Fehlberg Method (Embedded RK formula)

- 5^{th} order accuracy, variable time step, only 6 fn calls

Generally, RK(4+m) – m orders higher than 4 – requires more than m additional f^n calls, but not more than m+2.

Fehlberg discovered an RK5 with 6 f^n calls

$$k_1 = hf(y_n, x_n)$$
$$k_2 = hf(y_n + b_{21}k_1, \; x_n + a_2 h)$$
$$k_3 = hf(y_n + b_{31}k_1 + b_{32}k_2, \; x_n + a_3 h)$$
$$\vdots$$
$$k_6 = hf(y_n + b_{61}k_1 + b_{62}k_2 + \ldots + b_{65}k_5, \; x_n + a_6 h)$$

$$\begin{aligned} y_{n+1} &= y_n + c_1 k_1 + c_2 k_2 + c_3 k_3 + c_4 k_4 + c_5 k_5 + c_6 k_6 \\ &= y(x+h) + O(h^6) \qquad \text{RK5} \end{aligned}$$

that also yield a 4^{th} order method (RK4) with the <u>same</u> f^n calls

$$\begin{aligned} y^*_{n+1} &= y_n + c_1^* k_1 + c_2^* k_2 + c_3^* k_3 + c_4^* k_4 + c_5^* k_5 + c_6^* k_6 \\ &= y(x+h) + O(h^5) \qquad \text{RK4} \end{aligned}$$

The a_i's and b_{ij}'s are the same for both, but the c_i's and c_i^*'s are different.

The parameters commonly used nowadays are due to Cash and Karp (see e.g. Numerical Recipes)

We get an RK5 (5^{th} order method) <u>AND</u> an estimate of the error.

$$\Delta = y_{n+1} - y_{n+1}^{*} = \sum_{i=1}^{6} (c_i - c_i^{*}) k_i \sim \mathcal{O}(h^5)$$

with only 6 f^{n} calls. More efficient than RK4-based step halving/doubling.

As before, if stepsize $h_{current}$ produces $\Delta_{current}$ we want stepsize h_{new} that produces $|\Delta_{new}| = \epsilon$

$$\Delta \sim h^5 \rightarrow h_{new} = h_{current} \left| \frac{\epsilon}{\Delta_{current}} \right|^{1/n} \qquad n = 5, \quad 1/n = 0.2$$

In Numerical Recipes notation

$$h_0 = S h_1 \left(\frac{\Delta_0}{\Delta_1} \right)^{1/5} \qquad \text{with } S = 0.9$$

Practical implementation issues regarding Δ_0, the desired accuracy

- currently an absolutely value
- different for every component of $\vec{y}$
- different components may differ orders of magnitude
- not general for different problems

better: want solution "good to one part in 10^6"

set $\Delta_0 = \epsilon y$ with $\epsilon = 10^{-6}$ - relative error tolerance

but what if y oscillates (and $y=0$ sometimes)?

set $\Delta_0 = \epsilon \, y_{scale}$

In NR code

$$y_{scale}(i) = |y(i)| + |h \, \underset{\uparrow f}{dydx}(i)| + \underset{\nearrow 10^{-30}}{TINY}$$

$i = 1, \ldots n \rightarrow$ for each vector dimension

- if $|y(i)| \simeq 0$, y_{scale} is set by the derivative

$$h_{new} = S h \left| \frac{\Delta_0}{\Delta_1} \right|^{1/n}_{min(i=1,\ldots n)}$$

use largest relative error $\frac{\Delta_1}{\Delta_0}$ over all components

$$= S h \left(\min_i \left| \frac{\epsilon \, y_{scale}}{y_{err}} \right| \right)^{1/n}$$

$$= S h \left(\max_i \left| \frac{y_{err}}{\epsilon \, y_{scale}} \right| \right)^{-1/n}$$

use largest $\frac{y_{err}(i)}{\epsilon \, y_{scale}(i)}$ to control step size

Alternatively, may want to limit global error
(error at final integration point),

global error limit $\simeq N\Delta_0 \simeq \frac{t_f - t_i}{h}\Delta_0$

$\therefore$ want $\Delta_0 \sim h$ (target error at a step should scale with h)

choose $\Delta_0 = \epsilon h \, dy/dx$ as in NR's yscale

But then our formula $h_0 = h_1 \left|\frac{\Delta_0}{\Delta_1}\right|^{1/5}$ needs to

change (since global error $\sim \mathcal{O}(h^4)$)

to $h_0 = h_1 \left|\frac{\Delta_0}{\Delta_1}\right|^{1/4}$

Note that $1/4 = 0.25$
and $1/5 = 0.2$ are not too different.

NR takes a pragmatic, general, and conservative approach (not increasing or decreasing h too much per step).

When increasing step size $(\Delta_1 < \Delta_0)$

use $h_0 = S\, h_1 \left|\frac{\Delta_0}{\Delta_1}\right|^{0.20}$ (yields smaller h_0 than with exponent of 0.25)

but $h_0 \leq 5 h_1$ (upper limit on stepsize increase)

When decreasing step size ($\Delta_1 > \Delta_0$)

use $h_0 = S\, h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{0.25}$ (yields smaller h_0 than with exponent of 0.20)

but $h_0 \geq 0.1\, h_1$

lower limit on stepsize decrease

Also check that h does not become too small, ie, check that $x + h \neq x$

These aspects of the implementation are not rigourous, but are experience-based, practical steps to produce robust code that will solve most problems.

User must supply derivs function that returns dydx ($\vec{F}(\vec{y}, x)$)

and initialize $\vec{y}$.

Part of Project 2 deals with using NR code to solve a physics problem.

Appendix - Details on global error

Δ_1 - error estimate at each step $\sim \mathcal{O}(h^5)$

$$\Delta_1 = c\, h^5$$

Δ_0 - desired local error : $\Delta_0 = \epsilon h \frac{dy}{dx}$

Δ_1^g - approx. global error :

$$\Delta_1^g = N \Delta_1 \sim \frac{\Delta_1}{h} \sim \mathcal{O}(h^4) = D h^4$$

Δ_0^g - desired global error :

$$\Delta_0^g \doteq \epsilon h \frac{dy}{dx} N \sim \epsilon h \frac{dy}{dx} \frac{1}{h} = \epsilon \frac{dy}{dx}$$

$$\begin{aligned} h^4 D &= \Delta_1^g \\ h_{new}^4 D &= \Delta_0^g \end{aligned} \Rightarrow \frac{h^4}{h_{new}^4} = \frac{\Delta_1^g}{\Delta_0^g}$$

$$h_{new} = h\left(\frac{\Delta_0^g}{\Delta_1^g}\right)^{1/4} \sim h\left(\frac{\epsilon\, dy/dx}{\Delta_1/h}\right)^{1/4} = h\left(\frac{\epsilon h\, dy/dx}{\Delta_1}\right)^{1/4}$$

$$= h\left(\frac{\Delta_0}{\Delta_1}\right)^{1/4}$$