

INTRODUCTION TO COMPUTER METHODS FOR O.D.E.

0. Introduction. The goal of this handout is to introduce some of the ideas behind the basic computer algorithms to approximate solutions to differential equations. The general setting is the initial-value problem:

$$y' = f(t, y), \quad y = y(t), \quad y(a) = y_0,$$

where f is a real-valued function (in the case of a single equation) or a vector field in \mathbb{R}^d defining a system of d equations (in general nonlinear.)

The main *assumption* is that the (exact) solution $y(t)$ is defined on an interval $[a, b]$ (which is true in general only if $b - a$ is sufficiently small). All numerical methods involve *discretizing* the time variable t : we fix an integer N and partition the interval into N pieces (of the same length, for simplicity). The positive number $h = (b - a)/N$ is the *stepsize*, and the numerical methods find numbers (or vectors) y_n ($n = 0, \dots, N$). These are meant to approximate the value of the solution at the points $t_n = a + nh$. We may then define an approximate solution $\tilde{y}(t)$ for all $t \in [a, b]$, for example by linear interpolation from the y_n . A sensible method has the property that $\tilde{y}(t) \rightarrow y(t)$ as $h \rightarrow 0$ (equivalently, as $N \rightarrow \infty$). However, different methods converge at different rates.

This set-up is reminiscent of the Riemann integral, and with good reason: a special case of the o.d.e. problem is integration:

$$y' = f(t), \quad y(a) = y_0 \Rightarrow y(t) = y_0 + \int_a^t f(s) ds.$$

The numerical methods described here (Euler's method, midpoint Euler and fourth order Runge-Kutta) are inspired by classical methods to approximate integrals: respectively, left-endpoint Riemann sums, midpoint Riemann sums and 'Simpson's rule'. (All are normally seen in a calculus course, and are reviewed below.) These three methods have different 'orders of convergence' p ($p = 1, 2, 4$ resp.), and in each case we explain the origin of these rates, for the integration problem (detailed proofs for the DE methods themselves are given in the second part, which is more advanced.)

There is an accompanying (primitive) MATLAB m-file called **error-analysis.m** It includes calls to the elementary ODE solvers eul.m (Euler's method), rk2.m (midpoint Euler) and rk4.m (fourth order RK). These must be downloaded separately, as they are too primitive to be part of MATLAB. In the examples below, when an exact solution can't be found by hand, a call

to the MATLAB solver ode45 stands for the ‘exact’ solution. Also included are basic loops to produce graphs of the maximum error (over $n = 0, \dots, N$) as a function of k ($N = 2^k$) and of the \log_2 of the ratio of maximum errors for consecutive values of k . Let $E(h)$ be this maximum error. A method has ‘order p ’ if $E(h) \leq Ch^p$, for N large enough (or h small enough), where $C > 0$ is a constant. In this case, for large N we in fact expect $E(h) \sim Ch^p$ (greater p means faster convergence.) So we expect:

$$\log_2\left(\frac{E(h)}{E(h/2)}\right) \sim \log_2\left(\frac{Ch^p}{C(h^p/2^p)}\right) = p.$$

The m-file includes a plot of this \log_2 ratio as a function of k , with the goal of experimentally verifying the order of the given method.

EXAMPLES (from [Hubbard-West, Chapter 3].)

Ex.1 $y' = y$, $t \in [0, 2]$, $y(0) = 1$.

Ex.2 $y' = y^2 \sin t$ $t \in [0, \pi]$, $y(0) = 0.3$.

Ex.3 $y' = y^2$, $t \in [0, 1]$, $y(0) = 2$.

Ex.4 $y' = \sin(ty)$, $t \in [0, 2]$, $y(0) = 3$.

Ex.5 $y' = y^2 - t$, $t \in [0, 1]$, $y(0) = 2$.

Note that the first three examples have easily computable exact solutions, while the last two can only be visualized by numerical approximation (for example, using the MATLAB solver ode45).

1. Euler’s method. Given $N \in \mathbb{N}$ and the stepsize $h = (b - a)/N$, define the approximations by the recursion: $t_0 = a$,

$$t_{n+1} = t_n + h, \quad y_{n+1} = y_n + hf(t_n, y_n), \quad n = 0, \dots, N - 1.$$

Geometrically this means we find (t_{n+1}, y_{n+1}) on the line from (t_n, y_n) with slope (or direction vector) $f(t_n, y_n)$. If f is independent of y , the exact solution is $y(t) = y_0 + \int_a^t f(s)ds$, and then the Euler method corresponds to approximating by left-endpoint Riemann sums:

$$y_{n+1} = y_n + hf(a + nh) \Rightarrow y_n = y_0 + h \sum_{j=0}^n f(a + jh).$$

The error at the n th. step is defined as $e_n = y(t_n) - y_n$. For the integral, since

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t)dt,$$

we have:

$$e_{n+1} = e_n + \int_{t_n}^{t_{n+1}} f(t)dt - hf(t_n) = e_n + \int_{t_n}^{t_{n+1}} (f(t) - f(t_n))dt.$$

The mean-value theorem implies that if $|f'| \leq L$ on $[a, b]$, we have: $|f(t) - f(s)| \leq L|t - s|$ for $s, t \in [a, b]$. Therefore:

$$|e_{n+1} - e_n| \leq \int_{t_n}^{t_{n+1}} |f(t) - f(t_n)|dt \leq L \int_{t_n}^{t_{n+1}} (t - t_n)dt = L \frac{h^2}{2}.$$

Since, for each n : $e_n = e_0 + (e_1 - e_0) + \dots + (e_n - e_{n-1})$, this implies:

$$|e_n| \leq n \frac{Lh^2}{2} \leq (Nh) \frac{L}{2} h = \frac{L}{2} (b - a)h.$$

We conclude that, when approximating integrals by left-endpoint Riemann sums ‘the error depends linearly on the step size’. This is true for the Euler approximation in the general case: we can always find a constant $C > 0$ independent of n so that $|e_n| \leq Ch$.

2. Midpoint Euler approximation. (Also known as ‘second order Runge-Kutta’.) The approximations y_n are defined by the recursion: $t_0 = a$,

$$t_{n+1} = t_n + h, \quad y_{n+1} = y_n + hf(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n, y_n)).$$

Geometrically this means we follow the line from (t_n, y_n) with slope $f(t_n, y_n)$ for time $h/2$, to find the new slope m ; then follow the line from (t_n, y_n) with slope m to find (t_{n+1}, y_{n+1}) . Turning to the special case of integration, this corresponds to approximating $\int_a^t f(s)ds$ by ‘midpoint Riemann sums’:

$$y_{n+1} = y_n + hf(t_n + \frac{h}{2}) \Rightarrow y_n = y_0 + h \sum_{j=0}^n f(a + jh + \frac{h}{2}).$$

Again, since $y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t)dt$, we have:

$$e_{n+1} = e_n + \int_{t_n}^{t_{n+1}} f(t)dt - hf(t_n + \frac{h}{2}) = e_n + \int_{t_n}^{t_{n+1}} (f(t) - f(t_n + \frac{h}{2}))dt.$$

From this point on we adopt the notation: $s_n = t_n + \frac{h}{2}$.

As the reader may recall from Calculus, the midpoint approximation converges faster: the error decays like h^2 (or $1/N^2$). We review the argument

for this (in the scalar case, $d = 1$), based on the ‘second-order mean value theorem’: for $t \in [t_n, t_{n+1}]$,

$$f(t) = f(s_n) + (t - s_n)f'(s_n) + \frac{1}{2}(t - s_n)^2 f''(\theta_t),$$

for some (unknown) number θ_t , depending on t , located somewhere in the interval bounded by t and s_n . Now, by a change of variable ($u = t - s_n$) we see that:

$$\int_{t_n}^{t_{n+1}} (t - s_n)f'(s_n)dt = f'(s_n) \int_{-h/2}^{h/2} udu = 0.$$

In addition, assuming f'' is continuous on $[a, b]$, we have $|f''(t)| \leq M$ on $[a, b]$ (for some constant $M > 0$). We conclude:

$$|e_{n+1} - e_n| = \frac{1}{2} \left| \int_{t_n}^{t_{n+1}} (t - s_n)^2 f''(\theta_t) dt \right| \leq \frac{M}{2} \int_{t_n}^{t_{n+1}} (t - s_n)^2 dt = \frac{Mh^3}{24}.$$

Adding these estimates (as above) for $j = 0, \dots, n$, we find for each n :

$$|e_n| \leq n \frac{Mh^3}{24} \leq M(Nh) \frac{h^2}{24} = M(b - a) \frac{h^2}{24}.$$

This shows ‘the error decays quadratically with the step size’ for the midpoint approximation of the integral. The same is true in general for the error in the midpoint Euler method: we may find $C > 0$ so that $|e_n| \leq Ch^2$, for each n .

3.Fourth order Runge-Kutta. The fourth order Runge-Kutta approximations y_n are defined by the recursion: $t_0 = a$,

$$t_{n+1} = t_n + h, \quad y_{n+1} = y_n + hm.$$

$$\begin{aligned} m &= \frac{1}{6}(m_1 + 2m_2 + 2m_3 + m_4) \\ m_1 &= f(t_n, y_n) \\ m_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}m_1) \\ m_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}m_2) \\ m_4 &= f(t_n + h, y_n + hm_3) \end{aligned}$$

That is, we advance from (t_n, y_n) to (t_{n+1}, y_{n+1}) along a line with slope m , obtained as a weighted average of four slopes: one defined over t_n , two defined over $s_n = t_n + h/2$, and one defined over t_{n+1} . In the special case

of integration, the two slopes over s_n coincide, and the method reduces to the following approximation of the integral on the interval $[t_n, t_{n+1}]$:

$$\int_{t_n}^{t_{n+1}} f(t) dt \sim \frac{h}{6}(f(t_n) + 4f(s_n) + f(t_{n+1})).$$

This is the well-known ‘Simpson rule’ from Calculus. We review the geometric meaning of the coefficients and the error estimate.

There is a unique quadratic function $q(t)$ whose graph includes the points $(t_n, f(t_n)), (s_n, f(s_n)), (t_{n+1}, f(t_{n+1}))$. If we write it in the form

$$q(t) = a(t - s_n)^2 + b(t - s_n) + c,$$

the coefficients are easily found:

$$c = f(s_n), \quad b = \frac{1}{h}(f(t_{n+1}) - f(t_n)), \quad a = \frac{2}{h^2}(f(t_n + h) - 2f(s_n) + f(t_n)).$$

Given these expressions, direct integration yields:

$$\int_{t_n}^{t_{n+1}} q(t) dt = \frac{h}{6}(f(t_n) + 4f(s_n) + f(t_{n+1})).$$

This means the ‘Simpson’s rule’ approximation is *exact* for quadratic functions (this is where the ‘weights’ (1,4,1) in the formula come from.)

The error estimate is obtained by an argument similar to the previous case, except now we consider a fourth-order mean-value theorem. For $t \in [t_n, t_{n+1}]$:

$$f(t) = T_3(t) + \frac{1}{24}(t - s_n)^4 f^{(4)}(\theta_t),$$

$$T_3(t) = f(s_n) + (t - s_n)f'(s_n) + \frac{1}{2}(t - s_n)^2 f''(s_n) + \frac{1}{6}(t - s_n)^3 f'''(s_n).$$

If we integrate T_3 over $[t_n, t_{n+1}]$, we find (by the same change of variable as in the midpoint case) the integrals of the linear and cubic terms vanish, so we have:

$$\int_{t_n}^{t_{n+1}} f(t) dt = hf(s_n) + \frac{h^3}{24}f''(s_n) + \frac{1}{24} \int_{t_n}^{t_{n+1}} (t - s_n)^4 f^{(4)}(\theta_t) dt.$$

In other words, if $|f^{(4)}(t)| \leq M$ for $t \in [a, b]$, and $I_n = \int_{t_n}^{t_{n+1}} f(t) dt$, we have:

$$|I_n - (hf(s_n) + \frac{h^3}{24}f''(s_n))| \leq \frac{M}{24} \int_{t_n}^{t_{n+1}} (t - s_n)^4 dt = \frac{M h^5}{24 \cdot 80}.$$

Consider, on the other hand, the Taylor approximations of the terms occurring in the Simpson approximation $S_n = (h/6)(f(t_n) + 4f(s_n) + f(t_n + h))$ of I_n :

$$f(t_n) = f(s_n) - \frac{h}{2}f'(s_n) + \frac{1}{2}\left(\frac{h}{2}\right)^2 f''(s_n) - \frac{1}{6}\left(\frac{h}{2}\right)^3 f^{(3)}(s_n) + O(h^4)$$

$$f(t_n + h) = f(s_n) + \frac{h}{2}f'(s_n) + \frac{1}{2}\left(\frac{h}{2}\right)^2 f''(s_n) + \frac{1}{6}\left(\frac{h}{2}\right)^3 f^{(3)}(s_n) + O(h^4).$$

The odd-degree terms cancel upon addition, and we are left with:

$$S_n = hf(s_n) + \frac{h^3}{24}f''(s_n) + O(h^5),$$

where it is easy to see (from the same mean-value theorem) that the $O(h^5)$ term is bounded above by $2Mh(h/2)^4/(6 \times 24)$. We conclude:

$$|I_n - S_n| \leq \left(\frac{M}{1152} + \frac{M}{1920}\right)h^5 := Ch^5.$$

Adding this over the first n intervals, we find:

$$\left| \int_a^{t_n} f(t)dt - \sum_{j=0}^n S_j \right| \leq (nh)Ch^4 \leq (Nh)Ch^4 = (b-a)Ch^4,$$

showing that the error in Simpson's rule approximation 'decays like the fourth power of the step-size.' This is also true for the general 4th order Runge-Kutta approximation: $|e_n| \leq Ch^4$ for some $C > 0$, but the proof (given in the next section) is more technical.

4. Computers misbehaving. (I) Try the following experiment: run *erroranalysis.m* with N large (say 10, or 14), for any of the examples. You will notice that the 'experimental order' p (the last graph in the output) does not stabilize to 4, but rather begins to 'wobble', and even becomes negative! This is a very bad sign-it essentially means that once the mesh is fine enough, further refinements do not decrease the error.

It gets worse- if you remove the semi-colon next to the definition of eRK4(k), and add the command 'format long', you will get the errors (using RK4) for successive subdivisions, with 16 significant figures. After going down for a long time, you will notice (perhaps with horror) that the errors start to go UP! This happens in all the examples, and happens sooner in the 'unsolvable' ones. It would also happen for Euler and RK2, but you would have to take many more subdivisions.

Computers are finite-precision machines- rather than the (t, y) plane, what we really have is a (t, y) ‘grid’, with a positive least spacing δ between grid points. The computer doesn’t really store the *exact* value of (t, y) computed by the algorithm, but rather the closest grid point. Once the step size h starts getting close to δ , this choice of grid point does not necessarily lead to a smaller error with each successive subdivision (relative to a fixed ‘exact’ solution). The take-home message is that *there is an optimal step-size, below which further subdivision makes things worse.* (For a detailed discussion, see [Hubbard-West], pp. 133-148.)

(II) Instability- for long time intervals, the computer solutions often behave unpredictably under small changes of the step size. To explore this, use the short m-file *instability.m*, a function with the call:

instability($f, [a, b], y_0, h$).

f is best given as an inline function; for example:

$f = \text{inline}('y - y.^2, 't', 'y')$

defines:

$$f(t, y) = y - y^2.$$

Try this one, with $y_0 = 1.6$ over the interval $[0, 100]$. Begin with a step size $h = 1.0$ and increase it gradually (use the Euler method). For step size close to 1, after an oscillation near $t = 0$ the solution quickly converges to $\bar{y} = 1$ for large t (the correct behavior). This continues as the step size is increased (say, at 0.05 intervals) until somewhere between $h = 1.6$ and $h = 1.7$ something bizarre happens (try it), and after that the Euler solution is totally useless! In fact, similar behavior would be seen for RK2 and RK4- edit *instability.m* and verify this. For RK2 the behavior is erratic above $h = 2.0$, for RK4 just above 1.8. The theoretical cutoff value is $h = 2$.

Other examples to try (from [Hubbard-West], Chapter 5):

$$y' = -ty, \quad y = y(t), \quad t \in [0, 10], \quad y(0) = 1.$$

Recall the solution is $y(t) = e^{-t^2/2}$, and converges rapidly to 0. Check what happens for step size close to 0.5, for all three methods. (For RK4: transition between 0.4 and 0.5, for Euler: between 0.5 and 0.6, for RK2: just above 0.3)

$$y' = y^2 - t, \quad t \in [0, 20], y(0) = 0.5.$$

(We used `dfield7` to pick a good initial condition) RK2: transition between $h = 0.2$ and $h = 0.3$; for RK4 or Euler: between 0.3 and 0.4;

The explanation of this phenomenon is mathematically more involved (see [Hubbard-West], section 5.4). Any numerical method replaces evolution in continuous time t by an evolution in ‘discrete time’ n , following a law that depends on the step size h (and on the numerical method used):

$$(t_{n+1}, y_{n+1}) = f_h(t_n, y_n)$$

The stability properties of this evolution (that is, whether solutions remain bounded, or oscillate, or converge as $n \rightarrow \infty$) depend on h . It is perfectly possible for the ‘discrete evolution’ to have stability properties similar to those of the exact solution $y(t)$ for h small (over a fixed time interval), but to ‘blow up’ (diverge rapidly) after h reaches a critical value h_0 . The message here is that a value of h that is adequate for a given time interval may be completely useless for a longer time interval.

PROBLEMS. (The first item in problems 1-4 does not require use of MATLAB- 1pt. per problem for this item. The other items are also each worth one point per problem)

1,2,3,4 For the following first-order equations (for $y = y(t)$), given intervals and initial conditions (note the first two can be solved ‘by hand’, but not the last two):

(i) For $h = 0.1$, compute (by calculator) the approximation at $t_1 = a + h$, for each of the methods (Euler, RK2, RK4).

(ii) Using `dfield7`, obtain a plot of the solution and two nearby solutions;

(iii) Use `dfield7` and `erroranalysis.m` to find the smallest value of h (with the method RK2) for which the maximal error is less than 1 percent of the maximum value of $|y(t)|$ (the `dfield7` solution) over the given interval.

(iv) Use `erroranalysis.m` to find the optimal value of the step size h , with the method RK4. (Functions suggested by [Hubbard-West].)

1. $y' = 1 + y, \quad t \in [0, 2], \quad y(0) = 3.$

2. $y' = \frac{y^2+1}{t^2+1}, \quad t \in [0, 1], \quad y(0) = 2.$

3. $y' = \sin(y^2 - t), \quad t \in [0, \pi], \quad y(0) = 0.5.$

4. $y' = y^2 - t, \quad t \in [0, 3], \quad y(0) = 0.$

5.,6. (2 pts each) For the last two examples (i) and (ii) given in (II) above, use `instability.m` to verify the assertions made about threshold values of h in the intervals given, and obtain more precise values. First use `dfield7` plots to get a general idea of the integral curves.

REFERENCES.

1. J.Polking, D. Arnold, *Ordinary Differential Equations using MATLAB*, 3r. Ed.- Prentice Hall, 2004
2. J.H.Hubbard, B.H.West, *Differential Equations; A Dynamical Systems Approach* vol.1, Springer-Verlag 1997 (Texts in Applied Mathematics no.5).
- 3.A. Iserles, *A First Course in the Numerical Analysis of Differential Equations* 1st. ed., Cambridge U.P. 1996.