Lecture 3, Ordinary Differential Equations

1 The method of lines

We discretized the heat/diffusion equation in space and time. The space step was Δx and the time step was Δt. We often do the two discretizations separately, replacing the space operator $D\partial_x^2$ with a finite difference approximation

$$\partial_x^2 u(x,t) \approx \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2} \ .$$

If $U_k \in \mathbb{R}^n$ is the numerical approximation at time t, the time step was

$$U_{k+1} = U_k + \Delta t A U_k \ , \tag{1}$$

where A is the matrix representing the finite difference approximation of $D\partial_x^2$. The *method of lines* separates the problem of space discretization from the problem of evolution in time. It does this using the intermediate step where we discretize in space only but keep time continuous. In the present case, this would mean having a time dependent vector $U(t) \in \mathbb{R}^n$ with components meant to approximate the corresponding values of the exact PDE solution

$$U_j(t) \approx u(x_j,t) \ \ , \quad x_j = j\Delta x \ .$$

The *semidiscrete* approximation the PDE $\partial_t u = D\partial_x^2 u$ is

$$\dot{U} = AU \ . \tag{2}$$

Here, we write $\dot{U}$ for $\frac{d}{dt}U$ to emphasize that (2) is a system of ordinary differential equations. The *method of lines* replaces a PDE for a the evolution in time of a function of x with an ODE system for a discrete (in space) approximation.

The method of lines leads to the problem of solving (large) ODE systems. That is the problem addressed in this week's class. We will talk about linear equation systems (2) or non-linear ones that have the form

$$\dot{U} = F(U) \ . \tag{3}$$

Many of the methods and much of the analysis is completely generic, applying to any ODE system. But it is important to remember that many of our applications come from PDE discretizations that are large and have interesting and challenging special structure that we cannot ignore (if we want a good method) and may even take advantage of. For example, the specific matrix A we got by discretizing ∂_x^2, it has real negative eigenvalues ranging from $-\pi^2 D/L^2$, to $-n^2 D/L^2$. The ratio of largest to smallest (in the negative sense), which is the

condition number of A, is order n^2. Other PDE lead to ill conditioned matrices whose eigenvalues run up and down the imaginary axis. Methods for one type may or may not be good for other types.

Now, for the next while, forget the origin of the problems (2) or (3), mostly the latter. We consider *time stepping* algorithms to solve the *initial value problem* with *initial conditions* $U(0) = U_0$ given. We have a time step Δt and seek approximations $U_k \approx U(t_k)$. The simplest method is the one we just used, the *forward Euler* method

$$U_{k+1} = U_k + \Delta t F(U_k) \ .$$

This method is first order accurate, in that

$$\|U_k - U(t_k)\| \leq C_T \ , \quad \text{if } t_k \leq T.$$

The error constant depends on the time, T (very unfortunate and usually unavoidable), but is independent of Δt.

There are time stepping methods much better than forward Euler. Some are much more accurate. Others have vastly better stability properties for ill conditioned matrices. We discuss two broad classes of time stepping methods, *linear multistep* methods and *Runge Kutta* methods (which could be called *multi-stage* methods. It is possible to combine these to create multistep multi-stage methods, but such methods are not used much.

2 Linear multistep methods

We start with some motivation and examples, then present the general form of linear multistep methods. One way to arrive at a linear multistep method is to use a higher order approximation to the time derivative. The forward Euler method is based on the approximation

$$\frac{dU}{dt}(t) \approx \frac{U(t + \Delta t) - U(t)}{\Delta t} \ .$$

This one sided difference approximation is first order accurate. The simplest centered approximation is second order accurate

$$\frac{dU}{dt}(t) = \frac{U(t + \Delta t) - U(t - \Delta t)}{2\Delta t} + O(\Delta t^2) \ .$$

This leads to a second order time stepping method

$$U_{k+1} = U_{k-1} + 2\Delta t F(U_k) \ . \tag{4}$$

This method is called the *midpoint rule* or the *leapfrog*[1] method.

[1] Leapfrog is a children's game in which they crouch and pretend to be frogs. They take turns jumping (leaping) over each other.

The name "midpoint rule" comes from a different view of the formula (4). The exact solution satisfiles

$$\begin{aligned} U(t+\Delta t) &= U(t-\Delta t) + \int_{t-\Delta t}^{t+\Delta t} \dot{U}(s)\, ds \\ &= U(t-\Delta t) + \int_{t-\Delta t}^{t+\Delta t} F(U(s))\, ds \end{aligned}$$

The *midpoint rule* is to approximate the integral using the integrand at the midpoint (center of the interval):

$$\int_{t-\Delta t}^{t+\Delta t} f(s)\, ds \approx 2\Delta t f(t) \; .$$

The leapfrog method may be derived by applying the midpoint rule of integration to the integrand $f(t) = F(U(t))$. The forward Euler method also can be viewed as an integral approximation. This time it's

$$\int_{t}^{t+\Delta t} f(s)\, ds \approx \Delta t f(t) \; . \tag{5}$$

There is a family of linear multistep methods, called *Adams methods*, based on better approximations to the integrand $f(s)$ in (5). The approximation (5) may be "derived" by replacing $f(s)$ by $f(t)$. This approximation is first order accurate: $f(s) - f(t) = O(\Delta t)$ if $t \le s \le t + \Delta t$ and if $f(s)$ is differentiable. This is

$$\int_{t}^{t+\Delta t} f(s)\, ds \approx \int_{t}^{t+\Delta t} f(t)\, ds = \Delta t f(t) \; .$$

The higher order Adams methods are based on higher order approximations of $f(s)$ in the interval $[t, t + \Delta t]$. For example, you can use the estimate

$$\dot{f}(t) \approx \frac{f(t) - f(t-\Delta t)}{\Delta t} \; .$$

Then a better approximation to $f(s)$ for s near t is

$$\begin{aligned} f(s) &= f(t) + \dot{f}(t)(s-t) + O(\Delta t^2) \\ &= f(t) + \frac{f(t) - f(t-\Delta t)}{\Delta t}(s-t) + O(\Delta t^2) \; . \end{aligned} \tag{6}$$

You can integrate this to get

$$\begin{aligned} \int_{t}^{t+\Delta t} f(s)\, ds &= \Delta t f(t) + \frac{f(t) - f(t-\Delta t)}{\Delta t}\frac{\Delta t^2}{2} + O(\Delta t^3) \\ &= \frac{3}{2}\Delta t f(t) - \frac{1}{2}\Delta t f(t-\Delta t) + O(\Delta t^3) \; . \end{aligned}$$

We get an ODE time stepping scheme by using $f(s) = F(U(s))$:

$$U_{k+1} = U_k + \Delta t \left(\frac{3}{2} F(U_k) - \frac{1}{2} F(U_{k-1}) \right) . \tag{7}$$

This is the second order three level *Adams Bashforth* method.

The higher order Adams Bashforth methods are derived from a different interpretation of the approximation (6). The linear function

$$l(s) = f(t) + \frac{f(t) - f(t - \Delta t)}{\Delta t}(s - t)$$

is (obviously) a linear function of s and it interpolates the known values at times t and $t - \Delta t$:

$$l(t) = f(t) , \;\; l(t - \Delta t) = f(t - \Delta t) .$$

Linear interpolation of smooth functions is second order accurate, which leads to a second order accurate ODE solver.

The higher order Adams Bashforth methods use higher order polynomial interpolation of the known values $f(t)$, $f(t-\Delta t)$, ..., $f(t-r\Delta t)$. Using explicit interpolation formulas it is possible to integrate the interpolating polynomial of order r over the interval $[t, t + \Delta]$. The result is a time stepping method of the form

$$U_{k+1} = U_k + \Delta t \left(b_0 F(U_k) + b_1 F(U_{k-1}) + \cdots + b_r F(U_{k-r}) \right) . \tag{8}$$

The $r = 0$ method if forward Euler. The $r = 1$ method is the second order method above. In general, the method with r steps has order of accuracy $r + 1$. This gives first order for $r = 0$ (Euler) and second order for $r = 1$. These are examples of *linear multistep methods*. "Linear" means that the new value, U_{k+1}, is a linear combination of earlier computed values U_{k-j} and $F(U_{k-j})$. "Multistep" means that the scheme uses more than two time levels. These methods predict time level $k + 1$ from levels k down to $k - r$. You might call this an $r-$step method. More general linear multistep methods also use old U values:

$$\begin{aligned} U_{k+1} = {} & a_0 U_k + \cdots + a_r U_{k-r} \\ & + \Delta t \left(b_0 F(U_k) + b_1 F(U_{k-1}) + \cdots + b_r F(U_{k-r}) \right) . \end{aligned} \tag{9}$$

The leapfrog method (4) is another linear multistep method that is not in the Adams Bashforth family. It has $r = 1$, $a_0 = 0$, $a_1 = 1$, $b_0 = 2$, and $b_1 = 0$. Methods of the form (9) are *explicit* because they are explicit formulas for U_{k+1} in terms of already computed quantities.

3 Runge Kutta methods

Runge Kutta methods (named for Karl Runge and his student Kutta) are a different way to generalize the forward Euler method. They are one step methods,

which means that U_{k+1} is a function of U_k only. If you know U_k and not U_{k-1} (or $F(U_{k-1})$, you can compute U_{k+1}.

A simple Runge Kutta method involves two *stages*, which means two evaluations of $f(u)$. The midpoint rule for integration (which we just used) gives

$$U(t+\Delta t) = U(t) + \Delta t F(U(t+\tfrac{1}{2}\Delta t)) + O(\Delta t^3) \; .$$

This is based on the midpoint integral approximation

$$\int_t^{t+\Delta t} F(U(s))\, ds = \Delta t F(U(t+\tfrac{1}{2}\Delta t)) + O(\Delta t^3) \; .$$

This isn't useful as it stands because the midpoint value $U(t+\frac{1}{2}\Delta t)$ is not known. The Runge Kutta solution is to *predict* the unknown midpoint value using forward Euler. This leads to

$$\widetilde{U}_{k+\frac{1}{2}} = U_k + \frac{\Delta t}{2} F(U_k) \tag{10}$$

$$U_{k+1} = U_k + \Delta t F(\widetilde{U}_{k+\frac{1}{2}}) \; . \tag{11}$$

This is a two stage method. The first stage (10) predicts the midpoint value. The second stage (11), the *corrector* stage, uses the predicted midpoint value to take the time step.

An explicit *Runge Kutta* method with s stages has the form

$$\left.\begin{array}{rcl} \xi_1 &=& \Delta t F(U_k) \\ \xi_2 &=& \Delta t F(U_k + b_{11}\xi_1) \\ \xi_3 &=& \Delta t F(U_k + b_{21}\xi_1 + b_{22}\xi_2) \\ &\vdots& \\ \xi_s &=& \Delta t F(U_k + b_{s1}\xi_1 + \cdots + b_{s,s-1}\xi_{s-1} \end{array}\right\} \; . \tag{12}$$

$$U_{k+1} = U_k + a_1\xi_1 + \cdots + a_s\xi_s \; . \tag{13}$$

The number of stages is the number of evaluations of F. It is common that evaluating F is much more expensive than the rest of the Runge Kutta calculations, so the work for one time step is proportional to the number of stages. An s stage method is s times more work than a one stage method. There must be compelling arguments, involving accuracy or the size of the time step, for using a higher order multi-stage method. The forward Euler method has one stage. The second order (as we will see) predictor-corrector midpoint formula has $s = 2$ stages. The coefficients are $b_{11} = \frac{1}{2}$, and $a_1 = 0$, and $a_2 = 1$.

A general explicit s stage Runge Kutta method has lots of coefficients. These are chosen to achieve high order of accuracy or large time step stability or for other reasons. The specific calculations can be very complicated and, and not in a way most mathematicians call "beautiful". We present some accuracy calculations for the first few methods and then give a proof that a method with that formal order of accuracy in fact has error of the order of magnitude

suggested by the order of accuracy calculation. The corresponding theory for linear multi-step methods is in Lecture 4.

Here is some notation that we can use to state the order of accuracy claims. The Runge Kutta method above is a complicated function that computes U_{k+1} from U_k. That function will be written $U_{k+1} = G(U_k, \Delta t)$. The exact solution to the differential equation will be written $U(t + \Delta t) = S(U(t), \Delta t)$. The terminology is "motivated" by choosing G to represent a complicated function defined by applying F a few times. The S is for "solution". The function $U(t + s) = S(U(t), s)$ is also called the *flow*, or the *flow map* corresponding to the differential equation (3). The truncation error for a Runge Kutta method depends on the difference between the numerical update in time Δt and the update given by the exact solution of the ODE system. As we did for the heat equation, we define the truncation error with a factor of Δt pulled out. This definition of truncation error has the advantage that the truncation error has the same order (in Δt) as the actual error.

$$\Delta t R(U, \Delta t) = G(U, \Delta t) - S(U, \Delta t) \ . \tag{14}$$

We will show (not right now, but soon) that if $R = O(\Delta t^p)$ then $|U_k - U(t_k)| \leq C_T \Delta t^p$ if $t_k \leq T$. This is pretty much the same as the convergence theorem we had for the heat equation: the formal order of accuracy as given by truncation error is the same order as the actual error.

The three main steps in calibrating a Runge Kutta method are:

1. Calculate the Taylor series representation for $S(U, s)$ for small s.
2. Calculate the Taylor series representation for $G(U, s)$ for small s.
3. Identify terms order by order, and find the relations for the RK coefficients that make corresponding Taylor series terms equal, find solutions for the equations.

Here is how it works out for the forward Euler method.

Step 1, the exact solution: The Taylor series for the exact solution is

$$U(t + s) = U(t) + s\dot{U}(t) + \tfrac{1}{2}s^2\ddot{U}(t) + O(s^3) \ .$$

The differential equation (3) immediately gives

$$\dot{U}(t) = F(U(t)) \ .$$

You can find the second derivative by differentiating this with respect to time and using the chain rule:

$$\begin{aligned} \ddot{U}(t) &= \frac{d}{dt}\dot{U} \\ &= \frac{d}{dt}F(U(t)) \\ &= F'(U(t))\,\dot{U}(t) \qquad (15) \\ \ddot{U}(t) &= F'(U(t))\,F(U(t)) \ . \qquad (16) \end{aligned}$$

We need to be careful in interpreting this when U has many components. It isn't so hard in this case, but it gets harder for derivatives beyond the second. For the present second derivative calculation, (15) is just the chain rule from multivariate calculus, where F' is the $n \times n$ Jacobian matrix of partial derivatives and $\dot{U}$ is in the n component column vector. For clarity, we repeat this calculation with individual components and the less fancy version of the chain rule. The derivative of one component is

$$\frac{d}{dt}\dot{U}_j = \frac{d}{dt}F_j(U) = \sum_{k=1}^{n} \frac{\partial F_j(U)}{\partial U_k}\,\frac{dU_k}{dt} = \sum_{k=1}^{n} \frac{\partial F_j(U)}{\partial U_k}\,F_k(U) \ .$$

The sum on the right may be interpreted as component j of the product of the matrix with components $\partial_{U_k} F_j$ and the vector with components F_k. The matrix is the Jacobian matrix, written F'.

The conclusion of this calculation is

$$U(t+\Delta t) = S(U(t),\Delta t) = U(t) + \Delta t F(U(t)) + \Delta t^2 \frac{1}{2} F'(U(t))F(U(t)) + O(\Delta t^3) \ . \tag{17}$$

The order Δt term on the right is familiar. The order Δt^2 term, and its calculation using the chain rule, is the first indication of the complexity of Runge Kutta method algebra.

Step 2, the time stepper: For forward Euler, the time step is linear in Δt. This means that the first order Taylor series is exact:

$$G(U,\Delta t) = U + \Delta t F(U) \ , \quad \text{exactly.}$$

No other method is this simple, alas.

Step 3. order of accuracy. Putting together steps 1 and 2 gives

$$G(U,\Delta t) - S(U,\Delta t) = -\frac{1}{2}\Delta t^2 F'(U(t))F(U(t)) + O(\Delta t^3) \ .$$

The residual, therefore, satisfies

$$R(U,\Delta t) = -\frac{1}{2}\Delta t F'(U(t))F(U(t)) + O(\Delta t^2) = O(\Delta t) \ .$$

This is the analysis that shows the forward Euler method is first order accurate.

The analysis of two step methods will give you a clearer picture of Runge Kutta math. The analysis (17) is accurate enough for this purpose. The differences come in step 2. Although ξ_1 depends on Δt in a linear way, ξ_2 is some general nonlinear function of Δt. The Taylor series expansion is (explanations follow)

$$\begin{aligned}
\xi_2 &= \Delta t \left(F(U + b_{11}\xi_1) \right) \\
&= \Delta t \left(F(U) + b_{11}F'(U)\xi_1 + O(|\xi_1|^2) \right) \qquad (18) \\
&= \Delta t \left(F(U) + \Delta t b_{11} F'(U)F(U) + O(\Delta t^2) \right) \ . \qquad (19)
\end{aligned}$$

Equation (18) is the Taylor series expansion of $F(U+\xi_1)$. We know ξ_1 is small because it is of the order of Δt. The remainder estimate $O(|\xi_1^2|)$ is one of the forms of the remainder estimate for multi-variate Taylor expansions. Equation (19) just replaces $|\xi_1|$ with $O(\Delta t)$.

Step 3 also becomes (for the first time) non-trivial. Before we know the values of the parameters b_{11}, and a_1, and a_2, we have the general expression

$$G(U,\Delta t) - S(U,\Delta t) = a_1 \Delta t F(U) + a_2 \left(\Delta t F(U) + \Delta t^2 b_{11} F'(U)F(U) + O(\Delta t^3)\right) - \Delta t F(U) - \Delta t^2 \frac{1}{2} F'(U)F(U) + O(\Delta t^3) \; .$$

We want to chose the parameters to make the right side as small as possible (measured in powers of Δt. The largest term is the one proportional to Δt. It is

$$\Delta t \left(a_1 + a_2 - 1\right) F(U) \; .$$

Setting the coefficient of Δt to zero gives

$$a_1 + a_2 = 1 \; . \tag{20}$$

The stuff involving Δt^2 is

$$\Delta t^2 \left(a_2 b_{11} F'(U)F(U) - \frac{1}{2} F'(U)F(U) \right) \; .$$

Setting this to zero gives the equation

$$a_2 b_{11} - \frac{1}{2} = 0 \; . \tag{21}$$

If the equations (20) and (21) are satisfied then the residual will be second order in the sense that:

$$R(U,\Delta t) = O(\Delta t^2) \; .$$

There are many second order two stage explicit Runge Kutta methods. There are three unknown parameters (b_{11}, a_1, and a_2) and only two equations. Here are some of the possibilities:

- $a_1 = 0$, $a_2 = 1$, and $b_{11} = \frac{1}{2}$. This is the predictor/corrector midpoint rule method we discussed before. From $a_1 = 0$ and $a_2 = 1$, we have $U_{k+1} = U_k + \Delta t F(U_k + b_{11}\xi_1)$. But $U_k + \frac{1}{2}\Delta t F(U_k)$ is the forward Euler prediction $\widetilde{U}_{k+\frac{1}{2}}$ in (10).

- $a_1 = \frac{1}{2}$, $a_2 = \frac{1}{2}$, $b_{11} = 1$. This is a predictor/corrector form of the trapezoid rule for integration. The trapezoid rule is the approximation

$$\int_t^{t+\Delta t} f(s)\, ds = \Delta t \frac{1}{2} \left(f(t) + f(t+\Delta t)\right) + O(\Delta t^3) \; .$$

  The right side approximation is the area of the trapezoid (a "rectangle with flat bottom and sloped top) that touches the graph of f at t and $t+\Delta t$.

We could formulate a time stepping method by taking $f(s) = F(U(s))$ and then and we approximate $U(t_k)$ by U_k, then the trapezoid rule becomes

$$U_{k+1} - U_k \approx \int_{t_k}^{t_{k+1}} F(U(s))\, ds \approx \Delta t \frac{1}{2} \left(F(U_{k+1}) - F(U_k) \right) \ .$$

The time stepping algorithm based on these approximations is called the *trapezoid rule*:

$$U_{k+1} = U_k + \Delta t \, \frac{1}{2} (F(U_{k+1}) - F(U_k)) \ . \tag{22}$$

This is an example of an *implicit* method. It is implicit in the sense that you have to solve a system of equations to find U_{k+1}. The new value U_{k+1} defined implicitly by the equations rather than being given by an explicit formula. You can create an explicit approximation to the implicit method by using a *predicted* value of U_{k+1} on the right. The forward Euler prediction is

$$U_{k+1} \approx \widetilde{U}_{k+1} = U_k + \Delta t F(U_k) \ .$$

This is the *predictor* step of a *predictor/corrector* method. The *corrector* step is to use the prediction value $\widetilde{U}_{k+1}$ in the more accurate trapezoid rule formula. There are two evaluations of F, one for the prediction step and one for the correction step. This is the two stage Runge Kutta method.

- The method with $a_1 = \frac{1}{3}$ and $a_2 = \frac{2}{3}$ and $b_{11} = \frac{3}{4}$ is called *Heun's* method[2]. People say (e.g. the Wikipedia page on Runge Kutta methods) that Heun's method is the most accurate second order method, in the sense of having the smallest *error constant*. I don't know in what sense this is supposed to be true.

You get a better idea how the algebra gets complicated with three stage methods. For these, we have to calculate the next term in the Taylor series approximation of $S(U, \Delta t)$. We calculate the third derivative by differentiating the second derivative. Here, you have to use the product rule and the chain rule. We do it first informally (ignoring the fact that U and F have many components), then with indices to see the actual result.

$$\begin{aligned}
\frac{d^3}{dt^3} U(t) &= \frac{d}{dt} \ddot{U}(t) \\
&= \frac{d}{dt} \left[F'(U(t)) F(U(t)) \right] \\
&= \left(\frac{d}{dt} F'(U(t)) \right) F(U(t)) + F'(U(t)) \left(\frac{d}{dt} F(U(t)) \right) \\
&= F''(U(t)) \dot{U}(t) F(U(t)) + F'(U(t)) F'(U(t)) \dot{U}(t) \\
&= F''(U(t)) F'(U(t)) F(U(t)) + \left(F'(U(t)) \right)^2 F(U(t)) \ .
\end{aligned}$$

[2]Like Kutta, Heun was a PhD student of Runge. I think Heun came first with the second order method, while Kutta developed the fourth order method that is often called *the* Runge Kutta method.

The actual multi-variate result is like this, but with something like matrix products instead of ordinary multiplication. The multi-variate object F' is a matrix, which is an objet with two indices. The multi-variate object F'' has three indices.

$$F''_{i,jk} = \partial_{U_j} \partial_{U_k} F_i(U) \ .$$

The derivative calculation is, following the previous informal one,

$$\begin{aligned}
\frac{d}{dt}\ddot{U}_j &= \frac{d}{dt}\left(\sum_{k=1}^{n} \frac{\partial F_j(U)}{\partial U_k} F_k(U) \right) \\
&= \sum_{l=1}^{n}\sum_{k=1}^{n} \left(\partial_{U_l}\partial_{U_k} F_j(U)\right) F_l(U) F_k(U) + \sum_{k=1}^{n}\sum_{l=1}^{n} \left(\partial_{U_k} F_j(U)\right)\left(\partial_{U_l} F_k(u)\right) F_l(U) \\
&= T_1 + T_2 \\
T_1 &= \sum_{l=1}^{n}\sum_{k=1}^{n} \left(\partial_{U_l}\partial_{U_k} F_j(U)\right) F_l(U) F_k(U) \qquad (23) \\
T_2 &= \sum_{k=1}^{n}\sum_{l=1}^{n} \left(\partial_{U_k} F_j(U)\right)\left(\partial_{U_l} F_k(u)\right) F_l(U) \ . \qquad (24)
\end{aligned}$$

It happens that when you do the step 2 analysis of the of the discrete time step map $G(U, \Delta t)$ you get the same terms T_1 and T_2, multiplied by stuff that depends on the a_j and b_{jk}. Therefore, you have to satisfy two equations at order Δt^3 to make a Runge Kutta scheme third order accurate.

For a three stage explicit Runge Kutta scheme there are six parameters and four equations. The parameters are

$$a_1, \ a_2, \ a_3, \ b_{11}, \ b_{21}, \ b_{22} \ .$$

The equations are one for consistency (first order accuracy), one for second order, and two for third order. Since there are more parameters than unknowns, and because nature smiles on this calculation, there are again families of solutions.

3.1 Convergence of Runge Kutta methods

The convergence proof for Runge Kutta methods follows the consistency/stability template we used before. Consistency is the fact that the exact solution almost satisfies the discrete equation. Stability is the fact that if someone almost satisfies the discrete equation, then that someone is close to the discrete solution.

The consistency step for Runge Kutta methods is, but for notation, (14). The discrete equation is $U_{k+1} = G(U_k, \Delta t)$. The exact solution satisfies $U(t_{k+1}) = S(U(t_k), \Delta t)$. With (14), this gives

$$U(t_{k+1}) = G(U(t_k), \Delta t) + \Delta t R(U(t_k), \Delta t) \ .$$

A Runge Kutta method is formally order p if $|R| \leq C\Delta t^p$.

The stability of Runge Kutta methods must have hypotheses. The function $F(U)$ in the ODE (3) is *Lipschitz continuous*, with *Lipschitz constant* L_f, if

$$|F(U) - F(V)| \leq L_f \, |U - V| \ . \tag{25}$$

Your differential equations class [*that you should have taken should have*] explained that it is natural to assume that F is Lipschitz continuous. We assume this for now, and come back to discuss the more realistic situation where F is only *locally* Lipschitz continuous.

Runge Kutta stability is easy because if F is Lipschitz continuous, then G has the stability property

$$|G(U, \Delta t) - G(V\Delta t)| \leq (1 + L_g \Delta t) \, |U - V| \ . \tag{26}$$

The proof of this is an induction that relies on the fact that the coefficients b_{jk} and a_j are numbers with some specific values. Write η_j for the values of "ξ_j" in (12) you get starting with V instead of U. Assuming that F is Lipschitz continuous, the first stage of (12) satisfies

$$|\xi_1 - \eta_1| = \Delta t \, |F(U) - F(V)| \leq L_F \Delta t \, |U - V| \ .$$

The second stage satisfies

$$\begin{aligned} |\xi_2 - \eta_2| &= \Delta t \, |F(U + b_{11}\xi_1) - F(V + b_{11}\eta_1)| \\ &\leq \Delta t L_F \, |U - V + b_{11} \, (\xi_1 - \eta_1)| \\ &\leq \Delta t L_f \, (|U - V| + |b_{11}| \, |\xi_1 - \eta_1|) \\ &\leq \Delta t L_f \, (|U - V| + |b_{11}| \, \Delta t L_f \, |U - V|) \ . \end{aligned}$$

Since Δt is going to zero, we may Δt so small that $\Delta t \, |b_{11}| \, L_f \leq 1$. Then we get

$$|\xi_2 - \eta_2| \leq 2\Delta t L_f \, |U - V| \ .$$

This reasoning applies to the other stages. Finally, the update (13) satisfies

$$\begin{aligned} |G(U) - G(V)| &= \left| U - V + \sum_{j=1}^{s} a_j \, (\xi_j - \eta_j) \right| \\ &\leq |U - V| + \sum_j |a_j| \, |\xi_j - \eta_j| \\ &\leq |U - V| + CL_f \Delta t \, |U - V| \ . \end{aligned}$$

This is the stability inequality (26).

There are some important things to say about the stability inequality and its derivation. But first, here are the convergence proof and corresponding error estimate. Define the error bound C_T by

$$C_T = C \int_0^T e^{L_G t} \, dt \ . \tag{27}$$

We will show that

$$|U_k - U(t_k)| \leq \Delta t^p C_{t_k} \ .$$

The proof is by induction. It is true for $k = 0$ because $U_0 = U(t_0)$. Now we assume it for k and prove it for $k + 1$. We use the consistency bound (14) in the first step:

$$\begin{aligned} |U_{k+1} - U(t_{k+1})| &= |G(U_k, \Delta t) - (G(U(t_k), \Delta t) + \Delta t R_k)| \\ &\leq |G(U_k, \Delta t) - G(U(t_k), \Delta t)| + \Delta t \, |R_k| \\ &\leq \left[(1 + L_G \Delta t) \, C_{t_k} + \Delta t C \right] \Delta t^p \ . \end{aligned}$$

The integral (27) was chosen exactly so that

$$C_{t_{k+1}} \leq (1 + L_G \Delta t) \, C_{t_k} + \Delta t C \ .$$

To see this,

$$C_{t_{k+1}} = C_{t_k} + \int_{t_k}^{t_k + \Delta t}$$

Some comments on the proof and the stability estimate.

- The stability estimate we used in lecture 1 did not have the *grown term* $L_G \Delta t$. The heat equation does not have growth: $\|u(\cdot, t)\| \leq \|u(\cdot, 0)\|$. But solutions to differential equations can grow in time. The stability estimate here has to allow truncation errors R to be amplified as time progresses. It is essential for convergence that this growth rate remains bounded as $\Delta t \to 0$. A bounded growth rate (as $\Delta t \to 0$) is stability. If the growth rate for the numerical method goes to infinity as $\Delta t \to 0$, the numerical method would be unstable. The constants L_F, L_G, and C_T must be independent of Δt.

- (repeat, for emphasis) There is no error growth for our earlier discrete approximation to the heat equation, which is why we could have $\|U_{k+1}\| \leq \|U_k\|$ for the simple forward Euler in time, centered difference in space scheme. More complicated physical systems do have instabilities, which instabilities force the stability estimate to allow some error growth. In a time step of size Δt can allow growth by a factor of $(1 + C\Delta t)$ per time step. The proof above shows how that works.

- (for even more emphasis) There are chaotic dynamical systems that do have exponential amplification of small perturbations. If $|U(0) - V(0)| = \varepsilon$, then $|U(t) - V(t)| \sim \varepsilon e^t$. The *Lorenz attractor* is a famous example, as Wikipedia can explain.

- The differential equation (2) has a function $F(U) = AU$ with Lipschitz constant $L_F \sim \Delta x^{-2}$, because A is a finite difference approximation to ∂_x^2. In the convergence proof, we needed to assume that Δt is so small

that $L_f \Delta t$ is small. In our problem, that means Δt is at least as small as Δx^2. Generic time stepping methods for ODE systems (3) may or may not be good for a specific ODE system that comes from a semi-discrete approximation to a PDE.

4 Implicit methods

A time stepping method is *implicit* if you have to solve a system of equations to find U_{k+1}. Implicit methods are common, and commonly used to solve *stiff* systems of equations. An ODE system (2) is stuff if it has two properties:

1. There is a wide range of time scales in the problem. If λ_k are the eigenvalues of A, the corresponding solutions involve $e^{\lambda_k t}$, and the corresponding *time scale* is $\frac{1}{|\lambda_k|}$. The ratio of the fastest to slowest time scale is

$$\frac{\max |\lambda_k|}{\min |\lambda_k|} \ .$$

   If A is symmetric, this ratio is the condition number of A. It is commonly *an incorrectly* stated that a problem is stiff if A has a large condition number.

2. We want to solve (2) using a time step, Δt, that is large compared to the fastest time scale:

$$\Delta t \max |\lambda_k| \quad \text{is not small.} \tag{28}$$

   Be aware that if (28) is satisfied, then solutions $e^{\lambda_k t}$ will not be computed accurately. If we want large time steps (28) it is either because

   (a) we don't care whether we get the right answer, or

   (b) (more commonly) because the amplitude of the $e^{\lambda_k t}$ term is so small that it doesn't matter that we get it wrong.

   The semi-discrete heat equation is a typical example of situation 2(b). The fast time scale modes decay to zero so quickly (relative to the slow modes) that they are practically absent from the calculation most of the time. If the initial conditions are smooth, the fast decaying modes (which are the large k modes) were nearly absent from the beginning. We would like to take Δt small relative to the time scales present in the solution, but not small relative to time scales present in matrix A but absent in the solution we are calculating. If we do that with an explicit method, it will be unstable. High k modes will grow when they should decay. This is what makes a problem *stiff*.

The *backward Euler* method is a simple implicit method. It evaluates $\dot{U} = F(U)$ at time t_{k+1} instead of time t_k. The formal order of accuracy is still Δt:

$$U_{k+1} = U_k + \Delta t F(U_{k+1}) \ . \tag{29}$$

To do this scheme, you need a non-linear equation solver. Find V so that

$$V - \Delta t F(V) = U \ . \tag{30}$$

This is a system of d nonlinear equations for the unknown components of V. There are a lot of ideas out there on how to do this. Newton's method (from Numerical Methods I) is a good choice (if d is not too large) because the linear matrix $A(U) = F'(U)$ may be available, and because we often have a good initial guess. Here is the algorithm for one time step of the basic Newton/backward Euler method:

1. You have U_k, U_{k-1}, $\cdots$. Estimate U_{k+1} using either:
   (a) Constant "extrapolation": $V_0 = U_k$
   (b) Linear extrapolation: $V_0 = 2U_k - U_{k-1}$
   (c) Higher order extrapolation
   (d) Forward Euler: $V_0 = U_k + \Delta t F(U_k)$.
   (e) Something else
2. Solve the equations (30) using Newton's method. Set $j = 0$.
   (a) Evaluate the residual of the current approximation:
   $$R_j = V_j - \Delta t F(V_j) - U \ .$$
   (b) Evaluate the residual norm $r_j = \|R_j\|$. This can be the most subtle step, possibly because different components of R_j have different units (angles, positions, velocities, etc.).
   (c) If $r_j < \varepsilon$, stop and set $U_{k+1} = V_j$.
   (d) Calculate the next Newton iterate. Evaluate the linearization matrix $A_j = F'(V_j)$ and solve the linear system of equations
   $$(I - \Delta t A_j)\,(V_{j+1} - V_j) = R_j \ .$$
   This is the main work of the method.
   (e) Increment j (`j += 1` or `j++;`) and go to step (a).

Newton's method has local quadratic convergence. This means that if you have a good initial guess, you "converge" (satisfy your convergence criterion from step 2(c)) in just one or two iterations. Just one Newton iteration gives the linear stability properties of the fully implicit method. Be aware that the matrix $A = F'(U)$ may have Δx or Δx^2 in the denominator. Therefore, $\Delta t A$ may not be small. In fact, if $\Delta t A$ is small, then you probably don't need an implicit method[3].

[3]This remark is aimed at all those theory people who prove theorems about implicit methods using the hypothesis that $\Delta t A$ is small.

Lecture 4 will have more on how to chose an implicit method that's good for a specific problem. Until then, here is some terminology. A fully implicit Runge Kutta method with s stages would ask you to solve a system of sd equations for the s stage values with d components each. A *DIRK* (for "diagonally implicit Runge Kutta") method is a Runge Kutta method where the stages are implicit. That means you have to do s solves of problems of size d.

The implicit Adams methods are called *Adams Moulton*. The Adams Moulton method with r lags is one order of accuracy higher than the Adams Bashforth method of the same order. For example, consider the method with just two times t_k and t_{k+1}. The explicit method is forward Euler. The implicit method integrates the linear interpolant between $F(U_k)$ and $F(U_{k+1})$. The "area" of that "trapezoid" is $\Delta t \left(F(U_{k+1}) + F(U_k)\right)$. The resulting two point Adams Moulton is

$$U_{k+1} = U_k + \frac{\Delta t}{2} \left(F(U_{k+1}) + F(U_k)\right) \ .$$

This is the trapezoid rule we discussed before.

5 Adaptive methods

You have to make many choices to solve a specific practical problem: explicit/implicit, order of accuracy, Δt. The best choice depends on the behavior of the solution, which you (a) don't know in advance, and (b) can change from one time to another in a single solve. *Adaptive methods* are higher level algorithms that automate some of these choices, depending on the problem and/or the solution.

Here is an example that adjusts the time step of a Runge Kutta method. Suppose $G(U, \Delta t)$ is a Runge Kutta method that has order of accuracy r. Adaptive time steps are not all the same size, so we have time step sizes Δt_k and times $t_{k+1} = t_k + \Delta t_k$. The solution is advanced using those time steps: $U_{k+1} = G(U_k, \Delta t_k)$. These approximate the true solution at times t_k, which means $U_k \approx U(t_k)$. Our goal is time step sizes Δt_k that are small enough to achieve a desired target accuracy, but not a lot smaller than that. In the spirit of one-step (but multi-stage) Runge Kutta methods, Δt_k and U_{k+1} are functions of the data available at the start of time step k. That means U_k and Δt_{k-1}.

The motion of a comet is a problem that may explain the value of adaptive methods. Haley's comet (see Wikepedia). This comet spends a few months every 76 years in the inner solar system, during which time it's direction and speed changes noticeably from week to week. But it spends the majority of it's 76 year cycle much farther from the sun moving slowly and changing speed slowly. It is possible to compute the majority of its orbit using time steps that are far too big to get the inner approach accurately. An adaptive method can deliver comparable accuracy using a fraction of the number of time steps needed for an accurate solution using fixed time steps.

Error equidistribution is one approach to adaptive step size control. For each Δt_k there is a residual $\Delta t R_k = G(U_k, \Delta t_k) - S(U_k, \Delta t_k)$. Equidistribution

means that we choose Δt_k so as to keep the rate of error production roughly a constant, small, predetermined value, ε. Of course, we need to estimate R_k, because the exact R_k is not available.

It is possible to estimate $R(U, \Delta t)$ using *Richardson extrapolation*. Suppose $G(U, \Delta t)$ is a Runge Kutta method of order p. The derivations above show that $R(U, \Delta t)$ has an asymptotic expansion in powers of Δt of the form

$$R(U, \Delta t) = G(U, \Delta t) - S(U, \Delta t) = R_p \Delta t^p + O(\Delta t^{p+1}) \ . \tag{31}$$

You can estimate R_p by comparing the result of one time step with size Δt with two time steps of size $\frac{1}{2}\Delta t$. One time step of size Δt gives

$$U \overset{\Delta t}{\rightsquigarrow} V_1 = S(U, \Delta t) + \Delta t^{p+1} R_p + O(\Delta t^{p+2}) \ .$$

For the half step calculation, we change in U in a half time step as

$$\Delta U = G(U, \tfrac{1}{2}\Delta t) - U = S(U, \tfrac{1}{2}\Delta t) - U + O(\Delta t^{p+1})$$

Using two half size time steps gives

$$\begin{aligned} U &\overset{\frac{1}{2}\Delta t}{\rightsquigarrow} S(U, \tfrac{1}{2}\Delta t) + 2^{-(p+1)} \Delta t^{p+1} R_p + O(\Delta t^{p+2}) \\ &\overset{\frac{1}{2}\Delta t}{\rightsquigarrow} S(U + \Delta U, \tfrac{1}{2}\Delta t) + 2^{-(p+1)} \Delta t^{p+1} R_p + 2^{-(p+1)} \Delta t^{p+1} R_p + O(\Delta t^{p+2}) \ . \end{aligned}$$

I claim that we can replace $S(U + \Delta U, \frac{1}{2}\Delta t)$ with $S(U, \Delta t)$ within this level of approximation. Assuming this is true, two half steps give

$$U \overset{2\times\frac{1}{2}\Delta t}{\rightsquigarrow} V_2 = S(U, \Delta t) + 2^{-p} \Delta t^{p+1} R_p + O(\Delta t^{p+2}) \ .$$

This leads to the Richardson error estimate

$$R_{\text{est}} = \frac{1}{(1 - 2^{-p})\, \Delta t} \left(V_1 - V_2 \right) \ . \tag{32}$$

- The denominator on the right is not zero because the method is at least first order, $p \geq 1$.
- We used the relation $R \approx \Delta t^p R_p$. This gave $\Delta t^{p+1} R_p \approx \Delta t R$.

The adaptive time step algorithm can go like this:

1. Choose parameters
   - Δt (initial guess of time step at starting time)
   - U_0, $t = 0$ (initial conditions)
   - T (final time)
   - ε (desired truncation error rate)
   - m (number of time steps between adjusting Δt).

2. (outer loop) Calculate V_1 using one Δt time step and V_2 using two $\frac{1}{2}\Delta t$ time steps. Use the formula (32) to estimate R. Choose Δt to achieve truncation rate ε, which is

$$\|R_{\rm est}\|\,\Delta t^p = \varepsilon \qquad \Longrightarrow \qquad \Delta t = \left(\frac{\varepsilon}{\|R_{\rm est}\|}\right)^{1/p} . \tag{33}$$

3. Take m time steps with this Δt.

The error estimation step 2. uses three time steps that ultimately are not used. Therefore, we re-estimate the time step only after m steps. If $m = 6$, for example, there is 50% overhead from error estimation. There are more sophisticated adaptive strategies that have less overhead. One example is the code `RKF45`, which is based on the Runge Kutta Fehlberg method. Adams methods are often used adaptively, because the order of accuracy does not depend on the numbers Δt_k all being the same.

What I want you to get from this section on adaptive methods:

- The best algorithm for a problem is not necessarily very simple. Doing more complicated things, adaptivity in this case, can make a computational algorithm much more accurate, efficient, and reliable.

- Adaptive strategies do not have to be heuristic (as they often are when done by un-lisenced numerical computing "experts"). The best adaptive strategies are based on mathematical reasoning just as the best fixed Δt strategies are.

- Because of 1. and 2., the best strategy for you, if you have a generic problem, is to use a high quality ODE solver you download rather than writing one yourself. People have spend many more man/woman years developing these codes than you want to spend.