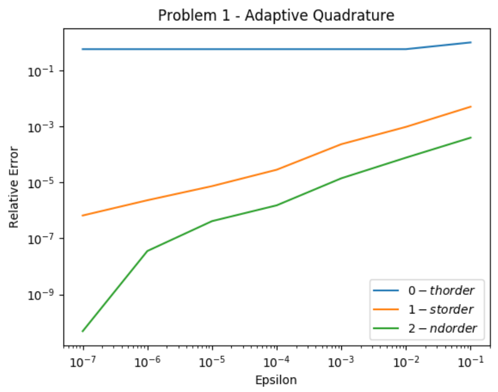
18.330 Pset 2

Kathleen Brandes

February 26, 2019

1 Problem 1

This is a plot shows the relative error of the adaptive quadrature algorithm using rectangular (0-th order), trapezoidal (1-st order), and Simpson's (2-nd order) rules compared to increasing ϵ for a test function of $f(x) = x^10e^{4x^3-3x^4}$ and a starting $N = 1$ on the interval $[0, 3]$.



To get an exact solution for this function, I used the solution to Simpson's Rule estimation with $n = 10^5$ iterations. We can see that as the termination constraint ϵ for the adaptive quadrature decreases towards zero, the relative error also decreases for each of the integration rules. As well, the error orders of each of the base Newton-Cotes quadrature rules used by the adaptive algorithm to estimate are reflected as well. My code for problem 1 is attached separately in the document problem1.jl.

2 Problem 2

2.1 Part A

Integrating $f(t)$ on the interval $t \in [a, b]$ can be approximated using the Euler's method for the function dependent only on t(and not also u as is the case for the initial value problem which it is normally used for), where it will have $u(t_0) = u(a) = 0$,$t_{i+1} = t_i + h$, and $\frac{du}{dt} = f(x)$, such that you get an approximation by doing a Riemann sum with the left endpoints as the height of each successive rectangle:

$u(t) = \int_a^b f(t)dt = u_0 + \sum_{i=0}^{N-1} hf(t_i) = \sum_{i=0}^{N-1} \frac{b-a}{N} f(t_i)$

This approximation would yield the exact same estimate as the Rectangular Newton-Cotes method for approximating integrals, which also uses the left points for each interval as the estimates for the values of the integral.

2.2 Part B

For integrating $f(t)$ on the interval $t \in [a, b]$, this can be estimated using the improved Euler method as well, where the function is only dependent on t and not also on u (which is the case for the initial value problem). The estimate uses $\frac{du}{dt} = f(t)$, $t_{i+1} = t_i + h$, and $u(t_0) = u(a) = 0$:

$u(t) = \int_a^b f(t)dt = u(t_0) + \sum_{i=1}^{N-1} \frac{h}{2}(f(t_i) + f(t_{i+1}) = \sum_{i=1}^{N-1} \frac{b-a}{2N}(f(t_i) + f(t_{i+1})$

This is explicitly the trapezoid Newton-Cotes integration rule for estimating integrals as it takes the two endpoints and multiplies them by $\frac{h}{2}$ for each iteration.

3 Problem 3

For the error analysis, let us assume that the true solution for the ODE is $u(t)$. Now, we can do a taylor expansion of this true solution with a step size of h centered around t_0, since that is the final set size for the midpoint method:

$u(t + 0 + h) = u(t_0) + hu'(t_0) + \frac{h^2}{2}(u''(t_0))$

Applying the chain rule and using the notation from class, we can get $u''(t_0) = \frac{df}{dt}|_{t_0,u_0} + \frac{df}{du}|_{t_0,u_0}\frac{du}{dt}|_{t_0,u_0} = f_t + f_u f(t_0, u_0)$, and we know from the problem definition that the ODE is $\frac{du}{dt} = f(t, u)$ such that $u'(t_0) = f(t_0, u_0) = f_0$. Using these two expansions, we can substitute these values into the taylor expansion of the true solution to get an expression in terms of the function f:

$u(t_0 + h) = u(t_0) + hf(t_0, u_0) + \frac{h^2}{2}(f_t + f_u f(t_0, u_0))$

$= u(t_0) + hf_0 + \frac{h^2}{2} f_t + \frac{h^2}{2} f_u f_0 + O(h^3)$

Now, we also want to determine what the midpoint method gets as a solution for the problem. We will taylor expand $f(t, u)$ around t_0, u_0 with step size h since we don't know what the exact solution will be:

$f(t_0 + h, u_0 + h) = \frac{du}{dt}|_{t_0,u_0} + h(\frac{df}{dt}|_{t_0,u_0} + \frac{df}{du}|_{t_0,u_0} = f_0 + hf_t + hf_u$

Now, we can also do a taylor expansion of the second evaluation of f with step size of $\frac{h}{2}$ at t_0, u_0:

$f(t_0 + \frac{h}{2}, u_0 + \frac{h}{2}f(t,u)) = f_0 + \frac{h}{2}hf_t + \frac{h}{2}f_0 f_u + O(h^2)$

We can then substitute this expansion back into the midpoint method for the estimate:

$u_{n+1} = u_n + h * f(t_0 + \frac{h}{2}, u_0 + \frac{h}{2}f(t,u)) = u_n + h[f_0 + \frac{h}{2}hf_t + \frac{h}{2}f_0 f_u + O(h^2)]$
$= u_n + hf_0 + \frac{h^2}{2}f_t + \frac{h^2}{2}f_0 f_u + O(h^3)$.

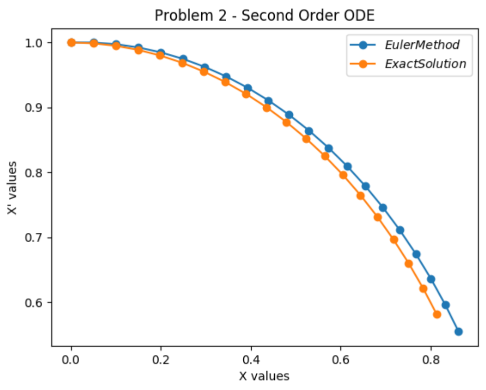
Now, we can calculate the error to be $u_{n+1}^{midpt} - u_{n+1}^{exact} = O(h^3)$ for a local single iteration of the algorithm. And then, since the algorithm has N total iterations, where $N = \frac{b-a}{h}$, so the global error of the algorithm is $O(h^3)N = O(h^2)$.

4 Problem 4

My code for problem 4 is attached separately in the document problem4.jl.

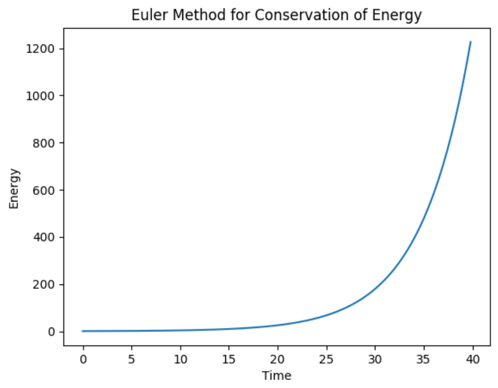
4.1 Part A

The Euler Method does not do a good job approximating this second order ODE $\ddot{x} = -x$ with initial conditions of $x(0) = 0$ and $\dot{x} = 1$. Here is a plot comparing the exact solution, which is $f(x) = sin(x)$, and the Euler approximation, given a step size of $h = .05$:



4.2 Part B

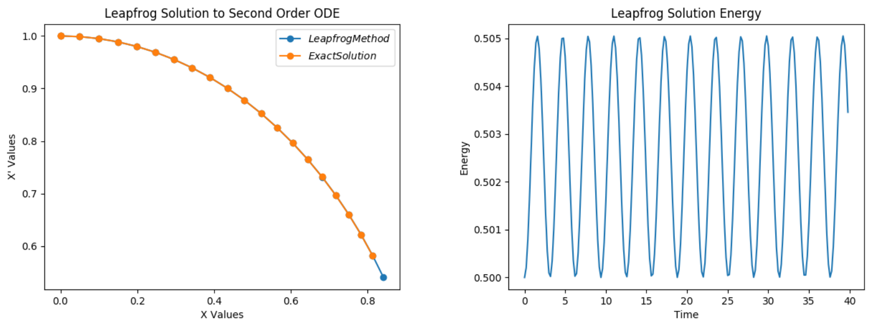
Additionally, we can see that the Euler method does not preserve conservation of energy. Here is a plot of the energy $E = .5\dot{x}^2 + .5x^2$ over time.



This plot shows that as time continues to increase, energy also increases, rather then remaining constant.

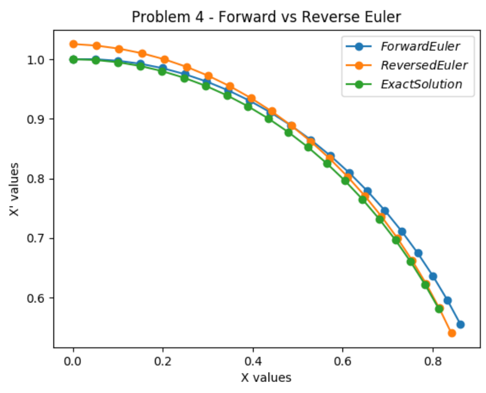
4.3 Part C

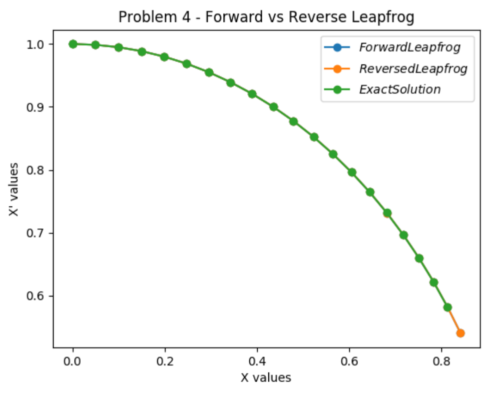
Using the leapfrog method to solve this initial value problem is much more successful. Here is a plot (left) of the exact solution and the solution found using the leapfrog method. In addition to being a much more accurate estimate, the leapfrog method also conserves energy. In the plot (right) below, we can see this because as time increases, the energy oscillates around zero for the leapfrog estimates.



4.4 Part D

Another flaw of the Euler method is that it does not respect time reversal symmetry. Here is a plot (left) showing the Euler method estimating the solution from $t = 0$ to $t = 1$ with a step size of $h = .05$ and then from $t = 1$ to $t = 0$ with a step size of $h = -.05$. In contrast, the leap frog method does respect time reversal symmetry, which is shown in the plot (right) for the same time interval and step sizes.





4.5 Part E

We can see this more exactly by computing each method going from (t_n, u_n) to $(t_n + h, u_{n+1})$ and then from (t_{n+1}, u_{n+1}) to $(t_{n+1} - h, u_n)$.
The Euler method:

Forward: $u_{n+1} = u_n + hf(t_n, u_n)$
Reverse: $u_n = u_{n+1} - hf(t_{n+1}, u_{n+1})$
$u_{n+1} = u_n + hf(t_{n+1}, u_{n+1})$
Clearly these are not equal because the two functions will not evaluate to the same thing for different values.
The Leapfrog method:

Forward: $v_{n+.5} = v_n + \frac{h}{2}F(x_n)$
$x_{n+1} = x_n + hv_{n+.5} = x_n + hv_n + \frac{h^2}{2}F(x_n)$
$v_{n+1} = v_{n+.5} + \frac{h}{2}F(x_{n+1}) = v_n + \frac{h}{2}F(x_n) + \frac{h}{2}F(x_{n+1})$

Reverse: $v_{n-.5} = v_{n+1} - \frac{h}{2}F(x_{n+1})$
$x_n = x_{n+1} - hv_{n-.5}$
$v_n = v_{n-.5} - \frac{h}{2}F(x_n) = v_{n+1} - \frac{h}{2}F(x_{n+1}) - \frac{h}{2}F(x_n)$
$v_{n+1} = v_n + \frac{h}{2}F(x_{n+1}) + \frac{h}{2}F(x_n)$

This shows that integrating forward and reverse using the leapfrog method is equivalent, and therefore the leapfrog method has time reversal symmetry.

5 Problem 5

5.1 Part A

Newton's Laws of three bodies with equal mass m and position vectors r_1, r_2, r_3 interacting under the force of gravity are as follows:

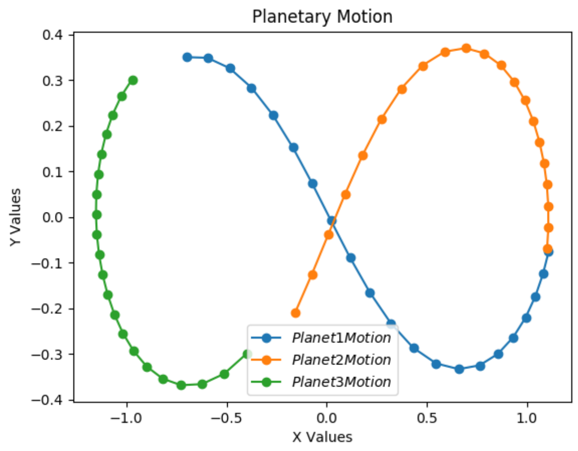
Planet 1: $\ddot{r_1} = -Gm(\frac{r_1 - r_2}{|r_1 - r_2|^3} + \frac{r_1 - r_3}{|r_1 - r_3|^3})$

Planet 2: $\ddot{r_2} = -Gm(\frac{r_2 - r_1}{|r_2 - r_1|^3} + \frac{r_2 - r_3}{|r_2 - r_3|^3})$

Planet 3: $\ddot{r_3} = -Gm(\frac{r_3 - r_2}{|r_3 - r_2|^3} + \frac{r_3 - r_1}{|r_3 - r_1|^3})$

5.2 Part B

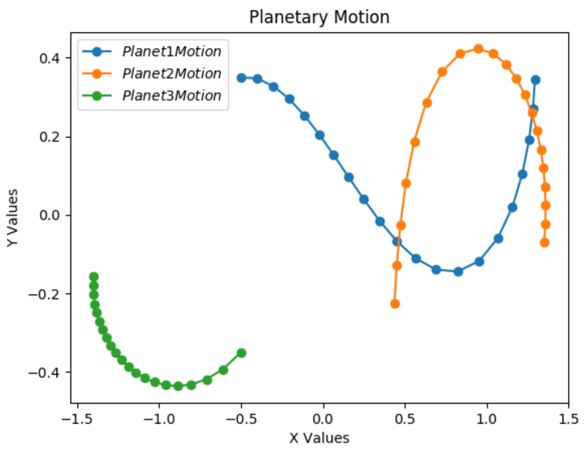
My code for plotting these ODEs estimated by the leapfrog integration method is attached in the file problem5.jl. Given the initial conditions, this is the outputted planetary motion in the x-y coordinate plane:



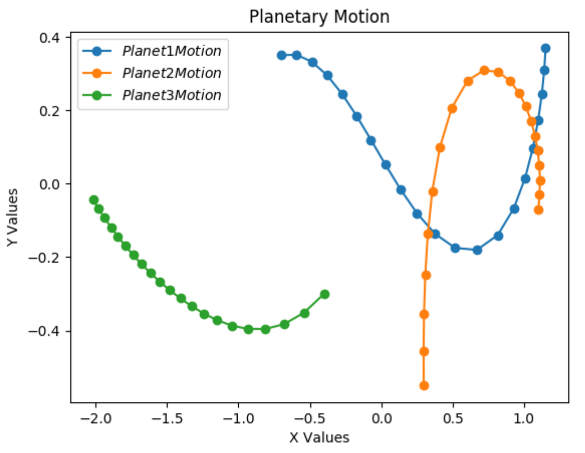
The shape of the leapfrog integrated solutions seems to accurately mimic planetary orbits and interactions between each other.

5.3 Part C

The first perturbation I did was to (x, y) pairs of coordinates, to give the initial starting conditions of $(x_1, y_1) = [-.5, .35], (x_2, y_2) = [1.35, -.07], (x_3, y_3) = [-.5, -.35]$ and the derivative coordinates (velocities) remained the same as part b. This produced planetary motion in the x-y plane:



The second perturbation I did was to the $(\dot{x}, \dot{y})$ pairs of coordinates, to give the initial starting conditions of $(\dot{x_1}, \dot{y_1}) = [.99, .108], (\dot{x_2}, \dot{y_2}) = [.125, .4], (\dot{x_3}, \dot{y_3}) = [-1.4, -.61]$ where the position coordinates remained the same as part b. This produced planetary motion in the x-y plane:



In both sets of perturbations, we can see that the planetary motion as found using leapfrog integration is very susceptible to small changes in both the x,y positions and the $\dot{x}, \dot{y}$ velocities. These small changes lead to drastically different orbits and trajectories of motion.