0 Instructions

This problem set is **due on Tuesday, Dec. 3 by midnight**. Submit your solutions on Canvas. Include the names of everyone you worked with on this problem set. Include any code you used to solve the problems as part of your submission.

1 Practice

We described the Adams multistep methods (the explicit methods are often called *Adams-Bashforth* methods while the implicit methods are often called *Adams-Moulton*) conceptually in class, though we did not give a concrete description. Here you will derive explicit descriptions for the three step explicit and implicit Adams multistep methods.

Problem 1.1. Give an explicit description of the update rule for the three step explicit multistep method discussed in class. Concretely, this means that you need to evaluate the integral in the equation below

$$w_{i+1} = w_i + \int_{t_i}^{t_{i+1}} u(t)dt$$

where $u(t)$ is the degree 2 interpolating polynomial to $f(t, y)$ which passes through the three points $(t_i, f(t_i, w_i)), (t_{i-1}, f(t_{i-1}, w_{i-1})), (t_{i-2}, f(t_{i-2}, w_{i-2}))$. (Hint: use the substitution $s = t - t_i$ in each of the three separate integrals you will need to compute.)

Problem 1.2. Give an explicit description of the update rule for the three step implicit multistep method discussed in class. Concretely, this means that you need to evaluate the integral in the equation below

$$w_{i+1} = w_i + \int_{t_i}^{t_{i+1}} u(t)dt$$

where $u(t)$ is the degree 3 interpolating polynomial to $f(t, y)$ which passes through the four points $(t_{i+1}, f(t_{i+1}, w_{i+1})), (t_i, f(t_i, w_i)), (t_{i-1}, f(t_{i-1}, w_{i-1})), (t_{i-2}, f(t_{i-2}, w_{i-2}))$. (Hint: use the substitution $s = t - t_i$ in each of the four separate integrals you will need to compute.)

The same general idea applies to find the coefficients for the explicit or implicit multistep methods of any order. In practice, if you wanted to implement this on a computer, you might do something like apply Gaussian integration to compute the values of the coefficients by integrating the Lagrange polynomials numerically.

2 DOPRI5

On the last homework, you implemented the Euler method and a Runge-Kutta method of order 4. This problem asks you to implement the adaptive method (often called DOPRI5 or ode45 in Matlab) described as follows. Note that while you could make a lot of optimizations to the following code, we will not try to optimize it heavily. First, consider the following 7-stage Butcher array

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & \frac{-56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & \frac{-25360}{2187} & \frac{64448}{6561} & \frac{-212}{729} & & & \\
1 & \frac{9017}{3168} & \frac{-355}{33} & \frac{46732}{5247} & \frac{49}{176} & \frac{-5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & \frac{-2187}{6784} & \frac{11}{84} & \\
\hline
w^5 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & \frac{-2187}{6784} & \frac{11}{84} & \\
\hline
w^4 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & \frac{-92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
$$

where the next-to-last line with w^5 on its left represents a fifth order Runge-Kutta method and the last line with w^4 on its left represents a fourth order Runge-Kutta method. In other words, you compute the stages $k_1, \ldots, k_7$ and then use them to obtain both a fourth order *and* fifth order estimate at each stage.

Now, suppose we're given a time interval $[a, b]$, a starting timestep size h, a function $f(t, y)$ modeling the differential equation $y' = f(t, y)$, an initial value $y(a) = y_0$, and a tolerance tol. Then the DOPRI5 method is as follows.

1. Initialize the starting point to the initial conditions $w_0 = y_0$.
2. Initialize $t = a$.
3. While $t < b$,
   (a) Use (t_i, w_i) to compute $k_1, \ldots, k_7$ as specified in the Butcher array above.
   (b) Use $k_1, \ldots, k_7$ to compute the two approximations w^5_{i+1} and w^4_{i+1} as specified in the Butcher array above.
   (c) Approximate the error by
   $$err = \frac{||w^5_{i+1} - w^4_{i+1}||_2}{tol\sqrt{n}}$$
   where n is the number of dimensions of the first order system (i.e. the number of equations, so if there are four equations then $n = 4$).
   (d) If $err \leq 1$, then update $w_{i+1} = w^5_{i+1}$ and $t = t + h$ and call this an accepted step. Otherwise, if $err > 1$, do not update w_{i+1} or t and call this a rejected step.

(e) Regardless of the result of the last step, compute the update factor

$$u = \min\left(5, \max\left(0.2, 0.8\left(\frac{1}{err}\right)^{0.2}\right)\right)$$

and update $h = uh$.

You should modify the description where necessary above to track the number of accepted and rejected steps.

Problem 2.1. Implement the DOPRI5 method as described above. Your function should take in the parameters a, b, h, y_0, f, tol as specified above.

Problem 2.2. Test your code with different choices of tolerances on the orbit system of differential equations

$$x'' = x + 2y' - d\frac{x+c}{((x+c)^2+y^2)^{\frac{3}{2}}} - c\frac{x-d}{((x-d)^2+y^2)^{\frac{3}{2}}}$$

$$y'' = y - 2x' - d\frac{y}{((x+c)^2+y^2)^{\frac{3}{2}}} - c\frac{y}{((x-d)^2+y^2)^{\frac{3}{2}}}$$

where $c = 0.012277471$ and $d = 1-c$ from the previous homework (problem 3.2 on homework 5). Recall that with the initial conditions

$$x(0) = 0.994, x'(0) = 0, y(0) = 0, y'(0) = -2.00158510637908252240537862224,$$

there is a periodic solution with period $P = 17.0652165601579625588917206249$. Compare the results of DOPRI5 (this homework), Euler (previous homework), and the Runge-Kutta method of order 4 (previous homework) qualitatively as well as by the total number of steps and computational time required to achieve “reasonable” output in each method.

3 Predictor-Corrector

As discussed in class, implicit multistep methods can be difficult to implement in practice due to the extra root-finding step needed in each timestep

$$w_{i+1} = w_i + \varphi(t_i, w_{i+1}, w_i, \dots).$$

The reason we need this extra root-finding step is because w_{i+1}, which is unknown, appears on both sides of the equation above and φ is typically complicated enough that we will not be able to solve for w_{i+1} algebraically. One cure for this is to use an explicit method to obtain a prediction for w_{i+1} and then to use the implicit method to obtain a correction:

$$w_{i+1}^{predictor} = w_i + \varphi(t_i, w_i, \dots, w_{i-m+1})$$

$$w_{i+1}^{corrector} = w_i + \varphi(t_i, w_{i+1}^{predictor}, w_i, \dots, w_{i-m+1}).$$

This is known as a predictor-corrector scheme, and it allows us to gain some (but not all) of the better accuracy of the implicit m-step multistep method while avoiding the computationally expensive root-finding step.

Problem 3.1. Use the three-step explicit and implicit multistep rules you derived in problems 1.1 and 1.2 to implement a predictor-corrector scheme as described above. Remember that for multistep methods, you need to compute the first few steps (in this case the first three steps) using a one-step method to get enough values to start the multistep method.

Problem 3.2. Test your code on the orbit system of differential equations from the previous problem.

If you compare the result of your predictor-corrector code to the explicit three-step multistep method by itself, you should observe that there is almost no difference between the resulting approximations. In other words, the "corrector" step was not as successful in this case as we could have hoped. One way we could have improved our results was to make use of the predictor-corrector scheme to devise an adaptive method.

Problem 3.3. Explain how you would modify the predictor-corrector scheme in order to turn it into an adaptive method. (Be careful when changing the size of a timestep!) You do not need to write code, but your explanation should be detailed enough to understand how to turn it into code.