SUBMITTED BY: BREITENBERGER, NICODEMUS

# 1 Exercise: Solving a simple model – system differential equations

Consider a water molecule. Assuming that the position of the oxygen atom is fixed and the angle between the hydrogen atom is also fixed the only thing that can change is the distance between the oxygen and hydrogen atoms. These are defined as $\rho_1$ and $\rho_2$. In the next steps described with $\rho_j$, $j \in \{1, 2\}$.

The connection of the oxygen und hydrogen atom can be approximated with a spring with a spring constant k.

Therefore equating Newton's force with the spring force results in:

$$m\partial_t^2 \rho_j = -k\rho_j$$

$$\partial_t^2 \rho_j + \frac{k}{m}\rho_j = 0$$

with the initial conditions:

$$\rho_j(0) = \tilde{\rho_j}$$

$$\partial_t \rho_j(0) = v_j$$

$\rho_j(t)$ has to fullfill the equation as well as the initial conditions. Assume an exponential function:

$$\rho_j(t) = ce^{\lambda t}$$

$$\partial_t^2 \rho_j(t) = c\lambda^2 e^{\lambda t}$$

Insert into differential equation:

$$c\lambda^2 e^{\lambda t} + c\frac{k}{m}e^{\lambda t} = 0$$

$$\lambda^2 + \frac{k}{m} = 0$$

$$\lambda_{1,2} = \pm\sqrt{-\frac{k}{m}} = \pm i\sqrt{\frac{k}{m}}$$

Add the two complex solutions:

$$\rho_j(t) = c_1 \cdot e^{i\sqrt{\frac{k}{m}}t} + c_2 \cdot e^{-i\sqrt{\frac{k}{m}}t}$$

Initial conditions:

$$\rho_j(0) = c_1 + c_2 = \tilde{\rho}_j$$

This is fullfilled for:

$$\rho_j(t) = \tilde{\rho}_j \cos\left(\sqrt{\frac{k}{m}}t\right)$$

with $c_1 = c_2 = \frac{1}{2}\tilde{\rho}_j$

$$\partial_t \rho_j(t) = -\tilde{\rho}_j \sqrt{\frac{k}{m}} \sin\left(\sqrt{\frac{k}{m}}t\right)$$

$$\partial_t \rho_j(0) = -\tilde{\rho}_j \sqrt{\frac{k}{m}} \sin(0) = 0$$

This means for the velocity at t = 0:

$$v_j(t = 0) = 0$$

If we approximate the binding between the oxygen and the hydrogen atoms each with a spring force the initial conditions lay down that at the starting point t=0 the distances are at their maximum $\tilde{\rho}_j$. At time t=0 the velocity is zero because the motion is at a turning point. This motion of the water molecule is called the symmetric stretch mode.

## 2 Exercise: Lennard–Jones

```
# Gnuplot script file for plotting lennard jones potential and
   force
set   autoscale                        # scale axes automatically
unset log                              # remove any log-scaling
unset label                            # remove any previous labels
set xtic auto                          # set xtics automatically
set ytic auto                          # set ytics automatically

set title "Lennard-Jones potential and approximation"
set xlabel "r"
set ylabel "V(r)"
```

```
11
12 set xrange [11:13]
13 set yrange [-3:0]
14
15 LJV(x) = 2.0*((12.0/x)**12.0-2.0*(12.0/x)**6.0)
16 V(x) = -2.0 + (6.0*2.0)/(12.0**2.0)*(13.0-7.0)*(x-12.0)**2.0
17
18 set multiplot layout 1,2
19 set size 1,0.5
20 set origin 0,0
21
22 plot V(x) title "Approximation close to equilibrium point", LJV(x
   ) title "Lennard-Jones Potential"
23
24 #set term png
25 #set output "LennardJones.png"
26 #replot
27 #set term x11
28
29 # o.b.d.A. vector r = |r|*(1,0,0)
30 # only in x direction because gnuplot does not provide
31 # a 3d vector plot of a function
32
33 LJF(x) = 12.0*2.0/x * ((12.0/x)**12.0 - (12.0/x)**6.0)
34 F(x) = -72.0*(x-12.0)/(12.0**2.0)
35
36 set title "Lennard-Jones force and approximation"
37 set xlabel "x"
38 set ylabel "F(x)"
39
40 set xrange [10:15]
41 set yrange [-20:20]
42
43 set size 1,0.5
44 set origin 0,0.5
45 plot F(x) title "Approximation close to equilibrium point", LJF(x
   ) title "Lennard-Jones Force"
46
47 unset multiplot
48
49
```

The plots verify that the approximation is reasonable close to the equilibrium point. Further away from this point the approximated curves are very different from the Lennard-Jones potential and force.

**Morse potential:** $V(r) = \epsilon \left(1 - e^{-\sigma(r-r_0)}\right)^2$

Corresponding force:

$$\vec{F}(\vec{r}) = -\nabla V = 2\epsilon\sigma e^{-\sigma(r-r_0)} \left(1 - e^{-\sigma(r-r_0)}\right) \frac{\vec{r}}{r}$$

Approximation: Taylor expansion of the potential as in the exercise description for minimum (equilibrium point $\tilde{r}$).

$$\left.\frac{\partial V}{\partial r}\right|_{r=\tilde{r}} = 0 = 2\epsilon\sigma e^{-\sigma(\tilde{r}-r_0)} \left(1 - e^{-\sigma(\tilde{r}-r_0)}\right) \Leftrightarrow \tilde{r} = r_0$$

$$\left.\frac{\partial^2 V}{\partial r^2}\right|_{r=\tilde{r}} = 2\epsilon\sigma^2 e^{-2\sigma(\tilde{r}-r_0)} - 2\epsilon\sigma^2 e^{-\sigma(\tilde{r}-r_0)} \left(1 - e^{-\sigma(\tilde{r}-r_0)}\right)$$

$$V(r) = V(r_0) + \frac{\partial V(r_0)}{\partial r}(r - r_0) + \frac{1}{2}\frac{\partial^2 V(r_0)}{\partial r^2}(r - r_0)^2 + O((r - r_0)^3)$$

$$V(r) \approx 0 + 0 + 2\epsilon\sigma^2(r - r_0)^2 = 2\epsilon\sigma^2(r - r_0)^2$$

$$\vec{F}(\vec{r}) = -\nabla V \approx 4\epsilon\sigma^2(r - r_0)\frac{\vec{r}}{r}$$

Then the linear approximation constant equals $k_M = 4\epsilon\sigma^2$.

# 3 Exercise: Euler Method – C++

```cpp
#include <string>
#include <iostream>
#include <fstream>
#include <sstream>

//could be replaced by literal constants; define is the C way
#define K 1
#define M 1
#define H 0.01
#define STEPS 1000
//some constants regarding x_0 and v_0
#define X_0 0
#define V_0 2

```

```
15
16 double computeNextForce(double x_n_1){
17
18   return -1 * K * x_n_1;
19
20 }
21
22 double computeNextVelocity(double v_n_1, double F_n){
23
24   return v_n_1 + H / M * F_n;
25
26 }
27
28 double computeNextPosition(double x_n_1, double v_n_1){
29
30   return x_n_1 + H * v_n_1;
31
32 }
33
34 int main(int argc, char* argv[]){
35
36   double x_n_1 = X_0;
37   double v_n_1 = V_0;
38   double f_n = 0;
39   std::ofstream eulerFile, gnuPlot;
40
41   if(argc == 2){
42
43     eulerFile.open(argv[1], std::ios::out);
44     gnuPlot.open("gp.gp", std::ios::out);
45
46   }else{
47
48     std::cout << "Wrong number of arguments.\n\n Useage: prog
       outfile" << '\n';
49     return 1;
50
51   }
52
53   int i = 1;
54   for(i = 1; i <= STEPS; i++){
55
56     f_n = computeNextForce(x_n_1);
57     double v_n_temp = computeNextVelocity(v_n_1, f_n);
58     double x_n_temp = computeNextPosition(x_n_1, v_n_1);
59
60     v_n_1 = v_n_temp;
61     x_n_1 = x_n_temp;
62
```

```
63      eulerFile << i * H << ' ' << f_n  << '\n';
64      eulerFile << i * H << ' ' << v_n_1  << '\n';
65      eulerFile << i * H << ' ' << x_n_1  << '\n';
66
67    }
68
69    eulerFile.close();
70
71    //preparing a gnu-script file to print the data to a png-file
72    //basically, i set a gnuplot file to read the generated file and
         print it to a png file
73    //most of the stuff is hardcoded. There has to be a more elegant
         way. For now, this will suffice
74    gnuPlot << "set terminal png" << '\n';
75    gnuPlot << "plot \"" << argv[1] << "\" using 1:2 title \"Euler
        Method Plot for Force, Velocity and Position\"" << '\n';
76    gnuPlot.close();
77
78    std::system("gnuplot gp.gp > eulerMethod.png");
79
80
81  }
```