1 Practice

Solution 1.1. First, use the Lagrange interpolation polynomial for the three given points.

$$f(t,y) \approx \frac{(t-t_{i-1})(t-t_{i-2})}{(t_i-t_{i-1})(t_i-t_{i-2})} f(t_i,w_i) + \frac{(t-t_i)(t-t_{i-2})}{(t_{i-1}-t_i)(t_{i-1}-t_{i-2})} f(t_{i-1},w_{i-1}) + \frac{(t-t_i)(t-t_{i-1})}{(t_{i-2}-t_i)(t_{i-2}-t_{i-1})} f(t_{i-2},w_{i-2}).$$

Next, we integrate all three terms above to obtain the coefficients for the update rule. In each case, we will use the substitution $s = t - t_i$ to simplify the algebra

$$\int_{t_i}^{t_{i+1}} \frac{(t-t_{i-1})(t-t_{i-2})}{(t_i-t_{i-1})(t_i-t_{i-2})} f(t_i,w_i)dt = \frac{f(t_i,w_i)}{(h)(2h)} \int_0^h (s+h)(s+2h)ds = \frac{f(t_i,w_i)}{(h)(2h)} \frac{23h^3}{6}$$

$$= \frac{23}{12} h f(t_i,w_i)$$

$$\int_{t_i}^{t_{i+1}} \frac{(t-t_i)(t-t_{i-2})}{(t_{i-1}-t_i)(t_{i-1}-t_{i-2})} f(t_{i-1},w_{i-1})dt = \frac{f(t_{i-1},w_{i-1})}{(-h)(h)} \int_0^h (s)(s+2h)ds = \frac{f(t_{i-1},w_{i-1})}{(-h)(h)} \frac{4h^3}{3}$$

$$= -\frac{4}{3} h f(t_{i-1},w_{i-1})$$

$$\int_{t_i}^{t_{i+1}} \frac{(t-t_i)(t-t_{i-1})}{(t_{i-2}-t_i)(t_{i-2}-t_{i-1})} f(t_{i-2},w_{i-2})dt = \frac{f(t_{i-2},w_{i-2})}{(-2h)(-h)} \int_0^h (s)(s+h)ds = \frac{f(t_{i-2},w_{i-2})}{(-2h)(-h)} \frac{5h^3}{6}$$

$$= \frac{5}{12} h f(t_{i-2},w_{i-2})$$

Note in particular that none of the coefficients depend on i, meaning that the update rule is uniform for each i:

$$w_{i+1} = w_i + h\left(\frac{23}{12} f(t_i,w_i) - \frac{4}{3} f(t_{i-1},w_{i-1}) + \frac{5}{12} h(t_{i-2},w_{i-2})\right)$$

Solution 1.2. First, use the Lagrange interpolation polynomial for the four given points and then proceed in the same way as last time.

$$f(t,y) \approx \frac{(t-t_i)(t-t_{i-1})(t-t_{i-2})}{(t_{i+1}-t_i)(t_{i+1}-t_{i-1})(t_{i+1}-t_{i-2})} f(t_{i+1},w_{i+1}) + \frac{(t-t_{i+1})(t-t_{i-1})(t-t_{i-2})}{(t_i-t_{i+1})(t_i-t_{i-1})(t_i-t_{i-2})} f(t_i,w_i) + \frac{(t-t_{i+1})(t-t_i)(t-t_{i-2})}{(t_{i-1}-t_{i+1})(t_{i-1}-t_i)(t_{i-1}-t_{i-2})} f(t_{i-1},w_{i-1}) + \frac{(t-t_{i+1})(t-t_i)(t-t_{i-1})}{(t_{i-2}-t_{i+1})(t_{i-2}-t_i)(t_{i-2}-t_{i-1})} f(t_{i-2},w_{i-2}).$$

Next we integrate all four terms above to obtain the coefficients for the update rule. In each case, we will use the substitution $s = t - t_i$ to simplify the algebra

$$\int_{t_i}^{t_{i+1}} \frac{(t-t_i)(t-t_{i-1})(t-t_{i-2})}{(t_{i+1}-t_i)(t_{i+1}-t_{i-1})(t_{i+1}-t_{i-2})} f(t_{i+1}, w_{i+1})dt = \frac{f(t_{i+1}, w_{i+1})}{(h)(2h)(3h)} \int_0^h s(s+h)(s+2h)ds$$

$$= \frac{9}{24} h f(t_{i+1}, w_{i+1})$$

$$\int_{t_i}^{t_{i+1}} \frac{(t-t_{i+1})(t-t_{i-1})(t-t_{i-2})}{(t_i-t_{i+1})(t_i-t_{i-1})(t_i-t_{i-2})} f(t_i, w_i)dt = \frac{f(t_i, w_i)}{(-h)(h)(2h)} \int_0^h (s-h)(s+h)(s+2h)ds$$

$$= \frac{19}{24} h f(t_i, w_i)$$

$$\int_{t_i}^{t_{i+1}} \frac{(t-t_{i+1})(t-t_i)(t-t_{i-2})}{(t_{i-1}-t_{i+1})(t_{i-1}-t_i)(t_{i-1}-t_{i-2})} f(t_{i-1}, w_{i-1})dt = \frac{f(t_{i-1}, w_{i-1})}{(-2h)(-h)(h)} \int_0^s (s-h)s(s+2h)ds$$

$$= -\frac{5}{24} h f(t_{i-1}, w_{i-1})$$

$$\int_{t_i}^{t_{i+1}} \frac{(t-t_{i+1})(t-t_i)(t-t_{i-1})}{(t_{i-2}-t_{i+1})(t_{i-2}-t_i)(t_{i-2}-t_{i-1})} f(t_{i-2}, w_{i-2})dt = \frac{f(t_{i-2}, w_{i-2})}{(-3h)(-2h)(-h)} \int_0^h (s-h)s(s+h)ds$$

$$= \frac{1}{24} h f(t_{i-2}, w_{i-2})$$

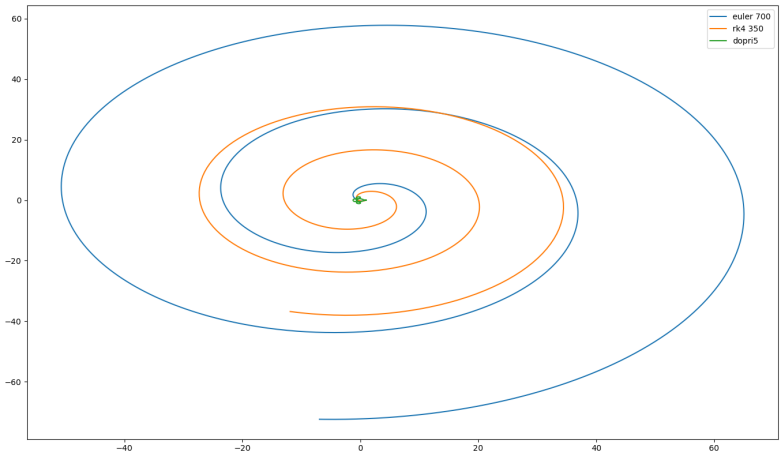
None of the coefficients depend on i, meaning that the update rule is uniform for each i:

$$w_{i+1} = w_i + h\left(\frac{9}{24} f(t_{i+1}, w_{i+1}) + \frac{19}{24} f(t_i, w_i) - \frac{5}{24} f(t_{i-1}, w_{i-1}) + \frac{1}{24} f(t_{i-2}, w_{i-2})\right).$$

2 Adaptive Solver

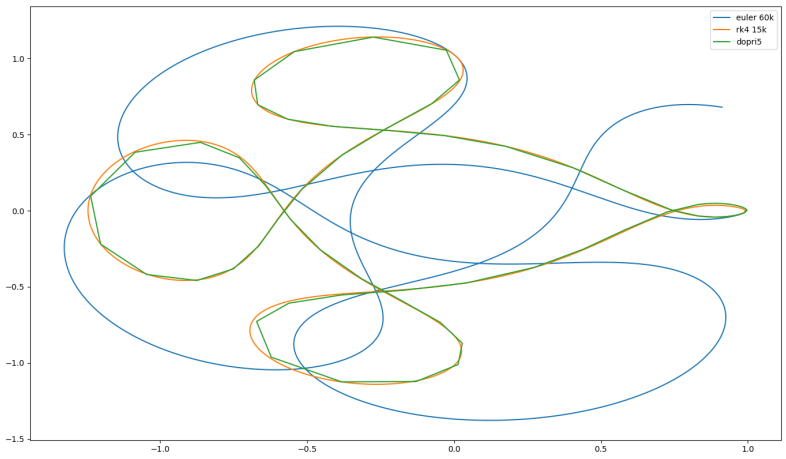
Solution 2.1. See the code on Canvas in ode2.py.

Solution 2.2. In my implementation, using similar computational time for all methods, I obtained the following plot



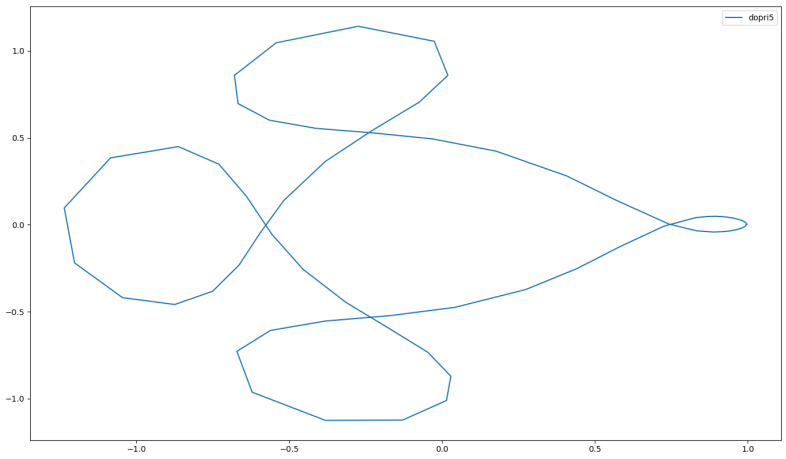
The computational times taken (in seconds) were 0.005911588668823242 for Euler's method with 1400 steps, 0.006769657135009766 for the order 4 Runge-Kutta method with 350 steps, and 0.005250453948974609 for DOPRI5 with 74 accepted steps and 25 rejected steps for a total of 99 steps. The number of steps in each case was chosen to yield roughly comparable computation times. As we can see from the graph however, the Euler and RK4 approximations do not converge to the actual solution due to insufficient numerical accuracy, while the DOPRI5 method does gives a good approximation.

Using more timesteps, I obtain the plot below



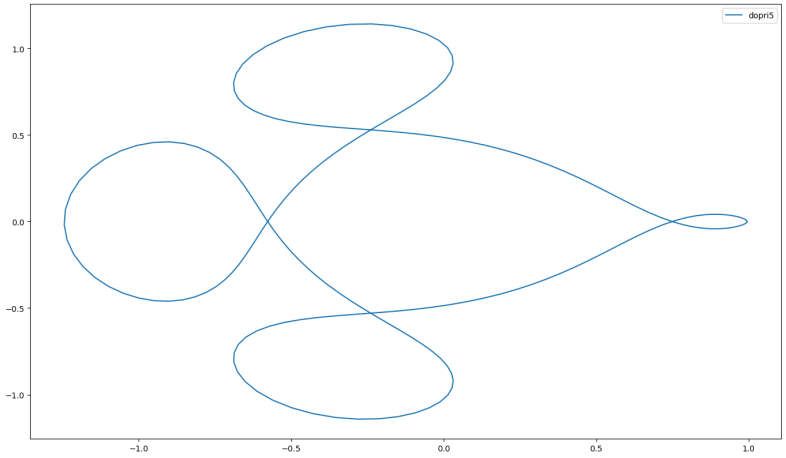
The computational times taken (in seconds) were 0.2579333782196045 for Euler's method with 60000 steps, 0.30963873863220215 for the order 4 Runge-Kutta method with 15000 steps, and the same DOPRI5 performance as before. As we can see, even allowing for more computation time, Euler's method remains somewhat ineffective while taking 50x more computation time while the RK4 method is reasonably effective, albeit with 60x more computation time.

As we can see, DOPRI5 massively outperforms Euler's method and even the fourth order Runge-Kutta method. Below is an isolated plot of the DOPRI5 approximation

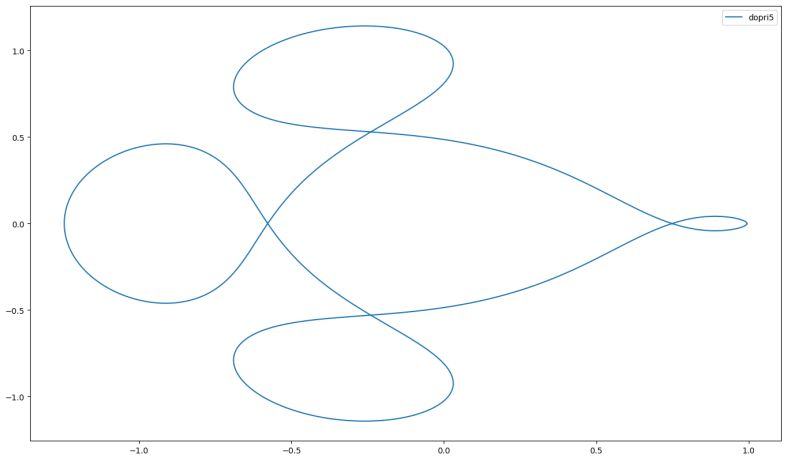


and you can see where we are allowed to take relatively large timesteps without losing much accuracy. In particular, note that near the right end, the curve looks relatively smooth so we need small timesteps in order to maintain good accuracy while in the three loops, we can see that the trajectory is very polygonal indicating the large size of tiemsteps allowed. For all the computations above, I used an initial $h = 10^{-4}$ for DOPRI5 with a tolerance of 10^{-4}.

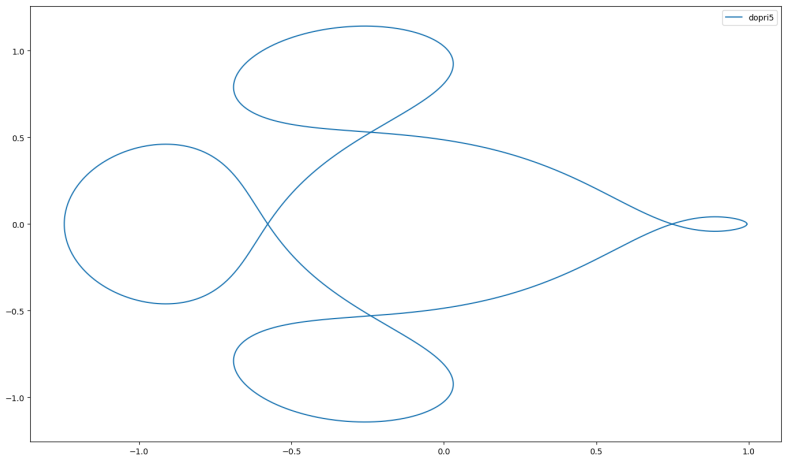
To test with other tolerances, I used $tol = 10^{-7}$ which had 249 accepted steps and 8 rejected steps with a runtime of 0.015631437301635742.



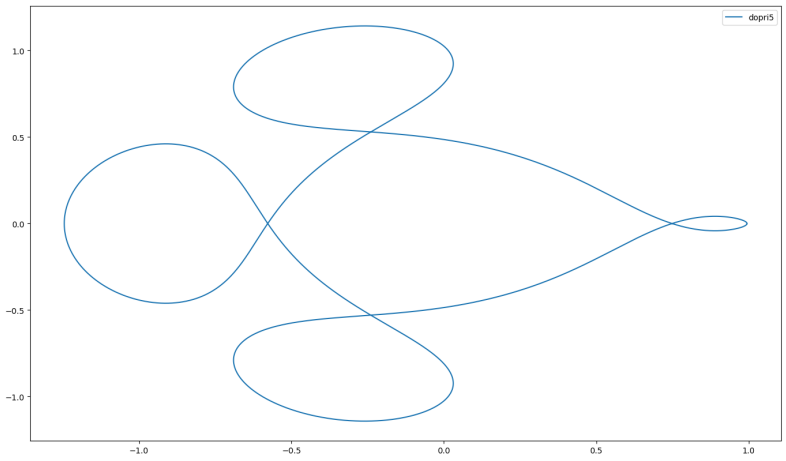
$tol = 10^{-10}$ had 974 accepted steps and 0 rejected steps with a runtime of 0.0521092414855957.



$tol = 10^{-13}$ had 2435 accepted steps and 0 rejected steps with a runtime of 0.1338825225830078.



$tol = 10^{-15}$ had 9723 accepted steps and 0 rejected steps with a runtime of 0.5526106357574463.

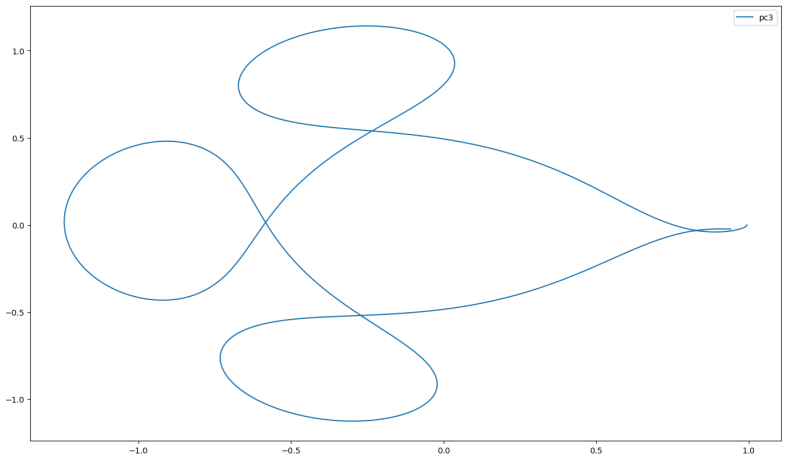


Note that we cannot ask for much smaller tolerances than this due to roundoff errors. If we did, the code would never be able to terminate. Furthermore, there were no rejected steps in the smaller tolerance cases because I selected the starting timestep size to be the same as the tolerance. This indicates that the most difficult (accuracy-wise) part of the differential equation to solve is near the beginning.

3 Predictor-Corrector

Solution 3.1. See the attached code in ode2.py. I selected the RK4 method from the previous homework to start the computation.

Solution 3.2. Using 200000 steps with a runtime of 3.4131035804748535, I obtained the following approximation



Note that the performance was not very good here, especially relative to DOPRI5 above. It is interesting to note in this case that there is essentially no improvement over using just

the explicit method by itself.

Solution 3.3. Recall from class that the implicit three-step multistep scheme is more accurate than the explicit three-step multistep scheme. This means that we can use their difference as a reasonable approximation for the true error at each step. Thus, if our approximation of the error is too large at a given timestep, then we need to reject that timestep and try again with a smaller step size. On the other hand, if our approximation of the error is sufficiently small at a given timestep, then we can try to increase the step size going forward.

Whenever we change the step size, we need to remember to recompute the next three timesteps using a one-step method (e.g. RK4) because our previously computed values were for a different step size. As a result of this additional computational expense, we would prefer to avoid changing the step sizes very frequently. Thus, whenever we do decide it is necessary to change step sizes, we might prefer to make a larger change (relative to the DOPRI5 rule) to avoid needing to change the step size again soon. More precisely, we will use the following steps for each timestep:

1. First compute the predictor $w_{i+1}^{predictor}$ using the explicit three-step multistep scheme.
2. Second compute the corrector $w_{i+1}^{corrector}$ using the implicit three-step multistep scheme with the predictor value to avoid a root-finding step.
3. Compute the distance between the two vectors as

$$\|w_{i+1}^{predictor} - w_{i+1}^{corrector}\|.$$

   If the distance is larger than some upper tolerance, then we need to reject this step, decrease the step size, recompute the next three values using a one-step method (RK4), and then try again.

   Otherwise if the distance is smaller than some lower tolerance, then we accept the step, increase the step size, recompute the next three values using a one-step method (RK4), and then go on to the next step.

   Finally, if the distance falls within the upper and lower tolerances, then simply go on to the next step without changing the step sizes.

The reason for having two tolerances here is so that we can try to avoid making too many changes to the step size, because this will lose us any time that we saved by using the adaptive method in the first place.