UNIVERSITY OF UTAH, MATHEMATICS

VOVO Method for Solving IVP ODEs

Nathan Taylor

April 5, 2018

1 DORMOND-PRINCE METHOD

The basis of the algorithm is the Dormond-Prince method of solving ODEs. Dormond-Prince is a form of Runge-Kutta method designed to minimize the number of function calls for high accuracy of non-stiff problems. Dormond-Prince is in fact the method used by Matlab's ODE45. The algorithm uses the following Butcher Tableau:

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

The beauty of this method is that the first solution under the line of the tableau is a solution of order 4 while the second is of order 5, this means that in order to compute the error we only need to do one more function call instead of having to redo the entire iteration. Because we are only interested in functions of one variable we can simplify the Dormond-Prince method by ignoring the y values. This turns the Dormond-Prince Runge-Kutta method into a form of

Simpsons rule. Doing so, the table above will yield the equations:

$$k_1 = f(t)$$
$$k_2 = f(t + \frac{h}{5})$$
$$k_3 = f(t + \frac{3h}{10})$$
$$k_4 = f(t + \frac{4h}{5})$$
$$k_5 = f(t + \frac{8h}{9})$$
$$k_6 = f(t + h)$$
$$k_7 = f(t + h)$$
$$y_{n+1} = y_n + h(\frac{35}{384}k_1 + 0K_2 + \frac{500}{1113}k_3 + \frac{125}{192}k_4 - \frac{2187}{6784}k_5 + \frac{11}{84}k_6)$$
$$\tau_{est} = h(\frac{71}{57600}k_1 + 0k_2 - \frac{71}{16695}k_3 + \frac{71}{1920}k_4 - \frac{17253}{339200}k_5 + \frac{22}{525}k_6 - \frac{1}{40}k_7$$

2 Optimization

2.1 Minimizing Function Calls

Our first goal is to minimize the number of function calls that need to me made. First we note that the t value of the next step will be the $t + h$ value of the last step so we can set $k_1 = k_6$ before resetting k_6. With the exception of the first step where $k_1 = f(a)$ with a being the starting point of the integral. Then we note that k_2 is not used if we take away the y variable, and so is not needed. And the obvious one, $k_6 = k_7$ so there is no need to calculate both. Assuming we take n steps, doing this will minimize the number of function calls from $7n$ to $4n + 1$ (the extra 1 being the first k_1 value).

2.2 Taking Advantage of the Error

Initially I was keeping each step under the allowable error τ however, it is possible to take advantage of the difference in each error, if one is positive and another is negative the errors will cancel each other out to some degree. The solution to this was to simply add up all the error estimates before taking the absolute value, and if the sum of estimates was under τ accept the step as though it's estimated error was under τ.

2.3 Accidental Improvements

I found that taking the sum helped most notably for equations that ended lower than they started. In order to take advantage of equations that ended higher than they started I also looked at the difference of all error estimates that came before. This could create a problem

as the program shouldn't be allowed to add some estimates and subtract others, so I needed to ensure that it would pick one and do it always. To do this I ran $f(a)$ and $f(b)$ with a and b being the start and end points respectively, and if $f(a) < f(b)$ I took the difference, and if $f(a) > f(b)$ I took the sum.
Conveniently I was already calculating $f(a)$ as it was the first k_1 value. However, calculating $f(b)$ cost me an extra function call. To remove this extra call I added an if statement to the calculation of k_6 to see if if was the final step, if it was I would simply set k_6 equal to the previously calculated $f(b)$.
Looking back I realized that this didn't quite make sense. Why would the absolute value of the difference be different than the absolute value of the sums? From testing I could see that it was, but couldn't figure out why. I think I finally figured it out. There are two reasons. First, I didn't set the original value to 0, I set it to τ to avoid issues of allowing the sum to be less than it should be. Second, I was using $f(b)$ instead of recalling the function on the last step. This means for each failed attempt at the last step I was using one fewer call than I would have otherwise, not a major help, but not nothing.

2.4 Changing Stepsize

Figuring out the best stepsize was a challenge. Initially I just doubled the stepsize after a successful step and halved it after a failed step. This worked but I wanted to be taking better advantage of flat functions and be more warry of steep ones. Initially I had a very messy set of Boolean variables keeping track of whether I had taken at least two successful steps in a row, when I did I would triple the stepsize, based on experimentation 3 was the most useful number to work with. Eventually I realized I could just create two variables to keep track of the number of successful/failed steps in a row and multiply/divide the stepsize by those variables accordingly. This approach was not always beneficial, but in all tests I ran the harm maxed out at roughly a 5% decrease in efficiency, and was able to give up to about a 20% gain when it helped. Both of these are edge cases, however the ratio of harm to benefit was consistent with the $\frac{5}{20}$ example.

3 Citation

3.1 Helpful Websites and Blogs

- Ordinary Differential Equation Solvers ODE23 and ODE45
  - By Cleve Moler
    - This resource was the most helpful for finding a starting place, it is a blog post by one of the people who wrote ODE23 for Matlab, it was the reason that I chose to base my method on ODE45 (the original program was based on ODE45 and ODE23, but ODE23 hurt more than it helped).
    - https://blogs.mathworks.com/cleve/2014/05/26/ordinary-differential-equation-solvers-ode23-and-ode45/

- Implementation of VSVO code for solving boundary value problems of Dirichlet and Neumann type
  - By Phang Pei See and Zanariah Abdul Majid
    - The end product has very little if any relation to this paper, but it helped to get the original program up and running before I gutted and fixed everything to better fit the specific problem.
    - http://ieeexplore.ieee.org/document/7357023/authors
- Dormand Prince method Wikipedia
  - multiple authors
    - I used many wikipedia articles in making this project, but this was far and away the most useful, at any given time in the last two months I can guarantee it was open on a tap in my computer. I mostly used it for the Butcher Tableau, but the text also helped clarify some of my issues along the way.
    - https://en.wikipedia.org/wiki/Dormand%E2%80%93Prince_method