UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 204 *Numerical methods*

The Dormand-Prince method

Douglas Wilhelm Harder, LEL, M.Math.
dwharder@uwaterloo.ca
dwharder@gmail.com

1

Introduction

- In this topic, we will
  - Derive and describe the adaptive Dormand-Prince method
  - Look at the implementation
  - See two examples

2

2

The Dormand-Prince method

Recap of 4th-order Runge-Kutta

- Given (t_k, y_k), the 4th-order Runge-Kutta method sampled the slope four times:

$$\begin{aligned} s_0 &\leftarrow f\left(t_k, y_k\right) \\ s_1 &\leftarrow f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}hs_0\right) \\ s_2 &\leftarrow f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}hs_1\right) \\ s_3 &\leftarrow f\left(t_k + h, y_k + hs_2\right) \end{aligned}$$

  - Note that two samples were inside the interval $[t_k, t_k + h]$
- We then took a weighted average of these slopes:

$$y_{k+1} \leftarrow y_k + h\frac{s_0 + 2s_1 + 2s_2 + s_3}{6}$$

3

3

The Dormand-Prince method

Dormand-Prince method

- Given (t_k, y_k), the method sampled the slope seven times:

$$\begin{aligned} s_0 &\leftarrow f\left(t_k, y_k\right) \\ s_1 &\leftarrow f\left(t_k + \tfrac{1}{5}h, y_k + \tfrac{1}{5}hs_0\right) \\ s_2 &\leftarrow f\left(t_k + \tfrac{3}{10}h, y_k + \tfrac{3}{10}h\frac{s_0 + 3s_1}{4}\right) \\ s_3 &\leftarrow f\left(t_k + \tfrac{4}{5}h, y_k + \tfrac{4}{5}h\frac{11s_0 - 42s_1 + 40s_2}{9}\right) \\ s_4 &\leftarrow f\left(t_k + \tfrac{8}{9}h, y_k + \tfrac{8}{9}h\frac{4843s_0 - 19020s_1 + 16112s_2 - 477s_3}{1458}\right) \\ s_5 &\leftarrow f\left(t_k + h, y_k + h\frac{477901s_0 - 1806240s_1 + 1495424s_2 + 46746s_3 - 45927s_4}{167904}\right) \\ s_6 &\leftarrow f\left(t_k + h, \underbrace{y_k + h\frac{12985s_0 + 64000s_2 + 92750s_3 - 45927s_4 + 18656s_5}{142464}}_{z}\right) \end{aligned}$$

$$y \leftarrow y_k + h\frac{1921409s_0 + 9690880s_2 + 13122270s_3 - 5802111s_4 + 1902912s_5 + 534240s_6}{21369600}$$

4

4

Dormand-Prince method

- Important:
  - For a single step, the error of the approximation z is $O(h^6)$
    - The literature refers to this one as *fifth-order*
    - This is similar to referring to 4^{th}-order Runge-Kutta which is actually $O(h^5)$ for a single step
  - Similarly, a single step of the approximation y is $O(h^5)$

5

5

Dormand-Prince method

- From analysis, y has an $O(h^5)$ error

$$2|z - y| \approx Ch^5$$

  - Solving this for C yields:

$$C \approx \frac{2|z - y|}{h^5}$$

- We want to choose the ideal ah so that the error is $\varepsilon_{\text{abs}}(ah)$

$$C(ah)^5 = \varepsilon_{\text{abs}}(ah)$$

  - Solving this for a yields

$$a^4 = \frac{\varepsilon_{\text{abs}}}{Ch^4}$$

  - Substituting in the approximation of C from above:

$$a = \sqrt[4]{\frac{\varepsilon_{\text{abs}} h}{2|z - y|}}$$

6

6

Implementation

- The implementation requires a few constants

$$s_i \leftarrow f\left(t_k + hc_i, y_k + hc_i\left(d_{i,0}s_0 + \cdots + d_{i,i-1}s_{i-1}\right)\right)$$

```
std::size_t const DIM{7};

double step[DIM - 1]{
        1.0/5.0,        3.0/10.0,      4.0/5.0,      8.0/9.0,        1.0,          1.0
};

double tableau[DIM - 1][DIM - 1]{
    {    1.0                                                                          },
    {    1.0/4.0,        3.0/4                                                        },
    {   11.0/9.0,      -14.0/3.0,      40.0/9.0                                       },
    { 4843.0/1458.0, -3170.0/243.0,  8056.0/729.0,  -53.0/162.0                       },
    { 9017.0/3168.0,  -355.0/33.0,  46732.0/5247.0,  49.0/176.0,  -5103.0/18656.0     },
    {   35.0/384.0,      0.0,        500.0/1113.0, 125.0/192.0,  -2187.0/6784.0,  11.0/84.0 }
};

double y_coeff[DIM]{
    5179.0/57600.0, 0.0, 7571.0/16695.0, 393.0/640.0, -92097.0/339200.0, 187.0/2100.0, 1.0/40.0
};
```


Implementation

- The implementation is only slightly more complex:

```
do {
    double s[DIM]{ qdy.back() };
    double z{};

    for ( std::size_t i{0}; i < DIM - 1; ++i ) {
        double slope{0.0};

        for ( std::size_t j{0}; j <= i; ++j ) {
            slope += tableau[i][j]*s[j];
        }

        z = qy.back() + h*step[i]*slope;
        s[i + 1] = f( qt.back() + h*step[i], z );
    }
```

The Dormand-Prince method

Implementation

```
double slope_y{0.0};

for ( std::size_t i{0}; i < DIM; ++i ) {
    slope_y += y_coeff[i]*s[i];
}

double y{ qy.back() + h*slope_y };

double a{ std::pow(
    eps_abs*h/(2.0*std::abs( z - y )), 0.25
) };

if ( (a > 1.0) || (h == h_rng.first) ) {
     qt.push( qt.back() + h );
     qy.push( z );
    qdy.push( f( qt.back(), z ) );
    found = true;
}

a *= 0.9;
```

9

9

The Dormand-Prince method

Implementation

- On slide 4, we represent the calculations as:

$$s_4 \leftarrow f\left(t_k + \tfrac{8}{9}h, y_k + \tfrac{8}{9}h\frac{4843s_0 - 19020s_1 + 16112s_2 - 477s_3}{1458}\right)$$

  – In the implementation, you will note it appears as

$$s_4 \leftarrow f\left(t_k + \left(\tfrac{8}{9}h\right), y_k + \left(\tfrac{8}{9}h\right)\left(\frac{4843}{1458}s_0 - \frac{19020}{1458}s_1 + \frac{16112}{1458}s_2 - \frac{477}{1458}s_3\right)\right)$$

  – You may be tempted to do the following:

$$s_4 \leftarrow f\left(t_k + \tfrac{8}{9}h, y_k + \tfrac{8}{9}h\frac{4843}{1458}s_0 - \tfrac{8}{9}h\frac{19020}{1458}s_1 + \tfrac{8}{9}h\frac{16112 - 477s_3}{1458}s_2 - \tfrac{8}{9}h\frac{477}{1458}s_3\right)$$

  – Issue: if h is small, this may result in a sum of denormalized numbers, which will magnify the error

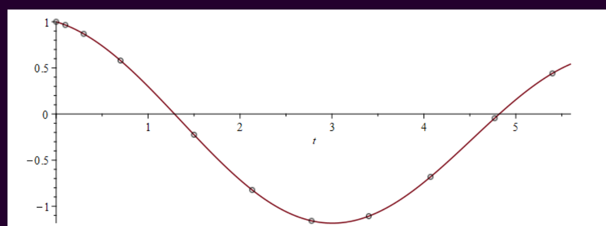
10

10

Example

- Suppose we have $y^{(1)}(t) = -0.2y(t) - \sin(t) - 0.1$

  $y(0) = 1$
  - With $h_{\text{min}} = 0.01$, $h_{\text{max}} = 1$ and $\varepsilon_{\text{abs}} = 0.00001$, we have



  - Now, the maximum h_{max} is more relevant, as we are using cubic splines to approximate values between these approximations

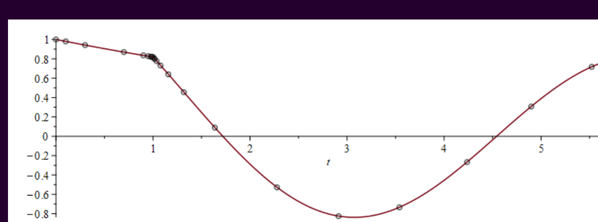
11

11

Example

- Suppose we have $y^{(1)}(t) = -0.2y(t) - \begin{cases} 0 & t < 1 \\ \sin(t) - 0.1 & t \geq 1 \end{cases}$

  $y(0) = 1$
  - With $h_{\text{min}} = 0.01$, $h_{\text{max}} = 1$ and $\varepsilon_{\text{abs}} = 0.00001$, we have



12

12

Summary

- Following this topic, you now
  - Understand the adaptive Dormand-Prince method
  - Are aware of the calculations required
  - Know the derivation of the appropriate scaling factor *a*
  - Have seen the implementation
  - Have seen two examples

13

13

References

[1] https://en.wikipedia.org/wiki/Dormand-Prince_method

[3] https://en.wikipedia.org/wiki/Adaptive_algorithm

[4] https://en.wikipedia.org/wiki/Adaptive_step_size

14

14

Acknowledgments

Tazik Shahjahan for pointing out typos.

Aristedes Jose B. Aquino Jr. from YouTube who pointed out that one of the coefficients on Slide 4 had

$$\cdots - 64000 s_2 + \cdots \text{ and not } \cdots + 64000 s_2 + \cdots$$

– The subsequent MATLAB code has the correct sign

15

15

Colophon

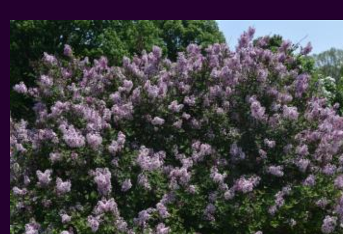
These slides were prepared using the Cambria typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.
Mathematical equations are prepared in MathType by Design Science, Inc.

Examples may be formulated and checked using Maple by Maplesoft, Inc.

The photographs of flowers and a monarch butter appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens in October of 2017 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.



16

16

Disclaimer

These slides are provided for the ECE 204 *Numerical methods* course taught at the University of Waterloo. The material in it reflects the author's best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.

17