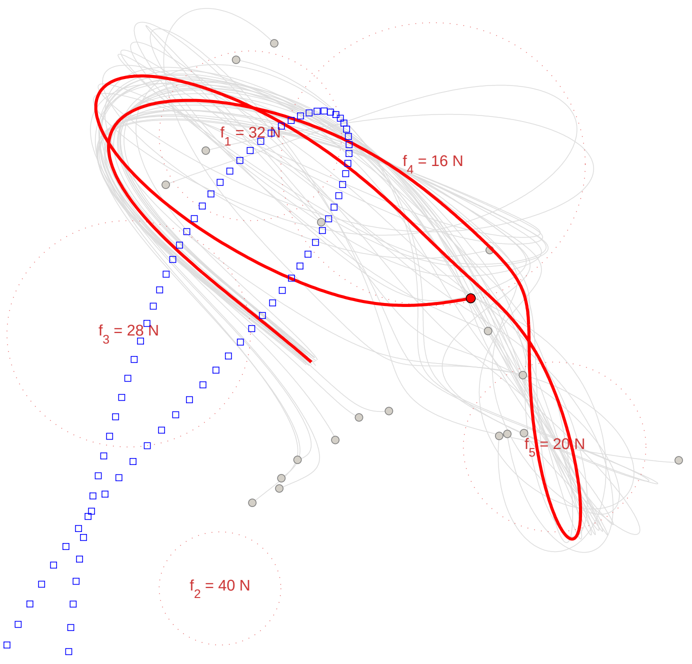
Computer Lab 1
Shadowing a Particle

Mikica Kocic
miko0008@student.umu.se



Mark what exercises you have done in this lab		*Points (by supervisor)*
Basic level (mandatory)	×	
Extra exercise #1: min, average and max final position displacement from reference solution	×	
Extra exercise #2: abs position displacement error correlation to time-step size	×	
Extra exercise #3: abs energy error correlation to time-step size	×	
Extra exercise #4: abs displacement and energy errors correlation to numerical integrator	×	
The report is brief and to the point but still coherent and well structured.		
Late report		
Total points		

Tutors: **Martin Servin** and **John Nordberg**

1 Background

The purpose of this lab was to study the numerical errors in the simulation of trajectory in a system of particles, and their relation to a) uncertainties in the initial conditions of the simulation, b) time-step size of the integration and c) choice of numerical integrator algorithm.

The simulated physical system was a 2D system of particles in the conservative force field which did not vary in time and without constraints and collisions. The model of the system was position-based governed by Newton's equations of motion.

The tasks in brief were to: 1) simulate and visualize trajectories of the particles, 2) analyze the dependency of the trajectory on disturbances in the initial conditions, 3) analyze the dependency of 3.a) the trajectory and 3.b) the total energy on the integration time-step size, and 4) analyze the dependency of the trajectory and the total energy on different numerical integrator algorithms.

2 Implementation and Simulation

The system is modeled using position-based dynamics where the state of each particle is described with its mass, position and velocity, and where positions and velocities are given with initial conditions. The ODE of motion are given in explicit form of the conservative force vector field that is composed of several spherically symmetric attractive force fields and the gravity (refer to equations (1)-(4) in the Lab1 assignment document).

The model is implemented using MATLAB and the simulation is organized in the following modules:

- **`lab1_odeSolver.m`**: implements `lab1_odeSolver` function that simulates the trajectory of a single particle system (without visualization) and tracks the total energy of the system. Its main usage is calculation of the reference trajectory. However, it is also used for sanity-checks of simulations performed in `lab1_main`.
- **`lab1_main.m`**: implements `lab1_main` function that simulates and visualizes trajectories of a larger number of particles and calculates/tracks/plots the total energy of the whole system. It also compares simulated trajectories to the reference trajectory.
- **`lab1_testSuite.m`**: contains a script that runs a series of simulations for both basic and advanced level tasks in the assignment and produces all required results.

Since both `lab1_odeSolver` and `lab1_main` do almost the same job, the following table summarizes main differences between them:

	lab1_odeSolver	**lab1_main**
Number of particles	$N_p = 1$	$N_p \geq 1$
Numerical integrator algorithms	Forward Euler Semi-implicit Euler 4th-order Runge-Kutta MATLAB ode45	Forward Euler Semi-implicit Euler – –
Uses matrices for calculations	–	Yes
Trajectory visualization	–	Yes
Total energy calculation	Yes	Yes
Energy levels visualization	–	Yes
Position displacement error statistics	–	Yes
Position displacement error calculation	Yes*)	Yes
Total energy abs. error calculation	Yes*)	Yes

*) can be derived from returned results

In this lab in general, all implemented models follow overall algorithm for a simulation of a mechanical system as presented in course literature (Servin M., *Notes on Discrete Mechanics*, 2011). Further sections in this document provide particular implementation details of individual modules.

Default definitions/physical parameters used throughout all simulation modules are:

```
%%% Physical constants

g = [ 0, -9.81 ];  % Acceleration of Gravity, m/s^2

%%% Particle system parameters

N_p  = 500;         % Number of particles in the system
m    = 1.0;         % Particle mass, kg
L    = 5.0;         % Characteristic length of the system, m
x_t0 = [ -L, -L ];  % Particle initial position, m
v_t0 = [  5, 10 ];  % Particle intital velocity, m/s

%%% Disturbances of the initial position and velocity

delX.lower = -0.02 * [L,L];  % lower bound for [x,y]
delX.upper =  0.02 * [L,L];  % upper bound for [x,y]
delV.lower = -0.02 * v_t0;   % lower bound for [Vx,Vy]
delV.upper =  0.02 * v_t0;   % upper bound for [Vx,Vy]

%%% Parameters of the spherically symmetric attractive force fields

fk = [  32,   40,  28,  16,  20 ]'     ;  % fk(k)   = strength of the force 'k'

rk = [  0.3, 0.2, 0.4, 0.5, 0.3 ]' * L ;  % rk(k)   = radius of the force 'k'

pk = [ -0.2,  0.8 ;  ... % pk1            % pk(k,:) = center position of the force 'k'
       -0.3, -0.8 ;  ... % pk2
       -0.6,  0.1 ;  ... % pk3
        0.4,  0.7 ;  ... % pk4
        0.8, -0.3    ... % pk5
     ] * L;

%%% Integration parameters

t_0  = 0;   % Initial time, s
t_f  = 3;   % Final (simulation) time, s
```

2.1 Module: `lab1_odeSolver.m`

Arguments:

`solver`	one of `'ode45'`, `'rk4'`, `'forwardEuler'` or `'semiEuler'`; default: `'rk4'`
`N_pts`	number of points to return (i.e. rows in the result matrix); default: `80`
`relError`	relative error tolerance parameter, used by `'ode45'` solver; default: `1e-7`
`h`	integration time-step parameter, used by other solvers; default: `1e-3`

Returns the solution as the matrix `N_pts`×`6` in the following format:

$$\begin{bmatrix} t_0 & x_0 & y_0 & v_{x,0} & v_{y,0} & E_{tot,0} \\ t_1 & x_1 & y_1 & v_{x,1} & v_{y,1} & E_{tot,1} \\ \vdots \\ t_f & x_f & y_f & v_{x,f} & v_{y,f} & E_{tot,f} \end{bmatrix}$$

This module simulates lab1 equations of motion using time integration and tracks the total energy of the system. The functional structure of the module is:

1. definitions / physical model parameters
2. call selected `ODEsolver` (solve particle trajectory)
3. calculate the total energy

where `ODEsolver` is either MATLAB's `ode45` function or one of the module's nested functions: `MyForwardEuler`, `MySemiEuler` or `MyRK4`. The Lab1 equations of motion are implemented in nested function `odeFunc` and passed as a function handle to `ODEsolver`.

Runge-Kutta algorithm that is implemented in `MyRK4` solver is described in (Mathews, 1992).

The module does not have other parameters beside the common physical model parameters.

Example usage:

```
>> lab1_odeSolver( 'rk4', 4, 0, 1e-2 )
ans =
         0   -5.0000   -5.0000    5.0000   10.0000   13.0088
    1.0000    0.1100    2.1882    3.0135    5.4953   13.0088
    2.0000   -1.4480    3.2412   -4.1377   -5.2590   13.0088
    3.0000   -3.9838   -5.1190   -0.8320  -11.3437   13.0088
```

Note: When called without arguments, `lab1_odeSolver` returns the reference trajectory obtained using the `rk4` algorithm with time-step `h = 1e-3`.

2.2 Module: `lab1_main.m`

Arguments:

`testSuiteNo`	ID of the test profile corresponding to specific parameter setup. If omitted, parameters are configured as required for the ‘basic level’ simulation in the assignment specification.

Returns the solution as matrix 2×6 in the following format:

$$\begin{bmatrix} t_0 & x_0 & y_0 & v_{x,0} & v_{y,0} & E_{tot,0} \\ t_f & x_f & y_f & v_{x,f} & v_{y,f} & E_{tot,f} \end{bmatrix}$$

This module simulates and visualizes trajectories of larger number of particles and tracks and plots total energy of the whole system. Simulated trajectories of the disturbed particles are animated and compared to the reference trajectory. The first particle can be optionally marked not to be disturbed.

The simulation is force based, i.e.: first, all internal and external forces are accumulated from which accelerations are computed based on Newton’s second law of motion, and second: the positions and velocities of the particles are updated using a time integration method.

The functional structure of the module is:

1. parameters, definitions and physical model constants and initialization of state variables
2. simulation loop with update of state and derived variables and trajectory animation
3. if $N_p > 1$, calculate displacement absolute error statistics (min, max, avg and std dev)
4. display of calculated data and optionally the figure with energy level plots

Additional parameters, beside the physical model parameters, are:

```
%%% Simulation and visualization control flags

flags.Disturbed    = true;  % Disturb position and velocities
flags.UndisturbN1  = true;  % Do not disturb particle #1
flags.Animate      = true;  % Render particles in real-time
flags.SaveFigures  = false; % Save figures (plots) as EPS files
flags.PlotEnergy   = true;  % Plot kinetic, potential and total energy over time

%%% Integration parameters

h          = 0.01;          % Time-step, s
stepper    = 'ForwardEuler'; % Forward Euler time-integration
%stepper   = 'SemiEuler';    % Semi-implicit Euler time-integration
```

Note that the `testSuiteNo` argument ‘suits’ default parameters to a specific pre-configured parameter set. E.g. suite nr `42` performs silent simulation (without animation and plots, but with textual reports) with the parameter set: `N_p = 500`, `h = 0.01` and `stepper = 'SemiEuler'`; overview of the test-suite IDs can be found in ‘Test Suite Setup’ section in `lab1_main.m`.

When called without arguments, `lab1_main` runs the basic level simulation as described in the assignment document.

Example usage:

```
>> lab1_main         % run basic-level simulation
>> lab1_main(11)     % run basic-level simulation and save figures as EPS files
```

Performance Considerations

The module `lab1_main` uses matrices for the representation of the state variables. Matrices for particle positions, velocities and forces of the system have dimension $N_p \times 2$ (one row per particle). Particle masses are also represented with $N_p \times 2$ dimensional matrix, so that the mass of the i-th particle is kept doubled in i-th row. This way, for example, forward Euler time-integration is implemented as:

```
for n = 1:NT
    X = X + h * V;            % solve positions
    V = V + h * F_tot ./ M;   % solve velocities
end
```

Further, matrices of the state variables can be easily ‘vectorized’ to $2N_p$-dimensional vectors when needed, like in the following example when calculating the kinetic energy:

```
E_k = 0.5 * ( ( M(:) .* V(:) )' * V(:) );   % calculate kinetic energy of the system
```

Other optimizations include the usage of `bsxfun` function, like in the following example used to calculate the force of the gravity field:

```
F_tot = bsxfun( @times, M, g );    % calculate gravity force
```

or the potential energy of the gravity field:

```
E_p = - trace( X' * bsxfun( @times, M, g ) );  % calculate potential energy of the gravity
```

On the other hand, the calculation of the total force is not vectorized – it is rather done by accumulating total force in nested loops for each particle and source field. However, implemented algorithm is optimized using MATLAB profiler so that 1) the usage of the `norm` function is removed, and 2) the ‘for each particle’ loop is placed inside ‘for each source’ loop, as in the following code-snippet from the module:

```
< 0.01  316 for k = 1:NS          % For each source in the force field
< 0.01  317     for i = 1:N_p     % For each particle
  0.58  324         X_p = X(i,:) - pk(k,:);
  0.09  325         sq_X_p = X_p(1)^2 + X_p(2)^2;
  0.87  327         F_tot(i,:) = F_tot(i,:) - fk... ;
  0.10  332     end
< 0.01  333 end
```

Usage of the `norm` function otherwise makes implementation **~4 times slower** (!) and reversing of nested loops makes implementation ~10 % slower, as shown in the following profiling report:

- Simulation with ‘for each particle’ loop inside ‘for each source’ loop and with `norm` function:

```
< 0.01  316 for k = 1:NS          % For each source in the force field
< 0.01  317     for i = 1:N_p     % For each particle
  0.67  324         X_p = X(i,:) - pk(k,:);
  3.62  325         norm_X_p = norm( X_p );
  2.05  327         F_tot(i,:) = F_tot(i,:) - fk... ;
  0.15  332     end
< 0.01  333 end
```

- Simulation with ‘for each source’ loop inside ‘for each particle’ loop and without `norm` function:

```
< 0.01  316 for i = 1:N_p         % For each particle
  0.05  317     for k = 1:NS      % For each source in the force field
  0.51  324         X_p = X(i,:) - pk(k,:);
  0.02  325         sq_X_p = X_p(1)^2 + X_p(2)^2;
  0.96  327         F_tot(i,:) = F_tot(i,:) - fk... ;
  0.10  332     end
  0.02  333 end
```

2.3 Module: `lab1_testSuite.m`

This module contains a script that runs a series of different simulations and produces all results as required by the assignment for both the basic and advanced level tasks.

The functional structure of the module is:

1. perform the basic-level and the simulation for advanced-level task 1
2. perform the simulations for advanced-level tasks 2-4

The only configurable parameter for this module is `flags.SaveFigures` boolean, which controls whether to save generated figures as EPS files or not.

Regarding the advanced tasks 2, 3 and 4, the script performs all required simulations in the single nested loop, since the simulations for tasks 2 and 3 differs from the simulations for task 4 only in a choice of numerical integrator.

Thus, the combined algorithm for the advanced tasks 2, 3 and 4 is implemented as:

```
algos = { 'ForwardEuler', 'SemiEuler', 'RK4' };  % Analyzed numerical integrators
H = [ 0.5, 0.1, 0.05, 0.01, 0.005, 0.001 ];  % Analyzed time-steps

for j = 1 : length(algos)
    for i = 1 : length(H)
        % Simulate lab1 ODE using given algorithm with given time-step

        sol = lab1_odeSolver( algos{j}, 2, 0, H(i) );

        X_n = sol(2,2:3);   % get final X(n)
        E_tot_0 = sol(1,6); % get initial E_tot(0)
        E_tot_n = sol(2,6); % get final E_tot(n)

        % Collect results:

        store in result matrix norm( X_n - X_f );        % abs err of X
        store in result matrix abs( E_tot_n - E_tot_0 ); % abs err of E_tot
    end
end
```

After the loop, data collected in the result matrix is printed in textual form and then shown on the log/log diagrams.

Running Simulation

To run the complete set of simulations, simply execute the script from the MATLAB command prompt:

```
>> lab1_testSuite
```

3 Results

3.1 Basic Level Tasks

The results of the main simulation are shown on Figure 3.1–1, where the shadow trajectories of disturbed particles are shown in gray and the reference solution is shown as blue boxes. The figure shows also the trajectory of the undisturbed particle highlighted in red. The reference trajectory is simulated using 4-th order Runge-Kutta ($h = 0.001$) and the shadow trajectories and the trajectory of the undisturbed particle are simulated using forward Euler integrator ($h = 0.01$).

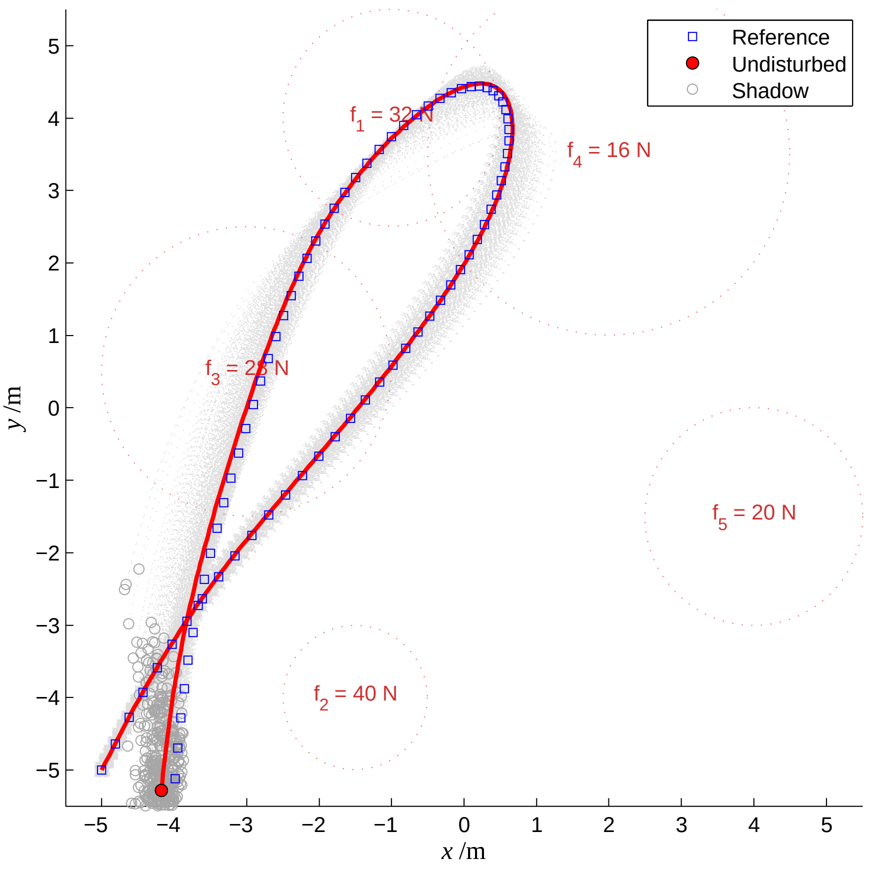


Figure 3.1–1: Shadow trajectories ($N_p = 500$) obtained using forward Euler integrator ($h = 0.01$).

For comparison, the position, velocity and the total energy of the reference particle over the time are summarized in the following table:

t /s	x /m	y /m	v_x /(m·s^{-1})	v_y /(m·s^{-1})	E_{tot} /J
0.000	−5.0000	−5.0000	5.0000	10.0000	13.0088
1.000	0.1100	2.1882	3.0135	5.4953	13.0088
2.000	−1.4480	3.2412	−4.1377	−5.2590	13.0088
3.000	−3.9838	−5.1190	−0.8320	−11.3437	13.0088

and the same variables of the undisturbed particle are summarized in the following table:

t /s	x /m	y /m	v_x /(m·s^{-1})	v_y /(m·s^{-1})	E_{tot} /J
0.000	−5.0000	−5.0000	5.0000	10.0000	13.0088
1.000	0.1280	2.2020	3.0465	5.4805	13.1660
2.000	−1.4094	3.2685	−4.4234	−5.4530	15.2254
3.000	−4.1784	−5.2787	−1.2088	−11.5980	15.3508

3.2 Advanced Level – Task 1

The minimum, maximum, average and standard deviation of the position displacement $\Delta\mathbf{x}_i = \|\mathbf{x}_i - \overline{\mathbf{x}}\|$ of the simulated particle positions $\mathbf{x}_i(t)$ at 1) initial time t_0 and 2) final time t_f as compared to the reference solution $\overline{\mathbf{x}}(t)$ respectively, are shown in the following tables:

Table 3.2-1: Statistics for *initial* displacements (random disturbances, uniform distribution)

	ΔX_0 /m	ΔV_0 /(m·s^{-1})
Minimum	0.004 979 781	0.010 911 177
Maximum	0.139 105 637	0.219 781 145
Average	0.077 727 295	0.115 459 682
Standard deviation	0.028 396 838	0.050 811 518

where:

$$\Delta X_0 = \left\{ \|\mathbf{x}_i(t_0) - \overline{\mathbf{x}}(t_0)\| : \quad i = 1,2,\ldots,N_\mathrm{p} \right\}$$

$$\Delta V_0 = \left\{ \|\mathbf{v}_i(t_0) - \overline{\mathbf{v}}(t_0)\| : \quad i = 1,2,\ldots,N_\mathrm{p} \right\}$$

Table 3.2-2: Statistics for *final* displacement errors

	ΔX_f /m	ΔV_f /(m·s^{-1})
Minimum	0.042 896 076	0.026 327 614
Maximum	3.867 415 831	3.844 067 968
Average	0.686 512 637	0.777 028 974
Standard deviation	0.460 222 492	0.443 570 870

where:

$$\Delta X_\mathrm{f} = \left\{ \|\mathbf{x}_i(t_\mathrm{f}) - \overline{\mathbf{x}}(t_\mathrm{f})\| : \quad i = 1,2,\ldots,N_\mathrm{p} \right\}$$

$$\Delta V_\mathrm{f} = \left\{ \|\mathbf{v}_i(t_\mathrm{f}) - \overline{\mathbf{v}}(t_\mathrm{f})\| : \quad i = 1,2,\ldots,N_\mathrm{p} \right\}$$

Sample data was obtained from the simulation for $N_\mathrm{p} = 500$ using forward Euler integrator ($h = 0.01$) and contains statistics also for the velocity displacements $\Delta\mathbf{v}_i = \|\mathbf{v}_i - \overline{\mathbf{v}}\|$ at initial and final time.

3.3 Advanced Level – Tasks 2, 3 and 4

The comparison of the trajectory displacement $\|\Delta\mathbf{x}\|$ and the total energy error $|\Delta E_\mathrm{tot}|$ for different numerical integrator methods and time-step sizes are shown in Table 3.3-1 and Table 3.3-2 bellow, as well on Figure 3.3–1 and Figure 3.3–2 on the next page.

Table 3.3-1: The position displacement absolute error $\|\Delta\mathbf{x}\|$ for different numerical integrator methods and time-step sizes

	$\|\Delta\mathbf{x}\|$ /m		
h /s	**Forward Euler**	**Semi-implicit Euler**	**4th-order Runge-Kutta**
0.5	4.097 178 388 089	30.402 340 339 477	1.309 387 059 813
0.1	2.101 261 688 784	3.059 395 399 101	0.002 253 353 932
0.05	1.192 057 934 307	1.245 691 511 441	0.000 119 572 636
0.01	0.251 707 365 762	0.218 720 946 370	0.000 000 155 805
0.005	0.126 242 649 645	0.107 801 257 932	0.000 000 009 436
0.001	0.025 294 730 218	0.021 320 108 622	0.000 000 000 015

Table 3.3-2: The total energy absolute error $|\Delta E_\mathrm{tot}|$ for different numerical integrator methods and time-step sizes

	$|\Delta E_\mathrm{tot}|$ /J		
h /s	**Forward Euler**	**Semi-implicit Euler**	**4th-order Runge-Kutta**
0.5	112.667 019 033 939	61.314 665 080 343	2.555 465 352 268
0.1	25.370 112 131 207	5.577 074 599 609	0.011 585 215 806
0.05	12.248 934 006 276	4.047 988 881 524	0.000 443 531 999
0.01	2.341 932 983 397	0.868 098 958 422	0.000 000 318 252
0.005	1.169 188 040 189	0.441 021 129 369	0.000 000 016 784
0.001	0.233 840 185 654	0.089 393 256 827	0.000 000 000 023

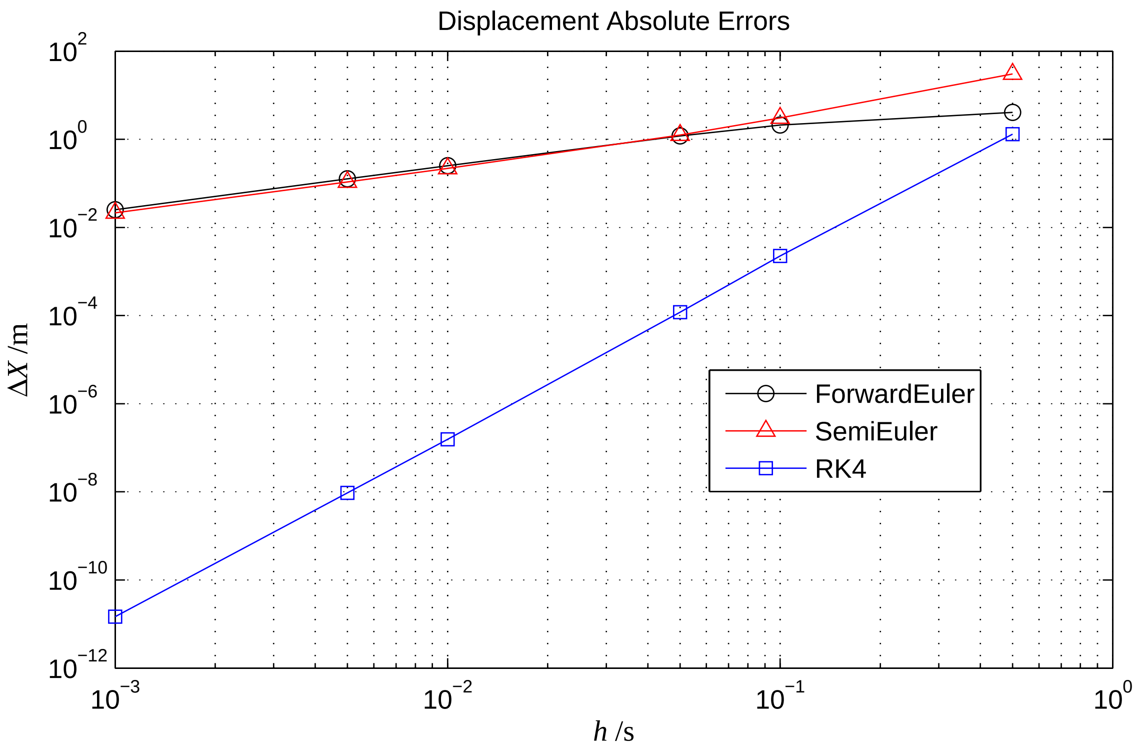


Figure 3.3–1: The position displacement absolute error $\|\Delta\mathbf{x}\|$ for different numerical integrator methods and time-step sizes

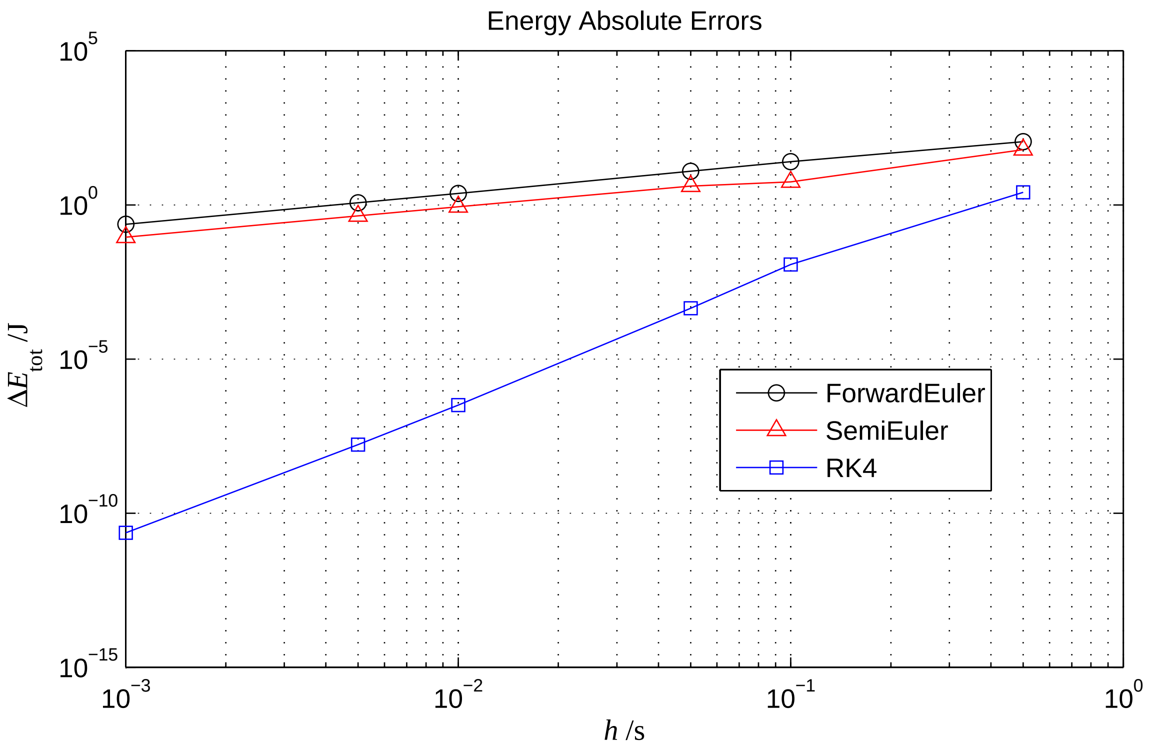


Figure 3.3–2: The total energy absolute error $|\Delta E_{\mathrm{tot}}|$ for different numerical integrator methods and time-step sizes

4 Discussion

The results show the dependence of the numerical errors from the uncertainties in the initial conditions, selected time-step size of the integration and the choice of numerical integrator algorithm.

Influence of the Initial Conditions

From the statistical results in Table 3.3-1 and Table 3.3-2, it can be seen that, for the modeled physical system during the period of time 3 s, the forward Euler integrator (h = 0.01) magnifies 1) the initial position disturbances from $\delta x_0 = 0.078(29)$ m to final $\delta x_f = 0.687(460)$ m, and 2) the initial velocity disturbances from $\delta v_0 = 0.115(50)$ m/s to final $\delta v_f = 0.777(443)$ m/s, where the number in parentheses is the numerical value of the standard deviation referred to the corresponding last digits of the quoted result; this gives approximately $\delta x_f/\delta x_0 \sim 9$ and $\delta v_f/\delta v_0 \sim 7$.

Note that during the simulation, the random initial conditions were distributed uniformly in the rectangular box in the phase space. It would be more appropriate, however, to use normally distributed spatially spherical disturbances, so the position and velocity displacement errors (calculated as norms) would be more meaningful.

Influence of the Time-Step Size

Data on figures Figure 3.3–1 and Figure 3.3–2 (and corresponding tables Table 3.3-1 and Table 3.3-2) shows that, for the analyzed numerical integrator algorithms, the numerical errors follow the power-law of the integration time-step size since the graphs on the referred figures are plotted in log/log scale and they resemble linear functions with the different slope for different numerical integrator.

Thus, it could be conjectured that $\log(\delta) = c + k \log(h)$ or that $\delta \propto h^k$, where k is the constant dependent of the numerical integrator and δ is the numerical error (either the position displacement absolute error $||\Delta\mathbf{x}||$ and total energy absolute error $|\Delta E_{\text{tot}}|$).

The respective values of k calculated from Table 3.3-1, i.e. from $\delta = ||\Delta\mathbf{x}||$ as a function of h, are:

	Forward Euler	**Semi-implicit Euler**	**4th-order Runge-Kutta**
k	0.959 725 045	1.078 423 098	4.088 369 076

and the respective values of k calculated from Table 3.3-2, i.e. from $\delta = |\Delta E_{\text{tot}}|$ as a function of h, are:

	Forward Euler	**Semi-implicit Euler**	**4th-order Runge-Kutta**
k	1.017 701 620	0.897 550 847	4.351 088 146

Influence of the Numerical Integrator Algorithm

Similarly, data on figures Figure 3.3–1 and Figure 3.3–2 (and corresponding tables) show the significant difference between the numerical errors obtained using different numerical integrator algorithms for the same time-step size.

Starting from the previous conclusion that the numerical error δ is proportional to h^k, it can be seen that the slope k is nearly the same for the forward Euler and the semi-implicit Euler integrators, while the slope k for the 4th-order Runge-Kutta integrator is roughly 4 times steeper than for the Euler integrators. This agrees with the theory, as k correlates to the order of the used numerical integrator methods.

Energy Conservation

Since the system is modeled without dissipative forces, it is expected that the total energy of the system is conserved; but—as previously concluded—there is the power-law dependency of the total energy absolute error of the time-step, meaning that the total energy eventually drifts away with time. On the other hand, how fast the total energy drifts away differs between numerical integrators.

The most interesting difference is notably between the forward Euler and semi-implicit Euler integrators, since both are first-order integrators and also have the same algorithmic complexity. Namely, simulated data shows that the semi-implicit Euler method almost conserves the energy, while the energy increases steadily using the standard Euler method.

This can be observed on Figure 3.3–2 where the semi-implicit Euler curve is slightly bellow the forward Euler curve, and also on the following figures showing how the energies of the undisturbed particle changes over time, for both integrators respectively:

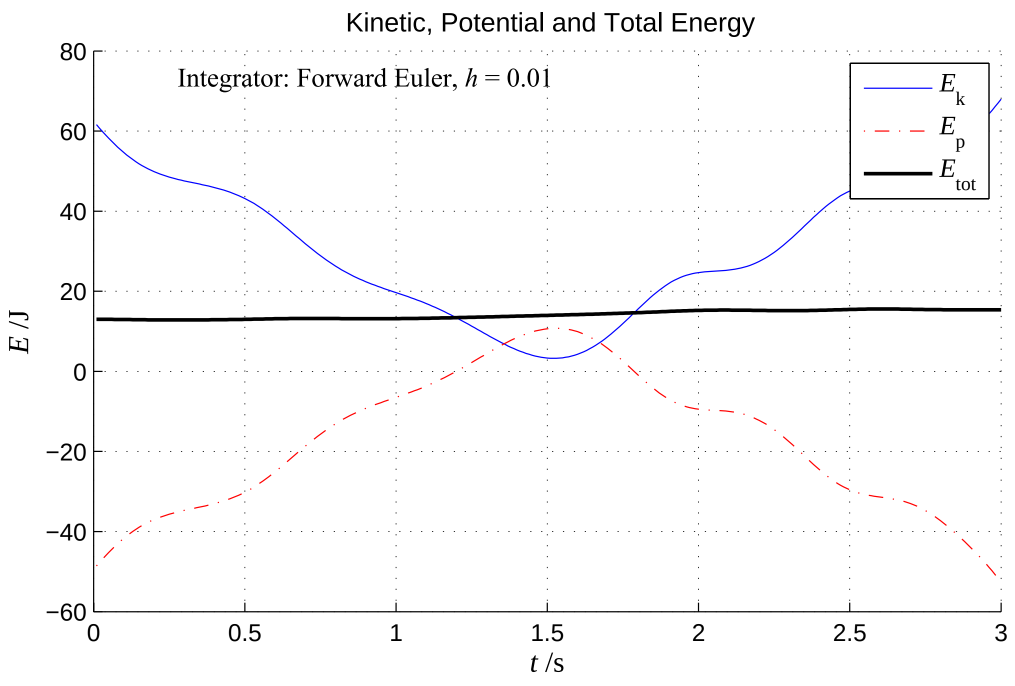


Figure 4–1: The energies of the undisturbed particle ($N_p = 1$) over the time, obtained using the forward Euler integrator ($h = 0.01$)

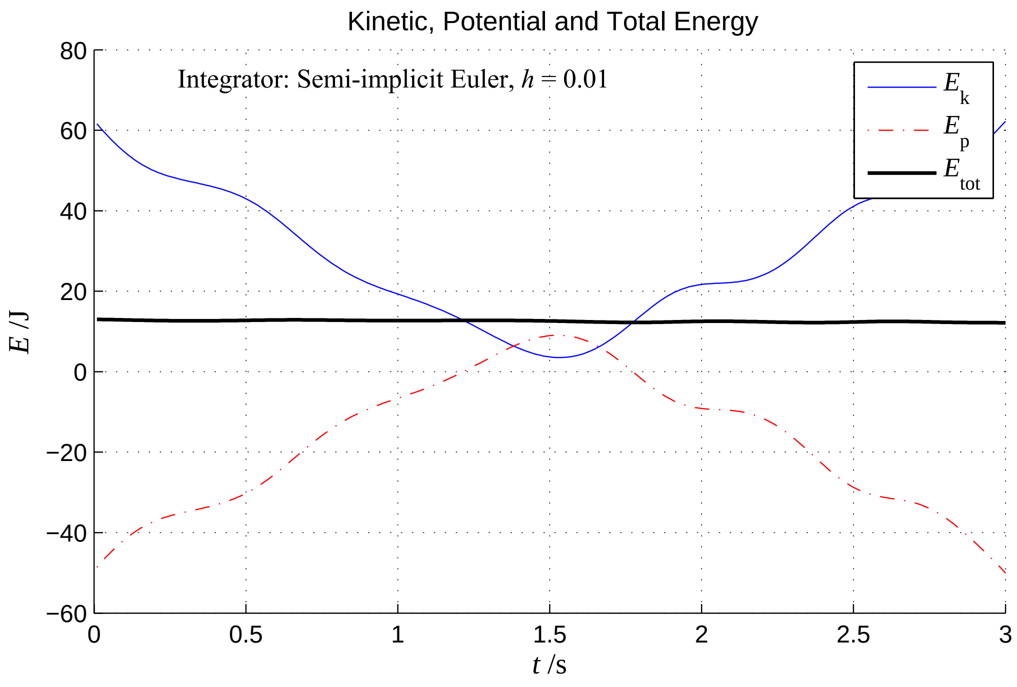


Figure 4–2: The energies of the undisturbed particle ($N_p = 1$) over the time, obtained using semi-implicit Euler integrator ($h = 0.01$)

At the same time, regarding the position displacement absolute errors, the semi-implicit Euler performs also slightly better than forward Euler for time-step sizes $h \leq 0.01$. This can be also observed on the following comparative figure showing trajectory simulations for the undisturbed particle for forward Euler (to the left) and semi-implicit Euler integrator (to the right):

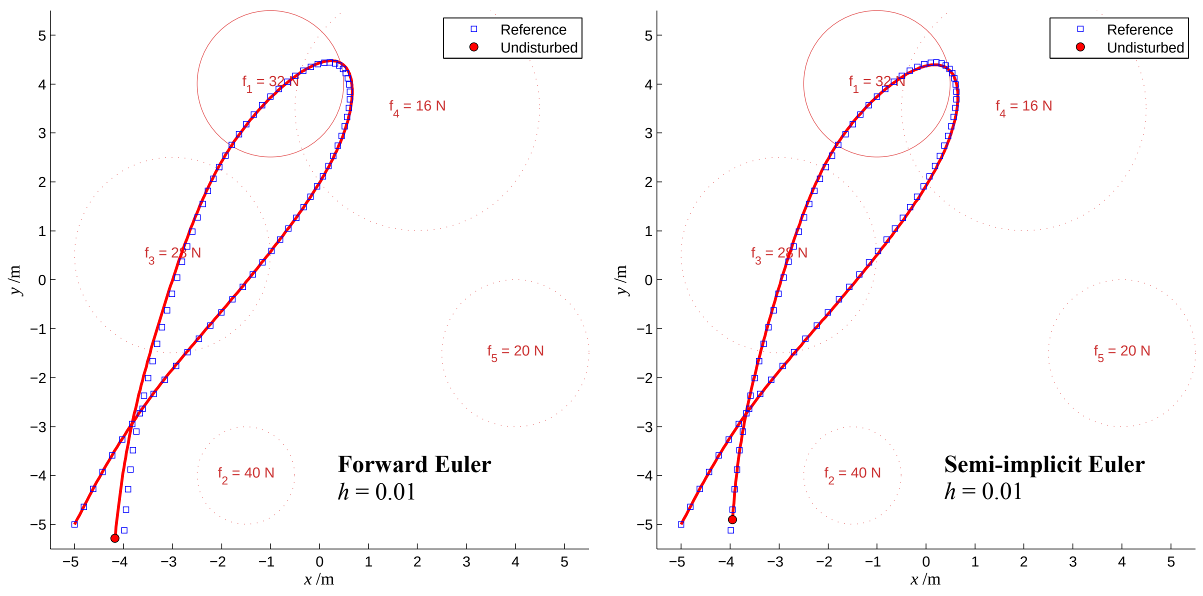


Figure 4–3: Trajectories of the undisturbed particle obtained using the forward Euler (left) and the semi-implicit Euler integrator (right) with $h = 0.01$

5 References

1. Mathews, John H., *Numerical Methods for Mathematics, Science and Engineering*, 2nd Ed, Prentice Hall, 1992, Section 9.5: Runge-Kutta Methods (p. 460)