Math 128A, Summer 2019

Lecture 20, Thursday 7/25/2019

1 Review

Strain opens with an example of how we read off a Butcher array. Suppose our roommate 'discovers' a new Runge-Kutta method, say:

$$\begin{aligned} u_{n+1} &= u_n + h\left[\frac{2}{3}\underbrace{f}_{k_3}\left(u_n + \frac{h}{3}\underbrace{f}_{k_2}\left(u_n + \frac{h}{3}\underbrace{f}_{k_1}(u_n)\right)\right) + \frac{1}{3}\underbrace{f}_{k_4}(u_{n+1})\right] \\ &= u_n + h\left(\frac{2}{3}k_3 + \frac{1}{3}k_4\right) \end{aligned}$$

WLOG we number the k's in the order, although the Butcher array could easily be permuted.
The butcher array here is

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & \frac{2}{3} & \frac{1}{3} \\ 0 & 0 & \frac{2}{3} & \frac{1}{3} \end{bmatrix}$$

Notice that the last row of the butcher array (b) is the same as the last row of the matrix.
Also notice the first step of the new step is the same as the last step of the previous step, so we don't actually take 4 computations, only 3. We call this 'First same as last'.
We write down the Butcher array so we can check the order conditions. It helps us not have to taylor expansion the f's a ton of times.
Recall from yesterday's lecture, the first (order) condition is:

$$\sum b_i = 1$$

and the second (order) condition is:

$$b^T A e = \frac{1}{2}$$

$c = Ae$ is simply the row-sums of the Butcher matrix; in our example above, we have:

$$c = Ae = \begin{bmatrix} 0 \\ \frac{1}{3} \\ \frac{1}{3} \\ 1 \end{bmatrix}$$

Then we check second-order accuracy:

$$b^T Ae = \begin{bmatrix} 0 & 0 & \frac{2}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 0 \\ \frac{1}{3} \\ \frac{1}{3} \\ 1 \end{bmatrix}$$

We only have first-order accuracy. But we invent reasons why this is better than existing methods, such as if we run on some trinary computer (as opposed to binary), our method would be exact.

2 Introduction: Deferred Correction

We used to talk about both extrapolation and deferred correction, but we found out that extrapolation isn't very good.
arbitrary-order
The idea is that given a low-order Runge-Kutta method for

$$y' = f(t, y) \qquad \to \qquad u_n \ \text{ approx solution}$$

that we derive an ODE for the error. But before this, we need a way to go from u_n to $u(t)$. To do this, we interpolate to get a differentiable function, $u(t)$.

Then, we get an ODE for error

$$e(t) = y(t) - u(t).$$

Next we solve by low-order method for e_n. Finally, we correct our solution given our found error:

$$u_n^1 = u_n + e_n$$

Hence we have a 'deferred' correction where we have an approximate solution, then subtract out the error at the end.
Essentially, we apply a low-order method (say first-order) to each of the solution and error, yielding a higher-order (second-order) result. We can iterate back up to Interpolation to the end, repeatedly.

2.1 Autonomizing: Suppressing t

$$\begin{aligned} y' &= f(t, y) \\ Y' := \begin{bmatrix} t \\ y \end{bmatrix}' &= \begin{bmatrix} 1 \\ y' \end{bmatrix} = F(Y) \end{aligned}$$

This is called 'autonomizing' the system, where we do not have t. Compare against $y' = f(t, y)$, which is a non-autonomous system.

3 Euler's Method

(1)

$$u_{n+1} = u_n + hf(t_n, u_n)$$

(2) We have a function and interpolate it with some polynomial $u(t)$ of some degree, say $n+1$ points.
(3) Now we want an ODE for error, $e(t) := y(t) - u(t)$. We know how to differentiate at all the mesh points; that is,

$$\begin{aligned} e'(t) &= y'(t) - u'(t) \\ &= f(t, y(t)) - u'(t) \end{aligned}$$

We rewrite: $e(t) = y(t) - u(t)$ to get $y(t) = u(t) - e(t)$ to get a differential equation from the above:

$$\begin{aligned} e'(t) &= f(t, u(t) - e(t)) - u'(t) \\ &= g(t, e(t)) \end{aligned}$$

Then (4) we use Euler's method:

$$\begin{aligned} e_{n+1} &= e_n + hg(t_n, e_n) \\ &= e_n + h\left[f(t_n, u_n + e_n) - u'(t_n)\right] \end{aligned}$$

and then (5), we have

$$u_n^1 = u_n + e_n,$$

and we ought to have a higher-order accuracy.

3.1 Example of Euler's Method

We work out the smaller details. Suppose we interpolate at points $n, n+1, \ldots, n+p$. We have a bunch of values, so we put a polynomial through it. Actually, we never need the polynomial itself; we need only its derivatives.
(1)

$$u_{n+j+1} = u_{n+j} + hf(t_{n+j}, u_{n+j})$$

For $p = 2$, we have:

$$u(t_{n+j}) = u_{n+j}, \qquad 0 \le j \le p = 2$$

This gives a quadratic polynomial (3 points). We have $j = 0, 1, 2$.
We say:

$$u'(t_{n+j}) = d_{j,0}u_n + d_{j,1}u_{n+1}d_{j,2}u_{n+2}.$$

Let's start with the best initial value and not correct it, only correcting our subsequent values (we accept the starting error and want to move forward as opposed to correcting the past). Hence:

$$\begin{aligned} e_n &:= 0 \\ e_{n+1} &= e_n + h\left[\overbrace{f(t_n, u_n + e_n)}^{d_0} - u'(t_n)\right] \\ e_{n+2} &= e_{n+1} + h\left[\underbrace{f(t_{n+1}, u_{n+1} + e_{n+1})}_{d_1} - u'(t_{n+1})\right] \end{aligned}$$

This gives:

$$\begin{aligned}
u^1_{n+1} &= u_{n+1} + e_{n+1} \\
u^1_{n+2} &= u_{n+2} + e_{n+2} \\
&= u_{n+1} + hf(t_{n+1}, u_{n+1}) + e_{n+1} + h\left[f(t_{n+1}, u_{n+1} + e_{n+1}) - u'(t_{n+1})\right] \\
&= u_n + hf(t_n, u_n) + hf(t_{n+1}, u_{n+1}) + h\left[f(t_n, u_n) - u'(t_n)\right] \\
&\quad + h\left[f\left(t_{n+2}, u_{n+1} + h\left[f(t_n, u_n) - u'(t_n)\right]\right) - u'(t_{n+1})\right]
\end{aligned}$$

where u^1_{n+2} is the final corrected second-order solution u_{n+2} at $t_{n+2} = t_n + 2h$.
We define the k's:

$$\begin{aligned}
k_1 &:= f(t_n, u_n) \\
k_2 &:= f(t_{n+1}, u_n + hk_1) \\
k_3 &:= f\left(t_{n+1}, u_n + hk_1 + h\left(k_1 - u'(t_k)\right)\right) \\
&= f\left(t_{n+1}, 2hk_1 - hu'(t_n)\right)
\end{aligned}$$

So we have:

$$\begin{aligned}
u'(t_n) &= d_{0,0}u_n + d_{0,1}u_{n+1} + d_{0,2}u_{n+2} \\
&= \left(d_{0,0} + d_{0,1} + d_{0,2}\right)u_n \\
&\quad + (d_{0,1} + d_{0,2})hf(t_n, u_n) \\
&\quad + d_{0,2}hf(t_{n+1}, u_{n+1})
\end{aligned}$$

but $d_{0,0} + d_{0,1} + d_{0,2} = 0$ because these are the coefficients of the result after differentiating. We define (as we did for Euler-Maclaurin):

$$\begin{aligned}
d_{0,0} &= \frac{-3}{2} \\
d_{0,1} &= 2 \\
d_{0,2} &= -\frac{1}{2}
\end{aligned}$$

Then this gives:

$$k_3 = f\left(t_{n+1}, u_n + h\left[(2 - d_{0,1} - d_{0,2})k_1 + d_{0,2}k_2\right]\right)$$

and by the same tokens as above, we have:

$$\begin{aligned}
u_{n+2} &= u_n + hk_1 + hk_2 + h(k_1 - u'(t_n)) + h[d_3 - u'(t_{n+1})] \\
&= u_n + hk_1 + hk_2 + hk_1 - (d_{0,1} + d_{0,2})h^2k_1 - d_{0,2}h^2k_2 \\
&\quad + hk_3 - (d_{1,1} + d_{1,2})h^2k_1 - d_{1,2}h^2k_2
\end{aligned}$$

We read off the Butcher array:

$$\begin{bmatrix}
0 & 0 & 0 \\
1 & 0 & 0 \\
2 - h(d_{0,1} + d_{0,2}) & -hd_{0,2} & 0 \\
2 - h(d_{0,1} + d_{0,2}) - h(d_{1,1} + d_{1,2}) & 1 - hd_{0,2} - hd_{1,2} & 1
\end{bmatrix}$$

Because this is an explicit scheme, we have zeroes on and above the diagonal.

Recall that we need to divide by two at the end (to standardize), because we are taking a step size $2h$, and the definition of the Butcher array uses a step size of h.
The point of the Butcher array is that the numbers within are independent of f, u, h.
Letting $d_{1,0} := \frac{-1}{2}, d_{1,1} := 0, d_{1,2} := \frac{1}{2}$, as we normally do for divided difference, we have our butcher array:

$$\begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Checking second-order accuracy, we have:

$$\sum b_i = 0 + \frac{1}{2} + \frac{1}{2} = 1$$
$$\sum b_i c_i = 0 \cdot 0 + \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2},$$

so we conclude we have second-order accuracy.
Breaktime.

4 (Iterated) Deferred Correction

We have a first-order solution,

$$\begin{aligned} u^0_{n+j+1} &= u^0_{n+j} + hf(t_{n+j}, u^0_{n+j}), \qquad 0 \le j \le p-1 \\ u^0_n &= u_n \\ \implies u_n(t_{n+j}) &= u_{n_j} \\ e_{n+j+1} &= e_{n+j} + h[f(t_{n+j}, u_{n+j} + e_{n+j}) - u'_n(t_{n+j})] \end{aligned}$$

We define our corrected solution to be:

$$\impliedby u^1_{n+j} = u^0_{n+j} + e_{n+j}, \qquad 1 \le j \le p$$

The limit to this iterative scheme is the interpolating polynomial; at some point, the interpolation will no longer correct anything. This iterative scheme $u^k \mapsto u^{k+1}$ looks like fixed point iteration.
Our first question is will this fixed-point iteration converge? Our zeroth question is if this converges, what will it net us?
We notice that if we have convergence, our error at each step (to be corrected) is zero; hence $e_{n+j+1} = e_{n+j} = 0$, hence the rest of the equation must be zero:

$$0 = h[f(t_{n+j}, u_{n+j} + e_{n+j}) - u'_n(t_{n+j})],$$

and at the limit, we have:

$$f(t_{n+j}, u_{n+j}) - u'_n(t_{n+j}) = 0, \qquad 0 \le j \le p-1$$

Remark: This is to say that the interpolating polynomial exactly satisfies the differential equation at the interpolating points.

We let d_i be δs times h, and we get:

$$\begin{aligned} hu'_n(t_{n+j}) &= d_{j,0}u_n + d_{j,1}u_{n+1} + \cdots + d_{j,p}u_{n+p} \\ hf(t_{n+j}, u_{n+j}) &= d_{j,0}u_n + \cdots + d_{j,p}u_{n+p}. \end{aligned}$$

This is a (fully) implicit scheme with p stages (where dependencies require us to solve everything at once), but our iterated scheme has p^2 stages; however, it is explicit.
Our second line is a Runge-Kutta method; however, it may not look like it. It expresses the stages in terms of the new values, whereas usually we express values in terms of the new stages. Define $k_j := f(t_{n+j}, u_{n+j})$. To get around this,

$$\begin{aligned} hk_j &= d_{j,0}u_n + \cdots + d_{j,p}u_{n+p} \\ &= d_{j,0}u_n + d_{j,1}(u_{n+1} - u_n + u_n) + \cdots + d_{j,p}(u_{n+p} - u_n + u_n) \\ &= d_{j,1}(u_{n+1}, u_n) + d_{j,2}(u_{n+2} - u_n) + \cdots + d_{j,p}(u_{n+p} - u_n), \end{aligned}$$

and this gives us:

$$\begin{aligned} h \begin{bmatrix} k_1 \\ \vdots \\ k_p \end{bmatrix} &= \begin{bmatrix} d_{1,1} & \cdots & d_{1,p} \\ \vdots & & \vdots \\ d_{p,1} & \cdots & d_{p,p} \end{bmatrix} \begin{bmatrix} u_{n+1} - u_n \\ \vdots \\ u_{n+p} - u_n \end{bmatrix} \\ hk &= D(u_{n+j} - u_n) \end{aligned}$$

and let $C := D^{-1}$, which then gives us:

$$\begin{bmatrix} u_{n+1} - u_n \\ \vdots \\ u_{n+p} - u_n \end{bmatrix} = hC \begin{bmatrix} k_1 \\ \vdots \\ k_p \end{bmatrix},$$

which looks a lot more like a Runge-Kutta method.

Remark: So in summary, iterated differed correction is a way to solve fully implicit Runge-Kutta methods.

Proving convergence of Runge-Kutta methods is incredibly difficult, so we skip over the proof.

Next week, we'll talk about stiff equations and multistep methods.