L9 Structural Bioinformatics

Exercise 1 - Algorithms

a) In the lecture you learned about the Verlet algorithm for numerical integration. This algorithm seems quite long-winded, an easier way would be simple Euler integration which we find by discretization of the differentials. First reconsider Newtons equation of motion.

$$\ddot{x} = \dot{v} = -\frac{\nabla V(x)}{m} := f(x) \tag{1}$$

This is a second order differential equation (it contains a second derivative), the Euler scheme only applies to first order differential equations. Therefore, we have to rewrite the second order diff. equation into a set of first order diff. equations as follows:

$$\dot{v} = f(x) \tag{2}$$
$$\dot{x} = v \tag{3}$$

What we find is a set of coupled differential equations which, as we found out on the pre-exercise, are not easy to handle. But now we have the power of numerics and modern computers. This is the point were we discretize the differentials:

$$\frac{\Delta v}{\Delta t} = \frac{v_{i+1} - v_i}{\Delta t} = f(x_i) \tag{4}$$
$$\frac{\Delta x}{\Delta t} = \frac{x_{i+1} - x_i}{\Delta t} = v_i \tag{5}$$

By simply reorganize the former equations we find an itterative solution to the differential equations.

$$v_{i+1} = v_i + f(x_i)\Delta t \tag{6}$$
$$x_{i+1} = x_i + v_i \Delta t \tag{7}$$

Reconsider problem a) of the pre-exercise and calculate 3 steps with the Euler algorithm and the Verlet algorithm, compare your results with the analytical solution, what did you find?

Hint: *Use a timestep of* $\Delta t = 0.3$

b) Another algorithm that is known as the leap frog algorithm is given as follows:

$$v_{i+1/2} = v_{i-1/2} + f(x_i)\Delta t \tag{8}$$
$$x_{i+1} = x_i + v_{i+1/2}\Delta t \tag{9}$$

Show by simple algebra that it is equal to the Verlet algorithm!

Exercise 2 - Programming

We now reconsider the whole problem on the computer. For this exercise you will need a working python environment, for this you could simply use the KIP server. In the following I give you a Code that contains the implementation of the numerical integration with the Verlet algorithm. We will use this implementation to solve the tasks from the Pre Exercise sheet.

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

# Verlet integrator:
# initial: array of shape [initial position, initial velocity]
# stepsize: float
# steps: int
# force: function that takes two inputs "def force(x,parameters):"
# parameters: parameters of the force and potential functions
# mass: float
# potential: either 0 if the energy should not be stored or a function similar
    to "force", that computes the potential energy (integral of the force
    function)
def verlet(initial, stepsize, steps, force, parameters, mass, potential = 0):
    initial = np.array(initial)
    trajectory = [initial[0]-initial[1]*stepsize, initial[0]]
    if potential != 0:
        energy = [mass*np.dot(initial[1],initial[1])/2 + potential(initial[0],
    parameters)]
    for k in range(steps):
        x = 2*trajectory[-1]-trajectory[-2] + force(trajectory[-1], parameters)/
    mass * stepsize**2
        if potential != 0:
            energy.append(mass*np.dot((x-trajectory[-1]),(x-trajectory[-1]))/
    stepsize**2/2 + potential(np.array(x),parameters))
        trajectory.append(x)
    if potential != 0:
        return np.array(trajectory), np.array(energy)
    else:
        return np.array(trajectory)
```

a) Reconsider problem a) from the Pre Exercise sheet, did you realise that this is actually a one dimensional problem? The particle will only move along the angle bisector between the x_1 and x_2 axis. For this specific case, this is easy to see, in a higher dimension one would run the simulation and reduce the movement of the particles to lower dimension by Principal Component Analysis. The code will run the simulation in two dimensions, if you have Python experience feel free to try to reduce it to one dimension. The integrator will work for both cases. Try different time steps to find the ideal balance between computational cost and accuracy. What happens if you make the time step very large?

```
# Force for the simple harmonic oszillator in 2D
# x: array with coordinates [x,y]
# parameters: array with parameters of the potential
def force_harmonic2D(x,parameters):
    return -2*parameters*x

# Here you should enter your stepsize and try around
stepsize = ???
steps = int(8/stepsize)
```

```
t = np.arange(0,(steps+0.5)*stepsize,stepsize)

trajectory = verlet([[1,-1],[0,0]], stepsize, steps, force_harmonic2D, np.
    array([1,1]), 1)

fig = plt.figure(figsize=(14,6))
ax0 = fig.add_subplot(121)
ax1 = fig.add_subplot(122, projection='3d')
ax0.plot(t, trajectory[1:,0], label = 'x')
ax0.plot(t, trajectory[1:,1], label = 'y')
ax0.legend(loc = "lower left", fontsize=15)
ax0.set_xlabel("t", fontsize=15)
ax0.set_ylabel("position", fontsize=15)

ax1.plot(trajectory[1:,0], trajectory[1:,1], pot_harmonic2D(np.array([
    trajectory[1:,0], trajectory[1:,1]]).T,np.array([1,1])))
ax1.set_xlabel("$x_1$", fontsize=15)
ax1.set_ylabel("$x_2$", fontsize=15)
ax1.set_zlabel("$V(x_1,x_2)$", fontsize=15)
ax1.view_init(elev=10, azim=-120)

plt.show()
```

b) Now reconsider problem b) from the Pre Exercise, use the ideal time step you found in part a) to solve the multi dimensional problem. What happens if you perturb the initial symmetry?

```
# Force field for the harmonic oszillator with interaction potential
# x: array with coordinates [x1,y1,x2,y2]
# parameters: array with parameters of the potential
def force_harmonic2D_coul(x,parameters):
    nenner = ((x[0]-x[2])**2 + (x[1]-x[3])**2)**1.5
    return parameters[2]*(x-np.array([x[2],x[3],x[0],x[1]]))/nenner-2*np.
    concatenate((parameters[:2]*x[:2],parameters[:2]*x[2:]))

# Here you should enter your stepsize from a)
stepsize = ???
initial = [[-1,1,1,1],[0,0,0,0]]
steps = int(8/stepsize)
t = np.arange(0,(steps+0.5)*stepsize,stepsize)

trajectory = verlet(initial, stepsize, steps, force_harmonic2D_coul, np.
    array([1,1,1]), 1)

fig = plt.figure(figsize=(14,12))
ax0 = fig.add_subplot(221)
ax0.plot(t, trajectory[1:,0], label = '$x_1$')
ax0.plot(t, trajectory[1:,1], label = '$x_2$')
ax0.legend(loc = "lower left", fontsize=15)
ax0.set_xlabel("t", fontsize=15)
ax0.set_ylabel("position", fontsize=15)
ax0.set_title("particle 1")


ax1 = fig.add_subplot(222, projection='3d')
ax1.plot(trajectory[1:,0], trajectory[1:,1], pot_harmonic2D(np.array([
    trajectory[1:,0], trajectory[1:,1]]).T,np.array([1,1])), label="
    particle 1")
```

```
ax1.plot(trajectory[1:,2], trajectory[1:,3], pot_harmonic2D(np.array([
    trajectory[1:,2], trajectory[1:,3]]).T,np.array([1,1])), label="
    particle 1")
ax1.set_xlabel("$x_1$", fontsize=15)
ax1.set_ylabel("$x_2$", fontsize=15)
ax1.set_zlabel("$V(x_1,x_2)$", fontsize=15)
ax1.set_xticks(np.arange(-1, 1.1, 0.5),np.arange(-1, 1.1, 0.5))
ax1.set_yticks(np.arange(-1, 1.1, 0.5),np.arange(-1, 1.1, 0.5))
ax1.set_zticks(np.arange(0, 2.1, 0.5),np.arange(0, 2.1, 0.5))
ax1.view_init(elev=10, azim=-120)
ax1.legend(loc="upper right")

ax2 = fig.add_subplot(223)
ax2.plot(t, trajectory[1:,2], label = '$x_1$')
ax2.plot(t, trajectory[1:,3], label = '$x_2$')
ax2.legend(loc = "lower left", fontsize=15)
ax2.set_xlabel("t", fontsize=15)
ax2.set_ylabel("position", fontsize=15)
ax2.set_title("particle 2")

ax3 = fig.add_subplot(224)
ax3.plot(trajectory[1:,0], trajectory[1:,1], label="particle 1")
ax3.plot(trajectory[1:,2], trajectory[1:,3], label="particle 1")
ax3.set_xlabel("$x_1$", fontsize=15)
ax3.set_ylabel("$x_2$", fontsize=15)
ax3.legend(loc="upper right")

plt.show()
```

Exercise 3 - Quantum Mechanics

This exercise is for the discussion in the tutorial, try to think about it so you can take part in the discussion!

In this exercise we will reconsider some quantum mechanics and why it is so computationally expensive to solve systems on this level. By now you should be familiar with the form of the Schrödinger equation and the formality of Quantum Mechanics in general (Knowledge that you gained from PC1 so far should be enough for this exercise.). We now look at problem 2 a) again but the particle behaves according to QM. This time we reduce the dimension so this results in the problem of solving the harmonic oscillator for a QM system. You already should know the solutions of this problem from PC1, even though you didn't solve the differential equation explicitly. The Schrödinger equation for this problem is given by:

$$-\frac{\hbar^2}{2m}\nabla^2\Psi(x) + \frac{m\omega^2}{2}x^2\Psi(x) = E\Psi(x) \tag{10}$$

By using atomic units we can set all the constants to 1.

$$-\Psi''(x) + x^2\Psi(x) = E\Psi(x) \tag{11}$$

We will try to use the Euler scheme to solve the equation. Therefore, we first have to rewrite the second order differential equation into two first order equations:

$$\Phi'(x) = (x^2 - E)\Psi(x) \tag{12}$$

$$\Psi'(x) = \Phi(x) \tag{13}$$

By also discretizing the derivative, the equation takes the form:

$$\frac{\Delta\Phi}{\Delta x} = \frac{\Phi_{i+1} - \Phi_i}{\Delta x} = (x_i^2 - E)\Psi_i \tag{14}$$

$$\frac{\Delta\Psi}{\Delta x} = \frac{\Psi_{i+1} - \Psi_i}{\Delta x} = \Phi_i(x) \tag{15}$$

The Euler scheme for integration results in:

$$\Phi_{i+1} = \Phi_i + (x_i^2 - E)\Psi_i\Delta x \tag{16}$$

$$\Psi_{i+1} = \Psi_i + \Phi_i(x)\Delta x \tag{17}$$

Think about problems that occur when you want to try numerical integration on the above formula!

Hint: *There are two major problems, try to think about a solution too!*