Errors in numerical approximations

Let $y(t)$ be the exact sol. of the $\overset{\text{diff'l equation}}{\text{problem}}$ [e.g. of $y' = f(t,y)$, $y(t_0) = y_0$]

Let Y_n = numerical approximation to $y(t_n)$, a finite-dim'l vector $\vec{Y} = (Y_0, Y_1, \dots, Y_N)$

= exact sol. of the Finite Difference scheme, after we discretize

Continuous problem

$$ODE[y] = y'(t) - f(t,y) = 0$$

$$y(t_0) = y_0$$

Discrete problem (e.g. Euler method)

$$\left(FDE[Y_n]:\ Y_{n+1} - Y_n + \Delta t \cdot f(t_n, Y_n) = 0\right)$$

$$Y_0 = y_0$$

better form: $FDE[Y_n] = \dfrac{Y_{n+1} - Y_n}{\Delta t} - f(t_n, Y_n) = 0$

Want to approximate $y(t_n)$, $n = 0, 1, \dots, N$, up to some time t_N $\left(\Delta t = \frac{t_N - t_0}{N}\right)$.
We discretize the continuous problem, i.e. replace it by a discrete problem, the FDE, and try to compute the discrete solution $\{Y_0, Y_1, \dots, Y_N\}$ of the FDE.

Definition: The discretization error at the n-th step is $de_n = y(t_n) - Y_n$.

Here Y_n is the exact sol of the $FDE[Y_n] = 0$, at ∞ precision!
This would be the error if we calculate Y_n from the FDE at infinite precision!
Since we have to use a computer with finite precision (finitely many floating point numbers!)
what we actually compute is only an approximation to Y_n due to rounding to machine numbers:

$$\tilde{Y}_n = \underset{\text{(floating point numbers)}}{\text{actual computed}} \text{ values} \approx Y_n$$

Definition: The roundoff error is $re_n = Y_n - \tilde{Y}_n$

So the actual total error at n-th step is

$$y(t_n) - \tilde{Y}_n = \underbrace{y(t_n) - Y_n}_{\text{discretization}} + \underbrace{Y_n - \tilde{Y}_n}_{\text{roundoff}}$$

$$de_n + re_n$$

There is yet another kind of error, that measures how far off is the FDE from the ODE! it is called the

Definition: The $\overset{\text{local}}{\text{truncation error}}$ (consistency error) is the amount by which the exact solution $y(t_n)$ of ODE fails to satisfy the FDE! $\frac{Y_{n+1} - Y_n}{\Delta t} = f(t_n, Y_n)$ to look like the ODE

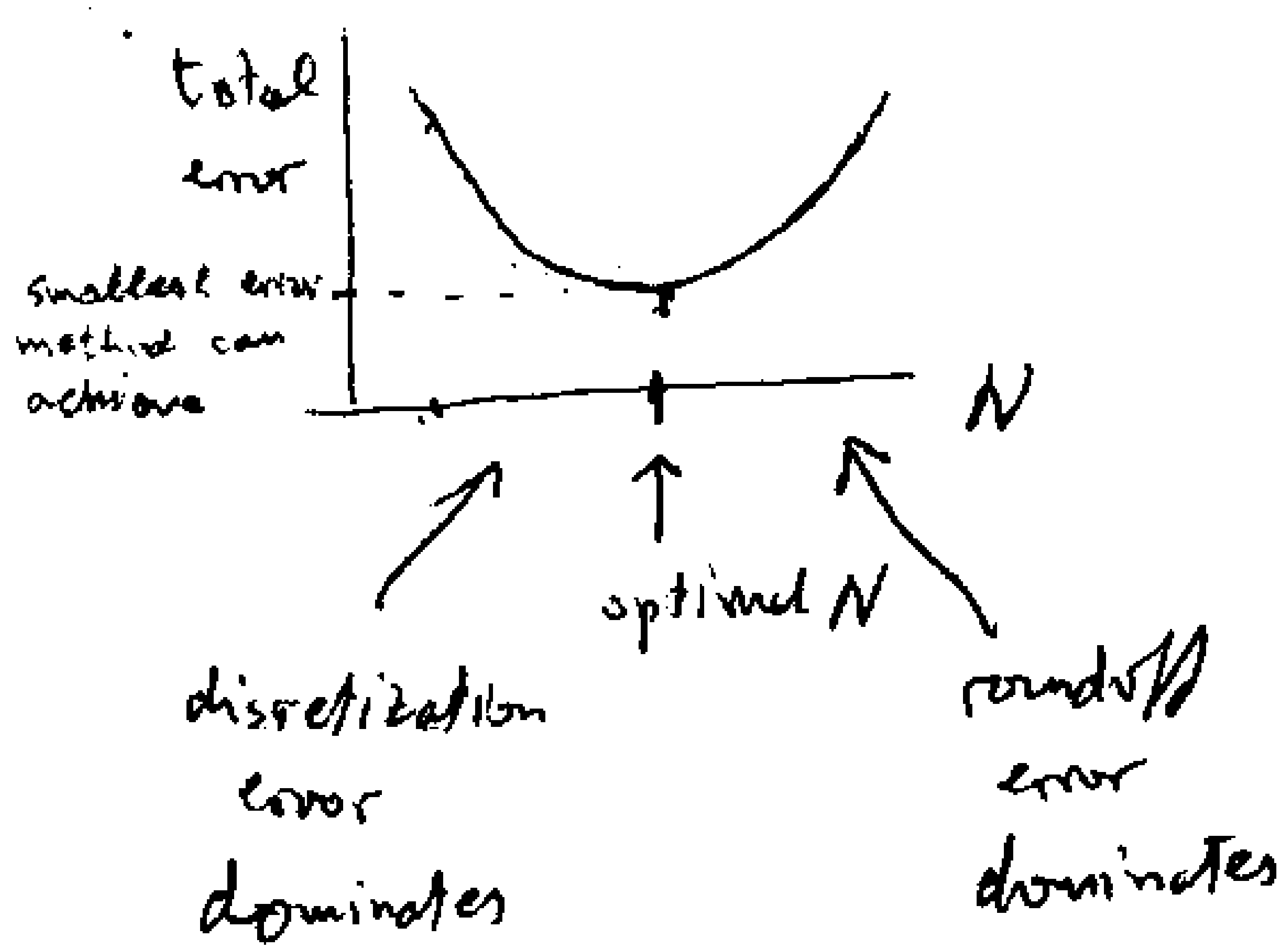
$$te_n := \frac{y(t_{n+1}) - y(t_n)}{\Delta t} - f(t_n, y(t_n))$$

$$= FDE[y(t_n)] - \overset{0}{ODE[y(t_n)]}$$ can be estimated by Taylor expansion

__But__, the more steps the more computations must be done, so the longer the run will take and __worse__ more roundoff errors may pile up!

The total error won't reduce as much, or it can even get worse due to roundoff!

The result is



as N grows, roundoff grows, it eventually takes over and we end up with worse total error! the roundoff takes over and accuracy __cannot__ improve further, ~~[illegible]~~ with this method!

To reduce roundoff: careful coding

double precision computation (arithmetic with more digits)

If the total error is still too large for our purposes, the only alternative is to use a __higher order method__! there are __many__, e.g. Runge-Kutta, multistep.

Errors ....

Accuracy of Euler Method (convergence estimate): The discritization error of Euler scheme satisfies

$$|y(t_n) - Y_n| \leq e^{K\cdot(t_n - t_0)} \cdot |y_0 - Y_0| + M_2 \cdot \frac{e^{K(t_n - t_0)} - 1}{2K} \cdot \Delta t$$

where $K =$ Lipschitz const. of $F \leq \sup\left|\frac{\partial F}{\partial y}\right| < \infty$, $M_2 = \sup|y''| \overset{\text{assumed}}{<} \infty$

Remarks: 1. Even a small initial error can become big for large t_n (long term computation) or large K

but always remains bounded.

2. Apart from the initial error, $|\text{error}| \leq (\text{const.})\, \Delta t = \mathcal{O}(\Delta t)$ ($\Rightarrow$ first order method)

So convergent $\therefore$ also stable.

3. 1^{st} order accuracy means error is proportional to Δt.
Halving Δt we expect half the error.

Big O notation: $f(x) = \mathcal{O}(g(x))$ as $x \to x_0$ means $\left|\frac{f(x)}{g(x)}\right| \leq M$ for x near x_0
i.e. $|f(x)| \leq (\text{const.})\,|g(x)|$ near x_0

Excellent ODE solvers are available: VODE, rksuite, ... in netlib.org
GSL

There is **no** best method for all ODEs or even for classes of ODEs.

Types of ODE integrators:

- explicit - implicit - BDF for "stiff"
- single step - multistep
- Taylor type - RK
- symplectic - nonsymplectic
- non-adaptive - adaptive

Matlab has 8 integrators: rk4, rk45, ode113, ode113s, ode15i, 15s, ode23, 23s, 23t, ode23tb