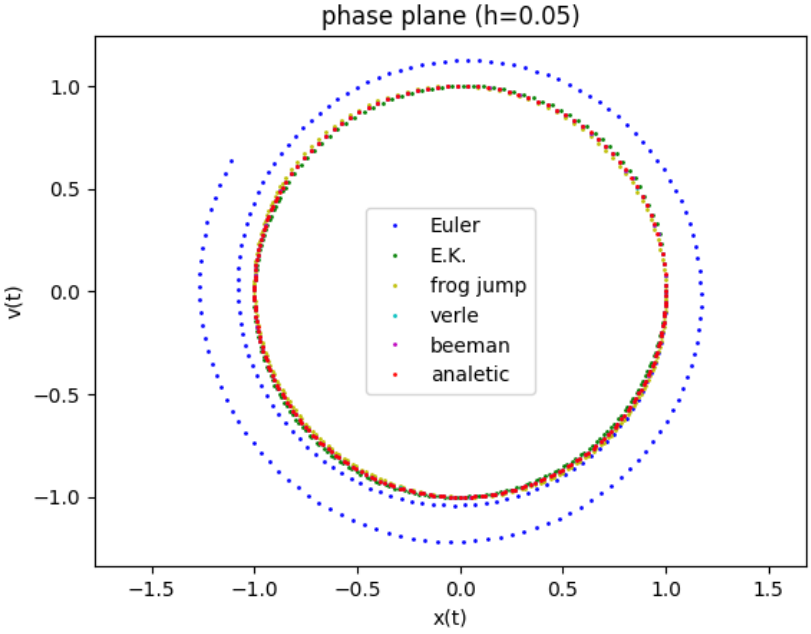



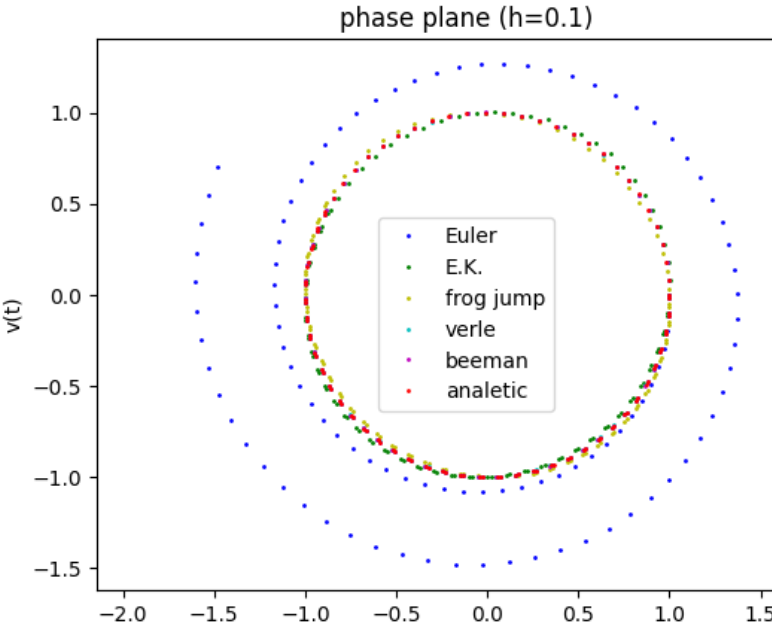


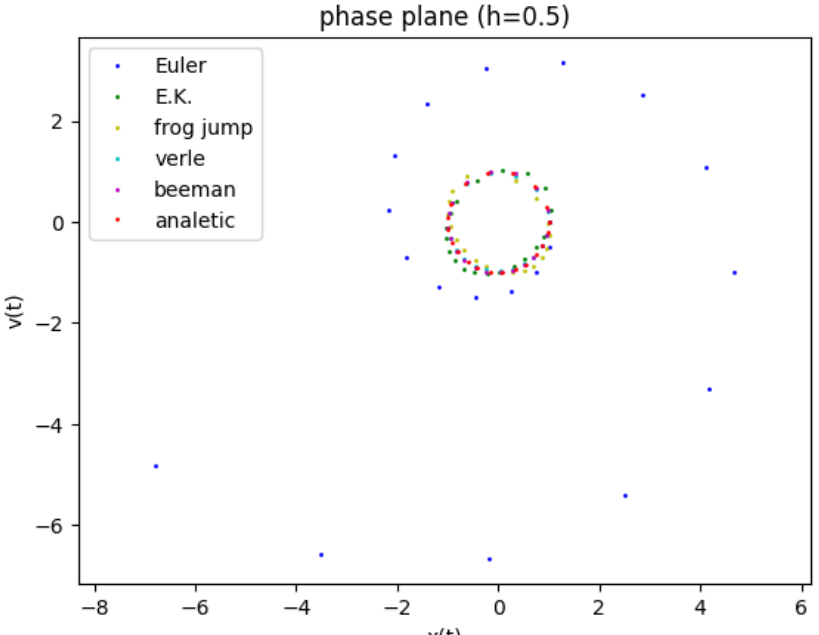




### 3.3.2. Phase Plane

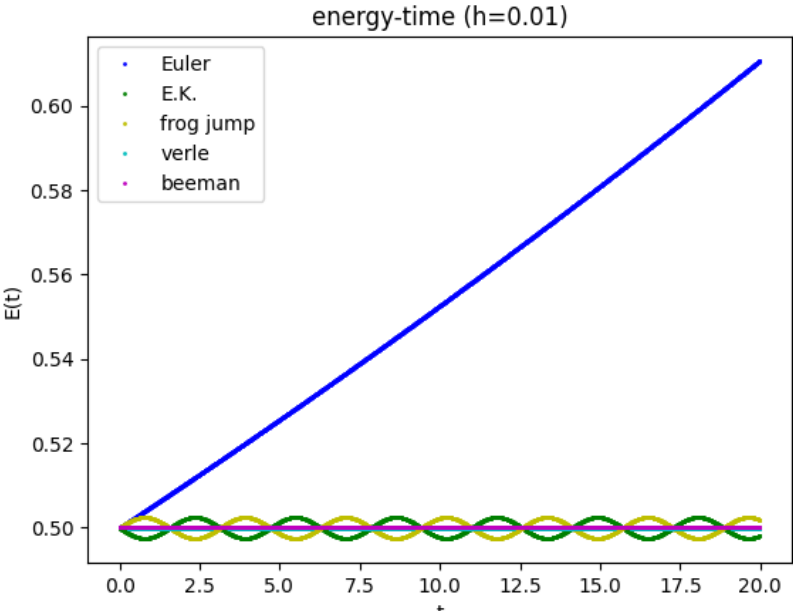


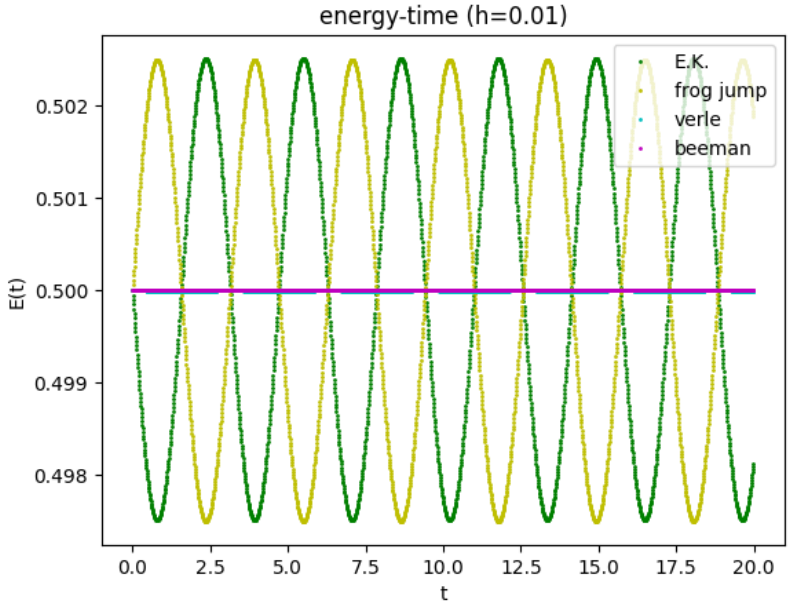



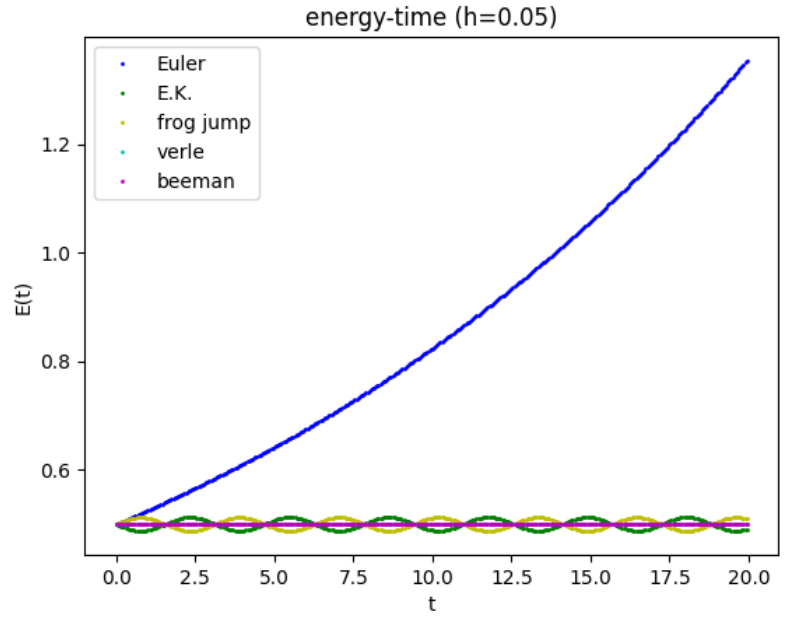



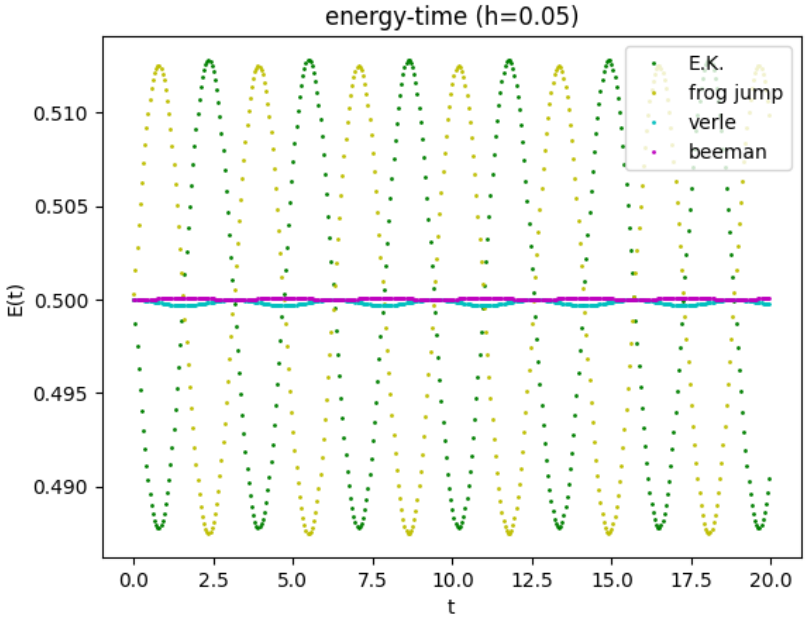



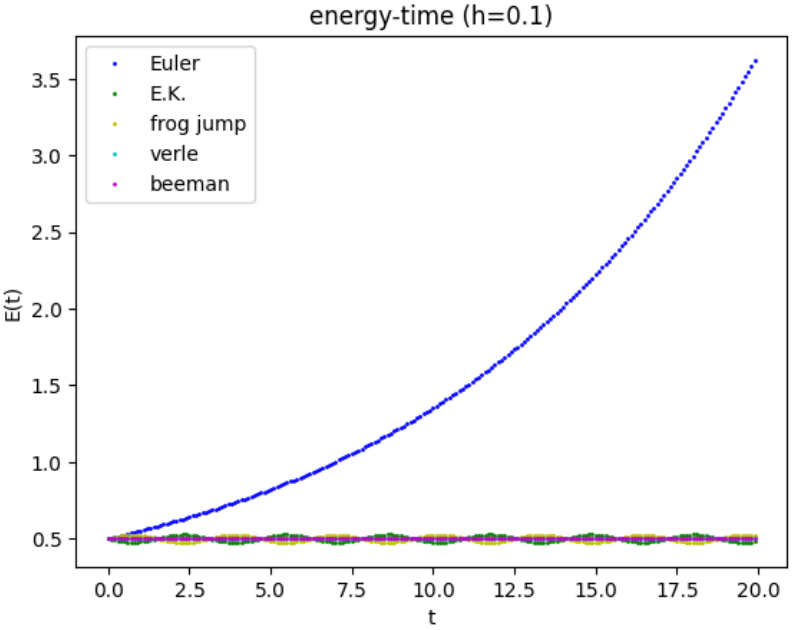
### 3.3.2 Conservation of Energy

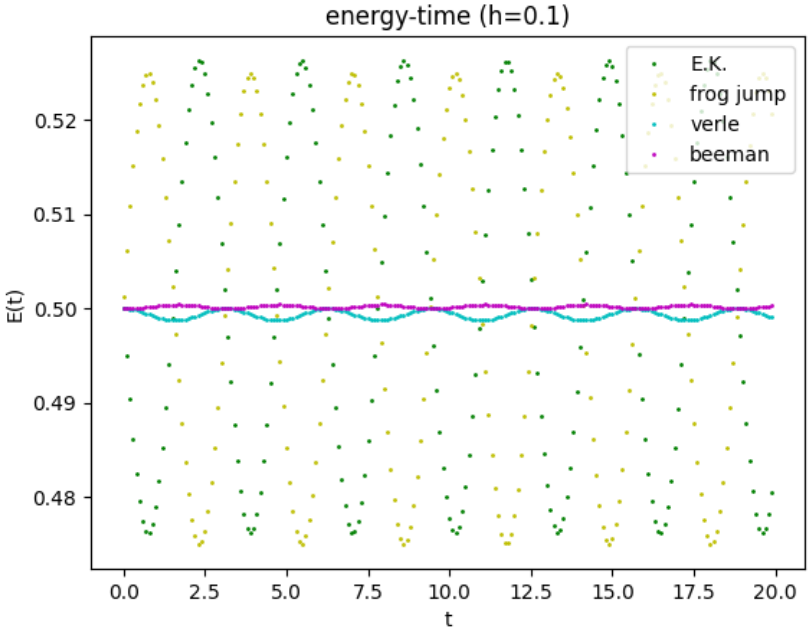


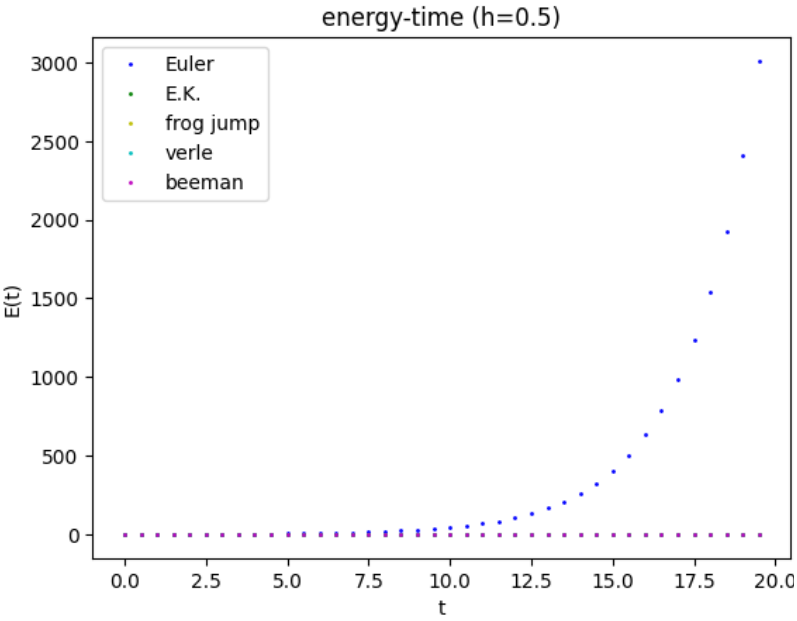



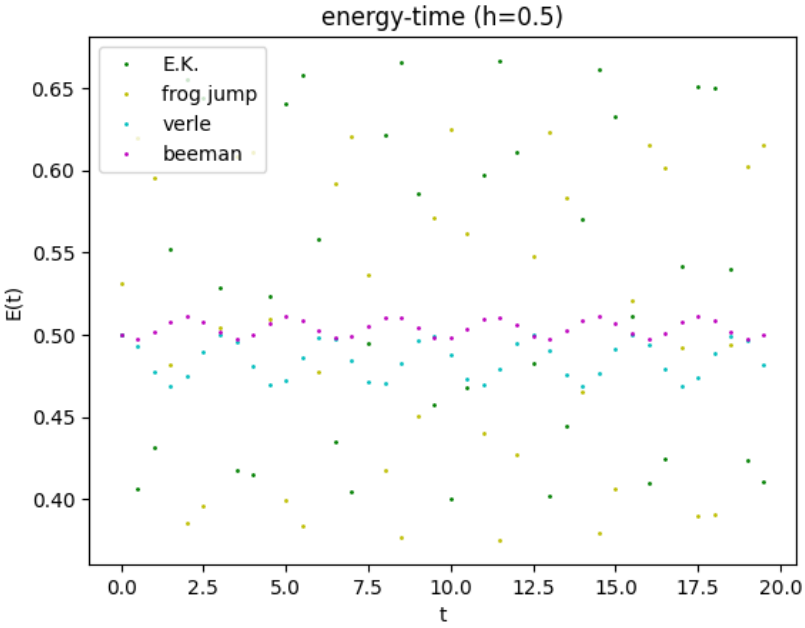






We learned that conservation of energy indicates the accuracy of an algorithm. As expected, smaller stepsizes show smaller changes in energy and hence better accuracy. At a given stepsize, we can see that Beeman algorithm is the most accurate, followed by verlet algorithm. Frog jump and Euler-Cromer algorithms are approximately the same, and Euler algorithm is the worst one.