Numerical Methods II, Courant Institute, Spring 2024
http://www.math.nyu.edu/faculty/goodman/teaching/NumericalMethodsII2024.html
Jonathan Goodman, February, 2024

ODE solvers, Runge Kutta methods

1 The problem

This is the first class on numerical methods for the *initial value problem* for ODE (ordinary differential equations). *Ordinary* means that the equation involves derivatives with respect to only one variable, which we call "time" and denote by t. The unknown is a d component vector that is a function of time, $x(t) \in \mathbb{R}^d$. The equation is *first order* if it involves first derivatives but not higher. The vector of first derivatives is written in various ways, including "x dot":

$$\frac{d}{dt}x(t) = \dot{x} = \begin{pmatrix} \dot{x}_1(t) \\ \vdots \\ \dot{x}_d(t) \end{pmatrix} \quad .$$

An *explicit* ODE system takes the form

$$\dot{x} = f(x) \; . \tag{1}$$

The function f has as d components, one for each component of x. Each component of f may depend on all components of x. Some examples are below.

In the *initial value problem*, we specify f and the value of x at some specific time. In mathematical discussions it is common to assume that that time is $t = 0$. We seek to compute $x(t)$, as a function of t. If we are given $x(0)$ and want $x(t)$ for $t > 0$, then $x(0)$ is the *initial value* (starting value), so finding $x(t)$ for $t > 0$ is the "initial value" problem.

1.1 Higher order equations

The *order* of an ODE is the degree of the highest derivative. A second order ODE involves second derivatives, as many problems that come from mechanical systems do. The *linear harmonic oscillator* is one example. We use $r(t)$ to represent the displacement of a mass from its neutral position, m for the mass, and k for the restoring force constant

$$m\frac{d^2r}{dt^2} = -kr \; . \tag{2}$$

One way to put this in the generic first order form (1) is to define variables

$$\begin{aligned} x_1(t) &= r(t) \\ x_2(t) &= \dot{r}(t) = \frac{dr}{dt} \; . \end{aligned}$$

With these definitions, one first order differential equation is just a statement of the definition of x_1 and x_2:

$$\dot{x}_1(t) = x_2(t) \; .$$

The other differential equation comes from $\dot{x}_2 = \ddot{x}_1 = \ddot{r}$. We can use the oscillator equation (2) to solve for $\ddot{r}$, which leads to

$$\dot{x}_2 = -\frac{k}{m} x_1 \; .$$

Both equations are linear, so they may be written as a single matrix/vector differential equation:

$$x(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} \quad , \quad \dot{x} = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{pmatrix} x \tag{3}$$

In the generic formulation (1), this oscillator has

$$f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix} \quad , \quad f_1(x) = x_2 \; , \; \; f_2(x) = -\frac{k}{m} x_1 \; .$$

Going further with this idea, suppose there are n "particles" (which can be electrons or stars, depending on the application) interacting with an inverse-square force law. Particle j has mass[1] m_j, and position $r_j \in \mathbb{R}^3$. The velocity of particle j is $\dot{r}_j \in \mathbb{R}^3$. All the position and velocity coordinates for all n particles make $d = 6n$ components of $x(t)$. The forces are a sum of an "external" force $F_e(r)$ and "interaction" forces for each pair $F_{jk}(r_j, r_k)$. The Newton equation of motion is

$$m_j \ddot{r}_j = F_e(r_j) + \sum_{k \neq j} C_{jk} \frac{r_j - r_k}{\left| r_k - r_j \right|^3} \tag{4}$$

The force is called "inverse-square" because its magnitude is proportional to the inverse of the square of the distance between particles:

$$|F_{jk}| = \left| \frac{r_j - r_k}{\left| r_k - r_j \right|^3} \right| = \frac{1}{\left| r_k - r_j \right|^2}$$

The force on particle j from particle k is in the direction of the line from particle k to particle j. It pulls particle j toward particle k in the planet application, so $C_{jk} > 1$. The electron application has a "repulsive" force, so $C_{jk} < 0$. When you code this, you need to calculate the sums (4). You also need to decide whether you put all the coordinate components first or all the coordinate and velocity components of each particle together. In the former case,

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = r_1 \; , \quad \begin{pmatrix} x_4 \\ x_5 \\ x_6 \end{pmatrix} = r_2 \; , \cdots , \quad \begin{pmatrix} x_{3n+1} \\ x_{3n+2} \\ x_{3n+3} \end{pmatrix} = \dot{r}_1 \; , \cdots \; .$$

[1]The parameters in actual applications have units and are not necessarily in the scale one to ten. The mass of an electron is $m \approx 9{\times}10^{-28}$ g. A sun sized star has $m \approx 2 \times 10^{33}$ g. Electrons all have the same mass while stars are all different.

In the other convention,

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = r_1 \; , \quad \begin{pmatrix} x_4 \\ x_5 \\ x_6 \end{pmatrix} = \dot{r}_1 \; , \quad \begin{pmatrix} x_7 \\ x_8 \\ x_9 \end{pmatrix} = r_2 \; , \cdots \; .$$

If you define F_j to be the sum on the right of (4) (the total force on particle j), then the right side of (1) could be

$$\begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} x_{3n+1} \\ x_{3n+2} \\ x_{3n+3} \end{pmatrix} \; , \quad \begin{pmatrix} x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} x_{3n+4} \\ x_{3n+5} \\ x_{3n+6} \end{pmatrix} \; , \cdots \; , \quad \begin{pmatrix} f_{3n+1} \\ f_{3n+2} \\ f_{3n+3} \end{pmatrix} = \frac{1}{m_1} F_1 \; , \cdots \; .$$

1.2 Time dependence

An ODE system can depend on time explicitly. In this case, the generic formulation (1) would change to

$$\dot{x} = f(x,t) \; .$$

Most ODE solvers designed for the time independent version (1) extend in a simple and natural way to the time dependent version.

Alternatively, the time dependent version can be reformulated in the time independent form by adding a component to x to represent time. If x has d components, you create a new vector with $d+1$ components with the understanding that the new one is time: $x_{d+1}(t) = t$. The new f just advances time at a linear rate:

$$\dot{x}_{d+1} = 1 \; .$$

With initial condition $x_{d+1}(0) = 0$, this makes $x_{d+1}(t) = t$. Thus, you can express the time dependent problem as

$$\frac{d}{dt} \begin{pmatrix} x_1 \\ \vdots \\ x_d \\ x_{d+1} \end{pmatrix} = \begin{pmatrix} f_1(x, x_{d+1}) \\ \vdots \\ f_d(x, x_{d+1}) \\ 1 \end{pmatrix} \; .$$

A common example is systems with specified external forcing (called "exogenous" in economics) such as the harmonically forced harmonic oscillator

$$m\ddot{r} = -kr + A\cos(\omega_0 t) \; .$$

The explicit time dependence could be more "structural". For example, consider an oscillator where the mass is decreasing with time (think of it as a melting ice cube attached to a spring, but there are less silly ways this can happen). You might get

$$\frac{m_0}{1+t^2}\ddot{r} = -kr \; .$$

1.3 Solution operator, flow map

The solution operator, often called *flow map* is the function defined by the ODE that takes as input the initial condition and gives as output the solution at time t. We call it $\Phi(x_0, t)$ and define it by saying that if x satisfies the ODE (1) then

$$x(t) = \Phi(x_0, t) \; . \tag{5}$$

We prefer "flow map" over "solution operator" because "operator" may suggest that Φ is a linear function of x_0, which it is not. It is common to say "map" for a function that takes a d component vector to another d component vector. You can wonder how this is related to the ordinary meaning of "map" as a diagram saying where things are.

The flow map is called the *fundamental solution* when the ODE is linear. In that case, there is a fundamental solution matrix $S(t)$ so that

$$\dot{x} = Ax \implies x(t) = S(t)x_0 \; . \tag{6}$$

The fundamental solution has initial condition $S(0) = I$ (why?) and satisfies the matrix differential equation

$$\frac{d}{dt}S(t) = AS(t) = S(t)A \; .$$

Note: matrices do not normally commute, but $SA = AS$, where there are several ways to see. Each column of $S(t)$ satisfies the vector ODE (6) (why?). You can compute column k by solving the vector ODE with initial condition $x_0 = e_k$, where e_k is the "standard basis vector" with all zero components except for a 1 in row k.

The fundamental solution may be thought of as the exponential of the matrix A. Just as the one component ODE $\dot{x} = ax$ has solution $x(t) = e^{at}x(0)$, the vector ODE (6) has solution

$$x(t) = S(t)x(0) = e^{tA}x_0 \; .$$

By analogy with the ordinary exponential, there is a power series formula for the matrix exponential

$$S(t) = e^{tA} = \sum_{n=0}^{\infty} \frac{t^n}{n!}A^n \; . \tag{7}$$

If you take the infinite series to be the definition of e^{tA}, you first check that the sum of matrices converges, then that it satisfies the matrix ODE with initial condition $e^{0A} = e^0 = I$ (the last is because all the terms in the sum are zero if $t = 0$ except for $n = 0$). For convergence, we write $M = \|A\|$ and use the inequality

$$\left\| \frac{t^n}{n!}A^n \right\| = \frac{t^n}{n!} \|A^n\| \leq \frac{t^n}{n!}M^n \; .$$

The matrix sum (7) converges because (look this up in a good calculus book) the corresponding scalar sum converges

$$e^{tM} = \sum_{n=0}^{\infty} \frac{t^n}{n!} M^n \ .$$

The sum satisfies the ODE because

$$\begin{aligned}
\frac{d}{dt} \sum_{n=0}^{\infty} \frac{t^n}{n!} A^n &= \sum_{n=0}^{\infty} \frac{\frac{d}{dt} t^n}{n!} A^n \\
&= \sum_{n=1}^{\infty} \frac{n t^{n-1}}{n!} A^n \\
&= \sum_{n=1}^{\infty} \frac{t^{n-1}}{(n-1)!} A^{n-1} A \\
&= \left(\sum_{n=1}^{\infty} \frac{t^{n-1}}{(n-1)!} A^{n-1} \right) A \\
\frac{d}{dt} e^{tA} &= e^{tA} A \ .
\end{aligned}$$

You get the version with Ae^{tA} by factoring A out on the other side, as $A^n = AA^{n-1}$. This is one of the ways to see that A commutes with $S(t)$.

Returning to the possibly nonlinear flow map, we use it for two purposes here. One involves power series approximations that apply for small t

$$\Phi(x,t) = I + t\Phi_1(x) + t^2 \Phi_2(x) + \cdots + t^n \Phi_n(x) + O(t^{n+1}) \ . \tag{8}$$

Each of the terms Φ_1, Φ_2, etc. has an expression in terms of $f(x)$ and its derivatives. The first few are simple but already Φ_4 is complicated. These expressions, and the calculations used to derive them, are the basis of *Runge Kutta* schemes for solving the ODE. You need to calculate up to Φ_n to derive a Runge-Kutta scheme of order n, which explains why those are hard to derive. Despite the cumbersome derivations, the methods themselves are not extremely complicated, and they can be extremely accurate.

Our second use of flow maps is *sensitivity analysis.* This tells you how sensitive the solution at time t is to small changes in the initial data. We use the following clumsy (apologies) notation. We fix a starting point x_0^* and its corresponding *trajectory*, $x^*(t)$. To say this in a different way, we look at $\Phi(x_0^* + y)$ for small y. The sensitivity matrix is the Jacobian matrix, in various notations

$$M(x_0^*, t) = \frac{\partial x(t)}{\partial x(0)} = \partial_{x_0} \Phi(x_0, t) \ .$$

Another notation for the Jacobian matrix of partial derivatives will prove helpful in what is coming below. Suppose $g(x) = g_1(x), \cdots, g_m(x))$ is m functions of d

variables. The Jacobian matrix may be denoted simply by dg. This is an $m{\times}d$ matrix with entries

$$(dg)_{jk} = \frac{\partial g_j}{\partial x_k} \; .$$

The first derivative Taylor approximation is

$$g(x+y) \approx g(x) + dg(x)\, y \; , \quad \text{if } y \text{ is small.}$$

A less informal statement is that if g is twice differentiable, then

$$\| g(x+y) - [\, g(x) + dg(x)\, y \,] \| \leq C \, \|y\| \; , \text{ if } \;\|y\|\; \text{ is small enough.}$$

In this notation, the sensitivity matrix is $M(x_0^*, t) = d\Phi(x_0^*, t)$.

This sensitivity may be found by linearizing the nonlinear dynamics (1) about the "baseline" trajectory $x^*(t)$. We subtract the baseline trajectory from both sides and use first order Taylor approximations to get

$$\begin{aligned}
\frac{d}{dt}\left(x(t) - x^*(t)\right) + \dot{x}^*(t) &= f(x^* + (x - x^*)) \\
&= f(x^*) + [\, f(x^* + (x - x^*)) - f(x^*) \,] \\
\frac{d}{dt}\left(x(t) - x^*(t)\right) &= [\, f(x^* + (x - x^*)) - f(x^*) \,] \\
\frac{d}{dt}\left(x - x^*\right) &\approx df(x^*)\, (x - x^*)
\end{aligned}$$

This suggests that the sensitivity matrix satisfies the *linearized*[2] equation

$$\dot{M}(x_0^*, t) = df(x^*(t))\, M(x_0^*, t) \; . \tag{9}$$

In this equation, the baseline trajectory $x_0^*(t)$ is already known. We could emphasize this by defining the linearized dynamics matrix

$$A(t) = df(x^*(t)) \; .$$

The sensitivity equation (9) is seen to be a linear ODE, but with time dependent coefficients

$$\dot{M} = A(t)\, M \; .$$

2 Time stepping, the setup

Many solution algorithms for the ODE initial value problem use *time stepping.* *Runge Kutta*[3] time stepping uses approximations to the flow map (5) that are

[2] *Linearization* means replacing a nonlinear equation with a linear equation that approximates it, generally using first derivative approximations.

[3] Karl Runge was *the* professor of applied mathematics at the university of Göttingen in the early $1900'^{s}$. He had several students who worked on time stepping methods of increasing complexity. Heun developed the second order method, and Kutta found the more general fourth order method described here. Runge is also know for his work on polynomial interpolation, including his discovery of the *Runge phenomenon.* In English, his name is often pronounced like "rung-ga".

accurate when t is small. We choose a small Δt and define discrete times

$$t_n = n\Delta t \ .$$

Computer solution is the sequence X_n, which approximates the true solution at the discrete times:

$$X_n \approx x(t_n) \ .$$

This approximation uses an approximate flow map $\Psi(x, \Delta t)$ in place of the exact one to "update" the solution from time t_n to time $t_{n+1} = t_n + \Delta t$.

$$x(t_{n+1}) = \Phi(\, x(t_n),\, \Delta t) \tag{10}$$

$$X_{n+1} = \Psi(\, X_n,\, \Delta t) \tag{11}$$

$$\Psi(x, \Delta t) = \Phi(x, \Delta t) + O(\Delta t^{p+1}) \ . \tag{12}$$

Both exact and approximate solutions use the exact initial data: $x(0) = x_0$, $X_0 = x_0$. Equation (10) says that the exact solution is updated using the exact flow map. This is the definition of the exact flow map. Equation (11) says that the numerical approximation solution is updated using the approximate flow map Ψ. In Runge Kutta methods, Ψ is computed using the f from the ODE (1). The number of *stages* in the Runge Kutta method is the number of times f is used in the definition of Ψ. This is described below. Equation (12) defines $p + 1$ as the *local order of accuracy* of the approximate flow map. We will see that the *global error* is order Δt^p. For this reason, p is the *order of accuracy* of the approximation. We will describe various approximate flow maps in Section 3.

The simplest method is the *forward Euler* method, which has $p = 1$. It's approximate flow map is

$$\Psi(x, \Delta t) = x + \Delta t f(x) \ . \tag{13}$$

The corresponding forward Euler *time stepping* method is

$$X_{n+1} = X_n + \Delta t f(X_n) \ . \tag{14}$$

We will first see that the forward Euler method satisfies the local error equation (12) with $p = 2$. Then we will define a version of "global error" and show that the global error is on the order of Δt. This will show that the order of accuracy is $p = 1$, which is usually called *first order* accuracy.

The local accuracy (11) follows from a Taylor approximation

$$x(\Delta t) = x_0 + \Delta t\, \dot{x}(0) + O(\Delta t^2) \ .$$

The ODE (1) gives the relation $\dot{x} = f(x)$, so

$$x(\Delta t) = x_0 + \Delta t\, f(x_0) + O(\Delta t^2) \ .$$

This implies that the forward Euler approximate flow map (13) satisfies the local error equation (11) with $p + 1 = 2$.

3 Runge Kutta methods

Runge Kutta methods are a family of approximate flow maps, mostly *higher order*, which means $p > 1$. As we have seen in earlier classes, higher order accuracy generally means getting the desired accuracy with less work. The higher order methods are more complicated to derive but not hard to code. Subsection 3.1 gives some motivation and introduces the Taylor series calculations involved. The following subsections extend these to third order so you can see how the derivations work, and possibly understand why many people find them distasteful.

Runge Kutta methods are *multi-stage* methods. Each *stage* requires evaluating $f(y)$ for some y. In most cases, evaluating f is more expensive than the other parts of the algorithm. This implies that the computer time per time step is proportional to the number of stages. A four stage method (common) is four times more work per time step than the forward Euler method. It is a "win" (less computer time for the ultimate answer) if you get the required accuracy with a four times larger Δt, which means four times fewer time steps.

3.1 A second order method

One such method is the *predictor/corrector trapezoid rule*. The trapezoid rule for integration is

$$\int_0^{\Delta t} y(t)\,dt \;\approx\; \frac{\Delta t}{2}\left(\, y(0) + y(\Delta t)\,\right) \;. \tag{15}$$

When working with ODE (as we will see a lot in when we do multi-step methods in Class 7) it can help to write the integral formulation of the ODE, which is

$$\begin{aligned} x(\Delta t) &= x(0) + \int_0^{\Delta t} \dot{x}(t)\,dt \\ &= x_0 + \int_0^{\Delta t} f(x(t))\,dt \;. \end{aligned} \tag{16}$$

The trapezoid rule approximation (15) applied to the integral formulation (16) gives the approximation

$$x(t+\Delta t) \approx x_0 + \frac{\Delta t}{2}\left[\, f(x_0) + f(x(\Delta t))\,\right] \;.$$

This leads to an ODE solver called the *trapezoid rule*

$$X_{n+1} = X_n + \frac{\Delta t}{2}\left[\, f(X_n) + f(X_{n+1}))\,\right] \;. \tag{17}$$

This method is *implicit* because it defines X_{n+1} implicitly through an equation but does not give an explicit formula for it. This structure is seen more clearly if you put all the X_{n+1} terms on the left side:

$$X_{n+1} - \frac{\Delta t}{2} f(X_{n+1}) = X_n + \frac{\Delta t}{2} f(X_n) \;. \tag{18}$$

To find X_{n+1}, you have to solve the equation

$$g(X_{n+1}) = a \ , \ \ g(y) = y - \frac{\Delta t}{2} f(y) \ , \ \ a = X_n + \frac{\Delta t}{2} f(X_n) \ .$$

The equation is trivial if $\Delta t = 0$ and easy to solve by iteration if Δt is small (g being a small perturbation of the identity function). Implicit methods are useful despite the extra complexity of equation solving at each time step, but we do not discuss them more until a future class.

Predictor/corrector methods are derived from implicit methods by using approximations to X_{n+1} on the left side of (18). In this case, we just use the forward Euler prediction

$$\widetilde{X}_{n+1} = X_n + \Delta t f(X_n) \ . \tag{19}$$

Then we use $\widetilde{X}_{n+1}$ in place of X_{n+1} on the right of (17), which gives

$$X_{n+1} = X_n + \frac{\Delta t}{2} \Big[f(X_n) + f(\widetilde{X}_{n+1})) \Big] \ . \tag{20}$$

This gives a *two stage* method, which means two stages per time step. In the first stage we use (19) to calculate $\widetilde{X}_{n+1}$. This requires one evaluation of f. In the second stage we use (20) to evaluate X_{n+1}. This requires another evaluation of f.

3.2 Explicit multi-stage methods

The predictor-corrector method of (19) and (20) is an example of a general explicit multi-stage method (also called Runge Kutta method). An explicit multi-stage method is an approximate flow map $\Psi(X, \Delta t)$ that is defined using a fixed number of evaluations of f and some linear vector operations. We present the general formalism here, then illustrate the calculations behind higher order methods in Subsections 3.3 and 3.4.

Each stage j of an explicit Runge Kutta method creates a new vector k_j, which is f evaluated at some point determined by the starting point X and the previous stage evaluations k_i. The two stage predictor-corrector trapezoid rule, in this notation, is

$$k_1 = f(X) \tag{21}$$

$$k_2 = f(X + \Delta t\, k_1) \tag{22}$$

$$\Psi(X) = X + \Delta t \left(\tfrac{1}{2} k_1 + \tfrac{1}{2} k_2 \right) \ . \tag{23}$$

The quantity k_1 is on the right side of (19). The argument of f in the second Runge Kutta state (22) is $\widetilde{X}_{n+1}$. A Runge Kutta method takes a step that is a linear combination of the stage values. In the example, the linear combination has weights $\frac{1}{2}$, which is corresponds to (20).

A general method of this type is defined by coefficients β_{ij} and α_j. Stage j computes k_j in terms of the previous stage values k_i using

$$k_j = f(X + \Delta t(\, \beta_{1j} k_1 + \cdots + \beta_{j-1,j} k_{j-1} \,)) \ . \tag{24}$$

In the example, (21) is the first stage ($j = 1$ with no $i < 1$ terms). The second stage corresponds to (22). It has $j = 2$ and only $i = 1$ on the right of (24). The step is a linear combination of the stages, with weights α_j:

$$\Psi(X) = X + \alpha_1 k_1 + \cdots + \alpha_n k_n \ . \tag{25}$$

There are implicit Runge Kutta methods. One kind of implicit method has implicit stages, where you have to solve equations to compute k_j once the k_i for $i < j$ are known. Such methods are called *DIRK*, for "diagonally implicit Runge Kutta". It is also possible to consider fully implicit Runge Kutta methods, in which each k_j is defined in terms of all the other k_i.

3.3 Two stage Runge Kutta methods

Explicit Runge Kutta methods may be found using Taylor series calculations related to the exact and approximate flow maps Φ and Ψ. The exact flow map (5) has small t (i.e., short time) Taylor approximations

$$x(t) = \Phi(x_0, t) = x_0 + t\dot{x}_0 + t^2 \tfrac{1}{2}\ddot{x}_0 + \cdots + t^p \frac{1}{p!}\left(\frac{d}{dt}\right)^p x(0) + O(t^{p+1}) \ . \tag{26}$$

Each of the derivatives on the right depends on f and derivatives of f. The approximate flow map also has Taylor approximations, with coefficients that we call y_j:

$$\Psi(x_0, t) = x_0 + t y_1 + t^2 y_2 + \cdots + t^p y_p + O(t^{p+1}) \ . \tag{27}$$

The method has order of accuracy p if the coefficients match up to order t^p. This means that for $k = 1, \cdots, p$, we arrange that

$$y_k = \frac{1}{k!}\left(\frac{d}{dt}\right)^k x(0) \ . \tag{28}$$

This turns out to be lots of algebra. Section 4 has some "review" of Taylor expansions, which can be hard to get right for higher order expansions with many variables.

The one stage forward Euler method is first order accurate because $\dot{x} = f(x)$. It is not second order accurate because forward Euler has $y_2 = 0$, which is not $\ddot{x}$ unless $x(t)$ is a linear function of t.

We achieve second order accuracy by finding a formula for $\ddot{x}$ and choosing a two stage method so that $y_1 = f(x)$ and $y_2 = \ddot{x}$, which involves some algebra. We can find a formula for $\ddot{x}$ by taking the time derivative of $\dot{x} = f(x)$ and using the chain rule.

$$\begin{aligned}
\ddot{x} &= \frac{d}{dt}\dot{x} \\
&= \frac{d}{dt} f(x(t)) \\
&= f'(x)\dot{x} \\
\ddot{x} &= f'(x) f(x) \ .
\end{aligned} \tag{29}$$

This expression involves the Jacobian matrix f', but this is only for the derivation. Runge Kutta methods themselves uses only f, not derivatives.

A two stage explicit Runge Kutta method has the form

$$k_1 = f(x) \tag{30}$$

$$k_2 = f(x + t\beta k_1) \tag{31}$$

$$\Psi(x,t) = x + t\,(\alpha_1 k_1 + \alpha_2 k_2) \tag{32}$$

We want a Taylor expansion for Ψ. The formula (30) is already a full Taylor expansion for k_1. An expansion for k_2 to the necessary order requires just the first order expansion of f:

$$\begin{aligned} k_2 &= f(x + t\beta k_1) \\ &= f(x) + f'(x)\,(t\beta k_1) + O\Big(\|t\beta k_1\|^2\Big) \\ &= f(x) + \beta t f'(x) f(x) + O\big(t^2\big) \end{aligned}$$

The expansion for Ψ comes from substituting this into the Ψ formula (32)

$$\begin{aligned} \Psi(x,t) &= x + t\big[\,\alpha_1 f(x) + \alpha_2\big(f(x) + \beta t f'(x) f(x) + O\big(t^2\big)\big)\,\big] \\ &= x + t\,(\,\alpha_1 f(x) + \alpha_2 f(x)\,) + t^2 \alpha_2 \beta f'(x) f(x) + O\big(t^3\big)\ . \end{aligned}$$

This gives the Taylor expansion (27) with

$$\begin{aligned} y_1 &= (\alpha_1 + \alpha_2)\, f(x) \\ y_2 &= \alpha_2 \beta f'(x) f(x)\ . \end{aligned}$$

Thus, the $k = 1$ accuracy condition (28) is satisfied if

$$\alpha_1 + \alpha_2 = 1\ . \tag{33}$$

Because of (29), second order accuracy condition ($k = 2$) is satisfied if

$$\alpha_2 \beta = \frac{1}{2}\ . \tag{34}$$

There are many ways to satisfy these two conditions. You can take $\alpha_1 = \alpha_2 = \frac{1}{2}$ and $\beta = 1$. This gives the predictor-corrector version of the trapezoid rule from Subsection 3.1. You can take $\alpha_1 = 0$, $\alpha_2 = 1$ and $\beta = \frac{1}{2}$. This is a predictor-corrector version of the *midpoint rule* of integration. The midpoint rule approximation of the integral on the right of (16) is

$$\int_0^t f(x(s))\,ds = t f(x(\tfrac{1}{2}t)) + O(t^3)\ .$$

We can use forward Euler to make the prediction

$$x(\tfrac{1}{2}t) \approx x(0) + \tfrac{1}{2} t f(x(0))\ .$$

This gives the two stage method

$$\begin{aligned} k_1 &= f(x) \\ k_2 &= f(x + \tfrac{1}{2} t k_1) \\ \Psi(x,t) &= x + t k_2 \ . \end{aligned}$$

This is the method with $\beta = \frac{1}{2}$, $\alpha_1 = 0$ and $\alpha_2 = 1$. A general two stage explicit Runge Kutta method has three free parameters (β, α_1, α_2). The method is second order accurate if the two constraints (33) and (34) are satisfied. With two equations and three unknowns, you expect that there is a one parameter family of solutions. That is the case here.

3.4 Three stage Runge Kutta methods

Please read Section (4) before going through the algebra here. This explains the *summation convention* that we use here. The summation convention saves some writing. More importantly, it makes the calculations clearer (in my opinion).

The general three stage explicit Runge Kutta method is

$$\begin{aligned} k_1 &= f(x) \\ k_2 &= f(x + t\beta_{11} k_1) \\ k_3 &= f(x + t(\beta_{21} k_1 + \beta_{22} k_2)) \\ \Psi(x,t) &= x + t\left(\alpha_1 k_1 + \alpha_2 k_2 + \alpha_3 k_3\right) \ . \end{aligned} \tag{35}$$

Keeping one more term in Taylor approximations will lead to one more order of accuracy, beyond second order. We expand k_2 to this order. We write f_i instead of $f_i(x)$, etc., to save space.

$$k_{2,i} = f_i(x) + t\beta_{11} \frac{\partial f_i}{\partial x_j} f_j(x) + t^2 \frac{1}{2} \beta_{11}^2 \frac{\partial^2 f_i}{\partial x_j \partial x_k} f_j f_k + O(t^3) \ . \tag{36}$$

Expanding k_3 in this way seems to give a giant mess of algebra, because it asks for the products of entries of k_2, which themselves involve products of entries of f. This would lead to products of four components of f and many indices.

Therefore, before substituting the k_2 expansion into the k_3 expansion, we figure out how accurate the k_2 approximations need to be in order for the final error term to be order t^3. We do not worry about k_1 because $k_1 = f$ is simple and exact. First, k_2 occurs in the formula for k_3 multiplied by t. Therefore, if the error in the k_2 is order t^2, the resulting k_3 error will be order t^3. To get this right, we define notation for the quantity in parentheses multiplying t:

$$l = \beta_{21} k_1 + \beta_{22} k_2 \ .$$

We use $k_1 = f$ and, for k_2, the approximation (36) keeping the order t term and taking the error term to be order t^2. This leads to approximations

$$l_i = \beta_{21} f_i + \beta_{22} f_i + t\beta_{22}\beta_{11} \frac{\partial f_i}{\partial x_j} f_j + O(t^2) \tag{37}$$

$$l_i = \beta_{21} f_i + \beta_{22} f_i + O(t) \ . \tag{38}$$

In terms of l, the Taylor approximation of k_3 is like the one for k_2:

$$k_{3,i} = f_i + t\frac{\partial f_i}{\partial x_j} l_j + t^2 \frac{1}{2}\frac{\partial^2 f_i}{\partial x_j \partial x_k} l_j l_k + O(t^3) \ .$$

We use the approximation (37) for the order t term and the simpler approximation (38) for the order t^2 term. This insures that the overall error will be order t^3. To avoid an index conflict (see the end of Section4) we re-write (37) as

$$l_j = (\beta_{21} + \beta_{22})\, f_j + t\beta_{22}\beta_{11}\frac{\partial f_j}{\partial x_k} x_k + O(t^2) \ .$$

The result is

$$\begin{aligned} k_{3,i} = f_i &+ t\,(\beta_{21}+\beta_{22})\,\frac{\partial f_i}{\partial x_j} f_j \\ &+ t^2\beta_{22}\beta_{11}\frac{\partial f_i}{\partial x_j}\frac{\partial f_j}{\partial x_k} f_k + t^2\frac{1}{2}\,(\beta_{21}+\beta_{22})^2\,\frac{\partial^2 f_i}{\partial x_j \partial x_k} f_j f_k \\ &+ O(t^3) \ . \end{aligned}$$

Finally, we use (35) and assemble all the terms just calculated. We organize the resulting terms by powers of t.

$$\begin{aligned} \Psi(x,t)_i = x_i &+ t\,(\alpha_1+\alpha_2+\alpha_3)\, f_i \\ &+ t^2\left[\alpha_2\beta_{11}\frac{\partial f_i}{\partial x_j} f_j + \alpha_3\,(\beta_{21}+\beta_{22})\,\frac{\partial f_i}{\partial x_j}\right] \\ &+ t^3\left[\alpha_2\frac{1}{2}\beta_{11}^2\frac{\partial^2 f_i}{\partial x_j\partial x_k} f_j f_k + \alpha_3\beta_{22}\beta_{11}\frac{\partial f_i}{\partial x_j}\frac{\partial f_j}{\partial x_k} f_k + \alpha_3\frac{1}{2}\,(\beta_{21}+\beta_{22})^2\,\frac{\partial^2 f_i}{\partial x_j\partial x_k} f_j f_k\right] \\ &+ O(t^4) \ . \end{aligned}$$

For the small t approximation to the exact flow map, we use (26), with the third derivative term given by (44) below.

$$\begin{aligned} \Phi(x,t)_i = x_i &+ t f_i + t^2\frac{1}{2}\frac{\partial f_i}{\partial x_j} f_j \\ &+ t^3\frac{1}{6}\left[\frac{\partial^2 f_i}{\partial x_j\partial x_k} f_j(x) f_k(x) \ + \ \frac{\partial f_i}{\partial x_j}\frac{\partial f_j}{\partial x_k} f_k(x)\right] \\ &+ O(t^4) \ . \end{aligned}$$

The method will be third order accurate if the corresponding terms in Ψ and Φ agree up to third order.

The order t agreement is

$$f_i = (\alpha_1+\alpha_2+\alpha_3)\, f_i \ .$$

This will hold no matter what f is provided the coefficients on the two sides agree. Therefore, the first accuracy condition is

$$1 = \alpha_1 + \alpha_2 + \alpha_3 \ . \qquad (39)$$

The order t^2 terms agree if

$$\frac{1}{2}\frac{\partial f_i}{\partial x_j} f_j = \left[\alpha_2\beta_{11} + \alpha_3\left(\beta_{21}+\beta_{22}\right)\right]\frac{\partial f_i}{\partial x_j} f_j \; .$$

The part that depends on f is the same on both sides. Therefore, we get second order accuracy if the coefficients satisfy

$$\frac{1}{2} = \alpha_2\beta_{11} + \alpha_3\left(\beta_{21}+\beta_{22}\right) \; . \tag{40}$$

These two conditions are simple extensions of the corresponding conditions for two stage methods, (33) and (34).

The order t^3 terms agree if

$$\frac{1}{6}\left[\frac{\partial^2 f_i}{\partial x_j \partial x_k} f_j(x) f_k(x) \; + \; \frac{\partial f_i}{\partial x_j}\frac{\partial f_j}{\partial x_k} f_k(x)\right]$$
$$= \alpha_2 \frac{1}{2}\beta_{11}^2 \frac{\partial^2 f_i}{\partial x_j \partial x_k} f_j f_k + \alpha_3\beta_{22}\beta_{11}\frac{\partial f_i}{\partial x_j}\frac{\partial f_j}{\partial x_k} f_k + \alpha_3 \frac{1}{2}\frac{\partial^2 f_i}{\partial x_j \partial x_k} f_j f_k$$

Here, there are two ways f appears:

$$\frac{\partial^2 f_i}{\partial x_j \partial x_k} f_j(x) f_k(x) \; , \quad \text{and} \quad \frac{\partial f_i}{\partial x_j}\frac{\partial f_j}{\partial x_k} f_k(x) \; .$$

If the combination matches for every f, the parts multiplying these two terms must match separately. This leads to two conditions:

$$\frac{1}{6} = \frac{1}{2}\alpha_2\beta_{11}^2 + \frac{1}{2}\alpha_3\left(\beta_{21}+\beta_{22}\right)^2 \tag{41}$$

and

$$\frac{1}{6} = \alpha_3\beta_{11}\beta_{22} \; . \tag{42}$$

This is a lot of work, but the reward is a third order accurate method.

More correctly, there are many ways to make a third order method. There are six parameters and only 4 equations. For example, you could imitate the second order predictor-corrector trapezoid rule by taking only k_3 in the end. This means $\alpha_1 = \alpha_2 = 0$ and $\alpha_3 = 1$.

4 Multi-variate multi-component Taylor approximations

These formulas are simple in principle but it can be a challenge to get the algebra right. One approach that works for me is the *Einstein summation convention.* The convention is that you don't write the summation symbol and assume that the expression is summed over any repeated variable. For example

$$x_k y_k \;\; \text{means} \;\; \sum_k x_k y_k \; .$$

The sum on the right is assumed to be over "all relevant values" of k. In this case, if x has n components, it is assumed that y also has n components, so

$$x_k y_k \text{ means } \sum_{k=1}^{n} x_k y_k \; .$$

As a more complicated example, suppose A is an $n \times m$ matrix with entries a_{ij} and B is an $m \times l$ matrix with entries b_{jk}. Suppose x is an l component vector and $y = Bx$ and $z = Ay = ABx$. The summation convention writes $y = Bx$ and $z = Ay$ as

$$y_j = b_{jk} x_k \; , \quad z_i = a_{ij} y_j \; .$$

You can substitute one expression into the other to get

$$z_i = a_{ij} b_{jk} x_k \text{ which means } z_i = \sum_{j=1}^{m} \sum_{k=1}^{l} a_{ij} b_{jk} x_k \; .$$

You can write this as

$$z_i = \left(a_{ij} b_{jk} \right) x_k = c_{ik} x_k \; , \quad \text{with } c_{ik} = a_{ij} b_{jk} \; . \tag{43}$$

This gives a formula for the elements of the matrix product $C = AB$.

The summation convention is a convenient way to express the chain rule. Suppose t is a single parameter and $x(t)$ is a smooth n component vector. Suppose $u(x)$ is a one component function of n variables. The chain rule, first without then with the summation convention, is

$$\frac{d}{dt} u(x(t)) = \sum_{j=1}^{n} \frac{\partial u}{\partial x_j} \, \dot{x}_j = \frac{\partial u}{\partial x_j} \, \dot{x}_j \; .$$

If $f(x)$ also has n components, then

$$\frac{d}{dt} f_i(x(t)) = \frac{\partial f_i}{\partial x_j} \, \dot{x}_j \; .$$

If $\dot{x} = f(x)$, this becomes

$$\frac{d}{dt} f_i(x(t)) = \frac{\partial f_i}{\partial x_j} \, f_j(x) \; .$$

The *jacobian* matrix f' has entries

$$\left(f' \right)_{ij} = \frac{\partial f_i}{\partial x_j} \; .$$

Using the jacobian matrix allows us to write the first time derivative formula in matrix/vector form and eventually get (29).

The summation convention derivation starts with

$$\dot{x}_j = \frac{d}{dt} x_j = f_j(x) \; .$$

Then the chain rule gives (29):

$$\begin{aligned}
\ddot{x}_i &= \frac{d}{dt}\dot{x}_i \\
&= \frac{d}{dt} f_i(x) \\
&= \frac{\partial f_i}{\partial x_j}\dot{x}_j \\
&= \frac{\partial f_i}{\partial x_j}\, f_j(x) \; .
\end{aligned}$$

The matrix/vector form (29) is simpler.

The next derivative shows the ease of working with the summation convention and the laws of calculus. The third derivative calculation is (using the product and chain rules)

$$\begin{aligned}
\left(\frac{d}{dt}\right)^3 x_i &= \frac{d}{dt}\ddot{x}_i \\
&= \frac{d}{dt}\left(\frac{\partial f_i}{\partial x_j}\, f_j(x)\right) \\
&= \left(\frac{d}{dt}\frac{\partial f_i}{\partial x_j}\right) f_j(x) + \frac{\partial f_i}{\partial x_j}\left(\frac{d}{dt} f_j(x)\right) \\
&= \left(\frac{\partial^2 f_i}{\partial x_j \partial x_k}\,\dot{x}_k\right) f_j(x) + \frac{\partial f_i}{\partial x_j}\left(\frac{\partial f_j}{\partial x_k}\dot{x}_k\right) \\
\left(\frac{d}{dt}\right)^3 x_i &= \frac{\partial^2 f_i}{\partial x_j \partial x_k} f_j(x) f_k(x) \;+\; \frac{\partial f_i}{\partial x_j}\frac{\partial f_j}{\partial x_k} f_k(x) \; . \qquad (44)
\end{aligned}$$

Both terms on the right are double sums, summing over j and k, but the summation convention says not to write the summation symbols. The two terms have similarities and differences. Both terms have three f factors "on top" and two x factors "on the bottom". The first term achieves this using one second derivative while the second factor uses two first derivatives.

The multi-dimensional Taylor approximations are

$$u(x+y) \approx u(x) + \frac{\partial u}{\partial x_i}\, y_i + \frac{1}{2}\frac{\partial^2 u}{\partial x_i \partial x_j}\, y_i y_j + \frac{1}{6}\frac{\partial^3 u}{\partial x_i \partial x_j \partial x_k}\, y_i y_j y_k + \cdots \; .$$

If t is a small number, there are error bounds such as

$$u(x+ty) \approx u(x) + t\frac{\partial u}{\partial x_i}\, y_i + t^2 \frac{1}{2}\frac{\partial^2 u}{\partial x_i \partial x_j}\, y_i y_j + O(t^2) \; .$$

Sometimes you need to change the names of indices when you substitute one expression into another. In the matrix multiplication example above, suppose we had

$$y_i = b_{ij} x_j \ , \ \ z_i = a_{ij} y_j \ .$$

To express z in terms of x, we need an expression for y_j. If we just take $i = j$ in the y_i expression, we get the non-sensical $y_j = b_{jj} x_j$. Instead, we change the names of the indices to avoid conflict of index names. Here, we could try changing $y_i = b_{ij} x_j$ to the equivalent $y_j = b_{ji} x_i$. This does not work because the i index in x_i conflicts with the i index in z_i. Therefore, we change one of them to k. We could choose $z_k = a_{kj} y_j$. With this, the substitution has no index conflicts and we get

$$z_k = a_{kj} b_{ji} x_i \ .$$

This expression is equivalent to the one we had earlier (43).

5 Convergence theory

Convergence theory starts with proving that the numerical "solution" converges to the actual solution in the limit $\Delta t \to 0$. The max error up to time T is

$$E_{\Delta t, T} = \max_{t_n \leq T} |X_n - x(t_n)| \ . \tag{45}$$

The method *converges* if $E_{\Delta t, T} \to 0$ as $\Delta t \to 0$ for any positive T. The method has order of accuracy p if the error has order Δt^p. That means that there is a C_T so that if Δt is small enough, then

$$E_{\Delta t, T} \leq C_T \Delta t^p \ . \tag{46}$$

Every theorem has hypotheses. One hypothesis is that the scheme has formal accuracy p. The other hypothesis is that the scheme has a Lipschitz property related to the Lipschitz property of the ODE. Here are some examples that explain the complexity of the hypotheses. One is "blow up". The solution of a nonlinear ODE can blow up at some "blow up time" T^*. An example of that involves the ODE (probably discussed in an undergrad ODE class)

$$\dot{x} = x^2 \ .$$

The solution with $x(0) = x_0$ is (check: this satisfies the ODE and the initial condition)

$$x(t) = \frac{1}{\frac{1}{x_0} - t} \ .$$

This solution blows up at $T^* = \frac{1}{x_0}$ (assuming $x_0 > 0$) in the sense $x(t) \to \infty$ as $t \to t^*$. Clearly the convergence theorems cannot apply unless $T < T^*$. A less subtle example is

$$\dot{x} = \log(x) \ .$$

This ODE makes sense only if $x > 0$. A theorem that applies to this case must restrict the range of x where the hypotheses apply – they cannot apply "for all x".

A function f is *Lipschitz continuous* in a domain Ω if there is an L (the *Lipschitz constant* so that if x and y are in Ω, then

$$|f(x) - f(y)| \leq L\,|x - y| \ .$$

A function does not have to be differentiable to be Lipschitz continuous. For example, $f(x) = |x|$ is Lipschitz continuous with Lipschitz constant $L = 1$ but $f'(0)$ does not exist. But if f is differentiable in some domain Ω that contains the line segment from x to y, and if $\|f'(x)\| \leq L$ for all x in Ω, then[4] $|f(x) - f(y)| \leq L\,|x - y|$.

With this motivation, we assume the exact trajectory $x(t)$ exists (does not blow up) up to and including time T. For any positive (probably small) ρ, we define a domain $\Omega_{\rho,T}$ to be the "width ρ neighborhood" of the trajectory. That means that $y \in \Omega_{\rho,T}$ if $|y - x(t)| \leq r$ for some t between 0 and T (inclusive). We assume that $f(x)$ is defined for all x in this neighborhood and that $\|f'(x)\| \leq L$ for all such x. More simply, we suppose f is good in a small neighborhood of the trajectory.

Explicit Runge Kutta methods all have the form

$$\Psi(x, \Delta t) = x + \Delta t \widetilde{\Psi}(x, \Delta t) \ , \tag{47}$$

where $\widetilde{\Psi}$ is found by "iterating" f (i.e., applying f to linear combinations of other values of f). Such a function has a derivative Lipschitz bound because of the chain rule. For example, if $y = f(x + \beta\Delta t f(x))$, then the derivative satisfies (a chain rule calculation)

$$\frac{\partial y}{\partial x} = f'(x + \beta\Delta t f(x))\,[\,I + \beta\Delta t f'(x)\,] \ .$$

If x is "on the trajectory" (that is, $x = x(t)$ for some $t \in [0, T]$) and Δt is small enough so that $|\beta\Delta t f(x)| \leq \rho$, then

$$\left\| \frac{\partial y}{\partial x} \right\| \leq L(\,1 + \beta\Delta t L\,) \ .$$

Therefore it makes sense to assume that there is an M so that for all $x \in \Omega_{\rho,T}$ and all Δt small enough,

$$\left\| \frac{\partial \widetilde{\Psi}(x, \Delta t)}{\partial x} \right\| \leq M \ . \tag{48}$$

[4]This follows from the 1D calculus theorem using a well-known trick. Define $u(t) = f(x + t(y - x))$. Then $u(0) = f(x)$ and $u(1) = f(y)$, and (chain rule) $u'(t) = f'(x + t(y - x))(y - x)$. Since $||f'(y - x)| \leq \|f'\| \cdot |y - x|$, this implies that $|u'(t)| \leq L\,|y - x|$. Therefore $|f(y) - f(x)| = |u(1) - u(0)| \leq L\,|y - x|$.

This is true for explicit Runge Kutta methods.

Here is a convergence proof that follows the *stability plus consistency* proof template. The idea is that the exact solution approximately satisfies the discrete equations (12). We think of this as saying that the exact solution is a perturbation[5] of the approximate solution, the perturbation being the *residual* of the scheme. We write $\Delta t^{p+1} r(x, \Delta t)$ for the residual, with r being defined by a slightly refined version of (12)

$$x(t + \Delta t) = \Psi(x(t), \Delta t) + \Delta t^{p+1} r(x, \Delta t) \; . \tag{49}$$

For example, we found that forward Euler has residual

$$\Delta t^2 r(x, \Delta t) \; , \quad r(x, \Delta t) = \frac{1}{2} f'(x) f(x) + O(\Delta t) \; .$$

We make the "formal order of accuracy" hypothesis that there is an overall bound R that works for all $x \in \Omega_{\rho,T}$ and small enough Δt:

$$|r(x, \Delta t)| \leq R \; . \tag{50}$$

We combine (47) with (49) and write

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + \Delta t \widetilde{\Psi}(x(t_n), \Delta t) + \Delta t^{p+1} r(x(t_n), \Delta t) \\ X_{n+1} \;&=\; X_n \;+ \Delta t \widetilde{\Psi}(X_n, \Delta t) \; . \end{aligned}$$

We use the definitions

$$D_n = x(t_n) - X_n \; , \;\; x(t_n) = X_n + D_n \; .$$

These combine to give

$$D_{n+1} = D_n + \Delta t \left[\widetilde{\Psi}(X_n + D_n) - \widetilde{\Psi}(X_n) \right] + \Delta t^{p+1} r(x(t_n), \Delta t) \; .$$

Given the assumptions, this leads to the inequality

$$|D_{n+1}| \leq (\, 1 + M \Delta t \,) \, |D_n| + R \Delta t^{p+1} \; . \tag{51}$$

This inequality incorporates *consistency* in that the forcing term is on the order of Δt^{p+1}. It incorporates *stability* in that the error is amplified in one time step by only $1 + M\Delta t$.

A little "analytical technique" (clever inequalities) turns the error propagation bound (51) into an error bound of the form (46). One observation is that D_n cannot be more than the solution of the error propagation equality

$$\widetilde{D}_n = (\, 1 + M \Delta t \,) \, \widetilde{D}_n + R \Delta t^{p+1} \; . \tag{52}$$

[5] *Perturbation* means "small change". More precisely, a small change with a small cause. If $g(x) = 0$ and $g(y) = \epsilon$, we say that ϵ is a small perturbation in the equation which (hopefully) makes y a small perturbation of x. In ordinary English, a person trying to study might be "perturbed" by a small noise. This suggests that the person not only is influenced by the noise, but is annoyed by it. Fortunately, mathematics uses the term "perturbation theory" rather than "annoyance theory".

Next, observe that this is the forward Euler approximation to the differential equation

$$\dot{d} = Md + R\Delta t^p \ .$$

This step is where the "local truncation error" of order Δt^{p+1} is turned into a "global error" (simply, error) of order Δt^p. The solution to the ODE, with initial condition $d(0) = 0$ (which we should have said all along: you start with the exact initial data $X_0 = x(0)$) is (using methods from ODE 101, check this)

$$d(t) = \frac{R\Delta t^p}{M}\left(e^{Mt} - 1 \right) \ . \tag{53}$$

This is a good estimate of the solution of the error propagation equality (52) when Δt is small, but is it an upper bound for it?

Yes, because of the inequality $1+s \leq e^s$, which is true for any s, or for a more "fundamental" reason that takes longer to explain. If we assume (induction hypothesis for induction on n) that $\widetilde{D}_n \leq d(t_n)$, then we first get $d(t_{n+1}) \leq \left(e^{Mt} - 1 \right) d(t_n) + R\Delta t^{p+1}$ and then $\widetilde{D}_n \leq d(t_n)$.

Conclusions

- Order Δt^{p+1} local truncation error leads to order Δt^p overall error.
- The error bound (53) grows exponentially with t so it may not be useful for long time simulations.
- The stability proof does not use detailed properties of the scheme or the ODE. The same proof would work for any explicit one-step method.

6 Error expansion, modified equation

A more refined analysis of the error can lead to post-processing tricks that lead to more accurate solutions. The *asymptotic error expansion* is a tool for this. The *modified equation* is a modification of the ODE system so that the flow map for the modified equation is closer to the trajectory of the ODE solver. The extra terms in the modified equation can explain slow but qualitative differences between the true ODE trajectory and the approximate trajectory produced by the solver.

6.1 Asymptotic expansion

An *asymptotic expansion* is a sequence of approximations with increasing accuracy. *Asymptotic* means that we do not know whether the sequence converges to the thing being approximated. Many expansions that do converge are used as asymptotic expansions. The fact that the sequence converges in the limit of infinitely many terms is not relevant if you only use the first few. If we only need the asymptotic expansion property, it is not necessary to know whether the sequence converges. In many interesting cases, the sequence does not converge.

Consider a function $A(h)$ that depends on a "small parameter" h. In applications here, h will be Δt (and later Δx). An asymptotic expansion in powers of h is written

$$A(h) \sim A_0 + hA_1 + h^2 A_2 + \cdots \; . \tag{54}$$

The asymptotic expansion property is that for any p, the terms up to order p give an approximation of $A(h)$ that is accurate to order p. Technically, if $p \geq 0$ is an integer, then there is an $h_p > 0$ and a C_p so that

$$\left| A(h) - (A_0 + \cdots + h^p A_p) \right| \leq C_p h^{p+1} \; , \text{ if } |h| \leq h_p \; . \tag{55}$$

This is the same as, for every p,

$$A(h) = A_0 + \cdots + h^p A_p + O(h^{p+1}) \; .$$

The "$\sim$" symbol in (54), rather than "$=$", means that for any given $h \neq 0$ we do not say that the sum on the right converges to $A(h)$, or converges at all.

Taylor expansions are asymptotic expansions in this sense. If $A(h) = f(x + h)$, then $A_n = \frac{1}{n!} f^{(n)}(x)$. One of the error bounds for Taylor approximations (Calculus II or Calculus III) is

$$f(x+h) = \sum_{n=0}^{p} \frac{1}{n!} f^{(n)}(x) h^n + \frac{1}{(n+1)!} f^{(n+1)}(\xi) h^{p+1} \; ,$$

$$\text{for some } \xi \text{ between } x \text{ and } x+h \; .$$

An asymptotic expansion that does not converge

Define

$$A(h) = \int_0^\infty \frac{e^{-\frac{x}{h}}}{1+x} dx \; .$$

When h is small the exponential is small unless x is very close to zero. This suggests that the small h behavior of $A(h)$ is determined by the small x behavior of $\frac{1}{1+x}$. For small x (actually, for any x less than 1) this is given by a convergent geometric series

$$\frac{1}{1+x} = 1 - x + x^2 - \cdots + (-1)^n x^n + \cdots \; .$$

This suggests that, for small h,

$$A(h) \approx \int_0^\infty e^{-\frac{x}{h}} \left(1 - x + x^2 \cdots \right) dx \; .$$

The integral of an exponential against a power of x is

$$\begin{aligned}\int_0^\infty e^{-ax}\,dx &= \frac{1}{a}\\ \int_0^\infty xe^{-ax}\,dx &= \frac{1}{a^2}\\ \int_0^\infty x^2e^{-ax}\,dx &= \frac{2}{a^3}\\ &\vdots\\ \int_0^\infty x^ne^{-ax}\,dx &= \frac{n!}{a^{n+1}}\end{aligned}$$

If you substitute these into the approximation for $A(h)$ without worrying about convergence, you get (with $a = \frac{1}{h}$)

$$A(h) \approx h - h^2 + 2h^3 - \cdots + (-1)^n n!\, h^{n+1} + \cdots \ . \tag{56}$$

It is possible to prove (if you "remember" the tricks of mathematical analysis) that this is an asymptotic expansion in the sense of (55), with $A_0 = 0$ and $A_p = (-1)^{p-1}(p-1)!$. However, the infinite sum diverges for any h, because eventually $n!$ grows faster than h^{n+1}.

The point of this example is not to make you worry about whether an asymptotic expansion converges. It's the opposite. This expansion diverges but it is just as useful for the purposes here as asymptotic expansion based on Taylor series that probably do converge. The point is to convince you not to worry about whether an asymptotic expansion converges.

6.2 Asymptotic error expansion

An *asymptotic error expansion* is an asymptotic expansion for the error in powers of Δt. Many computational tricks require asymptotic error expansions to exist. Some of those tricks actually calculate a term or two quantitatively. The proof that such expansions exist gives a method for calculating the terms, but this is so complicated that it is (almost) never used in practice directly. More bluntly, it is important to know *that* asymptotic error expansions exist, but it is less important to know *why* they exist.

An asymptotic error expansion for an ODE solver of order p has the form

$$X_n \sim x(t_n) + \Delta t^p y_p(t_n) + \Delta t^{p+1} y_{p+1}(t_n) + \cdots \ . \tag{57}$$

This expansion is based on an asymptotic expansion for the approximate flow map, which has (local truncation) error of order Δt^{p+1}:

$$\Psi(x,\Delta t) \sim \Phi(x,\Delta t) + \Delta t^{p+1}\Psi_{p+1}(x,\Delta t) + \Delta t^{p+2}\Psi_{p+2}(x,\Delta t) + \cdots \ . \tag{58}$$

The derivations of the various Runge Kutta methods make it clear that these flow map expansions exist, and that the formula for Ψ_{p+1} is likely to be complicated and not very informative.

We justify the asymptotic expansion (57) informally by showing how to find the leading error term y_p. A full proof would include arguments like those in Section 5. In these calculations we leave out the Δt argument all the time and other arguments some of the time to simplify expressions. We also use the facts that $\Psi(x,\Delta t) = x + \Delta t\widetilde{\Psi}(x,\Delta t)$, so $\Psi'(x)y = y + \Delta t\widetilde{\Psi}'(x)y$, and $x(t_{n+1}) = \Phi(x(t_n))$.

$$\begin{aligned}
X_{n+1} &= \Psi(X_n) \\
&= \Psi(\, x(t_n) + \Delta t^p\, y_p(t_n) + \cdots) \\
&\approx \Psi(\, x(t_n)) + \Delta t^p\, \Psi'(x(t_n))\, y_p(t_n) + \cdots \\
x(t_{n+1}) + \Delta t^p y_p(t_{n+1}) &\approx \Phi(\, x(t_n)) + \Delta t^p\, \Psi'(x(t_n))\, y_p(t_n) + \Delta t^{p+1}\Psi_{p+1}(x(t_n)) + \cdots \\
\Delta t^p y_p(t_{n+1}) &\approx \Delta t^p\, \Psi'(x(t_n))\, y_p(t_n) + \Delta t^{p+1}\Psi_{p+1}(x(t_n)) + \cdots \\
y_p(t_n) + \Delta t\dot{y}_p + \cdots &\approx y_p(t_n) + \Delta t\widetilde{\Psi}'(x)y_p + \Delta t\Psi_{p+1}(x(t_n)) + \cdots \\
\dot{y}_p &\approx \widetilde{\Psi}'(x)\, y_p + \Psi_{p+1}(x)\ .
\end{aligned}$$

Finally, to leading order $\Psi(x) = x + \Delta t f(x)$, so $\Psi'(x) = I + \Delta t f'(x)$. Therefore, to leading order, we may replace $\widetilde{\Psi}'(x)$ with f'. When you do that, none of the leading terms depend on Δt. Therefore, all the Δt corrections vanish in the limit and you are left with

$$\dot{y}_p(t) = f'(x(t))\, y_p(t) + \widetilde{\Psi}'(x(t))\ . \tag{59}$$

This is the leading order error propagation equation. It is the original ODE (1) linearized about the exact trajectory $x(t)$ and with the leading order truncation error as forcing.

7 Adaptive methods

A fancy computational algorithm tries to get the desired accuracy with the least work. For an ODE solver, this could mean integrating to a specified time T using the smallest number of time steps required to meet the specified accuracy. It is likely that "users" (the people who want the answer) does not know enough about the problem to choose Δt appropriately. They might not know whether a specified time step was small enough, or maybe smaller than necessary. *Adaptive* methods use the computer to help choose Δt.

8 References

On the FFT

The basic notation for Runge Kutta methods is in
Numerical Methods, Germund Dahlquist, Åke Björk

A more recent treatment that uses more modern terminology, and an excellent collection of references, is in
A First Course in the Numerical Analysis of Differential Equations by Arieh Iserles.