# Computational Physics - Exercise 4

Joel Schumacher, 309231, joel.schumacher@rwth-aachen.de

June 14, 2017

## Euler Integration

### Introduction

We assume a system of a single harmonic oscillator with $m = 1$ and $k = 1$:

$$\begin{aligned} V(x) &= x^2/2 \\ V'(x) &= x \\ a(x) &= -x \end{aligned}$$

The boundary conditions are:

$$\begin{aligned} x(0) &= 0 \\ v(0) &= 1 \end{aligned}$$

### Simulation model

To examine the time evolution of the formerly described system, we employ Euler integration, which algorithm is as follows:

$$\begin{aligned} x(t + \Delta t) &= x(t) + v(t)\Delta t \\ v(t + \Delta t) &= v(t) + a(x(t))\Delta t \end{aligned}$$

The euler algorithm is not symplectic, meaning it does not conserve phase space volume and therefore energy and is not stable.

### Simulation results

In figure 1 the unstable behaviour of the Euler algorithm can be observed for every time step size. A smaller timesteps merely corresponds to a slower divergence of the exact result.

## Euler-Cromer Integration

### Simulation model

Additionally the Euler-Cromer algorithm is used, which differs from the Euler algorithm, by either using the new velocity to calculate the new position or by using the new position to calculate the acceleration and consequently the new velocity. Formally:

$$\begin{aligned} v(t + \Delta t) &= v(t) + a(x(t))\Delta t \\ x(t + \Delta t) &= x(t) + v(t + \Delta t)\Delta t \end{aligned}$$

which will henceforth be referred to as “method A”. And also:

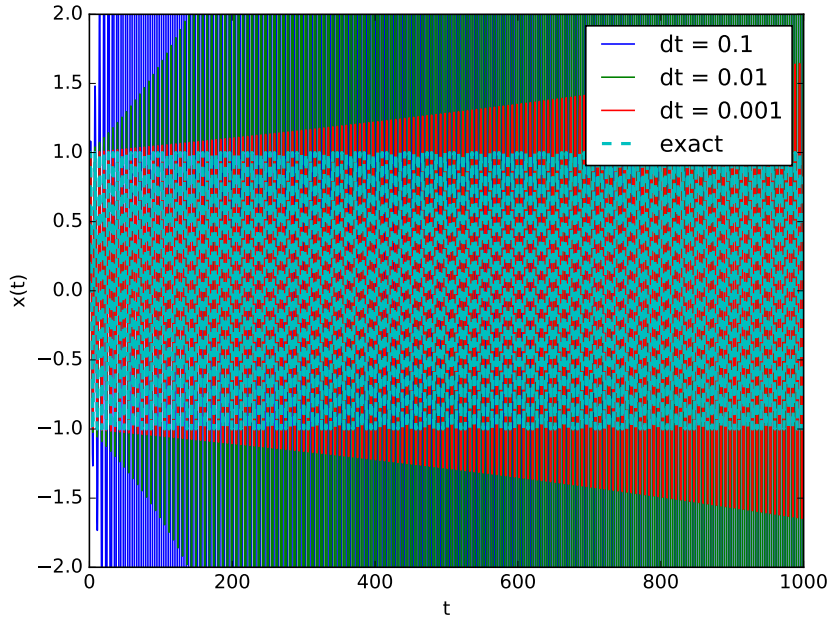


Figure 1: Time evolution of the position of the harmonic oscillator simulated using Euler integration.

$$x(t + \Delta t) = x(t) + v(t)\Delta t$$
$$v(t + \Delta t) = v(t) + a(x(t + \Delta t))\Delta t$$

from now on called "method B".

The Euler-Cromer variation of the algorithm is symplectic and significantly more stable than the regular Euler method.

### Simulation results

In figure 2 and 3 it is clearly visible that the very unstable behaviour disappeared and we no longer observe an obvious divergence from the exact solution for small and time step widths.

When looking at smaller time ranges and also considering the total energy of the system, we can observe that in general the simulation is in good agreement with the exact solution as can be seen in figure 4 and 5. The energy however shows slight deviations for both method A and B.

## Velocity Verlet Integration

### Simulation model

The velocity verlet integration is a second order integrator, which a widely used integrator for molecular dynamics, because it is computationally relatively inexpensive, symplectic and time-reversible. The algorithm is as follows:

$$x(t + \Delta t) = x(t)\Delta t + \frac{1}{2}a(x(t))(\Delta t)^2$$
$$v(t + \Delta t) = v(t) + \frac{1}{2}\left[a(x(t)) + a(x(t + \Delta t)\right]\Delta t$$

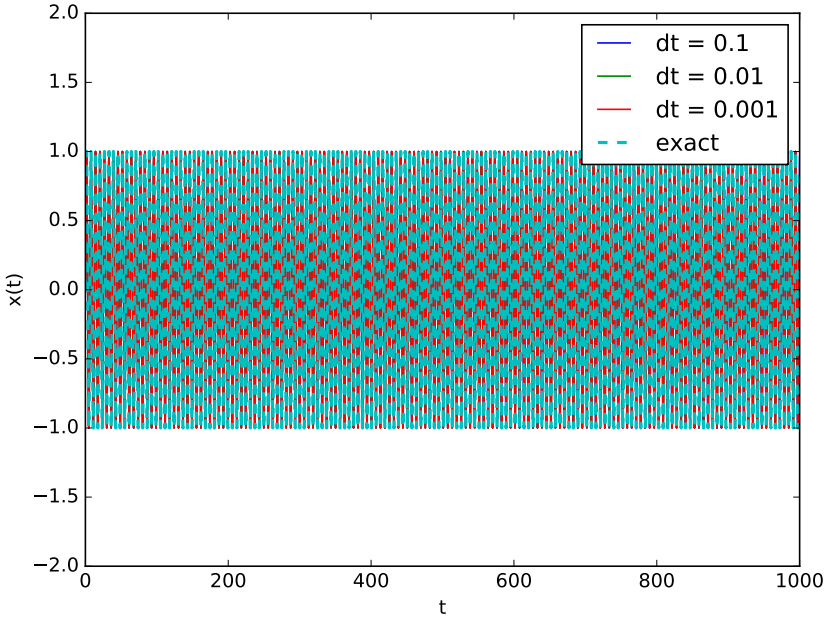


Figure 2: Time evolution of harmonic oscillator using the Euler-Cromer algorithm (method A).

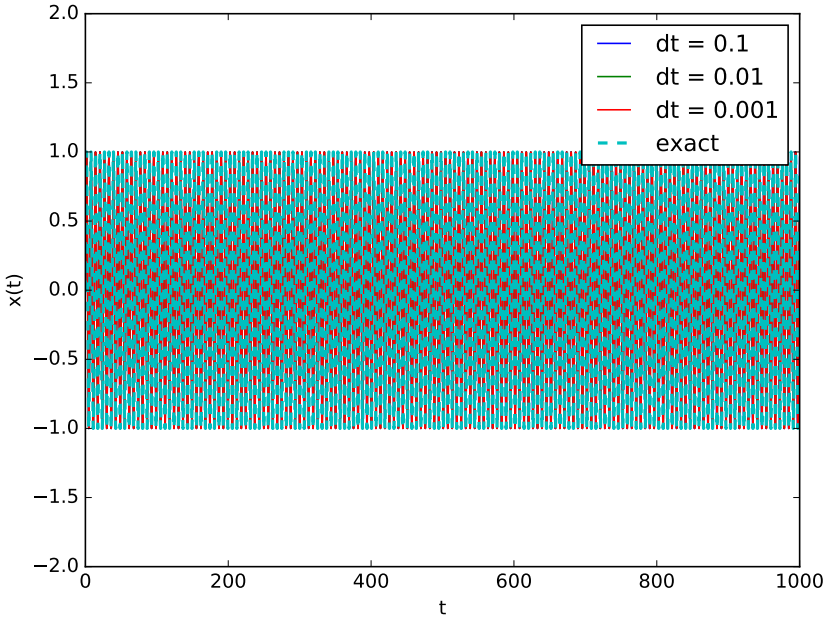


Figure 3: Time evolution of harmonic oscillator using the Euler-Cromer algorithm (method B).

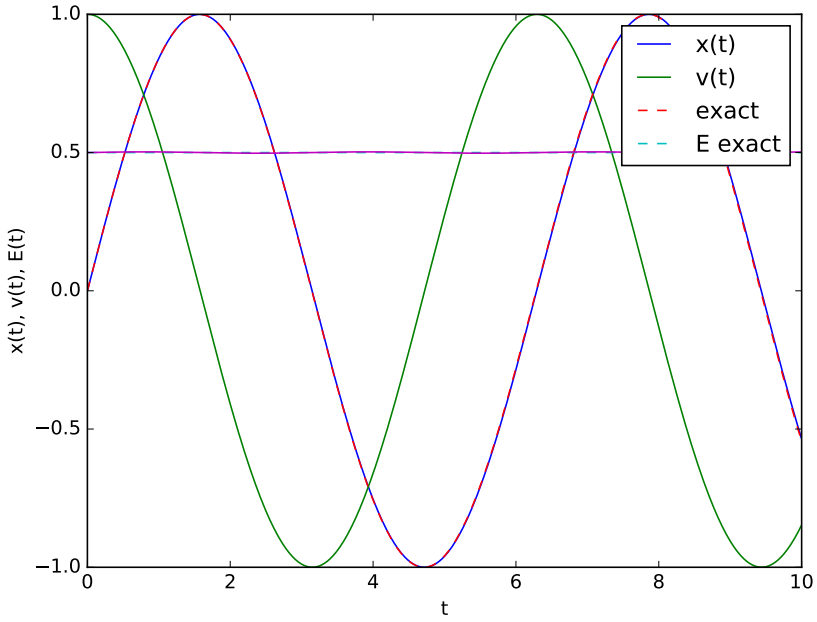


Figure 4: Time evolution of harmonic oscillator (position, velocity, energy) using the Euler-Cromer algorithm (method A).

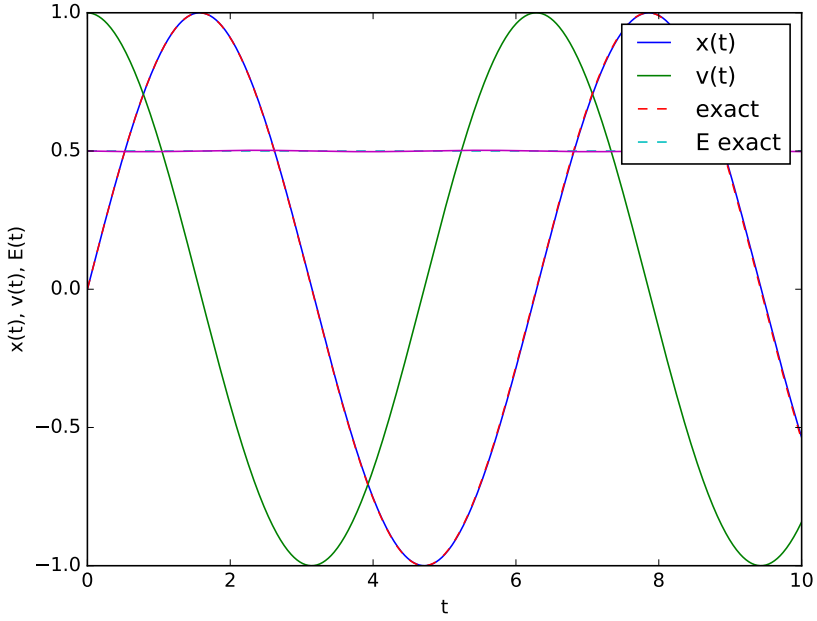


Figure 5: Time evolution of harmonic oscillator (position, velocity, energy) using the Euler-Cromer algorithm (method B).

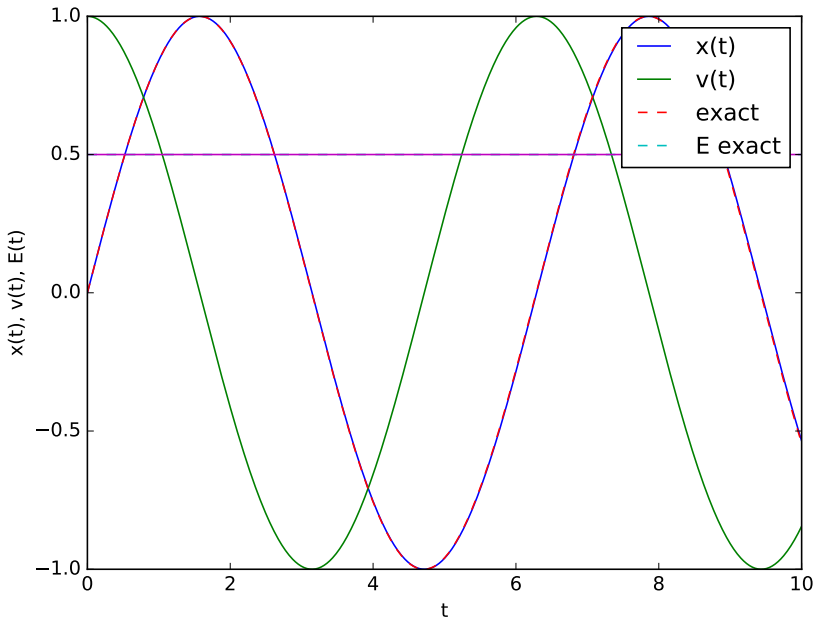


Figure 6: Time evolution of harmonic oscillator (position, velocity, energy) using the Velocity Verlet algorithm.

### Simulation results

In figure 6 we can clearly see a similarly good agreement with the exact solution, but also a constant energy throughout the whole integration time.

## Many-Body Velocity Verlet Integration

### Introduction

We consider a system with the following Hamiltonian:

$$
\begin{aligned}
H &= \frac{1}{2}\sum_{n=1}^{N} v_n^2 + \frac{1}{2}\sum_{n=1}^{N-1}(x_n - x_{n+1})^2 \\
\frac{\partial V}{\partial x_k} &= 2x_k - x_{k-1} - x_{k+1}, 1 < k < N \\
\frac{\partial V}{\partial x_1} &= x_1 - x_2 \\
\frac{\partial V}{\partial x_N} &= x_N - x_{N-1}
\end{aligned}
$$

The boundary conditions are one of the following:

$$
\begin{aligned}
v_i(0) &= 0 \forall i \\
x_i(0) &= 0 \forall i \neq N/2 \\
x_{N/2}(0) &= 1
\end{aligned}
$$

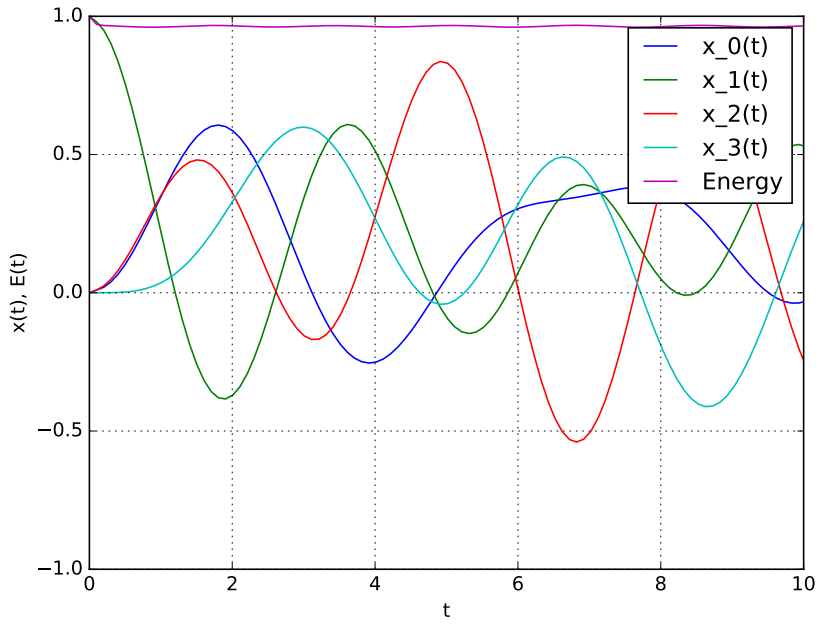


Figure 7: Time evolution of the position and total energy of 4 coupled harmonic oscillators simulated using Velocity Verlet integration with $\Delta t = 0.1$.

referred to as boundary conditions 1,

$$v_i(0) = 0 \forall i$$
$$x_i(0) = \sin \frac{\pi i}{N+1}$$

referred to as boundary conditions 2a and

$$v_i(0) = 0 \forall i$$
$$x_i(0) = \sin \frac{\pi N i}{2(N+1)}$$

called boundary conditions 2b.
For this system Velocity Verlet integration, as described in the previous section, is used.

## Simulation results

Comparing different timesteps ($\Delta t = 0.1$ and $\Delta t = 0.01$ in figure x and y respectively) it becomes clear, that the constancy of the total energy is worse for bigger time steps.

To compare the different boundary conditions a raster plot is used. The plots for the boundary conditions 1 can be found in figure 9. For boundary conditions 2a and 2b they can be found in figure 10 and 11 respectively.

## Discussion

As seen by the significant divergence of the simulation in the first section of this exercise the Euler method is not very useful, considering it's instability and bad energy convergence. A simple change in the form of the Euler-Cromer algorithm can give us symplecticity and therefore a much more stable behaviour. It's use is therefore much higher, but because of a lack of time-reversal symmetry energy conservation is still not guaranteed, as we can also see in the short time-scale plots of the Euler-Cromer based simulation. The Velocity Verlet integration on the other hand does have the time reversal property and therefore does guarantee energy conservation, which can even be

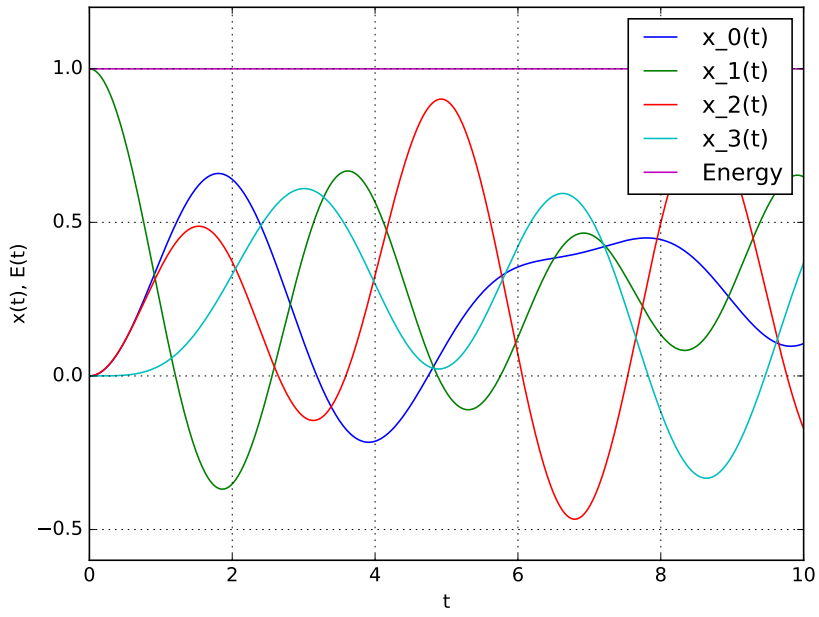


Figure 8: Time evolution of the position and total energy of 4 coupled harmonic oscillators simulated using Velocity Verlet integration with $\Delta t = 0.01$.

observed in the rather chaotic many body system. The instability of the total energy in the coupled oscillator plot for $\Delta t = 0.1$ can be attributed to the fact that the dynamics of the system have a shorter characteristic time scale than can be justified by our time step and therefore the degree of linearization of our numerical integration.

For the coupled system the boundary conditions 1 have $N$-independent total energy since only one oscillator is displaced initially. The system effectively behaves as if there was an energy source in the middle of the system which emits waves with a frequency that is independent of $N$ that are reflected at the boundaries, which is exactly what is observed. Boundary conditions 2a correspond to a very low frequency (with the wavelength corresponding to the scale of the system) wave propagating through the system. The total energy is very low, since the average distance between the oscillators is small. For boundary conditions 2b the total energy is very high, since the initial spatial frequency of the displacement is very high, i.e. the average distance between neighbouring oscillators is high as well. It is similar to the first case, but with a high frequency wave being propagated. In both cases each individual oscillator oscillates very regularly, like an individual free oscillator, with a frequency corresponding to the initial spatial frequency, i.e. the scale of the system.

In conclusion Euler integration, without Cromer's improvement is not very useful for physical simulations. Cromer's improvement makes it useful again, but should only be employed if very high performance is needed. In most cases the Velocity Verlet method shows significantly more accurate, unproblematic behaviour which can not be made up for by it's minorly higher demands computationally. Both the Euler-Cromer algorithm and Velocity Verlet algorithm only need one evaluation of the force per time step and only differ in the amount of arithmetic operations (Velocity Verlet needs one multiplication and two additions more).

# Appendix

## Program exercise 1 and first part of 2

```
1 import sys
2 import math
3 import numpy as np
4 import matplotlib.pyplot as plt
```

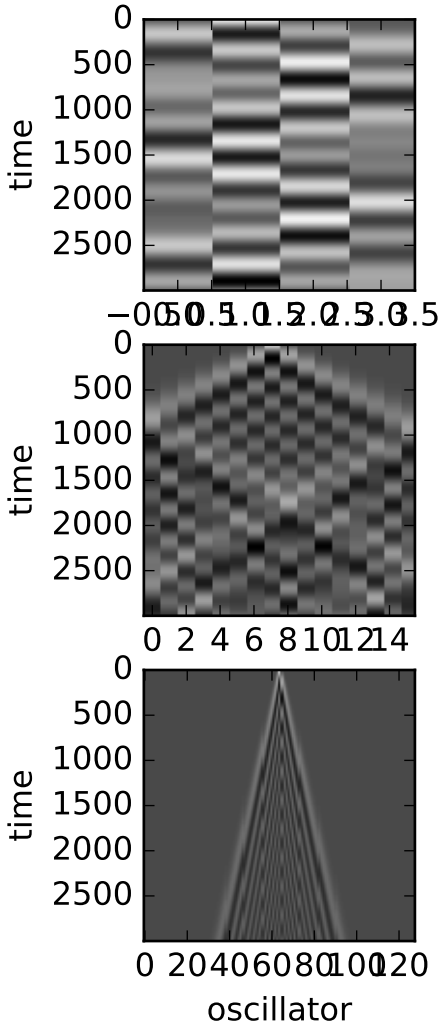


Figure 9: Raster plots showing the displacement of each oscillator on a column over time (down) for boundary conditions 1. Top is $N = 4$, middle is $N = 16$, bottom is $N = 128$. $\Delta t = 0.01$.

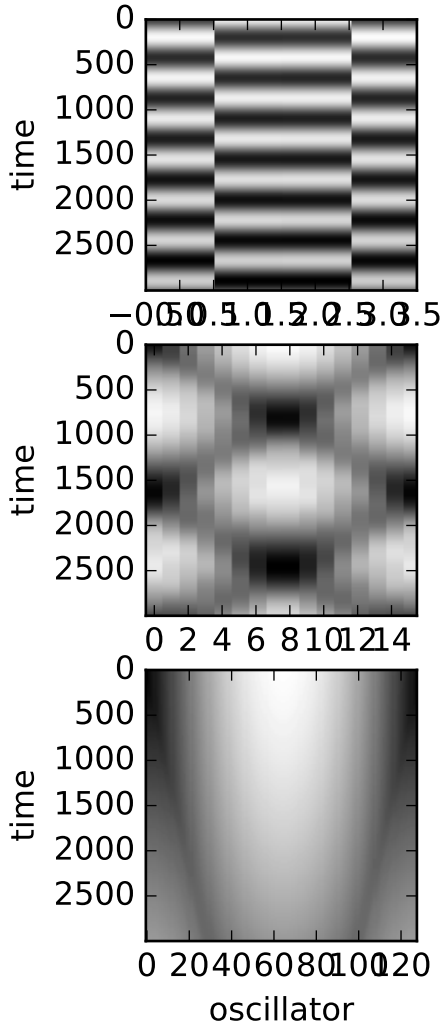


Figure 10: Raster plots showing the displacement of each oscillator on a column over time (down) for boundary conditions 2a. Top is $N = 4$, middle is $N = 16$, bottom is $N = 128$. $\Delta t = 0.01$.

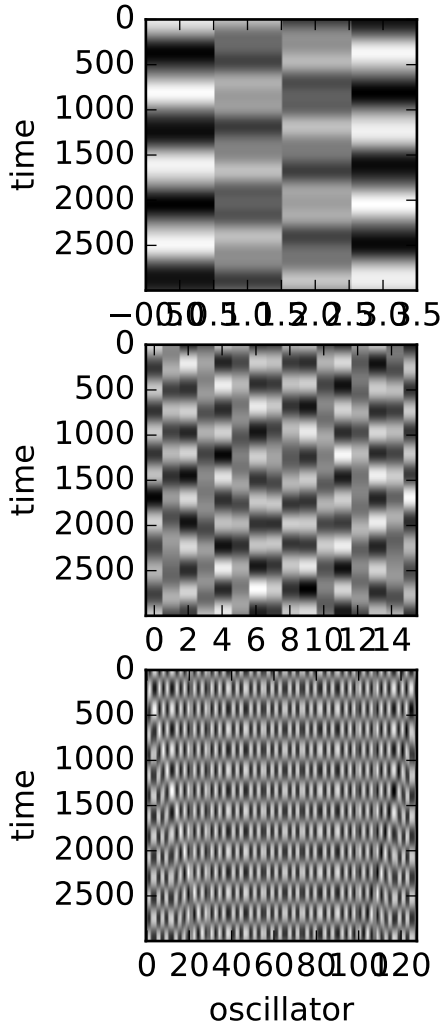


Figure 11: Raster plots showing the displacement of each oscillator on a column over time (down) for boundary conditions 2b. Top is $N = 4$, middle is $N = 16$, bottom is $N = 128$. $\Delta t = 0.01$.

```
5
6  if len(sys.argv) > 1:
7      INTEGRATION_TYPE = sys.argv[1]
8  else:
9      INTEGRATION_TYPE = "euler"
10
11 if INTEGRATION_TYPE not in ["euler", "eulerca", "eulercb"]:
12     print("Invalid integration type!")
13     quit()
14
15 def a(x):
16     return -x # k = 1
17
18 def integrate_euler(x, v, dt):
19     xn = x + v * dt
20     vn = v + a(x) * dt
21     return xn, vn
22
23 def integrate_euler_cromer_a(x, v, dt):
24     vn = v + dt * a(x)
25     xn = x + dt * vn
26     return xn, vn
27
28 def integrate_euler_cromer_b(x, v, dt):
29     xn = x + v * dt
30     vn = v + a(xn) * dt
31     return xn, vn
32
33 maxT = 1000
34 for dt in [0.1, 0.01, 0.001]:
35     N = math.floor(1000/dt)
36     t = np.linspace(0, N*dt, num=N)
37     x, v = np.zeros(N), np.ones(N)
38     for i in range(1, N):
39         if INTEGRATION_TYPE == "euler":
40             x[i], v[i] = integrate_euler(x[i-1], v[i-1], dt)
41         elif INTEGRATION_TYPE == "eulerca":
42             x[i], v[i] = integrate_euler_cromer_a(x[i-1], v[i-1], dt)
43         elif INTEGRATION_TYPE == "eulercb":
44             x[i], v[i] = integrate_euler_cromer_b(x[i-1], v[i-1], dt)
45         else:
46             print("Invalid integration type!")
47             quit()
48
49     plt.plot(t, x, label="dt = {}".format(dt))
50
51 t = np.linspace(0, maxT, 100000)
52 plt.plot(t, np.sin(t), linestyle="--", label="exact", linewidth=2)
53
54 plt.ylim(-2, 2)
55 plt.legend()
56 plt.xlabel("t")
57 plt.ylabel("x(t)")
58 plt.savefig("ex1_ex2_{}.pdf".format(INTEGRATION_TYPE))
59 plt.show()
```

### Program for exercise second part of 2 and 3

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

if len(sys.argv) > 1:
    INTEGRATION_TYPE = sys.argv[1]
else:
    INTEGRATION_TYPE = "verlet"

if INTEGRATION_TYPE not in ["verlet", "eulerca", "eulercb"]:
    print("Invalid integration type!")
    quit()

def a(x):
    return -x # k = 1

def integrate_euler_cromer_a(x, v, dt):
    vn = v + dt * a(x)
    xn = x + dt * vn
    return xn, vn

def integrate_euler_cromer_b(x, v, dt):
    xn = x + v * dt
    vn = v + a(xn) * dt
    return xn, vn

dt = 0.01
N = 1000
maxT = N * dt
t = np.linspace(0, maxT, num=N)
x, v = np.zeros(N), np.ones(N)

if INTEGRATION_TYPE == "verlet":
    # one step euler (-cromer A)
    v[1] = v[0] + a(x[0]) * dt
    x[1] = x[0] + v[1] * dt
    lastAccell = a(x[1])
    startIndex = 2
else:
    startIndex = 1

for i in range(startIndex, N):
    if INTEGRATION_TYPE == "verlet":
        # velocity verlet
        x[i] = x[i-1] + v[i-1]*dt + 0.5 * lastAccell * dt*dt
        accell = a(x[i])
        v[i] = v[i-1] + 0.5*(lastAccell + accell)*dt
        lastAccell = accell
    elif INTEGRATION_TYPE == "eulerca":
        x[i], v[i] = integrate_euler_cromer_a(x[i-1], v[i-1], dt)
    elif INTEGRATION_TYPE == "eulercb":
        x[i], v[i] = integrate_euler_cromer_b(x[i-1], v[i-1], dt)
    else:
        print("Invalid integration type!")
```

```
56          quit()
57
58  plt.plot(t, x, label="x(t)".format(dt))
59  plt.plot(t, v, label="v(t)".format(dt))
60
61  plt.plot(t, np.sin(t), linestyle="--", label="exact")
62
63  Ee = np.full(N, 0.5)
64  plt.plot(t, Ee, linestyle="--", label="E exact")
65  plt.plot(t, np.power(v, 2)/2 + np.power(x, 2)/2)
66
67  plt.ylim(-1, 1)
68  plt.legend()
69  plt.xlabel("t")
70  plt.ylabel("x(t), v(t), E(t)")
71  plt.savefig("ex2_ex3_{}.pdf".format(INTEGRATION_TYPE))
72  plt.show()
```

### Program for exercise 4 (coupled oscillators)

```
1   import sys
2   import math
3   import numpy as np
4   import matplotlib.pyplot as plt
5
6   if len(sys.argv) > 1:
7       BOUNDARY_TYPE = int(sys.argv[1])
8   else:
9       BOUNDARY_TYPE = 2
10  print("Boundary type:", BOUNDARY_TYPE)
11
12  def a(x, i):
13      if i == 0:
14          return -(x[0] - x[1])
15      elif i == x.size - 1:
16          return -(x[i] - x[i-1])
17      else:
18          return -(2*x[i] - x[i-1] - x[i+1])
19
20  x_results = []
21
22  for n_i, N in enumerate([4, 16, 128]):
23      #for dt in [0.1]:
24      for dt in [0.1, 0.01]:
25          maxT = 30
26          Nt = math.floor(maxT / dt)
27          t = np.linspace(0, maxT, num=Nt)
28
29          v = np.zeros((Nt, N))
30          if BOUNDARY_TYPE == 1:
31              x = np.zeros((Nt, N))
32              x[0][math.floor(N/2-1)] = 1
33          elif BOUNDARY_TYPE == 2:
34              x = np.zeros((Nt, N))
35              for i in range(N):
36                  x[0][i] = np.sin(np.pi * (i+1) / (N + 1))
37          elif BOUNDARY_TYPE == 3:
```

```
38              x = np.zeros((Nt, N))
39              for i in range(N):
40                  x[0][i] = np.sin(np.pi * N/2 * (i+1) / (N + 1))
41          else:
42              print("invalid BOUNDARY_TYPE!")
43              quit()
44
45          # one euler step, prepare lastAccell
46          lastAccell = np.zeros(N)
47          for n in range(N):
48              v[1][n] = v[0][n] + a(x[0], n) * dt
49              x[1][n] = x[0][n] + v[1][n] * dt
50              lastAccell[n] = a(x[1], n)
51
52          # verlet integration
53          for i in range(2, Nt):
54              for n in range(N):
55                  x[i][n] = x[i-1][n] + v[i-1][n]*dt + 0.5 * lastAccell[n] * dt*
                        dt
56
57              for n in range(N):
58                  accell = a(x[i], n)
59                  v[i][n] = v[i-1][n] + 0.5*(lastAccell[n] + accell)*dt
60                  lastAccell[n] = accell
61
62      # plot for last dt
63      toPlot = [0, 1, 2, 3]
64      for i in toPlot:
65          plt.plot(t, x[:,i], label="x_{}(t)".format(i))
66
67      # shift, leave out last column
68      energy = np.sum(np.power(x[:,:-1] - np.roll(x, -1, axis=1)[:,:-1], 2), axis
            =1)/2 + \
69          np.sum(np.power(v, 2), axis=1)/2
70      plt.plot(t, energy, label="Energy")
71
72      plt.legend()
73      plt.xlabel("t")
74      plt.ylabel("x(t), E(t)")
75      plt.grid()
76      plt.xlim(0, 10)
77      plt.savefig("ex4_N={}_dt={}_boundaries={}_curves.pdf".format(N, dt,
            BOUNDARY_TYPE))
78      #plt.show()
79      plt.close()
80
81      x_results.append((N,x))
82
83  # Raster plots
84  f, ax = plt.subplots(3,1)
85  for i, (N,x) in enumerate(x_results):
86      ax[i].imshow(x, cmap="gray", interpolation="nearest", aspect=N/Nt)
87      ax[i].set_xlabel("oscillator")
88      ax[i].set_ylabel("time")
89
90  plt.savefig("ex4_dt={}_boundaries={}_raster.pdf".format(dt, BOUNDARY_TYPE))
```

```
91  plt.show()
92  plt.close()
```