# Week 3: Numerical Integration and Units

**2MMN40: Introduction to Molecular Modeling and Simulation**
Last update: November 2, 2020

## Contents

## 1 The Assignment

Last week we have seen how to calculate forces within a molecule. This week we will use the forces to simulate the time evolution of the molecule. The assignment consists of five parts.

1. Reuse the code you wrote last week to calculate intra-molecular forces,
2. Implement different integrators (euler, verlocity verlet and one other integrator) in python and use them to integrate the forces over time,
3. Output XYZ trajectories of your simulation and a file with velocities,
4. Plot phase space diagrams for the various integrators and compare to theoretical phase space behaviour and
5. Visualize the trajectory in VMD (you might want to do this before looking at the phase space, to check if the molecule behaves nicely).

## 2 Which Molecules to Simulate

### 2.1 Diatomic Molecules

- Simulate the oscillation of $H_2$, $O_2$, CO. Think about how these molecules differ. What kind of differences would you expect between the two? What kind of similarities?
- Use three different integrators to simulate your systems. Two of them must be Euler and Velocity Verlet. You may choose any other third integrator you wish.
- Plot the phase space diagrams for the systems above and compare the phase space behaviour for the different integrators. Compare to the theoretical phase space behaviour for the $H_2$ case.
- Visualize the trajectories of the molecules with VMD.

### 2.2 Triatomic Molecules

Simulate an $H_2O$ molecule. Check your results by visualizing the trajectory in VMD.

## 3 Numerical Integration

### 3.1 Getting Parameters

Last week we provided you with all the necessary parameters for the computation of the forces, this week you have to find your own (it is not hard: google). You will for example need the masses of atoms, try to find them. As a starting point it is probably a good idea to begin with force constants equal to 1, then you can change them later on when looking at different cases. This does not apply to equilibrium angles, angles should be close to what the molecule's angle truly is. E.g the angle is about 104.52° for water.

### 3.2 Units

One of the big problems with numerics is handling very small or very large numbers. They result in floating point errors. To work around that issue it is a good idea to choose a system of units in which all variables are around unity. As an example take distance. The average bond length is on the order of $10^{-10}$m, storing this value in meters is therefore a terrible idea, using Ångstrom however makes this unity. Which units to use is up to you, but be consistent and mention it in a comment in the code. A helpful tool that can help you to calculate a system of units can be found here http://cbio.bmt.tue.nl/pumma/index.php/Manual/ReducedUnits.

### 3.3 Integration Time Step

The time step you use should be much smaller than the smallest oscillation period of your system (What is the period of a simple harmonic oscillator?). You may experiment with time steps in the exercises below (HINT: Bonds tend to be *stiffer* than angles...). Choose an appropriate value and stick to it for all your results in the report.

### 3.4 Initial Conditions

You should use appropriate initial conditions for your systems to see any oscillation. There are two easy ways to do this. You may randomly deform the molecule slightly by moving individual atoms around. Or you may give the atoms a small random velocity. As for initial positions, we recommend you use the initial configurations we gave you. For example, use the water molecule from WaterSingle.xyz from before.

Adding a small random velocity:

```python
# Generate a random velocity:
# first get a random unit vector (direction)
u = np.random.uniform(size=3)
u /= np.linalg.norm(u) # normalize
vRand = 0.01*u
```

## 4 Visualization with VMD

We strongly recommend you visualize your molecules in VMD, to make sure that your molecules look like they are moving properly. You may google for videos of various molecules to compare. This part should be easy, you have already seen how to do this in week 1. If you forgot revisit the VMD tutorial on canvas.

## 5 Implementation Tip

We ask you to implement different integrators in this assignment. The easiest way to allow for different integrators is to define every integrator in its own function. What an integrator does on an abstract level is update the state of the system according to some rules for a small time step $dt$, i.e.

$$x_{i+1} = \text{Integrator}(x_i, dt) \tag{1}$$

where $x_i$ is the state of the whole system at time step $i$. The easiest way to implement an integrator is with the same idea, see the following example.

```python
def integrator(x, v, a, others2, dt):
        """ Implementation of a single step for this integrator. """
        return(x, v, a, others)

time = 0
```

```
while(time<=endTime):
        x, v, a, others = integrator(x, v, a, others2, dt)
        time += dt
```

I used `x` for position, `v` for velocity, `a` for acceleration (this is optional depending on your implementation) and `others` and `others2` to indicate other optional arguments (depending on you implementation you might need masses for example) but not necessarily the same arguments.

## 6 Final Project Tip

The investigation of the numerical integrators that you do/did this week is part of you final report. You can already have a look at the final project description and see which parts of this weeks assignment you can use in the final project.