Week 11: aim to cover

- Derivation of RK methods
- Linear stability of RK methods
- Variable time-step RK methods `ode23`, `ode45`
- Other MATLAB solvers (brief)

Systematic derivation

Are there any more such methods? Are they really 2nd order?
We look for explicit 2-stage methods of the form:

$$s_1 = f(t_n, y_n) \tag{1}$$

$$s_2 = f(t_n + c_2 h, y_n + h a_{21} s_1) \tag{2}$$

$$y_{n+1} = y_n + h(b_1 s_1 + b_2 s_2) \tag{3}$$

which is displayed in a Butcher tableau, after J.Butcher (Auckland)

$$\begin{array}{c|cc} 0 & 0 & 0 \\ c_2 & a_{21} & 0 \\ \hline & b_1 & b_2 \end{array}$$

Extend Taylor series

To find conditions for 2nd order consistency, match the local error from the Taylor series starting from $y(t_n) = y_n$:

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{1}{2}h^2 y''(t_n) + O(h^3)$$

$$y(t_{n+1}) = y_n + hf(t_n, y_n) + \frac{1}{2}h^2 \frac{d}{dt} f(t, y(t)) \mid_{t_n} + O(h^3)$$

$$y(t_{n+1}) = y_n + hf(t_n, y_n) + \frac{1}{2}h^2 [f_t + f_y y'] \mid_{t_n} + O(h^3)$$

$$y(t_{n+1}) = y_n + hf(t_n, y_n) + \frac{1}{2}h^2 [f_t + f_y f] \mid_{t_n} + O(h^3)$$

2-stage method

Now compare with our 2-stage method: (expand about (t_n, y_n))

$$s_1 = f_n \equiv f(t_n, y_n)$$

$$s_2 = f_n + f_t c_2 h + f_y h a_{21} s_1 + O(h^2)$$

$$y_{n+1} = y_n + h(b_1 s_1 + b_2 s_2)$$

so

$$\begin{aligned} y_{n+1} &= y_n + hb_1 f_n + hb_2(f_n + f_t c_2 h + f_y h a_{21} s_1 + O(h^2)) \\ &= y_n + h(b_1 + b_2) f_n + h^2 (b_2 c_2) f_t + h^2 b_2 a_{21} f_y f_n + O(h^3) \end{aligned}$$

For this to match the Taylor series to $O(h^3)$

$$y(t_{n+1}) = y_n + hf_n + \frac{1}{2} h^2 [f_t + f_y f] \mid_{t_n} + O(h^3)$$

need to match terms, which gives ...

Order conditions

$$
\begin{aligned}
b_1 + b_2 &= 1 \\
b_2 c_2 &= \frac{1}{2} \\
b_2 a_{21} &= \frac{1}{2}
\end{aligned}
$$

We have 3 equations in 4 unknowns $\implies$ a 1-parameter family of methods: Let $c_2 = \alpha$

$\implies a_{21} = \alpha, b_2 = 1/(2\alpha), b_1 = 1 - 1/(2\alpha)$

Any such method is consistent of order 2, by construction

but disagrees with Taylor series at next term, so only 2nd order

RK2

$$
\begin{aligned}
s_1 &= f(t_n, y_n) && (1)\\
s_2 &= f(t_n + \alpha h, y_n + \alpha h s_1) && (2)\\
y_{n+1} &= y_n + h\left(\left(1 - \frac{1}{2\alpha}\right)s_1 + \frac{1}{2\alpha}s_2\right) && (3)
\end{aligned}
$$

Any such method is called a second order (explicit) Runge-Kutta method **RK2**

Example

- $\alpha = 1/2 \to$ explicit midpoint method
- $\alpha = 1 \to$ modified Euler or explicit trapezoid method
- $\alpha = 1/3 \to$ RK2 with lowest local error (Heun)

Convergence of RK methods

By construction, our RK2 methods are consistent of order 2
Recall the Big Theorem

Consistency + 0-Stability → Convergence

Luckily, all RK methods are 0-stable and so

All RK methods are convergent

hence RK2 are 2nd order convergent methods
But what about numerical stability (behaviour at finite h)?

Linear stability

While convergence proofs are comforting, we actually run ODE codes at finite h. We want numerical solution to have damped errors, when the true solutions are contractive i.e. $J < 0$.
The simplest theory for this is **Linear stability**
Consider an autonomous linear system

$$\mathbf{y_1}' = \mathbf{A}\mathbf{y_1} + \mathbf{b}(t);\ \ \mathbf{y_1}(0) = \mathbf{y}_0$$

Then a nearby solution with different IC satisfies

$$\mathbf{y_2}' = \mathbf{A}\mathbf{y_2} + \mathbf{b}(t);\ \ \mathbf{y_2}(0) = \mathbf{y}_0 + \delta$$

The difference $\mathbf{z}$ satisfies

$$\mathbf{z}' = \mathbf{A}\mathbf{z};\ \ \mathbf{z}(0) = \delta$$

Model equation

Assume **A** is diagonalizable: then $\mathbf{A} = \mathbf{S}\Lambda\mathbf{S}^{-1}$; $\lambda_i \in \mathbb{C}$. So by changing variables

$$\mathbf{w} = \mathbf{S}^{-1}\mathbf{z}$$

we get the system

$$\mathbf{w}' = \Lambda\mathbf{w}$$

which is decoupled

$$w_i' = \lambda_i w_i \quad \lambda_i \in \mathbb{C}$$

since Λ is diagonal.

This explains why, for linear stability, we use the **model scalar equation**

$$y' = \lambda y, \quad \lambda \in \mathbb{C}$$

For linear stability, ask how the method behaves on the model equation i.e. $J = \lambda$

Region of Absolute Stability

For contractive solutions, need $Re(\lambda) < 0$
For numerical errors to be damped $\rightarrow$ we demand

$$| y_{n+1} | < | y_n |$$

For Euler's Method:

$$y_{n+1} = y_n + hf(t_n, y_n) = y_n + h\lambda y_n = (1 + h\lambda)y_n$$

We call $\{h\lambda \in \mathbb{C} : | 1 + h\lambda | < 1\}$ the **region of absolute stability RAS**

its intersection with the real axis $= (-2, 0)$ is the **interval of absolute stability**

RAS for RK2

Do same for RK2: apply method to the model equation

$$
\begin{aligned}
s_1 &= f(t_n, y_n) = \lambda y_n \\
s_2 &= \lambda(y_n + \alpha h s_1) = \lambda(y_n + \alpha h \lambda y_n) \\
y_{n+1} &= y_n + h((1 - \frac{1}{2\alpha})s_1 + \frac{1}{2\alpha}s_2) \\
&= y_n + h((1 - \frac{1}{2\alpha})\lambda y_n + \frac{1}{2\alpha}\lambda(y_n + \alpha h \lambda y_n)) \\
&= y_n[1 + h\lambda + \frac{1}{2}(h\lambda)^2]
\end{aligned}
$$

$\rightarrow$ Region of absolute stability: $\mid 1 + h\lambda + \frac{1}{2}(h\lambda)^2 \mid < 1$
$\rightarrow$ Interval of absolute stability: $(-2, 0)$ (again)

Moral: Use RK2 for better accuracy, not improved stability

Note: since exact solution
$y(t_{n+1}) = e^{\lambda h} y_n = [1 + h\lambda + \frac{1}{2}(h\lambda)^2 + O(h^3)]y_n$
$\rightarrow$ RK2 is only 2nd order, not higher

A-stability

Ideally, we would like method numerical errors to be damped whenever solutions are contractive.
If region of A-stability includes $Re(\lambda) < 0$ (whole LH of complex plane)
$\rightarrow$ method is A-stable

Theorem

No explicit RK method is A-stable

Proof.

The region of absolute stability for any explicit RK method is given by $|P(h\lambda)| < 1$ where P is some polynomial. Since $|P(h\lambda)|$ must $\rightarrow \infty$ as $\lambda \rightarrow -\infty$ the RAS can never extend to infinity — it must always be a bounded domain. $\square$

RK3

Similarly look for 3rd order methods using 3 stages:

$$
\begin{aligned}
s_1 &= f(t_n, y_n) && (4) \\
s_2 &= f(t_n + c_2 h, y_n + h a_{21} s_1) && (5) \\
s_3 &= f(t_n + c_3 h, y_n + h a_{31} s_1 + h a_{32} s_2) && (6) \\
y_{n+1} &= y_n + h(b_1 s_1 + b_2 s_2 + b_3 s_3) && (7)
\end{aligned}
$$

$$
\begin{array}{c|ccc}
0 & 0 & 0 & 0 \\
c_2 & a_{21} & 0 & 0 \\
c_3 & a_{31} & a_{32} & 0 \\
\hline
 & b_1 & b_2 & b_3
\end{array}
$$

match with Taylor series $\rightarrow$ 3 1-parameter families, all RK3 with 3 stages

RK4

Similarly look for 4th order methods using 4 stages:

Example

classical RK4 (Kutta 1905)

$$
\begin{array}{c|cccc}
0 & & & & \\
1/2 & 1/2 & & & \\
1/2 & 0 & 1/2 & & \\
1 & 0 & 0 & 1 & \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
$$

Can go on, but for $p > 4$ need $s > p$

Example

for RK5, need 6 stages

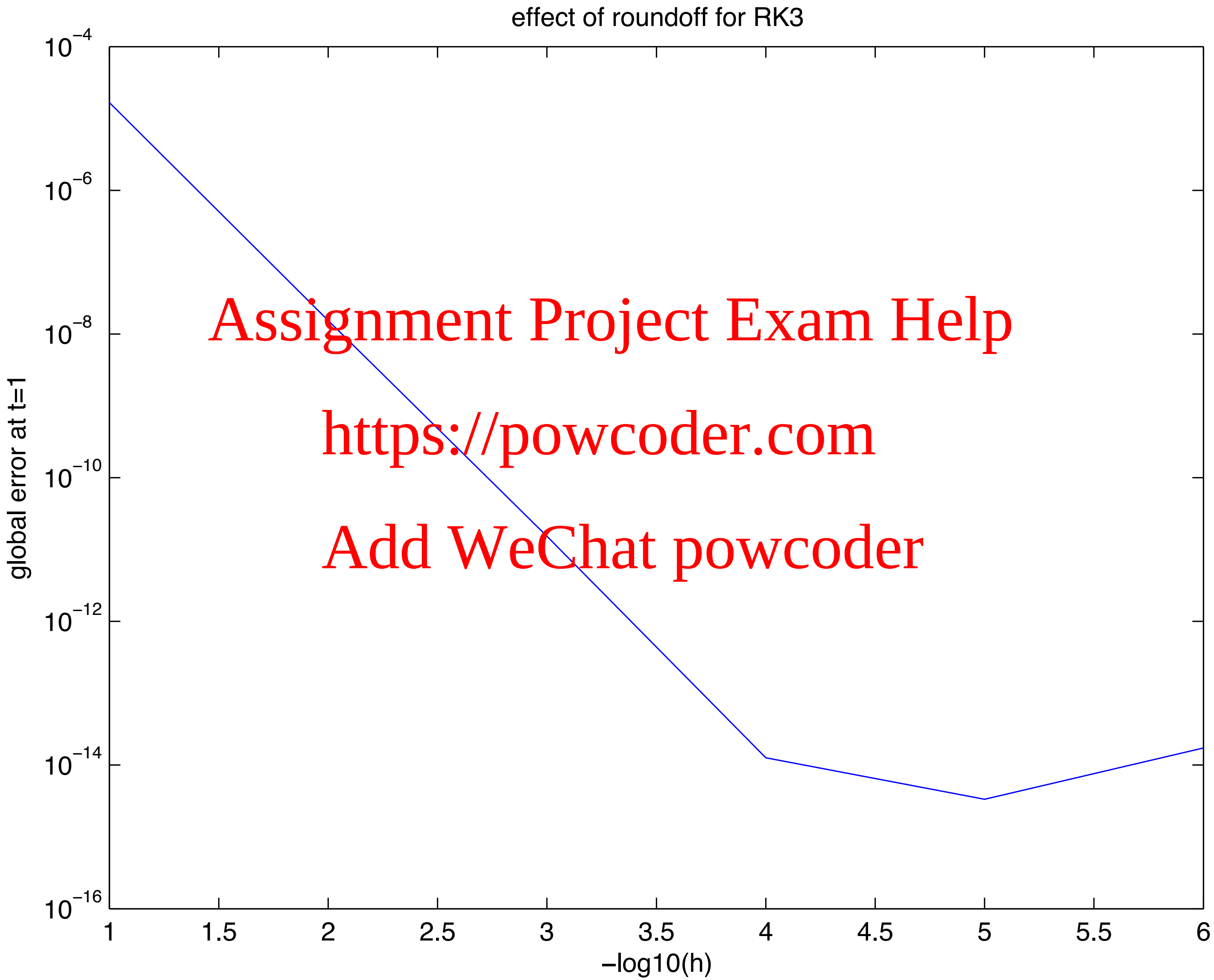
Effect of roundoff for RK methods

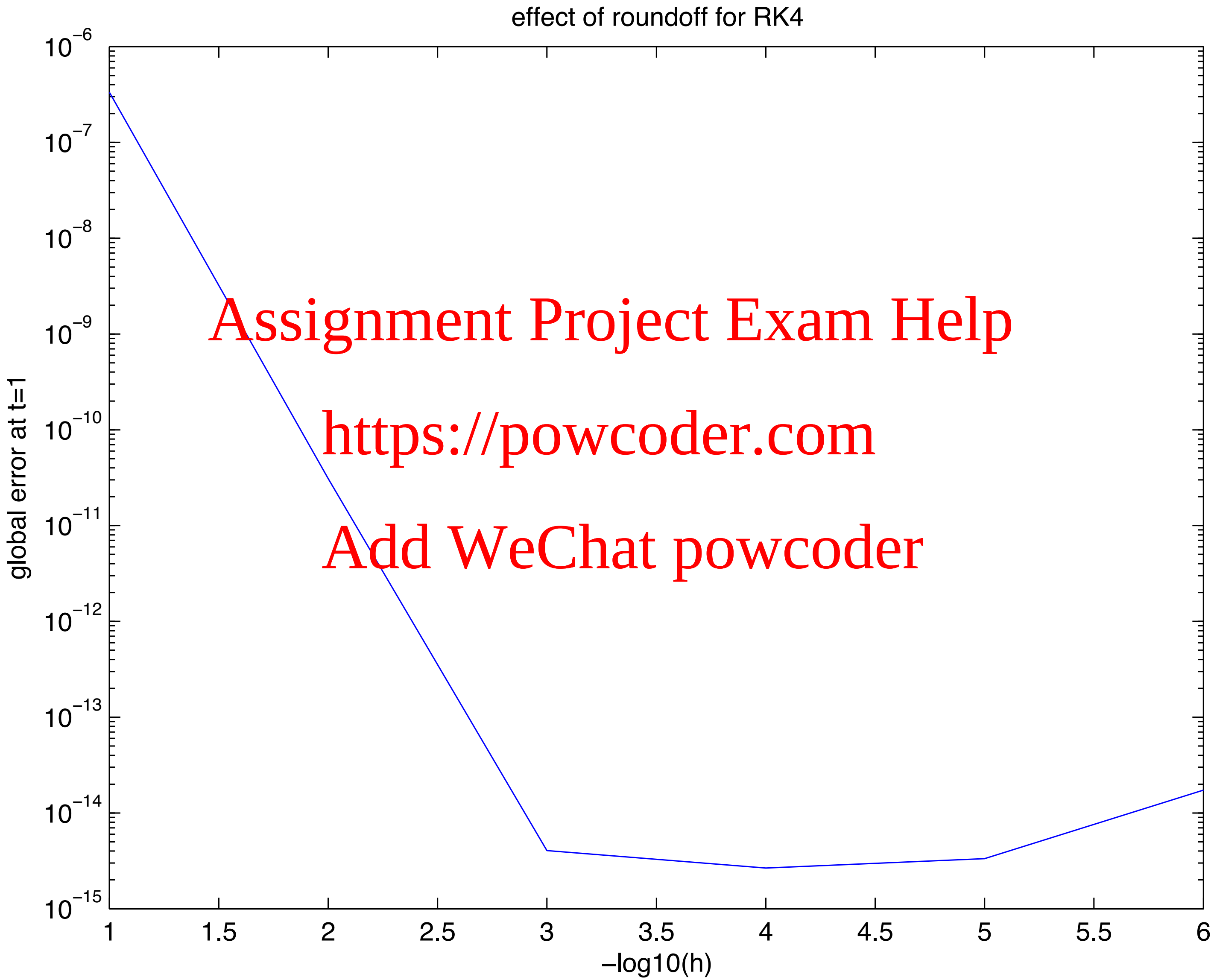
For each method, GE $= O(h^p)$ after $n = T/h$ steps: **truncation error** in exact arithmetic

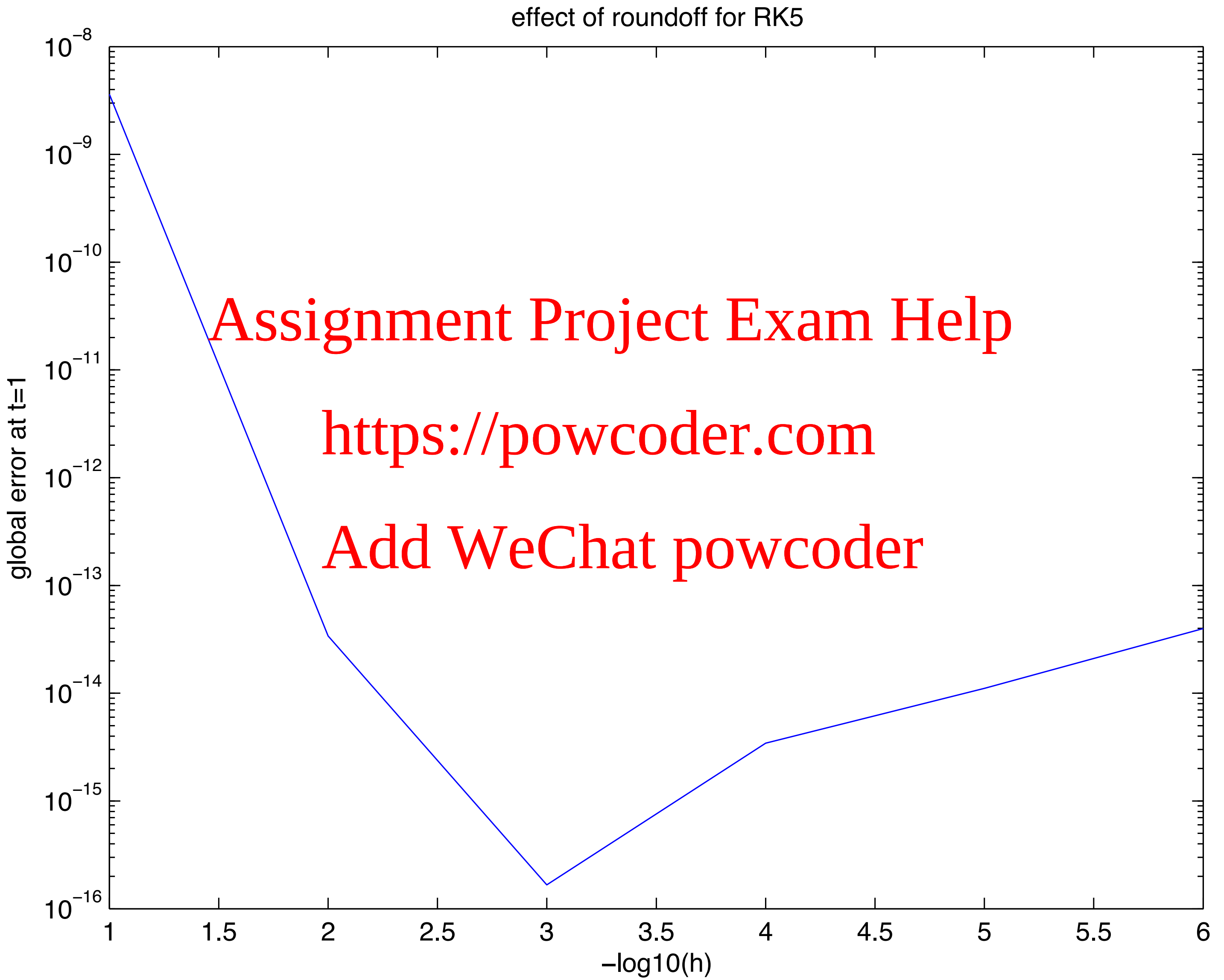
If roundoff errors add randomly (as observed): get extra error $\sim un^{1/2} \sim uh^{-1/2}$
$\to$ optimal h just like numerical differentiation, with

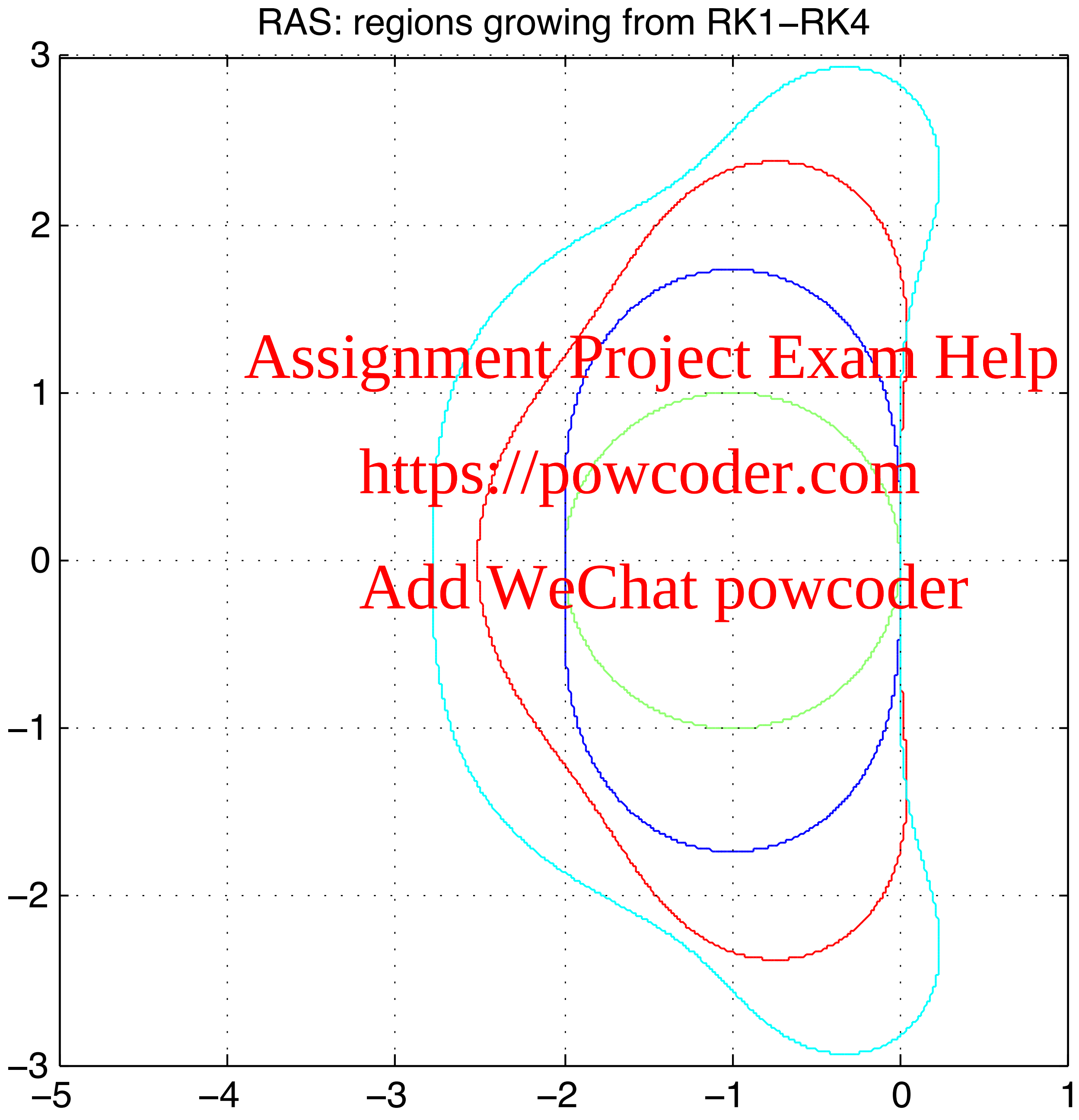
$$h_{\text{opt}} \sim u^{2/(2p+1)}$$

$\to h_{\text{opt}} \sim 10^{-5}, 10^{-4}, 10^{-3}$ for RK3, RK4, RK5 in double precision


effect of roundoff for RK3
global error at t=1
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6
−log10(h)


effect of roundoff for RK4
global error at t=1
−log10(h)
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{-15}


effect of roundoff for RK5
global error at t=1
-log10(h)
10^-8
10^-9
10^-10
10^-11
10^-12
10^-13
10^-14
10^-15
10^-16
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6


RAS: regions growing from RK1−RK4
3
2
1
0
−1
−2
−3
−5
−4
−3
−2
−1
0
1

Summary of RK methods

Method	Order	Local error	Fevals/step	h_{opt}
Euler = RK1	1	h^2	1	$\sim 10^{-10}$
RK2	2	h^3	2	$\sim 5 \times 10^{-7}$
RK3	3	h^4	3	$\sim 10^{-5}$
RK4	4	h^5	4	$\sim 10^{-4}$
RK5	5	h^6	6	$\sim 10^{-3}$

Variable step methods

So far, everything has been fixed-step. The user has to choose n or h.

Instead, the user should choose a tolerance (absolute or relative) and the method should choose h at each step to achieve an error smaller than the tolerance $\rightarrow$ a variable-step method.

The basic problem is that it's hard to estimate the global error (depends on J, which may change over time) but we can estimate the local error. The idea is to step from t_n to t_{n+1} twice, using 2 methods with different h (e.g. h and $h/2$) or different order. Then from 2 results, estimate the local error and use this to control the stepsize.

Embedded Runge-Kutta methods

One clever idea (Fehlberg 1969) is to use 2 RK methods of different order, with same $\mathbf{c}, \mathbf{A}$ from the Butcher tableau (same evaluation points) so a lot of the function evaluations are shared between the 2 methods $\rightarrow$ saves work!

Example

MATLAB's `ode23,ode45`

We use 2 estimates from methods of different order $p, p+1$ e.g. $= 2, 3$
y^p_{n+1} has local error $\sim Ch^{p+1}$
y^{p+1}_{n+1} has local error $\sim \bar{C}h^{p+2}$
for usual values of h, $\bar{C}h^{p+2} \ll Ch^{p+1}$ so we estimate error in y^p_{n+1} (the worse method) by

$$\text{err} = | y^p_{n+1} - y^{p+1}_{n+1} |$$

and demand that err $<$ Atol

Local extrapolation

If err $<$ Atol, keep that step, using y_{n+1}^{p+1} (the better estimate)

If not, cut down stepsize h so err $<$ Atol with the new stepsize

Control stepsize using error estimate of worse method but keep better estimate — called *local extrapolation*

We hope this local extrapolation makes up for controlling the local error, not the global error, but it's not guaranteed.

Rescaling h

We want err $<$ Atol and we know err $\sim h^{p+1}$
$\implies$ we will achieve the desired tolerance with a new stepsize $= qh_{old}$, provided

$$err_{new} < \frac{\text{Atol}}{\text{err}} err_{old}$$

But

$$\frac{err_{new}}{err_{old}} \sim \frac{C(qh)^{p+1}}{Ch^{p+1}} = q^{p+1}$$

so we choose

$$q = 0.8(\frac{\text{Atol}}{\text{err}})^{1/(p+1)}$$

where 0.8 is a safety factor to ensure new h is easily small enough.
Similar idea for a relative tolerance.

ode23

For a simplified version, see `ode23tx.m` and Moler §7.5, 7.6.

- uses 3rd order 3-stage RK3
- and (4-stage!) RK2 which uses s_1, s_2, s_3 from RK3 (no extra work)

- and $s_4 = f(t_{n+1}, y_{n+1})$
  Note: $s_4 \mapsto s_1$ on next step (**First Same As Last**) so this costs nothing extra if step is accepted (i.e. most of the time)

$\rightarrow$ a 3rd order method + error estimator for $\sim$ 3 stages of work!
In fact, we don't bother forming y^p at all — just form the local error estimator $\mid y^3 - y^2 \mid$

ode45

- uses a 5th order 6-stage RK
- + 4th order 7-stage **First Same As Last** RK

→ 5th order method + error estimator for ∼ 6 stages of work!

Embedded RK methods are good nonstiff 1-step solvers — prob. first methods to try.

MATLAB suggests `ode45` as the first method to try.

End of Lecture 21