The original assignment can be found here.

Background

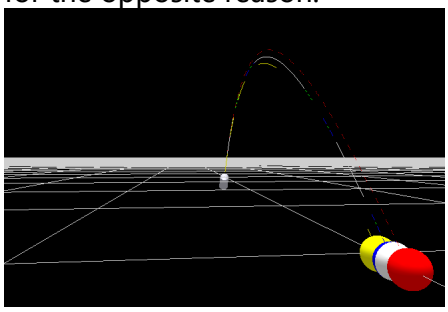
The purpose of this assignment is to compare the stability and accuracy of various integrators. As such, I spent more time on implementation and analysis than on the UI. The viewer is admittedly very simple - cylindrical mouse navigation, ctl-click-drag to zoom. There's no proper GUI, all simulation interaction is done through a console. This system is the same for both parts - ballistic motion and the spring-mass system. Also the same between both parts are the integrators used: I implemented Euler's (Red), the Midpoint (Green), 4th Order Runge-Kutta methods (Blue), and Backwards Euler's (Yellow).

Simulations

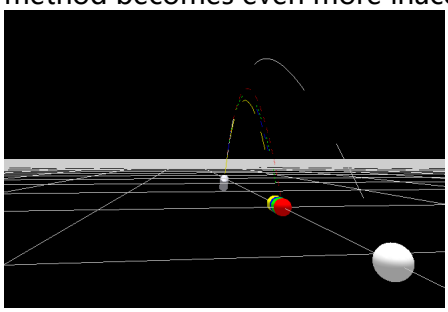
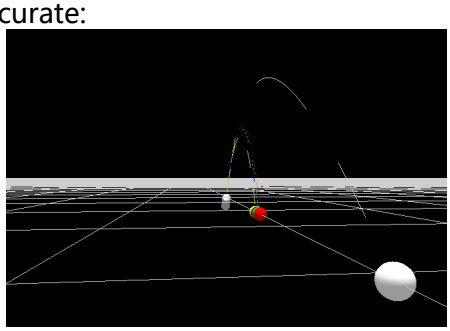
Ballistic Motion

A cannon shoots a projectile from the origin with a certain barrel azimuth and elevation. The projectile has a certain mass and is propelled by a certain amount of powder. While flying, it is subject to air friction. Where will it land?

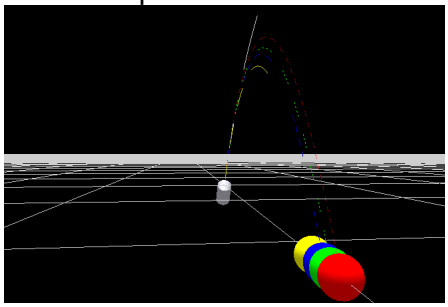
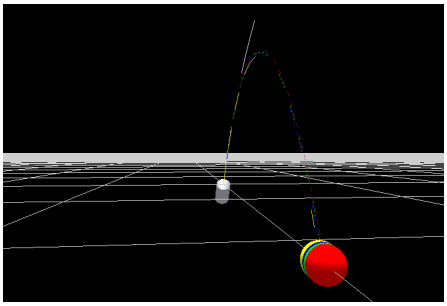
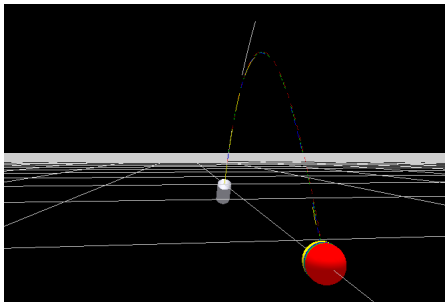
Without air friction, this problem can be solved trivially analytically. This solution is the white sphere/path in these simulations. Here is an image showing a timestep of 0.05s and no air friction, indicating that the analytical solution (white) agrees with RK4 (blue) and Midpoint (green). Euler's method (red) overshoots the correct solution due to constantly over-estimating the velocity, and the Backwards Euler's undershoots for the opposite reason:

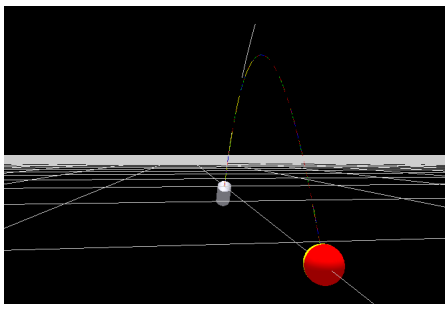


When air friction (50kg/s left, 100kg/s right) is introduced, RK4 and Midpoint begin to diverge, and Euler's method becomes even more inaccurate:

Decreasing the time step (0.05, 0.01, 0.005, 0.0001, L-R) alleviates this problem, but introduces linearly more computation:

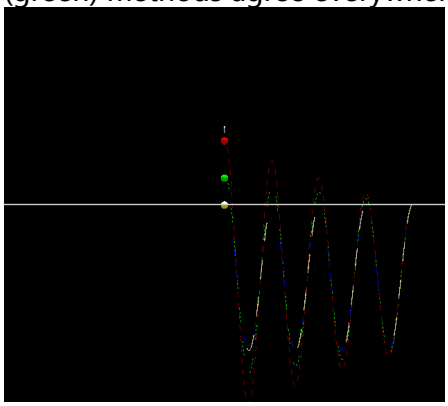


Spring-Mass System

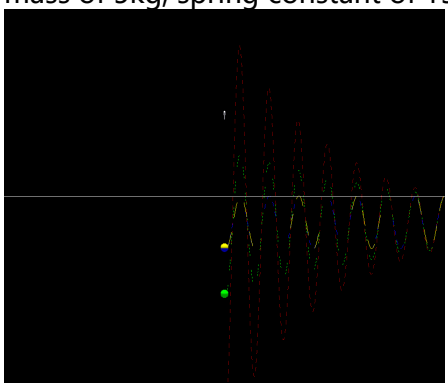
A spring (with spring constant *k*) hangs at equilibrium, and an object with mass *m* is attached to the free end and allowed uninterrupted motion. What will happen to the system?

The equation governing this system is that relating the Force on the mass (*m*) to its acceleration (*a*):
F=ma
And, in the case of a vertical spring:
F=-kx+w (where *x* is vertical displacement and *w* is the mass's weight)
So, the acceleration can be found as follows:
a=-kx/m - 9.8
And for a damped spring (with damping coefficient *b*):
a=-kx/m - 9.8 - b*v

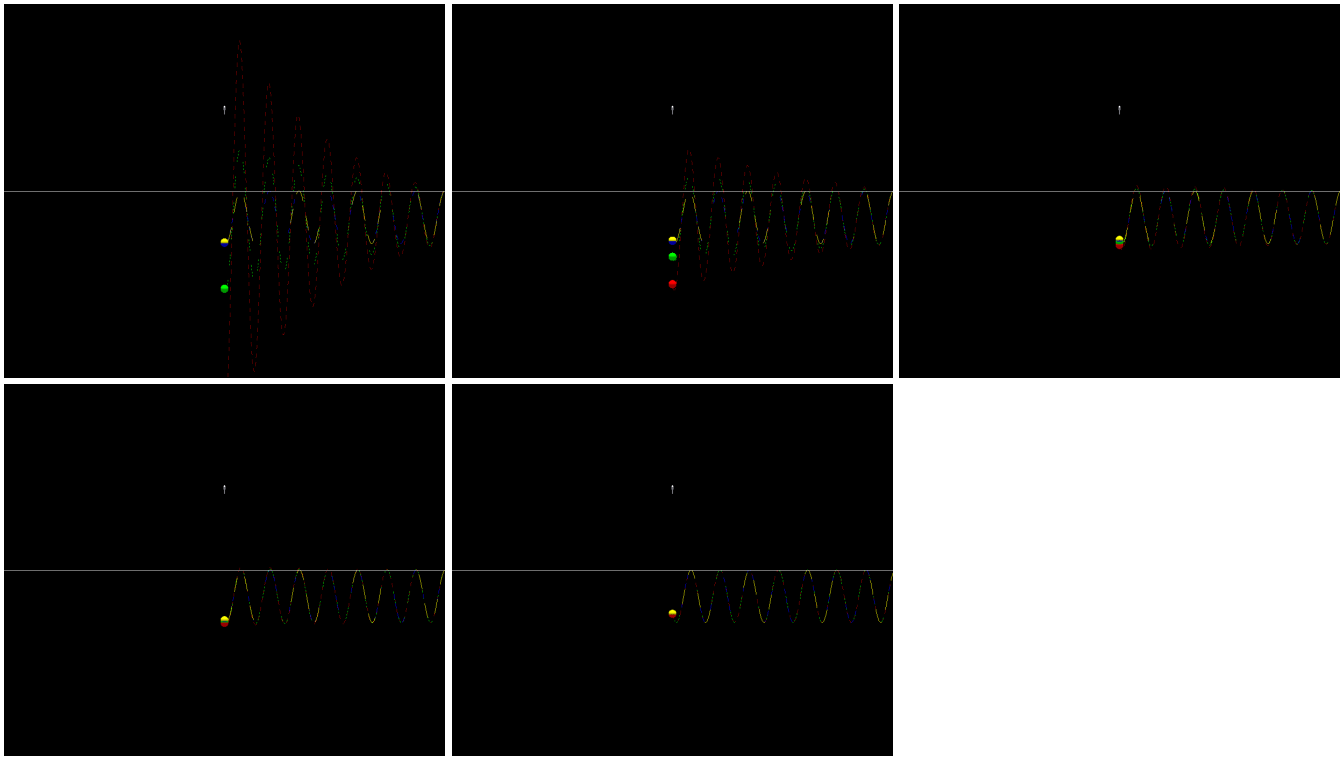
Here's a simulation running with a mass of 1kg, spring constant of 1N/m, no damping, and a timestep of 0.05. It shows that the RK4 (blue) and Backwards Euler's (yellow) methods agree with the analytic result (white) (analytic results are omitted for the rest of the experiments), and the Euler's (red) and Midpoint (green) methods agree everywhere except for the minima and maxima:



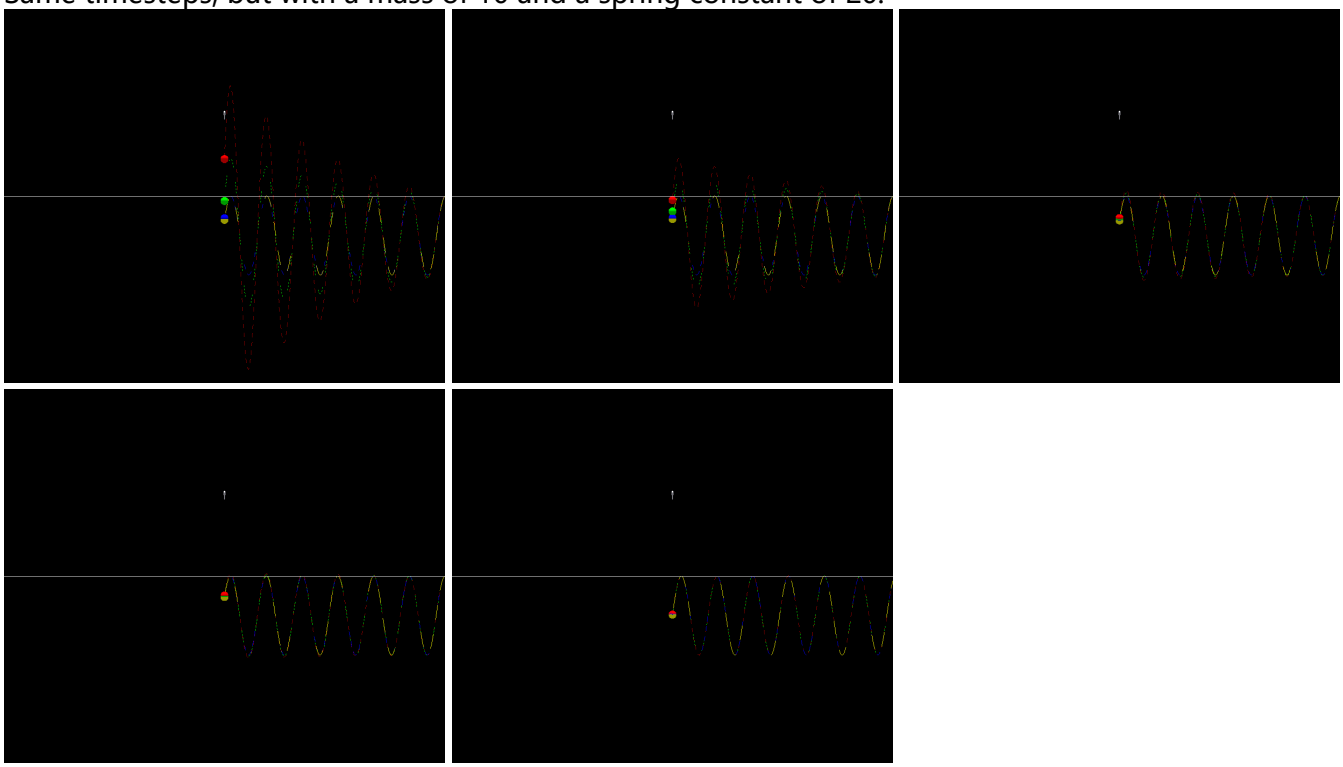
With such oscillating systems, the most common pitfall is instability. Here's an example, simulated with a mass of 5kg, spring constant of 15, no damping, and a timestep of 0.05:



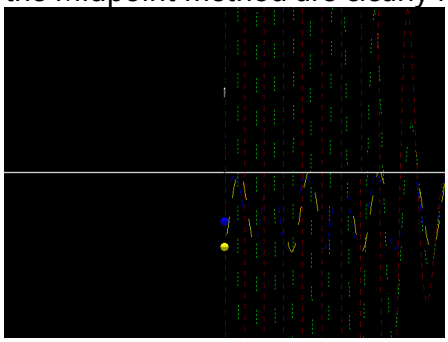
A simple approach to solving this is to use a very small timestep. Here are simulations with the same parameters as before, but with timesteps of 0.05, 0.025, 0.005, 0.0025, and 0.0001 from left to right. (View the images in a new tab/window to see the detail.) Notice how there is still significant overshoot in both Euler's method and the Midpoint method, even when the timestep is 0.005.



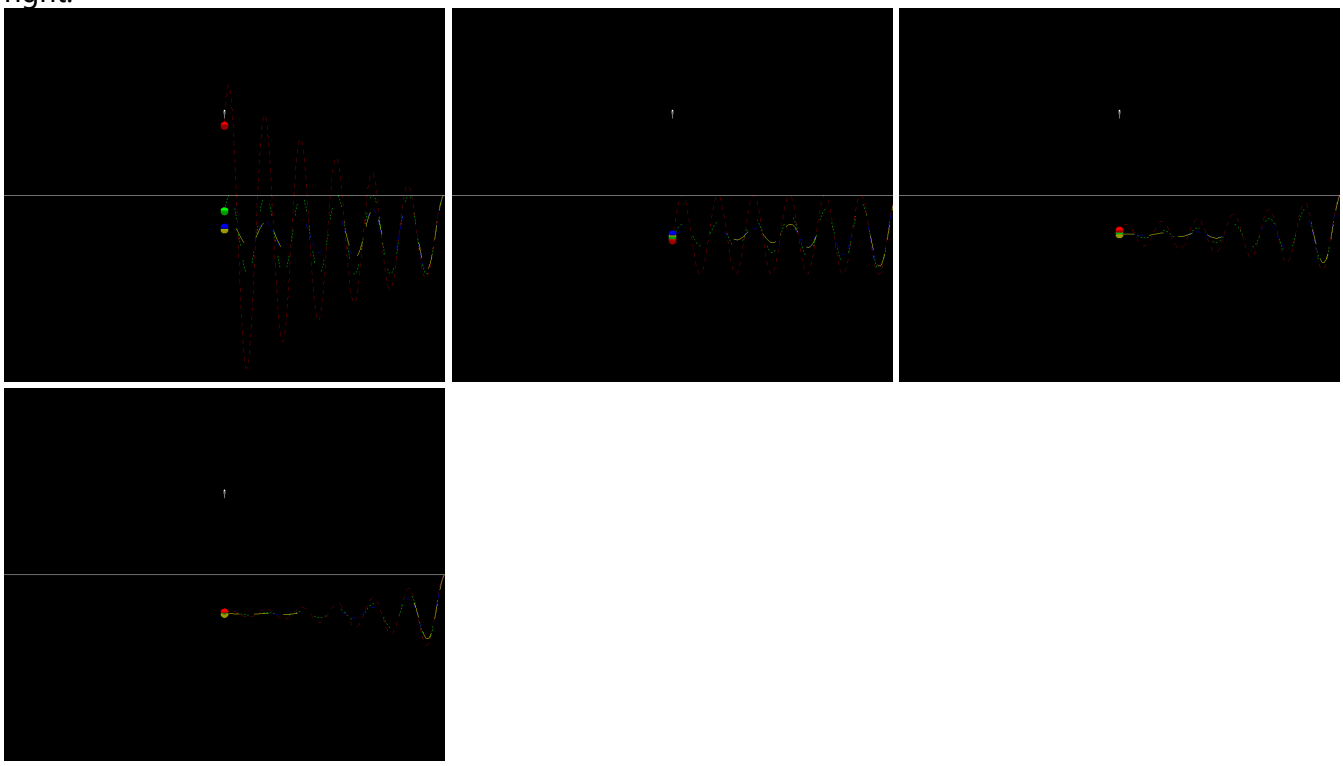
Same timesteps, but with a mass of 10 and a spring constant of 20:



On the other hand, 4th-Order Runge-Kutta and Backwards Euler's could be used with much higher timesteps. Here's a shot with a timestep of .3831 seconds - both are stable and bounded, but Euler's and the Midpoint method are clearly not stable after an iteration or two:



When damping was included in the simulation, the two previously unstable methods could be used at higher timesteps, though they still were not particularly accurate, merely stable. These simulations were performed with the same parameters, but with damping coefficients of 0.1, 0.2, 0.3, and 0.4 from left to right:



Analysis

Projectile Motion

Clearly, Euler's Method is only useful in very simple simulations. (Strictly, it's only correct when the velocity is constant over each timestep, but it can be used in more simulations with some success.) In the projectile motion case, it consistently over-estimated the velocities over the timestep, since the velocity at the beginning will always be higher than the velocity at the end. So, the projectile was simulated to go farther than it actually should. This effect was compounded when air resistance was added, since the X and Z velocities were also decreasing, not just the Y velocity due to gravity. Air resistance also made clear the difference between the Midpoint and RK4 methods. Just like Euler's, the Midpoint method overshoots the actual value due to overestimating velocity. However, since it is doing *some* velocity interpolation, it comes closer to the truth. Again, Backwards Euler's method undershoots, since it chooses the velocity at the *end* of the timestep, which will always be the smallest velocity within the timestep.

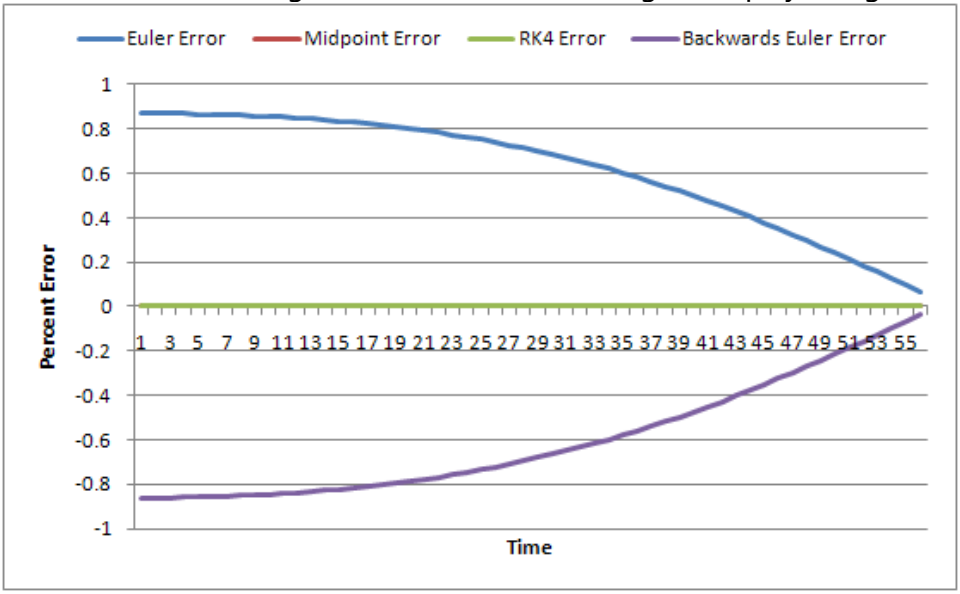
Table 1: Final distance determined by simulations at given timestep.
(Mass: 0.5, Powder: 250, Drag: 100)

Timestep	Euler	Midpoint	Backwards Euler	RK4
0.05	23.237	22.706	21.496	22.254
0.01	22.452	22.345	22.103	22.255
0.005	22.354	22.300	22.179	22.255
0.0001	22.257	22.256	22.254	22.255

Table 2: % Error in final distance determined by simulations at given timestep
(where smallest-timestep RK4 is ground truth).

Timestep	Euler	Midpoint	Backwards Euler	RK4
0.05	4.692	2.027	-3.410	-0.005
0.01	0.885	0.404	-0.683	0.000
0.005	0.445	0.202	-0.341	0.000
0.0001	0.009	0.005	-0.005	0.000

Figure 1: Percent error of the different simulators as a function of time. The reference is the analytical solution, the time step is 0.05 seconds. The errors decrease as the time increases due to the definition of error in this case: the difference between the *total* distance traveled. At the end of the simulations, they are all at or near a height of 0, while errors in height will play a large role sooner in the simulations.



Spring-Mass System

When the spring was undamped, Euler's method was unstable for every time step except for the smallest and few iterations. Even then, given enough iterations, the simulation will become unstable as errors compound. Adding damping, necessary to model a real spring, gives stability to all the integrators at a reasonable time step (very dependent on the values of the constants).

Table 3: Position of the mass at time 9.4 as determined by different simulators at different time steps.

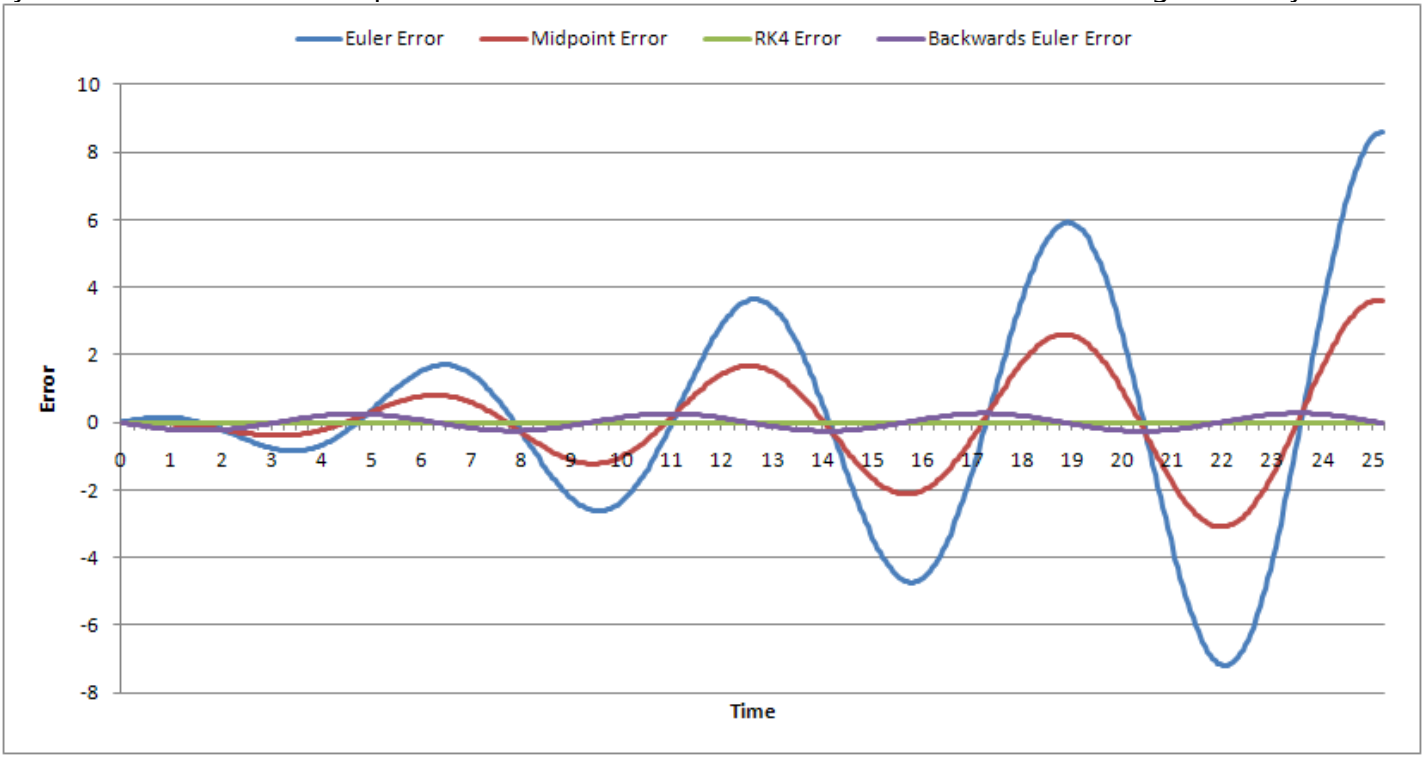
Timestep	Euler	Midpoint	Backwards Euler	RK4	Analytic
0.05	-22.2061	-20.8221	-19.5902	-19.5969	-19.5969
0.025	-20.8249	-20.1946	-19.5999	-19.6	-19.5324
0.005	-19.8312	-19.7142	-19.5986	-19.5981	-19.5356
0.0025	-19.7134	-19.6554	-19.5978	-19.5976	-19.5179
0.0005	-19.6202	-19.6087	-19.5972	-19.5971	-19.5995

Table 4: Percent error of the position of the mass at time 9.4 as determined by different simulators at different time steps. The ground truth is considered to be the Analytic solution with a time step of 0.0025s: the change in the trend at dt=0.0005 seems like it might be caused by truncation errors in floating point summation. (The overall simulation time was kept track of by simply adding dt to a running total at each iteration.)

Timestep	Euler	Midpoint	Backwards Euler	RK4	Analytic
0.05	13.773	6.682	0.370	0.405	0.405
0.025	6.696	3.467	0.420	0.421	0.074
0.005	1.605	1.006	0.413	0.411	0.091
0.0025	1.00	0.704	0.409	0.408	0.000

0.0005	0.524	0.465	0.406	0.405	0.418

Figure 2: Absolute error of the different simulators as a function of time. The reference is the analytical solution, the time step is 0.05 seconds. RK4 and Backwards Euler's methods show bounded and constant errors, respectively, indicating that they are stable techniques and usable in practice, even for undamped systems. Euler's and the Midpoint methods, however, show unbounded errors, indicating instability.



Conclusion

While Euler's method is very easy to implement and simple to compute, its high errors make it unusable for all but the simplest simulations. The Midpoint method is marginally more accurate and robust, but takes roughly twice as much computational power to simulate. Backwards Euler's method, a type of implicit method, is no more complicated than the regular Euler's method, but proves to be much more stable in simple harmonic systems. Finally, the 4-th Order Runge-Kutta method is by far the most accurate and stable, though it takes around four times as many computations during simulation. This is a small price to pay for a stable and accurate solution, though, especially when the ability to use a larger time step is factored in. For example, by using a time step that is ten times that used for Euler's method, RK4 is actually 2.5x cheaper.

The code and compiled binaries (along with necessary DLLs [for screenshot capturing]) can be found here.