Math 260: Python programming in math

Fall 2020

More on ODEs:
Stiff equations, implicit methods

Stiff ODEs: the problem

Solving

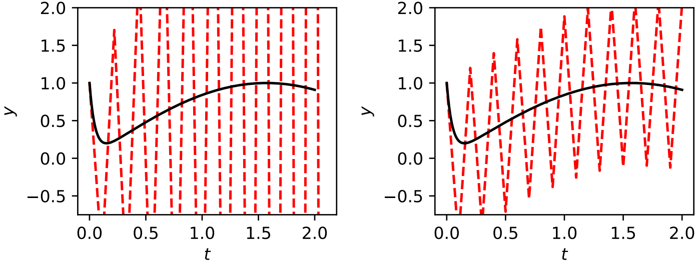
$$y' = -20(y - \sin t) + \cos t, \quad y(0) = 1$$

with exact solution

$$y(t) = Ce^{-20t} + \sin t.$$

The numerical solution should approach $\sin t$!

FE with $h = 0.12$ and $h = 0.1$:



Oscillations grow in t (disaster!); bounded when $h = 0.1$.

Stiff ODEs: the problem

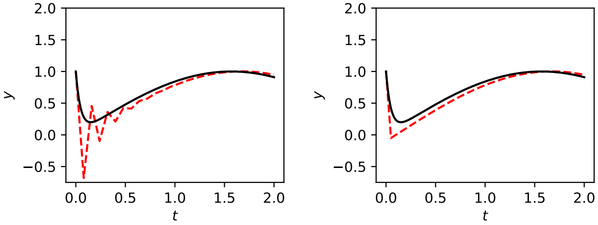
Solving

$$y' = -20(y - \sin t) + \cos t, \quad y(0) = 1$$

with exact solution

$$y(t) = Ce^{-20t} + \sin t.$$

... and with $h = 0.08$ and $h = 0.05$:

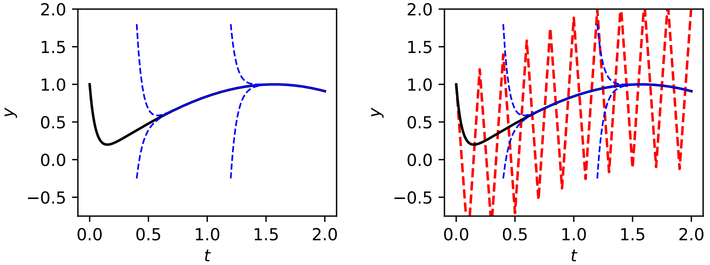


Suddenly, the solver does fine (approaches $\sin t$).

Stiff ODEs: stability

What's going on?

- The $y(t) \approx \sin t$ solution hides the ODE behavior
- **nearby** solutions (starting near $\sin t$) quickly approach $\sin t$



- Forward Euler 'follows' the slope of these solutions
- It overshoots repeatedly - it sees the sharp slopes of the nearby solutions.
- How can we do better? Use a different method!

Stiff ODEs: stability

Forward and backward Euler:

$$\text{Forward Euler: } u_{n+1} = u_n + hf(t_n, u_n)$$
$$\text{Backward Euler: } u_{n+1} = u_n + hf(t_n, u_{n+1})$$

Applied to the linear constant coefficient ODE $y' = \lambda y$...

$$\begin{aligned} \text{FE: } u_{n+1} &= u_n + h\lambda u_n \\ \text{BE: } u_{n+1} &= u_n + h\lambda u_{n+1} \end{aligned} \quad \Longrightarrow \quad \begin{aligned} \text{FE: } u_{n+1} &= (1 + h\lambda)u_n \\ \text{BE: } u_{n+1} &= \frac{1}{1 - h\lambda} u_n \end{aligned}$$

Example: take $\lambda = -10$ and $h = 0.3$. Then $h\lambda = -3$ so

$$\text{FE} : u_{n+1} = 2u_n, \quad \text{BE} : u_{n+1} = \frac{1}{4} u_n$$

and the exact solution is $y = y_0 e^{-10t}$.

Key point:

The qualitative behavior (grows? decays?) **can be different** from the ODE.

- Growth decay depends on h and λ
- Not related to convergence - a new type of concern
- Sharp transition from 'bad' to 'good' (stability)

Stiff ODEs: linear systems

More generally, for **linear systems** like

$$\mathbf{x}' = A\mathbf{x}, \quad \mathbf{x}(0) = \vec{c},$$

we can directly solve' for the next step...

$$\text{BE: } \mathbf{u}_{n+1} = \mathbf{u}_n + hF(t_n, \mathbf{u}_{n+1})$$

$$\implies \mathbf{u}_{n+1} = (I - hA)^{-1}\mathbf{u}_n.$$

In practice, we use a linear system solver (e.g. LU factorization).

- to go from $t_n \to t_{n+1}$...
- Solve $(I - hA)\mathbf{u}_{n+1} = \mathbf{u}_n$ for the unknown $\mathbf{u}_{n+1}$.

For the trapezoidal rule:

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \frac{h}{2}(F(t_n, \mathbf{u}_n) + F_{n+1}(t_{n+1}, \mathbf{u}_{n+1}))$$

$$\implies (I - \frac{h}{2}A)\mathbf{u}_{n+1} = (I + \frac{h}{2}A)\mathbf{u}_n.$$

This method (in the context of PDEs) is called **Crank-Nicolson**.