3 Finite difference approximations for ODEs.

Let $y(t)$ be a solution of

$$\frac{dy(t)}{dt} = f(y(t), t). \tag{3.1}$$

Consider the solution at discrete values of the 'time' variable, $t = t_i,\ i = 1, 2, ...,$ taking for definiteness a uniform grid along the t-axis so that $t_i = t_0 + (i-1)h$. The step h is constant. We have $y(t_{i+1}) = y(t_i + h)$ and, assuming that $y(t)$ is sufficiently smooth and h is small,

$$y(t_{i+1}) = y(t_i) + h\frac{dy(t_i)}{dt} + O(h^2). \tag{3.2}$$

From (3.1) and (3.2) we then have

$$\frac{y(t_{i+1}) - y(t_i)}{h} = f(y(t_i), t_i) + O(h). \tag{3.3}$$

Now consider the difference equation,

$$\frac{y_{i+1} - y_i}{h} = f(y_i, t_i) \tag{3.4}$$

which, when solved, determines a sequence of numbers, $y_i, i = 1, 2, ...$. In general, $y_i \neq y(t_i)$ however the difference between the two is expected to be small when h is small.

Equation (3.4) is a finite-difference approximation of the differential equation (3.1). The error of approximation in the equation is of order $O(h)$ or first-order in powers of h. This error is also called a local truncation error because only local information about the functions involved is used.

Note that a local truncation error in the solution is of order $O(h^2)$.

Replacing ODE (3.1) with the finite-difference scheme (3.4) is called the Euler method. In the exercise below we find that a global truncation error (error in the solution on a finite interval in t) for the Euler method is $O(h)$ which is why the method is said to be first-order accurate.

The method is explicit. Given y_i, the value of y_{i+1} at the next step is computed immediately,

$$y_{i+1} = y_i + hf(y_i, t_i). \tag{3.5}$$

As soon as the value of y_{i+1} is determined we can use the same algorithm (3.5) to compute y_{i+2}, y_{i+3} and so on. As a result, given a starting value, y_0 from the initial condition supplied for the original differential equation, for instance $y(0) = y_0$, an approximate solution is computed on the grid t_i.

The approximation for the derivative in (3.4) is said to be forward; the derivative at the current t-value, t_i, is computed using the current and the next y-values, y_i and y_{i+1}.

Backward Euler method.

The equation (3.1) is again considered at time t_i but now for approximating the derivative we wish to use the current and previous points in the discrete time, $y(t_i)$ and $y(t_{i-1}) = y(t_i - h)$. We have

$$y(t_{i-1}) = y(t_i) - h\frac{dy(t_i)}{dt} + O(h). \tag{3.6}$$

The approximation for the derivative,

$$\frac{dy(t_i)}{dt} = \frac{y(t_i) - y(t_{i-1})}{h} + O(h), \tag{3.7}$$

is said to be backward. We arrive then at the backward Euler algorithm in the form,

$$\frac{y_i - y_{i-1}}{h} = f(y_i, t_i), \tag{3.8}$$

or

$$y_i - hf(y_i, t_i) = y_{i-1}. \tag{3.9}$$

The last formula may not seem useful for the method is implicit - there is no immediate calculation of y_i given the solution at the previous time step. However this algorithm has its advantages as we learn later. Backward Euler method is first order accurate.

Central approximation of the derivative. Leap-frog method.

Let us see what happens if we combine forward and backward approximations. We have, including second-order terms explicitly,

$$y(t_{i+1}) = y(t_i + h) = y(t_i) + h\frac{dy(t_i)}{dt} + \frac{h^2}{2}\frac{d^2y(t_i)}{dt^2} + O(h^3), \tag{3.10}$$

$$y(t_{i-1}) = y(t_i - h) = y(t_i) - h\frac{dy(t_i)}{dt} + \frac{h^2}{2}\frac{d^2y(t_i)}{dt^2} + O(h^3), \tag{3.11}$$

therefore

$$\frac{dy(t_i)}{dt} = \frac{y(t_{i+1}) - y(t_{i-1})}{2h} + O(h^2). \tag{3.12}$$

Using this approximation for the derivative in (3.1) we have a second-order finite difference algorithm with a central approximation of the derivative,

$$y_{i+1} = y_{i-1} + 2hf(y_i, t_i). \tag{3.13}$$

This is known as a leap-frog method. It is explicit. What is new here compared to the forward and backward Euler algorithm is that this is a multi-step algorithm.

Exercise. Design a second-order accurate procedure to solve the equation (**??**) for $t > 0$ with the initial condition, $y(0) = y_0$, using the leap-frog method.

The trick here is that the algorithm (3.13) requires two previous values, y_{i-1} and y_i, in order to compute y_{i+1}. However we only have one initial condition and therefore only one starting point at the beginning of calculation, y_0. To start the leap-frog algorithm, we need to compute y_1.

Exercise. Second-order, forward, approximation of the first-order ODE.
We shall simplify the task a little just to make the notation simpler. At $t = 0$, write down a second-order forward approximation of the equation (3.1) using three points, $t_0 = 0, t_1 = h$ and $t_2 = 2h$. This will give us a formula involving y_0, y_1 and y_2.

To compute a second-order forward approximation to the first derivative dy/dt we use Taylor expansions as follows.

$$y(h) = y(0) + h\frac{dy(0)}{dt} + \frac{h^2}{2}\frac{d^2y(0)}{dt^2} + O(h^3), \tag{3.14}$$

$$y(2h) = y(0) + 2h\frac{dy(0)}{dt} + \frac{(2h)^2}{2}\frac{d^2y(0)}{dt^2} + O(h^3), \tag{3.15}$$

and, eliminating the second derivative, $d^2y(0)/dt^2$, we have

$$\frac{dy(0)}{dt} = \frac{-3y(0) + 4y(h) - y(2h)}{2h} + O(h^2). \tag{3.16}$$

After truncating and using the original ODE (3.1) we have

$$-3y_0 + 4y_1 - y_2 = 2hf(y_0, t_0). \tag{3.17}$$

As an extra exercise, you can now generalize the last formula for any point on the grid t_i.

Global truncation error (GTE) for the Euler method.
We will show that the forward Euler's method has GTE of $O(h)$. Suppose the equation (3.1) is solved with the initial condition, $y(t_0) = y_0$, in the range $t_0 \le t \le t_n$ on a grid $t_i = t_0 + ih$, $0 \le i \le n$. The forward Euler algorithm is used as given in (3.5)

We start with the Taylor expansion for the exact solution of the ODE,

$$y(t_{i+1}) = y(t_i + h) = y(t_i) + h\frac{dy(t_i)}{dt} + \frac{h^2}{2}\frac{d^2y(\xi)}{dt^2}, \tag{3.18}$$

where $t_i \le \xi \le t_{i+1}$. Let e_i be the error, i.e. the difference between the finite-difference and exact solutions, at time t_i,

$$e_i = y_i - y(t_i). \tag{3.19}$$

From (3.18), using $dy(t_i)/dt = f(y(t_i), t_i)$, $y_{i+1} = y_i + hf(y_i, t_i)$, we have

$$e_{i+1} = y_{i+1} - y(t_{i+1}) \tag{3.20}$$

$$= e_i + h[f(y_i, t_i) - f(y(t_i), t_i)] - \frac{h^2}{2}\frac{d^2y(\xi)}{dt^2}. \tag{3.21}$$

At the starting point, $e_0 = 0$.

The accuracy of the Euler method depends on the smoothness of the function $f(y, t)$ in the ODE. We shall assume that the right-hand side in the ODE satisfies a Lipschitz condition,

$$|f(y_1, t) - f(y_2, t)| \leq L|y_1 - y_2|, \tag{3.22}$$

with some constant L for the entire integration interval, $t \in [t_0, t_n]$, and $(y_1, y_2) \in (-\infty, \infty)$. Another assumption is that

$$\left|\frac{d^2y(t)}{dt^2}\right| \leq M, \tag{3.23}$$

for some constant M. Then, from the triangle inequality,

$$|e_{i+1}| \leq |e_i| + hL\,|y_i - y(t_i)| + \frac{Mh^2}{2} \tag{3.24}$$

$$= |e_i|\,(1 + hL) + \frac{Mh^2}{2}. \tag{3.25}$$

The maximum error occurs when the equality holds. We then have a geometric progression for the coefficients $|e_i|$. To remind, if

$$x_i = A + Bx_{i-1}, \tag{3.26}$$

then

$$x_n = A\frac{B^n - 1}{B - 1} + B^n x_0. \tag{3.27}$$

Applied to (3.25) this gives us

$$|e_n| = \frac{Mh}{2L}[(1 + hL)^n - 1] + e_0, \text{ with } e_0 = 0. \tag{3.28}$$

As

$$(1 + hL)^n \leq e^{nhL}, \tag{3.29}$$

for $hL \geq 0$, we now have

$$|e_n| \leq \frac{Mh}{2L}(e^{nhL} - 1). \tag{3.30}$$

This error estimate has a maximum at the last point of the domain in t and therefore gives us an estimate for the GTE. Noting that $h = (t_n - t_0)/n$, in summary we have

$$\text{GTE} = |e_n| \leq \frac{Mh}{2L}(e^{L(t_n - t_0)} - 1). \tag{3.31}$$

Hence the GTE is of order $O(h)$.

Discussion: the significance of the curvature, M, and the time range, $t_n - t_0$.

Discussion: truncation error vs round-off error - can the finite-difference solution be made arbitrarily accurate by taking progressively smaller steps h?

Discussion: The GTE estimate in (3.31) in practice can be a gross overestimation. Can you think of a function, $f(y,t)$, for which this estimate would be close to reality?

Note: the Euler method is an efficient tool in analysis of differential operators however, for practical computations, the first order accuracy is often not sufficient. Higher order methods are preferable.

Taylor series methods.

Although of little practical significance, Taylor series methods provide a convenient starting point for the Runge-Kutta methods. We consider the initial-value problem for the first-order ODE,

$$\frac{dy}{dt} = f(y(t),t),\ y(t_0) = y_0. \tag{3.32}$$

Assuming that the necessary partial derivatives exist, the pth order Taylor expansion for $y(t_i + h)$ is

$$y(t_i + h) = y(t_i) + h\frac{dy(t_i)}{dt} + \frac{h^2}{2}\frac{d^2y(t_i)}{dt^2} + ... \tag{3.33}$$

$$+\frac{h^p}{p!}\frac{d^py(t_i)}{dt^p} + \frac{h^{p+1}}{(p+1)!}\frac{d^{p+1}y(\xi)}{dt^{p+1}}, \tag{3.34}$$

where $t_i \le \xi \le t_i + h$. Now, from (3.32),

$$y'(t_i) = f(y(t_i),t_i), \tag{3.35}$$

$$y''(t_i) = f'(y(t_i),t_i), \tag{3.36}$$

$$y'''(t_i) = f''(y(t_i),t_i), \tag{3.37}$$

$$... \tag{3.38}$$

$$y^{(p)}(t_i) = f^{(p-1)}(y(t_i),t_i), \tag{3.39}$$

where we use the dash to denote a total derivative with respect to t, for example

$$f'(y(t),t) = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{dy}{dt} = \frac{\partial f}{\partial t} + f\frac{\partial f}{\partial y}. \tag{3.40}$$

By truncating the Taylor expansion (3.34) and replacing the exact values, $y(t_i)$, with their approximations, y_i, the following numerical scheme is obtained,

$$y_{i+1} = y_i + hf(y_i,t_i) + \frac{h^2}{2}f'(y_i,t_i) + ... + \frac{h^p}{p!}f^{(p-1)}(y_i,t_i). \tag{3.41}$$

The LTE in (3.41) is $O(h^{p+1})$ and the GTE is $O(h^p)$. Note that with $p = 1$ this approximation reduces to the forward Euler method. With $p = 2$ we have, writing out the time derivative f' explicitly,

$$y_{i+1} = y_i + hf(y_i, t_i) + \frac{h^2}{2}\left[\frac{\partial f}{\partial t} + f\frac{\partial f}{\partial y}\right]_{t=t_i}. \tag{3.42}$$

The main drawback of the Taylor method when used for high values of p is that at each time step many partial derivatives need to be computed. This is usually a computationally intensive task.

Runge-Kutta methods.

The second-order Taylor algorithm (3.42) is

$$y_{i+1} = y_i + hf(y_i, t_i) + \frac{h^2}{2}\left.\frac{\partial f}{\partial t}\right|_{t_i} + \frac{h^2}{2}\left.\left(f\frac{\partial f}{\partial y}\right)\right|_{t_i}. \tag{3.43}$$

The aim is to construct an explicit algorithm with the same LTE as in (3.43) but not involving computation of the derivatives $\partial f/\partial t, \partial f/\partial y$. We are looking for an approximation of the form,

$$y_{i+1} = y_i + h[\omega_1 f(y_i, t_i) + \omega_2 f(y_i + Y, t_i + T)]. \tag{3.44}$$

In essence, by computing the function f at two neighbouring points (we assume Y and T to be small parameters) we aim to reproduce the linear and quadratic terms in h. Specifically, we take

$$Y = \alpha h f_i = \alpha h f(y_i, t_i), \tag{3.45}$$

$$T = \lambda h, \tag{3.46}$$

with constant α and λ. We have used a shorthand, f_i for $f(y_i, t_i)$.

Substituting into (3.44) and expanding for small h,

$$y_{i+1} = y_i + h(\omega_1 + \omega_2)f_i + h^2\omega_2\left[\alpha f_i \left.\frac{\partial f}{\partial y}\right|_{t_i} + \lambda \left.\frac{\partial f}{\partial t}\right|_{t_i}\right] + O(h^3). \tag{3.47}$$

The equations (3.43) and (3.47) are identical to $O(h^3)$ if

$$\omega_1 + \omega_2 = 1, \tag{3.48}$$

$$\alpha\omega_2 = \frac{1}{2}, \tag{3.49}$$

$$\lambda\omega_2 = \frac{1}{2}. \tag{3.50}$$

Since three equations are to be satisfied by the four parameters we have an infinite number of choices. Two popular choices are:

Heun's method

$$\omega_1 = \omega_2 = \frac{1}{2}, \qquad \alpha = \lambda = 1, \tag{3.51}$$

$$y_{i+1} = y_i + \frac{h}{2}\left[f(y_i, t_i) + f(y_i + hf(y_i, t_i), t_i + h)\right], \tag{3.52}$$

Modified Euler method

$$\omega_1 = 0, \qquad \omega_2 = 1, \qquad \alpha = \lambda = \frac{1}{2}, \tag{3.53}$$

$$y_{i+1} = y_i + hf(y_i + \frac{h}{2}f_i, t_i + \frac{h}{2}). \tag{3.54}$$

Second-order Runge-Kutta methods have LTE of $O(h^3)$ and GTE of $O(h^2)$.

RK4 - fourth-order algorithm.

This method of solving ODEs is well known for its speed and efficiency. It gives LTE of $O(h^5)$ and GTE of $O(h^4)$. In its classical implementation the algorithm reads:

$$k_1 = hf(y_i, t_i) \tag{3.55}$$

$$k_2 = hf(y_i + \frac{k_1}{2}, t_i + \frac{h}{2}) \tag{3.56}$$

$$k_3 = hf(y_i + \frac{k_2}{2}, t_i + \frac{h}{2}) \tag{3.57}$$

$$k_4 = hf(y_i + k_3, t_i + h) \tag{3.58}$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{3.59}$$

Linear difference equations.

Finite-difference methods often lead to replacing a differential equation with a difference equation. A linear difference equation of order N is of the form,

$$y_{n+N} + a_n^{(N-1)} y_{n+N-1} + a_n^{(N-2)} y_{n+N-2} + ... + a_n^{(0)} y_n = b_n. \tag{3.60}$$

Here are some examples:

$$y_{n+1} - y_n = 1, \text{ for } n \geq 0, \qquad \text{solution is } y_n = n + C; \tag{3.61}$$

$$y_{n+1} - y_n = n, \text{ for } n \geq 0, \text{ solution is } y_n = Cn(n-1)/2; \tag{3.62}$$

$$y_{n+1} - (n+1) y_n = 0, \text{ for } n \geq 0, \qquad \text{solution is } y_n = Cn!. \tag{3.63}$$

In the examples above C is an arbitrary constant.

If the coefficients in (3.60) are constant the equation can be written as

$$y_{n+N} + a_{N-1} y_{n+N-1} + a_{N-2} y_{n+N-2} + ... + a_0 y_n = b_n. \tag{3.64}$$

If, in addition, the equation is homogeneous ($b_n = 0$), a trial solution of the form $y_n = \beta^n$ is used. Substitution into (3.64) gives the characteristic equation,

$$P(\beta) = \beta^N + a_{N-1}\beta^{N-1} + a_{N-2}\beta^{N-2} + ... + a_0 = 0. \tag{3.65}$$

There are N roots of the characteristic equation and, similar to ordinary differential equations with constant coefficients, three options arise.

N **distinct real roots,** $\beta_1, \beta_2, ..., \beta_N$. By linearity, the general solution containing N arbitrary constants, $C_1, ..., C_N$ is of the form,

$$y_n = C_1\beta_1^n + C_2\beta_2^n + ... + C_N\beta_N^n. \tag{3.66}$$

Pairs of complex conjugate roots. Suppose there are two complex conjugate roots, $\beta_1$and β_2. In polar form, we write $\beta_1 = re^{i\theta}$ and $\beta_2 = re^{-i\theta}$. The contribution to the general solution due to these two roots is

$$y_n = A_1\beta_1^n + A_2\beta_2^n = r^n[C_1 \cos n\theta + C_2 \sin n\theta], \tag{3.67}$$

where $C_1$and C_2 are real valued.

Double roots. Let $P(\beta_1) = 0$ and $P'(\beta_1) = 0$. Then the contribution to the general solution from the double root is

$$y_n = C_1\beta_1^n + C_2 n\beta_1^n. \tag{3.68}$$

The arbitrary constants, $C_1, ..., C_N$, are determined by the initial (or boundary conditions). For a difference equation, specifying N initial conditions means giving explicit values to the first N terms in the sequence y_n, for example

$$y_0 = a_0, y_1 = a_1, ..., y_{N-1} = a_{N-1}, \tag{3.69}$$

where $a_i$are known.

Example 1. Solve

$$y_{n+3} - 2y_{n+2} - y_{n+1} + 2y_n = 0, \tag{3.70}$$

with the initial conditions, $y_0 = 0, y_1 = 1$ and $y_2 = 1$.

Substituting $y_n = \beta^n$ leads to

$$\beta^3 - 2\beta^2 - \beta + 2 = 0, \tag{3.71}$$

with roots $\beta_{1,2} = \pm 1$ and $\beta_3 = 2$. The general solution is then

$$y_n = C_1(+1)^n + C_2(-1)^n + C_3 2^n = C_1 + C_2(-1)^n + C_3 2^n. \tag{3.72}$$

Computing the constants from the initial condition gives the final solution,

$$y_n = -\frac{1}{3}(-1)^n + \frac{1}{3}2^n. \tag{3.73}$$

Example 2. Solve

$$y_{n+2} - 2y_{n+1} + 2y_n = 0. \tag{3.74}$$

The general solution is found to be

$$y_n = (\sqrt{2})^n \left(C_1 \cos \frac{n\pi}{4} + C_2 \sin \frac{n\pi}{4} \right). \tag{3.75}$$

Non-homogeneous equation,

$$y_{n+N} + a_{N-1}y_{n+N-1} + a_{N-2}y_{n+N-2} + ... + a_0 y_n = b_n, \tag{3.76}$$

with $b_n \neq 0$. Now we need to find a particular solution which agrees with the right-hand side in the equation, $y_n^{(P)}$, say, and add the general solution of the homogeneous equation. The form of the particular solution is often found by trial, for instance,

$$\text{if } b_n = n \text{ then try} \qquad y_n^{(P)} = An + B; \tag{3.77}$$

$$\text{if } b_n = n^2 \text{ then try} \qquad y_n^{(P)} = An^2 + Bn + C; \tag{3.78}$$

$$\text{if } b_n = \alpha^n \text{ then try} \qquad y_n^{(P)} = A\alpha^n. \tag{3.79}$$

In each case the constants, A, B, C, are supposed to be uniquely determined by the equation.

For practice - see past examination papers.

Discussion: is the particular solution unique?

Stability.

Consider the initial value problem for the 1st order ODE,

$$\frac{dy}{dt} = -30y, \qquad y(0) = \frac{1}{3}, \qquad 0 \le t \le 1.5. \tag{3.80}$$

The exact solution is

$$y(t) = \frac{1}{3}e^{-30t}. \tag{3.81}$$

At $t = 1.5$, the following results are obtained,

$$\text{Exact solution:} \qquad y(t = 1.5) = 9.54 \times 10^{-21}; \tag{3.82}$$

$$\text{Euler with } h = 0.1: \qquad y(t = 1.5) \approx -1.1 \times 10^{4}; \tag{3.83}$$

$$\text{RK4 with } h = 0.1: \qquad y(t = 1.5) \approx 3.96 \times 10^{1}. \tag{3.84}$$

Clearly, numerical solutions fail. The reason is that both the Euler and RK4 finite-difference schemes prove to be unstable in this case.

There are two sides to the issue of stability in using finite-difference approximations of differential equations. One is related to the behaviour of the solutions of the ODE itself. The other is to do with instability brought about due to finite-difference approximations.

The ODE, $dy/dt = f(y, t)$, is called stable if the Jacobian, $J = \partial f/\partial y$, is negative, $J < 0$.

As a rough justification, let $y = y^*(t)$ be a solution of our ODE. We consider small perturbations, $y = y^*(t) + u(t)$. Substituting into the ODE,

$$\frac{dy^*}{dt} + \frac{du}{dt} = f(y^* + u, t), \tag{3.85}$$

and then, expanding the right-hand side,

$$\frac{dy^*}{dt} + \frac{du}{dt} = f(y^*, t) + u\frac{\partial f(y^*, t)}{\partial y} + O(u^2). \tag{3.86}$$

To first order in u, the perturbation is then governed by

$$\frac{du}{dt} = uJ, \tag{3.87}$$

where $J = \partial f(y^*, t)/\partial y$ is the Jacobian. Locally, if y^* can be treated as approximately constant, the solution for the perturbation becomes

$$u = e^{Jt}, \tag{3.88}$$

which shows that if $J > 0$ then perturbations grow and the original solution, $y = y^*(t)$ is locally unstable. If $J < 0$ then perturbations decay. If J is complex valued then

the growth/decay is determined by the real part, ReJ. In the case $ReJ = 0$ with $ImJ \neq 0$ perturbations oscillate.

We thus have a classification of ODEs as locally stable or unstable. The instability of this type is local and tells us something about divergence or convergence of neighbouring solutions. Some authors prefer to avoid the term instability and classify the ODEs as growth or decay type (or oscillatory). Regardless of the terminology, in essence, an unstable or growth type problem is harder to solve numerically as local truncation errors tend to grow (exponentially!). The same is true for rounding errors.

Example 1. Consider

$$\frac{dy}{dt} = y(y-1), \qquad y(0) = 1. \tag{3.89}$$

The exact solution, $y(t) = 1$ is unstable because

$$J = \frac{\partial f}{\partial y} = 2y - 1 = 1 > 0. \tag{3.90}$$

Indeed, if we perturb the initial condition, $y(0) = 1+\epsilon$, perturbed solution trajectories diverge from the exact solution (they also diverge between themselves when we take several different values of ϵ. However for solutions with $y < 1/2$ the trajectories tend to converge to each other and solutions become locally stable.

Example 2. Consider

$$\frac{dy}{dt} = \alpha y, \qquad y(0) = y_0, \qquad \alpha \text{ and } y_0 \text{are constants.} \tag{3.91}$$

The exact solution is

$$y = y_0 e^{\alpha t} \tag{3.92}$$

which either grows exponentially ($\alpha > 0$) or decays ($\alpha < 0$). In the original example (3.80) we have $\alpha = -30$ so the equation is stable, $J = -30 < 0$. Let us see what went wrong in the numerical solution. The forward Euler method for the finite-difference solution on the grid $t_i = ih, i = 0, 1, 2, ...,$ reads

$$y_{i+1} = y_i + h\alpha y_i, \tag{3.93}$$

or $y_{i+1} = (1 + \alpha h)y_i$ with $y_0 = y(0)$ given. Hence

$$y_i = y_0(1 + \alpha h)^i. \tag{3.94}$$

Compare this with the exact solution computed on the grid t_i,

$$y(t_i) = y_0 e^{i\alpha h}. \tag{3.95}$$

There are two cases to consider. If $\alpha > 0$ then both solutions grow with i although the numerical solution (3.94) grows slower than the exact one in (3.95). Of course in the limit $h \to 0$ the finite-difference solution converges to the exact solution.

If $\alpha < 0$, the exact solution is a monotonically decaying curve. As long as $|1+\alpha h| < 1$ the same is true for the numerical solution (3.94). However if $|1+\alpha h| > 1$ the numerical solution will grow in magnitude as we march along the time axis, and also oscillate (since for $\alpha < 0$ the actual value of $1 + \alpha h$ will be negative. For our illustration we took $\alpha = -30$ and $h = 0.1$ giving $1 + \alpha h = -2$.

The phenomenon we observe here is what is known as numerical (computational) instability. It can be cured by taking a sufficiently small step. If we repeat the computations for (3.80) with $h = 0.001$ we have

$$\text{Exact solution:} \qquad y(t = 1.5) = 9.54 \times 10^{-21}; \tag{3.96}$$

$$\text{Euler:} \qquad y(t = 1.5) \approx 4.79 \times 10^{-21}; \tag{3.97}$$

$$\text{RK4:} \qquad y(t = 1.5) \approx 9.61 \times 10^{-21}. \tag{3.98}$$

Notice how well the 4th order Runge-Kutta algorithm performs compared to the Euler method.

The conclusion to draw from this exercise is that finite-difference approximations can bring in their own instability which is not associated with instability of the differential operator. In practical computations, numerical instability is often seen as vigorous oscillations from point to point along the computational grid (although instability may also be monotone).

Numerical methods which are stable irrespective of the step size are called unconditionally stable. Explicit methods are usually conditionally stable, i.e. stability is observed for sufficiently small step sizes.

Discussion. Unconditionally unstable methods exist. Are they useful or not?

General stability analysis.

Suppose an ODE or some other mathematical formulation is approximated as a finite-difference algorithm,

$$y_{n+1} = T(y_n, t_n). \tag{3.99}$$

When the solution y_n is perturbed by an amount ϵ_n the solution at the next grid point will respond with a perturbation ϵ_{n+1} so that

$$y_{n+1} + \epsilon_{n+1} = T(y_n + \epsilon_n, t_n). \tag{3.100}$$

After linearization for small ϵ_n and neglecting higher order terms,

$$y_{n+1} + \epsilon_{n+1} = T(y_n, t_n) + \epsilon_n \frac{\partial T(y_n, t_n)}{\partial y_n}, \tag{3.101}$$

or

$$\epsilon_{n+1} = g_n \epsilon_n, \tag{3.102}$$

where the growth factor, g_n, is given by

$$g_n = \frac{\partial T(y_n, t_n)}{\partial y_n}. \tag{3.103}$$

For stability of the numerical algorithm we require $|g_n| < 1$. In the case $|g_n| > 1$ we observe instability. The intermediate case, $|g_n| = 1$ may be interpreted as neutrally stable.

In a somewhat untidy notation, the stability condition for the algorithm (3.99) may be written as

$$\left| \frac{\partial y_{n+1}}{\partial y_n} \right| < 1. \tag{3.104}$$

Example. Euler's method,

$$y_{n+1} = y_n + hf(y_n, t_n). \tag{3.105}$$

Therefore $T(y_n, t_n) = y_n + hf(y_n, t_n)$ and

$$g_n = \frac{\partial T(y_n, t_n)}{\partial y_n} = 1 + h\frac{\partial f(y_n, t_n)}{\partial y_n}. \tag{3.106}$$

Depending on the form of $f(y_n, t_n)$ we have several cases.

(a) Decay-type problems with $\partial f(y, t)/\partial y < 0$,

$$g_n = 1 - h\left| \frac{\partial f}{\partial y_n} \right|, \tag{3.107}$$

and for stability

$$-1 < 1 - h\left| \frac{\partial f(y_n, t_n)}{\partial y_n} \right| < 1. \tag{3.108}$$

The constraint on the left gives

$$h < \frac{2}{\left| \frac{\partial f(y_n, t_n)}{\partial y_n} \right|}, \tag{3.109}$$

which means that stability requires sufficiently small time step. The constraint on the right simply states that $h > 0$.

(b) Growth problems, $\partial f(y, t)/\partial y > 0$. Here we have

$$g_n = 1 + h\frac{\partial f(y_n, t_n)}{\partial y_n} > 1, \tag{3.110}$$

therefore Euler's method is unconditionally unstable.

(c) The function f is complex valued. We shall consider a special case of an oscillatory problem with the ODE of the form,

$$\frac{dy}{dt} = -i\omega y. \tag{3.111}$$

[Note that i here is the imaginary unit, not the grid counter used before]. Then $g_n$is complex valued so we proceed as follows.

$$g_n = 1 - i\omega h, \tag{3.112}$$

$$|g_n|^2 = g_n g_n^* = 1 + h^2\omega^2 > 1, \tag{3.113}$$

hence instability for the oscillatory case.

Example. RK2 is given by

$$y_{n+1/2} = y_n + \frac{h}{2} f(y_n, t_n) \tag{3.114}$$

$$y_{n+1} = y_n + hf(y_{n+1/2}, t_{n+1/2}) \tag{3.115}$$

To calculate g_n,

$$g_n = \frac{\partial y_{n+1}}{\partial y_n} = 1 + h\frac{\partial f}{\partial y_{n+1/2}}\frac{\partial y_{n+1/2}}{\partial y_n} \tag{3.116}$$

$$= 1 + h\frac{\partial f(y_{n+1/2}, t_{n+1/2})}{\partial y_{n+1/2}}\left[1 + \frac{h}{2}\frac{\partial f(y_n, t_n)}{\partial y_n}\right]. \tag{3.117}$$

For small h we can take

$$\Delta = h\frac{\partial f(y_n, t_n)}{\partial y_n} \approx h\frac{\partial f(y_{n+1/2}, t_{n+1/2})}{\partial y_{n+1/2}}, \tag{3.118}$$

hence

$$g_n = 1 + \Delta + \frac{\Delta^2}{2}. \tag{3.119}$$

For stability, $|g_n| < 1$ if $-2 < \Delta < 0$. This shows that for growth problems the RK2 method is unconditionally unstable, whereas for decay problems the method is stable for a sufficiently small step size,

$$h < \frac{2}{\left|\frac{\partial f(y_n, t_n)}{\partial y_n}\right|}. \tag{3.120}$$

The notion of stability discussed above is sometimes termed absolute stability and the calculations of the region of stability the quantity $\Delta = h\frac{\partial f(y_n, t_n)}{\partial y_n}$ is allowed to take complex values. Then the region of absolute stability is identified in the complex plane Δ (or h). For example, for the Euler method the region of absolute

stability in the complex plane Δ is a disk of radius 1 centered at -1, as follows from the condition $|1 + \Delta| < 1$.

Example. Leap-frog method.

$$y_{n+1} = y_{n-1} + 2hf(y_n, t_n). \tag{3.121}$$

Differentiating with respect to y_n,

$$\frac{\partial y_{n+1}}{\partial y_n} = \frac{\partial y_{n-1}}{\partial y_n} + 2h\frac{\partial f(y_n, t_n)}{\partial y_n}. \tag{3.122}$$

Calculating the growth factor,

$$g_n = \frac{\partial y_{n+1}}{\partial y_n}, \tag{3.123}$$

we again use the approximation

$$\frac{\partial y_n}{\partial y_{n-1}} = g_n + ..., \tag{3.124}$$

for small h, hence

$$g_n^2 = 1 + 2\Delta g_n, \tag{3.125}$$

where $\Delta = h\partial f(y_n, t_n)/\partial y_n$. For real Δ, the roots of the quadratic are

$$g_n = \Delta \pm \sqrt{1 + \Delta^2}. \tag{3.126}$$

It follows that one of the roots always has $|g_n| > 1$ therefore the method is unstable.

For purely imaginary $\Delta = i\beta$,

$$g_n = i\beta \pm \sqrt{1 - \beta^2} \tag{3.127}$$

and we have neutral stability with $|g_n| = 1$ for

$$\beta = h\left|\frac{\partial f(y_n, t_n)}{\partial y_n}\right| < 1, \tag{3.128}$$

which can be viewed as a restriction on the time step, h.

Example. Backward Euler method. This is an implicit method given by

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1}). \tag{3.129}$$

For the growth factor we find,

$$g_n = 1 + g_n\Delta, \tag{3.130}$$

where

$$\Delta = h\frac{\partial f(y_{n+1}, t_{n+1})}{\partial y_{n+1}}. \tag{3.131}$$

Hence

$$g_n = \frac{1}{1-\Delta}, \tag{3.132}$$

and, for real Δ, the requirement $|g_n < 1|$ shows unconditional stability for decay-type problems with $\Delta < 0$. For growth-type problems we observe instability if $0 < \Delta < 2$ (the point $\Delta = 1$ is clearly singular). However the range $\Delta > 2$ is stable.

Exercise. Sketch the region of absolute stability of the backward Euler method in the complex plane Δ.

Exercise. The Crank-Nicolson (trapezoidal) algorithm is given by

$$y_{n+1} = y_n + \frac{h}{2}\left[f(y_n, t_n) + f(y_{n+1}, t_{n+1})\right]. \tag{3.133}$$

Verify that the algorithm gives a second-order accurate approximation of the differential equation

$$\frac{dy}{dt} = f(y, t). \tag{3.134}$$

Examine stability of the Crank-Nicolson method.

Hint. Show that

$$g_n = \frac{1+\Delta}{1-\Delta}, \tag{3.135}$$

where

$$\Delta = \frac{1}{2}h\frac{\partial f(y_n, t_n)}{\partial y_n}. \tag{3.136}$$

Predictor-corrector methods.

We shall illustrate the idea using the Crank-Nicolson algorithm (3.133). In general, we cannot find y_{n+1}immediately (unless the function f is very simple). We construct then an iterative procedure, starting with a predictor step based on an explicit method, for instance forward Euler:

$$y_{n+1}^{(0)} = y_n + hf(y_n, t_n). \tag{3.137}$$

The predictor is corrected using a formula derived from the implicit relation (3.133):

$$y_{n+1}^{(1)} = y_n + \frac{h}{2}\left[f(y_n, t_n) + f(y_{n+1}^{(0)}, t_{n+1})\right]. \tag{3.138}$$

It can be shown that the predictor-corrector procedure gives a second-order accurate algorithm overall. However if we wish we can continue iterations until solution of (3.133) is computed with any desired accuracy:

$$y_{n+1}^{(p+1)} = y_n + \frac{h}{2}\left[f(y_n, t_n) + f(y_{n+1}^{(p)}, t_{n+1})\right]. \tag{3.139}$$

The iterations terminate when $|y_{n+1}^{(p+1)} - y_{n+1}^{(p)}| < \epsilon$, where p is the number of iteration and ϵ is the desired precision.

Newton-Raphson method.

Instead of predictor-corrector type iterations, we can find solution from an implicit algorithm using Newton-Raphson method. In general, if we need to find a root of the equation,

$$F(x) = 0, \tag{3.140}$$

we need an approximate value x_0 as a starting point for calculations. Then the starting value is improved,

$$x_1 = x_0 + \Delta x, \tag{3.141}$$

with the aim to get

$$F(x_0 + \Delta x) = 0. \tag{3.142}$$

Assuming that Δx is small in magnitude, we have

$$F(x_0) + \Delta x F'(x_0) + ... = 0, \tag{3.143}$$

so that

$$\Delta x = -\frac{F(x_0)}{F'(x_0)}. \tag{3.144}$$

We then have the formula for the first iteration,

$$x_1 = x_0 - \frac{F(x_0)}{F'(x_0)}. \tag{3.145}$$

To continue the iterations, we use

$$x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)}. \tag{3.146}$$

Iterations terminate when $|x_{n+1} - x_n| < \epsilon$ for some sufficiently small tolerance ϵ.

Discussion. A higher order ODE can be written as a 1st order system of N equations,

$$\frac{d\mathbf{Y}}{dt} = \mathbf{G}(t, \mathbf{Y}), \tag{3.147}$$

where $\mathbf{Y}$ is a $N-$vector of the unknowns, $\mathbf{Y} = (y_1(t), ..., y_N(t))$, and $\mathbf{G} = (g_1(\mathbf{Y}, t), ..., g_N(\mathbf{Y}, t))$ is the vector of right-hand sides in the equation. By applying the techniques developed above we can generate a finite-difference scheme which approximates the system, for example,

$$\mathbf{Y}_{n+1} = \mathbf{Y}_n + h\mathbf{G}(t_n, \mathbf{Y}_n), \tag{3.148}$$

for a forward Euler approximation. How do we approach the question of stability for a system of finite-difference equations? Suggest your own criteria: (i) for the system of differential equations to be of decay type (you may have already seen similar things

in the Dynamical Systems course) and (ii) for the discrete version of the system to be stable. Hint: refer to the Jacobian and its eigenvalues.

Discussion. Convergence. Consistency and stability. Lax theorem.

RK4 for two dependent variables.

For reference, here we reproduce the RK4 algorithm for the case of two first-order equations of the form,

$$\frac{dy}{dt} = f(y, z, t) \tag{3.149}$$

$$\frac{dz}{dt} = g(y, z, t). \tag{3.150}$$

In the usual notation,

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{3.151}$$

$$z_{n+1} = z_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4) \tag{3.152}$$

where

$$k_1 = hf(y_n, z_n, t_n) \tag{3.153}$$

$$l_1 = hg(y_n, z_n, t_n) \tag{3.154}$$

$$k_2 = hf(y_n + k_1/2, z_n + l_1/2, t_n + h/2) \tag{3.155}$$

$$l_2 = hg(y_n + k_1/2, z_n + l_1/2, t_n + h/2) \tag{3.156}$$

$$k_3 = hf(y_n + k_2/2, z_n + l_2/2, t_n + h/2) \tag{3.157}$$

$$l_3 = hg(y_n + k_2/2, z_n + l_2/2, t_n + h/2) \tag{3.158}$$

$$k_4 = hf(y_n + k_3, z_n + l_3, t_n + h) \tag{3.159}$$

$$l_4 = hg(y_n + k_3, z_n + l_3, t_n + h) \tag{3.160}$$

Discussion. Stiff ODEs.

Discussion. Boundary-value problems for ODEs. Eigenvalue problems. Shooting methods.