Math 128A, Summer 2019
Lecture 22, Tuesday 7/30/2019

1 Review: Homework

```
1 u_{n+j+1} = u_{n+j} + hf(t_{n+j}, u_{n+j})
2 0 \leq j \leq p - 1
3 e_{n+j+1} = e_{n+j} + h \[ f(t_{n+j}, u_{n+j} + e_{n+j}
4     + e_{n+j}) - u'(t_{n+j}) \]
5 0 \leq j \leq p - 1
6 e_n = 0
7 u_{n+j}^{1} = u_{n+j} + e_{n+j}
```

Then once we have line 7, u_{n+j}, we stick it back into line 3, u_{n+j}. We solve for the error and add it onto our approximation, then solve for error again and repeat. We don't change u_n over the different passes.
Our problem with the orbit is about the natural (perpetual) figure-8 orbit around the earth and the moon.

2 Review Lecture 21: Multi-Step Methods

These may not have the same charm as Runge-Kutta methods, perhaps because they're truly simple. We take:

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} p(s)\, ds,$$

where we have the general practice of replacing things we don't know with things we **do** know, which is the interpolating polynomial $p(s)$. Effectively, we interpolate some polynomial through points $t_{n+1-k}, \ldots, t_n$ and then integrate past our interval, over $[t_n, t_{n+1}]$ which is not illegal and grants us an estimate (k-step explicit scheme).

Alternatively, we can put $q(s)$ which also interpolates at t_{n+1}, so we are integrating within our interpolating interval ($k-1$-step implicit scheme), and this ought to be more accurate even when using the same number of points, $t_{n+2-k}, \ldots, t_n, t_{n+1}$.

$$\begin{aligned} v_{n+1} &= u_n + h_n \sum_{j=0}^{k-1} p_j f_{n-j} \\ u_{n+1} &= u_n + h_n \left[q_{-1} f(t_{n+1}, v_{n+1}) + \sum_{j=0}^{k+2} q_j f_{n-j} \right], \end{aligned}$$

where we explicitize the implicit scheme in the second line by taking the problematic term p_j by using the explicit scheme as a predictor and the implicit scheme as a corrector.

If we feed the exact solution y into the above, we have:

$$y_{n+1} = y_n + h_n \sum_{j=0}^{k-1} p_j y'_{n-j} + h_n \tau^p$$

$$y_{n+1} = y_n + h_n \left[q_{-1} f(t_{n+1}, y_{n+1}) + \sum_{j=0}^{k+2} q_j y'_{n-j} \right] + h_n \tau^q,$$

now with the local truncation errors.
To get this result, Strain ponders:

$$v_{n+1} = y_n + \int_{t_n}^{t_{n+1}} p(s)\ ds = y_{n+1} - h\tau^p$$

$$u_{n+1} = y_n + \int_{t_n}^{t_{n+1}} q(s)\ ds = y_{n+1} - h\tau^q,$$

and subtracting these equations gives

$$\begin{aligned} \frac{v_{n+1} - u_{n+1}}{h} &= \tau^p - \tau^q \\ &= (p_k - q_k)\, h^k \end{aligned}$$

However, subtracting these may return 0, which does not give us any insight into how large each of them are.
This is why we are using one step less than our original explicit scheme, so we are the same order of accuracy from the true solution.
To see that $p_k - q_k \neq 0$, we look at:

$$\int_{t_n}^{t_{n+1}} \underbrace{t - t_{n+1}}_{q} \underbrace{[(t-t_n)\cdots(t-t_{n-k+2})]}_{p(t)}\ dt$$

Hence we conclude:

Remark:

$$q_k h^k = \tau^q = \frac{q_k}{p_k - q_k} \frac{v_{n+1} - u_{n+1}}{h}$$

estimates error in the corrected solution, with step size control.

Suppose we want error $\leq$ tolerance. We just took step $t_n \to t_{n+1}$ and error estimate $E = q_k h^k$.

(1) If $|E| \leq tol$, then we want to increase the next step (we didn't waste computation time, but we may have taken too small a step and consequently may require too many steps; we may not get the solution by the time we need it). We increase by some factor, say

$$\theta := \min \left\{ 1 + \frac{1}{k}, 0.9 \left(\frac{tol}{|E|} \right)^{1/k} \right\}; \qquad h_{n+1} := h_n \theta$$

(2) If $|E| \geq tol$, we must reject our step and re-do that step with a smaller step-size, say $h_{n+1} := \theta h_n$ with:

$$\theta := \max \left\{ 0.1, \left(\frac{tol}{|E|} \right)^{1/k} \right\}$$

If we look at the textbook GGK, we can see that adaptive stepsize control is ridiculously more efficient and better-performing than equidistant. For our purposes, in the interest of time (summer), we will not be asked to programatically implement adaptive stepsize control, as it is difficult to debug. We can get about 9-digit accuracy with an equidistant Adams predictor-corrector.

3 Stability: The Second Piece in Proving Convergence

Let's suppose we have k initial values, $u_0, \ldots, u_{k-1}$ to some degree of accuracy. How do we prove that u_n converges to y_n on some fixed interval of time?

Via extrapolating, we have:

$$u_{n+1} = u_n + h\left[p_0 f_n + \cdots + p_1 f_{n-k+1}\right]$$

As we recall (lecture 18), there are two ingredients: (1) consistency in that $\tau = O(h^k)$ and our method converges to the equation, and (2) stability in that as $h \to 0$, errors don't explode.
We checked already that $\tau_n = O(h^k)$ via:

$$y_{n+1} = y_n + h\left[p_0 f(t_n, y_n) + \cdots + p_{k-1} f(t_{n-k+1}, y_{n-k+1})\right] + h\tau_n$$

and additionally for u,

$$u_{n+1} = u_n + h\left[p_0 f(t_n, u_n) + \cdots + p_{k-1} f(t_{n-k+1}, u_{n-k+1})\right] + h\tau_n$$

and subtracting these,

$$|e_{n+1}| \le |e_n| + h\left(|p_0|\underbrace{|f(y_n) - f(u_n)|}_{L|y_n - u_n| = L|e_n|} + \cdots + |p_{k-1}|\ \underbrace{|f(y_{n-k+1} - f(u_{n-k+1})|}_{L|y_{n-k+1} - u_{n-k+1}| = L|e_{n-k+1}|}\right) + h|\tau_n|.$$

How about bounding these by the largest error?

$$E_n := \max_{0 \le j \le n} |e_j|$$

This gives

$$\begin{aligned} |e_{n+1}| &\le \left[1 + h\underbrace{(|p_0| + \cdots + |p_{k-1}|)L}_{\text{indep. of } h}\right] E_n \\ &\le E_{n+1} \quad \text{(to see this, consider two cases:)} \end{aligned}$$

If $|e_{n+1}| \le E_n$, then $E_{n+1} = E_n \le RHS$ (this case is not very likely). On the other hand, if $|e_{n+1}| > E_{n+1} \le RHS$.

Notice this is just like Euler's method, where

$$E_n \le \frac{e^{pLnh} - 1}{hL}\left(h\tau - E_k\right)$$
$$p = |p_0| + \cdots + |p_{k-1}|$$

So as $h \to 0$, we have $E_n \to 0$ if:
(1) $\tau = O(h^k)$ which we already know, and
(2) $\frac{1}{h}E_{k-1} \to 0$, $\frac{1}{h}(u_0 - y_0) \to 0$, $\ldots$, $\frac{1}{h}(u_{k-1} - y_{k-1}) \to 0$. We compute these guys via Taylor expansion or a Runge-Kutta method with fewer steps. Notice that $\frac{1}{h} \to +\infty$, so the factor $(u_j - y_j)$ must go down faster to offset this.

4 Getting Started (Programatically)

For $n = 0 \to n = 1$, we have to use a 1-step method, say Euler-predictor:

$$\begin{aligned} v_n &= u_0 + hf(t_0, u_0) \quad \text{(predictor)} \\ u_1 &= u_0 + hf(t_1, v_1) \quad \text{(implicit Euler, corrector)} \end{aligned}$$

The problem has $O(h_0^2)$ error.
Eventually, we want $O(h^k)$ error.

$$h_0^2 = h^k \implies h_0 = T^{1-(k/2)}h^{k/2} << h$$

We have options. We can keep the time-step the same and increase the order. Alternatively, with a low-order method, we can double the time-step without doing any harm (this is a small win); hence we can increase the order and time-step simultaneously. For our purposes, we can just keep the step-size the same:

$$\begin{aligned} v_2 &= u_1 + h_0\left[\frac{3}{2}f_1 - \frac{1}{2}f_0\right] \\ u_2 &= u_1 + h\left[\frac{1}{2}f(t_2, v_2) + \frac{1}{2}f_1\right] \\ f_2 &= f(t_2, u_2) \\ &\vdots \\ f_{k-1} &= \cdots \end{aligned}$$

When we evaluate $f(t_1, v_1)$ for u_1, we need to compute and store $f_1 = f(t_1, u_1)$ as the predictor. We predict, evaluate, correct, then evaluate (PECE).
We use a 2-step method for the predictor v_2, so we use 1-step for u_2. Our time-steps are accurate to order h^{k+1} accuracy. The idea is that the process of getting started (with small steps), then gradually increasing our step-size, trying our steady increase θ from earlier until at some point, we want to check how much distance we have yet to cover and how much time we have to compute, then to re-define h so that we march along.
Essentially we might as well be coding a multi-step methods, but Strain doesn't want a riot!

5 Stiff Equations

In stiff equations, like weather, we have wind and chemistry that happens much faster than wind, and even smaller contributions such as fires.
We have a stiff equation in that if we follow the many events over so much detail over geological times, we can't use anything like multistep methods. It turns out, Runge-Kutta methods generally don't work either. We need to use methods specifically designed for stiff equations!

$$\begin{aligned} y_1'(t) &= -y_1(t) \\ y_2'(t) &= -100y_2(t) \end{aligned}$$

Euler's method here won't be too bad for y_1; however, for this same time-step, Euler's method will get worse and worse for y_2. To see this,

$$\begin{aligned} u_{n+1} &= u_n - hu_n = (1-h)u_n \\ v_{n+1} &= v_n - 100hv_n = (1-100h)v_n = -v_n \end{aligned}$$

and we say $h \approx 0.02$. If we had a factor 10000 instead of 100, our timestep h has to 'chase' the smallest unit.
The **cure** for stiff equations is to use implicit methods!

$$\begin{aligned} u_{n+1} &= u_n - hu_{n+1} \\ v_{n+1} &= v_n - 100hv_{n+1} \\ (1+100h)v_{n+1} &= v_n \\ v_{n+1} &= \frac{1}{1+100h}v_n \end{aligned}$$

In general, we'll have to solve nonlinear equations (as opposed to the linear solution here).
The problem is now that we have to solve an implicit (i.e. Euler) equation, say:

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

We solve this in one of three ways (the Holy Trinity): Bisection, Fixed Point, Newton. This is probably a system in that u_n has many components that we cannot simultaneously find all at once. Hence we consider the latter two.
Fixed point says:

$$\begin{aligned} v = g(v) &= v_n - 100hv \\ g'(v) &= -100h, \end{aligned}$$

and we have $|g'(v)| \leq \frac{1}{2}$ if $h \leq \frac{1}{200}$, which is exactly what we were trying to avoid in our first example. Hence we're left with Newton, and it better work!
Newton delivers us exactly:

$$v_{n+1} = \frac{1}{1+100h}v_n$$

Now, the problem is that for Newton, we need f', and if this is a system, then we need the Jacobian matrix $Df(u)$ (it's a fact, we need this, unless we cheat via differentiation coefficients or interpolation). Else, we can take a course on optimization to solve this without knowing the derivative.

Lecture ends here.

We'll talk more about Stiff equations this week, as they occur a lot in real life. Next week we'll get started with Linear Algebra.