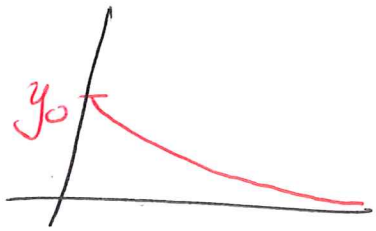
Stability (Continued)

Having ensured that a given ODE is stable (intrinsically) we can attempt to design a numerical scheme that approximates it effectively.

We will start, again, by the model stable ODE

$$y'(t) = \lambda\, y(t) \quad [= f(y)] \quad \text{with } \lambda < 0$$

Analytic solution $y(t) = y_0 e^{\lambda t}$



As a minimum requirement for "stability" we would need a numerical approximation to match its asymptotic behavior as $t \to \infty$ (i.e. $y(t) \to 0$). That is, if a numeric scheme produces approximations $y_k \approx y(t_k)$ we would require that $\lim_{k\to\infty} y_k \to 0$.

For simplicity assume a constant time step size

$$t_{k+1} - t_k = h = \text{const} \quad (\text{i.e } t_k = t_0 + k \cdot h)$$

Remember how we derived candidate approximation schemes:

$$y_{k+1} = y_k + \int_{t_k}^{t_{k+1}} (\lambda y)\, dt$$

$\Rightarrow$ Using $\int_a^b f \approx (b-a) f(a) \Rightarrow$

$$\boxed{y_{k+1} = y_k + h \cdot \lambda y_k} \quad (1)$$

or $y_{k+1} = y_k + h f(y_k)$

"Forward Euler"

$\Rightarrow$ Using $\int_a^b f \approx (b-a) f(b) \Rightarrow$

$$\boxed{y_{k+1} = y_k + h \lambda y_{k+1}} \quad (2)$$

or $y_{k+1} = y_k + h f(y_{k+1})$

"Backward Euler"

$\Rightarrow$ Using $\int_a^b f \approx \frac{b-a}{2}[f(a)+f(b)] \Rightarrow$

$$\boxed{y_{k+1} = y_k + h\lambda \frac{y_k + y_{k+1}}{2}} \quad (3)$$

or $y_{k+1} = y_k + \frac{h}{2}\{f(y_k) + f(y_{k+1})\}$

"Trapezoidal Rule"

Let's analyze options (1,2,3) for the model equation:

Forward Euler

Eq. (1) yields $\Rightarrow y_{k+1} = (1+h\lambda) y_k$

$\Rightarrow y_k = (1+h\lambda)^k y_0$

Thus, it is easy to determine when y_k exhibits the desired asymptotic behavior $y_k \xrightarrow[k\to\infty]{} 0$

This will happen provided that

$$|1+h\lambda| \le 1 \iff -1 \le \underbrace{1+h\lambda \le 1}_{\text{always true if } \lambda<0} \iff$$

$$\iff -h\lambda \le 2 \iff h|\lambda| \le 2 \implies \boxed{h \le \frac{2}{|\lambda|}}$$

This is the stability condition for forward Euler

Note #1 If we have a linear system of ODE's

$$Y'(t) = A \cdot Y(t)$$

Then, the corresponding stability condition for Forward Euler is

$$\|1 + h\lambda_i\| < 1 \quad \forall i = 1, 2, \ldots, n$$

($\|\cdot\|$: complex magnitude)

where $\lambda_1, \lambda_2, \ldots, \lambda_n$ are all eigenvalues of A.

Note #2 If we have a nonlinear system of ODE s

$$Y(t) = F(Y(t)) \qquad Y : \mathbb{R} \to \mathbb{R}^n$$
$$F : \mathbb{R}^n \to \mathbb{R}^n$$

Then, the condition $\|1+h\lambda_i\| < 1$ must hold for all eigenvalues of the Jacobian $J = \partial F / \partial Y$

(optional exercise)

⇒ What would be the stability condition for the Hookean spring ODE

$$\begin{pmatrix} x \\ v \end{pmatrix}' = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{pmatrix} \begin{pmatrix} x \\ v \end{pmatrix} \quad ?$$

Backward Euler

Applying the rule $\boxed{y_{k+1} = y_k + h f(y_{k+1})}$ to the model ODE $y'(t) = f(y) = \lambda y$, yields:

$$y_{k+1} = y_k + h\lambda y_{k+1} \Rightarrow (1-h\lambda) y_{k+1} = y_k \Rightarrow$$

$$\Rightarrow y_{k+1} = \frac{1}{(1-h\lambda)} y_k \Rightarrow y_k = \frac{1}{(1-h\lambda)^k} y_0$$

The condition under which we can guarantee $y_k \xrightarrow[k\to\infty]{} 0$ is now $|1-h\lambda| > 1$, which holds for any $h>0$ (when $\lambda < 0$)

Thus Backward Euler is unconditionally stable for the model ODE, and - under very reasoble assumptions - for any stable ODE, even nonlinear ones.

Trapezoidal Rule (TR)

As before

$$y' = f(y) = \lambda y$$

$$y_{k+1} = y_k + \frac{h}{2}\left[f(y_k) + f(y_{k+1})\right] \Rightarrow y_{k+1} = y_k + \frac{\lambda h}{2}(y_k + y_{k+1})$$

$$\Rightarrow \left(1 - \frac{\lambda h}{2}\right) y_{k+1} = \left(1 + \frac{h\lambda}{2}\right) y_k$$

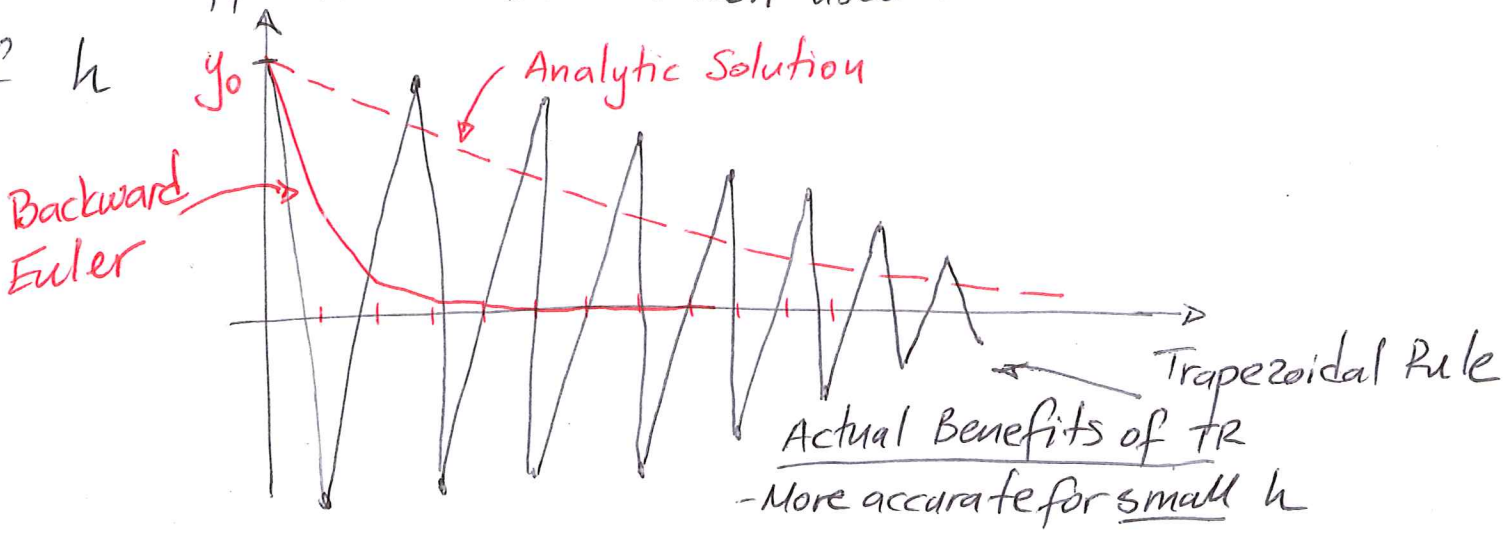
$$\Rightarrow y_{k+1} = \left(\frac{1+\lambda h/2}{1-\lambda h/2}\right) y_k \Rightarrow y_k = \left(\frac{1+\lambda h/2}{1-\lambda h/2}\right)^k y_0$$

Again, we can show that $\left|\frac{1+\lambda h/2}{1-\lambda h/2}\right| < 1$ for any $h > 0$

so this method is also unconditionally stable.

Comparison of TR v.s. BE

Although both stable under very large time steps, TR & BE exhibit different behavior when used with excessive values of h



Practical Backward Euler for Elasticity

For simplicity we will concentrate on a *linear* force model (e.g. linear elasticity) and *linear* (e.g. Rayleigh) damping — we will extend to nonlinear materials later

Thus we have

$$f_{el}(\underline{x}) = -K(\underline{x} - \underline{X})$$ (implemented as we saw in demos)

↑ elastic force

$$f_d(\underline{v}) = -\gamma K \underline{v}$$ (likewise)

↑ Rayleigh Damping force

We really should have written $f_d(\underline{x}, \underline{v})$, but here no dependence on $\underline{x}$

One more generalization: Instead of a *uniform* mass for every particle, we allow $m_1, m_2, \ldots, m_N$ to be different, and define a diagonal matrix $M = \text{diag}(m_1, m_2, \ldots, m_N)$

Then the equations of motion are

$$M\underline{x}''(t) = f_{el}(\underline{x}) + f_d(\underline{x}, \underline{v}) \Rightarrow$$

$$\underbrace{\begin{bmatrix} x(t) \\ v(t) \end{bmatrix}'}_{Y'(t)} = \underbrace{\begin{bmatrix} v(t) \\ M^{-1}[f_{el}(x) + f_d(\underline{x}, v)] \end{bmatrix}}_{F(Y(t))}$$

Backward Euler then becomes:

$$Y_{k+1} = Y_k + h F(Y_{k+1}) \Rightarrow$$

$$\underline{x}_{n+1} = \underline{x}_n + h \underline{v}_{n+1} \quad (4)$$

$$\underline{v}_{n+1} = \underline{v}_n + h M^{-1} \left\{ f_{el}(\underline{x}_{n+1}) + f_d(\underline{v}_{n+1}) \right\}$$

$$= \underline{v}_n + h M^{-1} \left\{ -K(\underline{x}_{n+1} - X) - \gamma K \underline{v}_{n+1} \right\} \quad (5)$$

$$(4) \Rightarrow \underline{v}_{n+1} = \frac{\underline{x}_{n+1} - \underline{x}_n}{h} = \frac{\Delta \underline{x}}{h} \quad (6) \quad \left(\Delta \underline{x} := \underline{x}_{n+1} - \underline{x}_n \right)$$

$$(5,6) \Rightarrow \frac{\Delta \underline{x}}{h} = \underline{v}_n + h M^{-1} \left\{ -K\left(\underbrace{\underline{x}_{n+1} - \underline{x}_n}_{\Delta \underline{x}} + \underline{x}_n - X \right) - \gamma K \frac{\Delta \underline{x}}{h} \right\}$$

$$\Rightarrow \frac{1}{h^2} M \Delta \underline{x} = \frac{1}{h} M \underline{v}_n + \left\{ -K \Delta \underline{x} - \underbrace{K(\underline{x}_n - X)}_{f_{el}(\underline{x}_n)} - \gamma K \frac{\Delta \underline{x}}{h} \right\}$$

$$\Rightarrow \boxed{\left[\left(1 + \frac{\gamma}{h}\right) K + \frac{1}{h^2} M \right] \Delta \underline{x} = \frac{1}{h} M \underline{v}_n + f_{el}(\underline{x}_n)} \quad (7)$$

After equation (7) has been solved, $\underline{x}_{n+1} \leftarrow \underline{x}_n + \Delta \underline{x}$

and $\underline{v}_{n+1} \leftarrow \frac{\Delta \underline{x}}{h}$ The challenge now shifts to solving equation (7)!

About solving equation (7)

→ Computing the right-hand-side is easy!
(we're done most of it, in code)

→ Left-hand-side is more tricky...

* How do we compute K?
* Where did we store it?
* How expensive is to solve the system even after we have formed it?

(would we use Gauss Elimination, for example?)
(would an L-U factorization work).

Alternative (preferred) method

We can "solve" $Ax = b$ by providing an initial guess x_0, and applying an iterative "improvement" procedure to get improved approximations $x_1, x_2, \ldots x_k$

"good enough"

Benefits

* Can stop early
* We dont need to "form" the matrix A...
as long as we know how to multiply by it :)

The Conjugate Gradients algorithm

Pre-requisites: Symmetric, positive definite A.
Initial guess x_0

In pseudocode:

$r_0 \leftarrow b - A x_0$
if r_0 is small, terminate
$p_0 \leftarrow r_0$
$k \leftarrow 0$
repeat
  $\alpha_k \leftarrow \dfrac{r_k^T r_k}{p_k A p_k}$
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $r_{k+1} \leftarrow r_k - \alpha_k A p_k$
  if r_{k+1} is small, terminate
  $\beta_k \leftarrow \dfrac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
  $p_{k+1} \leftarrow r_{k+1} + \beta p_k$
  $k++$