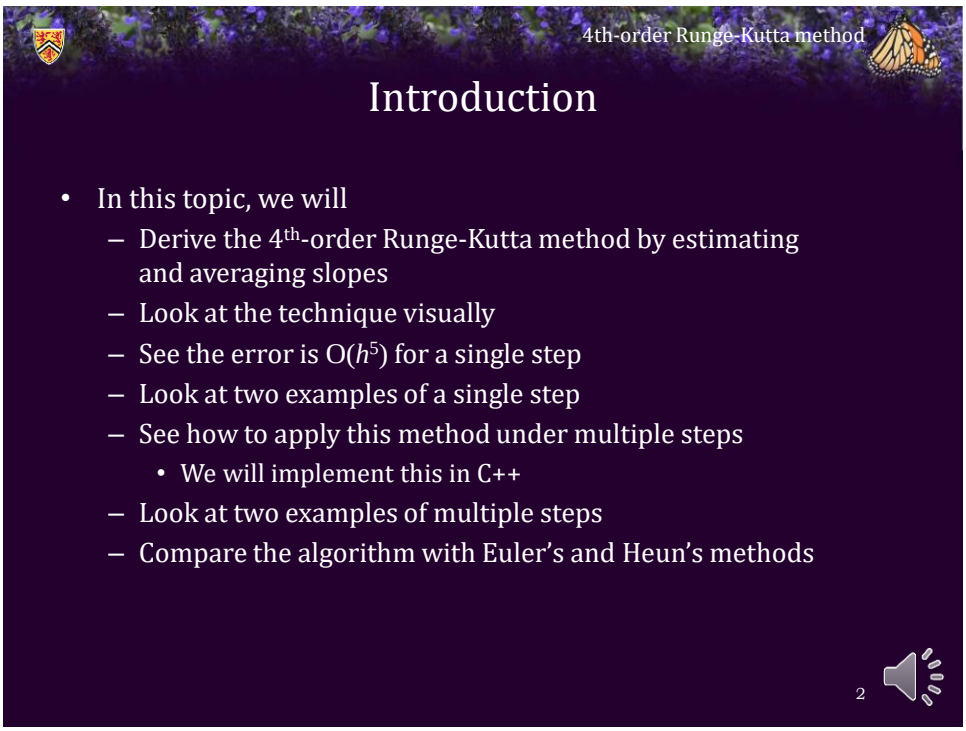

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering
ECE 204 Numerical methods
4th-order Runge-Kutta method
Douglas Wilhelm Harder, LEL, M.Math.
dwharder@uwaterloo.ca
dwharder@gmail.com
BY NC SA

1


4th-order Runge-Kutta method
Introduction
• In this topic, we will
– Derive the 4th-order Runge-Kutta method by estimating and averaging slopes
– Look at the technique visually
– See the error is O(h5) for a single step
– Look at two examples of a single step
– See how to apply this method under multiple steps
• We will implement this in C++
– Look at two examples of multiple steps
– Compare the algorithm with Euler's and Heun's methods
2

2

Simpson's rule

- A simple Reimann sum approximates an integral with one value:

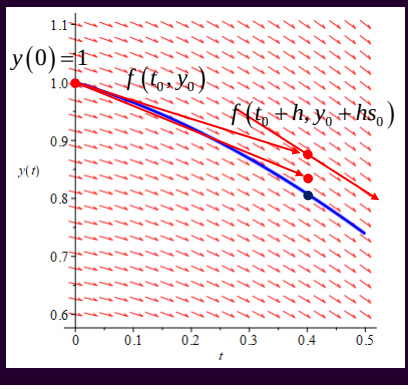
$$\int_a^b f(x)\,dx \approx f(a)(b-a)$$

- The value of the function, however, changes across [*a*, *b*], so Simpson's rule integrates an interpolating quadratic:

$$\int_a^b f(x)\,dx \approx \frac{f(a)+4f\left(\frac{a+b}{2}\right)+f(b)}{6}(b-a)$$

4th-order Runge-Kutta method

- Heun's method uses two slopes
  - We'd like to take more samples of the slope…



$$y^{(1)}(t) = -0.2y(t) - \sin(t) - 0.1$$
$$y(0) = 1$$

4th-order Runge-Kutta method

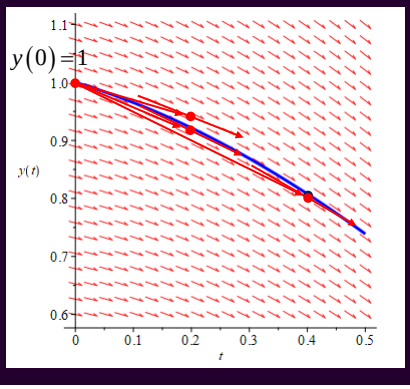
- Without justification,
  4th-order Runge-Kutta says to proceed as follows:

$$s_0 \leftarrow f\left(t_k, y_k\right)$$
$$s_1 \leftarrow f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}hs_0\right)$$
$$s_2 \leftarrow f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}hs_1\right)$$
$$s_3 \leftarrow f\left(t_k + h, y_k + hs_2\right)$$
$$y_{k+1} \leftarrow y_k + h\frac{s_0 + 2s_1 + 2s_2 + s_3}{6}$$

5

4th-order Runge-Kutta method

- Visually, we proceed as follows



$$y^{(1)}(t) = -0.2y(t) - \sin(t) - 0.1$$
$$y(0) = 1$$

6

One step of the 4th-order Runge-Kutta method

- What is the error? O(h^4) or O(h^5) or better?
  - We will look at two initial-value problems and approximate $y(t_0 + h)$ for successively smaller values of h
  - For example, approximate $y(0.4)$ with:

$$y^{(1)}(t) = -y(t) \qquad y(0) = 1$$

$$s_0 \leftarrow f(0, 1) = -1$$
$$s_1 \leftarrow f(0.2, 1 + 0.2s_0) = -0.8$$
$$s_2 \leftarrow f(0.2, 1 + 0.2s_1) = -0.84$$
$$s_3 \leftarrow f(0.4, 1 + 0.4s_2) = -0.664$$

$$e^{-0.4} = 0.6703200460356393$$

$$y_1 \leftarrow y_0 + h\frac{s_0 + 2s_1 + 2s_2 + s_3}{6}$$
$$= 1 + 0.4\frac{(-1) + 2(-0.8) + 2(-0.84) + (-0.664)}{6} = 1 + 0.4(-0.824)$$
$$= 0.6704$$

One step of the 4th-order Runge-Kutta method

- Let's approximate the solution at $y(0 + h)$ to

$$y^{(1)}(t) = -y(t)$$
$$y(0) = 1$$

n	$h = 2^{-n}$	Exact	Approximation	Error	Ratio
1	0.5	0.606530659712633	0.606770833333333	–0.0002402	
2	0.25	0.7788007830714049	0.7788085937500000	–0.000007811	0.03252
3	0.125	0.8824969025845955	0.8824971516927084	–0.0000002491	0.03189
4	0.0625	0.9394130628134758	0.9394130706787109	–0.000000007865	0.03157
5	0.03125	0.9692332344763441	0.9692332347234091	–0.0000000002471	0.03141
6	0.015625	0.9844964370054085	0.9844964370131493	–0.000000000007741	0.03133
7	0.0078125	0.9922179382602435	0.9922179382604858	–0.0000000000002423	0.03130
8	0.00390625	0.9961013694701175	0.9961013694701251	–0.000000000000007550	0.03116
9	0.001953125	0.9980487811074755	0.9980487811074757	–0.000000000000002220	0.02941
10	0.0009765625	0.9990239141819757	0.9990239141819757	0	0

One step of the 4th-order Runge-Kutta method

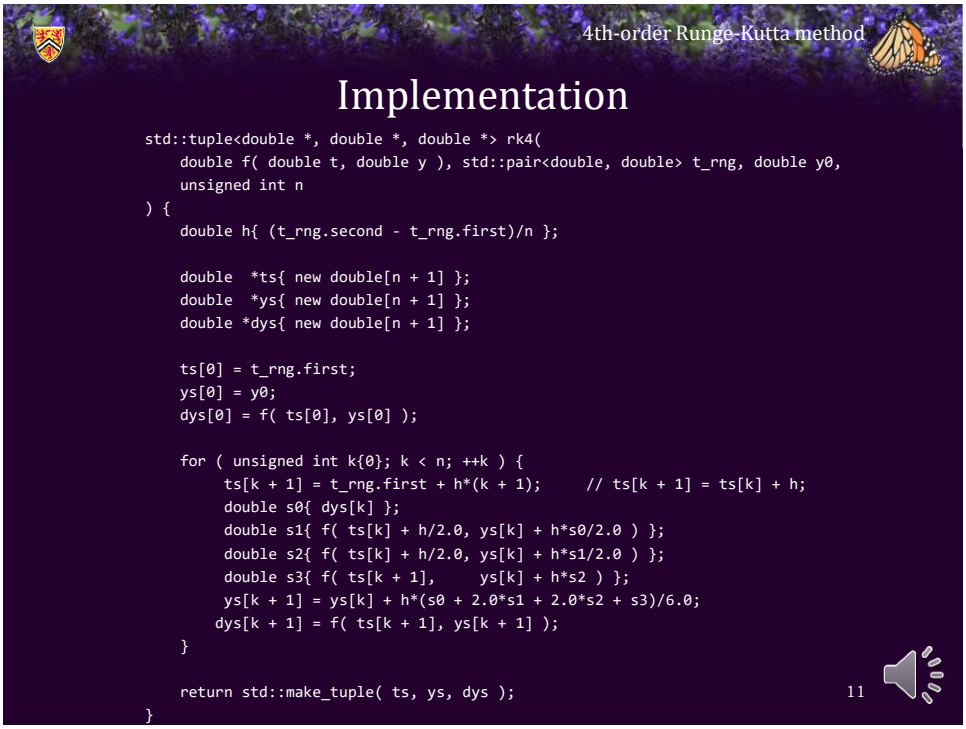
- Let's approximate the solution at $y(0 + h)$ to

$$y^{(1)}(t) = -0.2y(t) - \sin(t) - 0.1 \qquad y(t) = \frac{-13 + 25\cos(t) - 5\sin(t) + 14e^{-\frac{t}{5}}}{26}$$
$$y(0) = 1$$

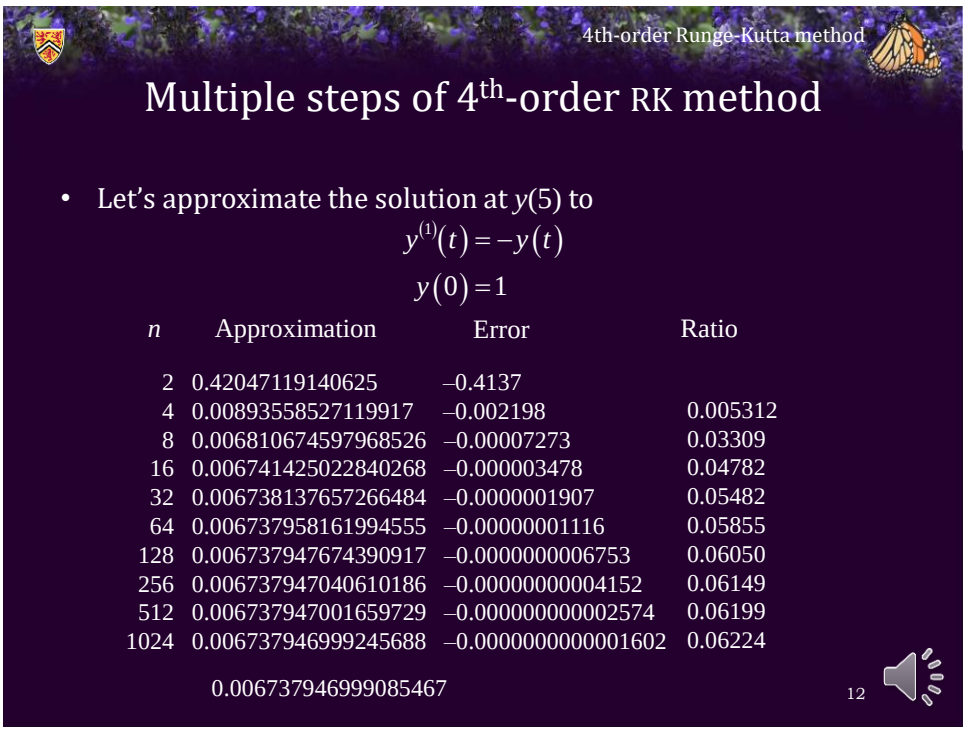
n	$h = 2^{-n}$	Exact	Approximation	Error	Ratio
1	0.5	0.738852315643913	0.738856449702695	–0.000004134	
2	0.25	0.8962693342116731	0.8962695046719316	–0.0000001705	0.04123
3	0.125	0.9552271839309132	0.9552271898849072	–0.000000005954	0.03493
4	0.0625	0.9794223225078814	0.9794223227035564	–0.0000000001957	0.03286
5	0.03125	0.9901670100357426	0.9901670100420059	–0.000000000006263	0.03201
6	0.015625	0.9951978758220640	0.9951978758222620	–0.0000000000001981	0.03162
7	0.0078125	0.9976275785667972	0.9976275785668035	–0.000000000000006328	0.03195
8	0.00390625	0.9988209552460877	0.9988209552460880	–0.0000000000000003331	0.05263
9	0.001953125	0.999412269826320	0.999412269826320	0	0

Multiple steps of Heun's method

- Can we see that the error is indeed $O(h^5)$
- As with Euler's method:
  - First, we will implement a function to find the n approximations by dividing a range $[t_0, t_f]$ into n sub-intervals
  - For two IVPs with the initial condition $y(0) = 1$, we will approximate $y(5)$ by using 2^n intervals


4th-order Runge-Kutta method
Implementation
std::tuple<double *, double *, double *> rk4(
double f(double t, double y), std::pair<double, double> t_rng, double y0,
unsigned int n
) {
double h{ (t_rng.second - t_rng.first)/n };
double *ts{ new double[n + 1] };
double *ys{ new double[n + 1] };
double *dys{ new double[n + 1] };
ts[0] = t_rng.first;
ys[0] = y0;
dys[0] = f(ts[0], ys[0]);
for (unsigned int k{0}; k < n; ++k) {
ts[k + 1] = t_rng.first + h*(k + 1); // ts[k + 1] = ts[k] + h;
double s0{ dys[k] };
double s1{ f(ts[k] + h/2.0, ys[k] + h*s0/2.0) };
double s2{ f(ts[k] + h/2.0, ys[k] + h*s1/2.0) };
double s3{ f(ts[k + 1], ys[k] + h*s2) };
ys[k + 1] = ys[k] + h*(s0 + 2.0*s1 + 2.0*s2 + s3)/6.0;
dys[k + 1] = f(ts[k + 1], ys[k + 1]);
}
return std::make_tuple(ts, ys, dys);
}
11

11


4th-order Runge-Kutta method
Multiple steps of 4th-order RK method
• Let's approximate the solution at y(5) to
$y^{(1)}(t) = -y(t)$
$y(0) = 1$
n Approximation Error Ratio
2 0.42047119140625 –0.4137
4 0.00893558527119917 –0.002198 0.005312
8 0.006810674597968526 –0.00007273 0.03309
16 0.006741425022840268 –0.000003478 0.04782
32 0.006738137657266484 –0.0000001907 0.05482
64 0.006737958161994555 –0.00000001116 0.05855
128 0.006737947674390917 –0.0000000006753 0.06050
256 0.006737947040610186 –0.00000000004152 0.06149
512 0.006737947001659729 –0.000000000002574 0.06199
1024 0.006737946999245688 –0.0000000000001602 0.06224
0.006737946999085467
12

12

Multiple steps of 4th-order RK method

- Let's approximate the solution at $y(5)$ to

$$y^{(1)}(t) = -0.2y(t) - \sin(t) - 0.1 \qquad y(t) = \frac{-13 + 25\cos(t) - 5\sin(t) + 14e^{-\frac{t}{5}}}{26}$$
$$y(0) = 1$$

n	Approximation	Error	Ratio
2	0.1469019038207984	0.008348	
4	0.1548307896015398	0.0004188	0.05016
8	0.1552239200410955	0.00002570	0.06119
16	0.1552479334528051	0.000001612	0.06291
32	0.1552494441496338	0.0000001015	0.06294
64	0.1552495392562453	0.000000006371	0.06278
128	0.1552495452276594	0.0000000003991	0.06265
256	0.1552495456018131	0.00000000002498	0.06258
512	0.1552495456252274	0.000000000001563	0.06257
1024	0.1552495456266942	0.00000000000009581	0.06131

0.1552495456267901

Comparison

- We could compare Euler, Heun and 4th-order Runge-Kutta for 1024 intervals
  - Issue: This isn't really fair, as
    - Euler's method uses one function evaluation per interval
    - Heun's uses two
    - 4th-order Runge-Kutta uses four
  - For 1024 function evaluations, how accurate is each?

Euler	1024	0.006655931188587414	0.00008202
Heun	512	0.006738486441915978	–0.0000005394
RK4	256	0.006737947040610186	–0.00000000004152
Euler	1024	0.152997481619969	0.002252
Heun	512	0.1552516585204115	–0.000002113
RK4	256	0.1552495456018131	0.00000000002498

Comparison

- Another question is:
  - How many function evaluations are required to get the same accuracy as Euler's method with 1024 intervals?

		$y(5) =$ 0.00682130435141457		
Euler	1024	0.006655931188587414	0.00008202	1024
Heun	43	0.006737946999085467	–0.00008336	86
RK4	8	0.006810674597968526	–0.00007273	32

		$y(5) =$ 0.1552495456267901		
Euler	1024	0.152997481619969	0.002252	1024
Heun	16	0.153866775462848	–0.002317	32
RK4	3	0.153866775462848	0.0013827	12

Summary

- Following this topic, you now
  - Understand the 4th-order Runge-Kutta method for approximating a solution to a 1st-order initial-value problem
  - Are aware of a visual interpretation with respect to slopes
  - Understand the error is $O(h^5)$ for a single step
  - Are aware that we must apply this technique multiple times to estimate the solution on a larger interval
  - Know that the error drops in this case to $O(h^4)$
  - Have seen a number of examples and an implementation
  - Understand how much better the algorithm is, even when we consider the number of function evaluations

References

[1] https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods

Acknowledgments

None so far.

Colophon

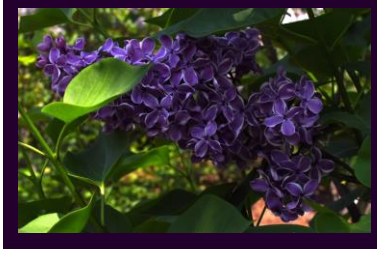
These slides were prepared using the Cambria typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`. Mathematical equations are prepared in MathType by Design Science, Inc.

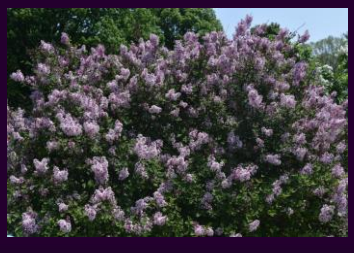
Examples may be formulated and checked using Maple by Maplesoft, Inc.

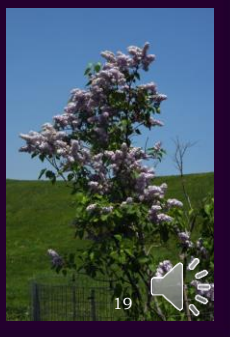
The photographs of flowers and a monarch butter appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens in October of 2017 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.







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.