Review of Backward Euler Implementation

On a single spring

On a mass-spring system

$$x^{n+1} = x^n + dt\, v^{n+1} \quad (1)$$

$$v^{n+1} = v^n + \frac{dt}{m} f(x^{n+1}, v^{n+1})$$

$$= v^n + \frac{dt}{m}\left\{-k(x^{n+1} - x_0) - b v^{n+1}\right\}$$

Linear in x^{n+1} & v^{n+1}

$$v^{n+1} = v^n + dt M^{-1}\left\{g + f^{el}(x^{n+1}) + f^{d}(x^{n+1}, v^{n+1})\right\}$$

$$= v^n + dt M^{-1}\left\{g + f^{el}(x^{n+1}) + G(x^{n+1}) v^{n+1}\right\}$$

Nonlinear!

Approximations:

I. $f^{el}(x^{n+1}) \approx f^{el}(x^n) + K(x^{n+1} - x^n)$

[Matrix K to be defined later!]

II. $G(x^{n+1}) v^{n+1} \approx G(x^n) v^{n+1}$

$$\Rightarrow v^{n+1} = v^n + dt M^{-1}\left\{g + f^{el}(x^n) + K(x^{n+1} - x^n) + G(x^n) v^{n+1}\right\}$$

For convenience, introduce $\delta x = x^{n+1} - x^n$ (The "position change")

i.e. $\boxed{x^{n+1} = x^n + \delta x}$

Also from (1) $\boxed{v^{n+1} = \frac{1}{dt} \cdot \delta x}$

} Given δx, these equations compute x^{n+1} & v^{n+1}

$\Rightarrow$ Focus on finding δx

$$\frac{1}{dt}\delta x = v^n + \frac{dt}{m}\left\{-k(x^n + \delta x - x_0) - b\frac{1}{dt}\delta x\right\}$$

$$\frac{1}{dt}\delta x = v^n + dt M^{-1}\left\{g + f^{el}(x^n) + K\delta x + G(x^n)\frac{1}{dt}\delta x\right\}$$

Single Spring

$$\left[\frac{1}{dt} - \frac{k\,dt}{m} - \frac{b}{m}\right]\delta x =$$

$$= v^n + \frac{dt}{m}\left\{-k(x^n - x_0)\right\}$$

$$= v^n + \frac{dt}{m} \cdot f^{el}(x^n)$$

Mass-Spring System

$$\left[\frac{1}{dt} I - M^{-1} G(x^n) - dt M^{-1} K\right] \delta x$$

$$= v^n + dt M^{-1}\left\{f^{el}(x^n) + g\right\}$$

↳ Solve linear system for δx!

Implementation considerations:

- A "true" Backward Euler method, i.e. without any of the previously described approximations would offer strong guarantees for stability. Due to these approximations, we cannot expect a 100% robustness in the case of huge dt's, and may need to be somewhat restrained in case instabilities appear. Generally, though, this approach is quite robust, and certainly enables way larger dt's than explicit & semi-implicit methods.
- Even if the method appears stable for large dt, the physical fidelity may not be as good as if we took somewhat smaller dt's. This is a consequence of (a) Backward euler being less accurate in approximating the ODE due to large dt (b) The effect of various modeling approximations become more severe.

□ We want to use Conjugate Gradients to solve the linear system for δx. For this, we need to guarantee symmetry. We achieve this as follows

$$\frac{1}{dt}M\times\left\{\frac{1}{dt}I - M^{-1}G(x^n) - dtM^{-1}K\right\}\delta x = \frac{1}{dt}M\left\{v^n + dtM^{-1}[g+f^{el}(x^n)]\right\}$$

$$\Rightarrow \left\{\frac{1}{dt^2}M - \frac{1}{dt}G(x^n) - K\right\}\delta x = \frac{1}{dt}Mv^n + \left\{g+f^{el}(x^n)\right\}$$

K is symmetric, as we see next, thus the entire matrix multiplying δx is symmetric.

□ The nature of matrix K

We previously used the approximation

$$f^{el}(x^{n+1}) \cong f^{el}(x^n) + K(x^{n+1} - x^n)$$

or $$f^{el}(x^n+\delta x) \cong f^{el}(x^n) + K\delta x$$

What is such a matrix that would make an approximation like that be reasonable?

For a 1D function $f(x)$, Taylor's 1st order approximation gives

$$f(x_0+\delta x) \approx f(x_0) + f'(x_0)\,\delta x$$

when $f\in\mathbb{R}^n$ & $x\in\mathbb{R}^n$ the respective Taylor Approximation is

$$f(x_0+\delta x) \approx f(x_0) + Jf(x_0)\cdot\delta x$$

Thus we can generate an approximation

$$f^{el}(x^n+\delta x) \approx f^{el}(x^n) + K\delta x$$

By taking $K := Jf^{el}(x^n)$

The Jacobian $Jf^{el}(x^n)$ is an $N \times N$ matrix (N = degrees of freedom, not # of particles) with elements

$$\left[Jf^{el}(x^n)\right]_{ij} = \frac{\partial f^{el}_{(i)}}{\partial x_{(j)}}(x^n)$$

We don't need to worry at this point about computing these partial derivatives "on paper". We will see a ready-to-use formula later on...

A note on naming

$$\underbrace{f(x^n+\delta x) - f(x^n)}_{\text{Force difference, resulting from a position change } \delta x} \approx \underbrace{K\delta x}_{\text{Force differential}}$$

Force differential
i.e. the approximation of the change in force, using Taylor's formula

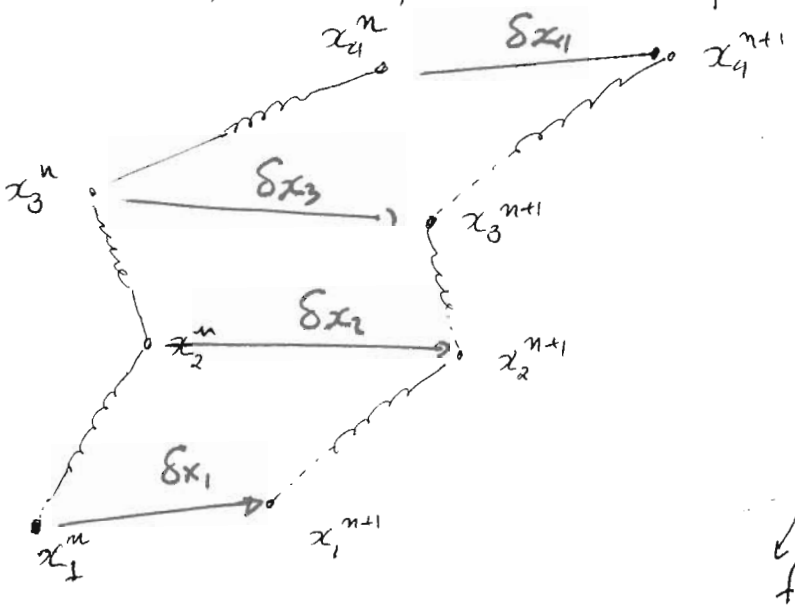
We use the notation $\delta f := K\delta x$ for the force differential

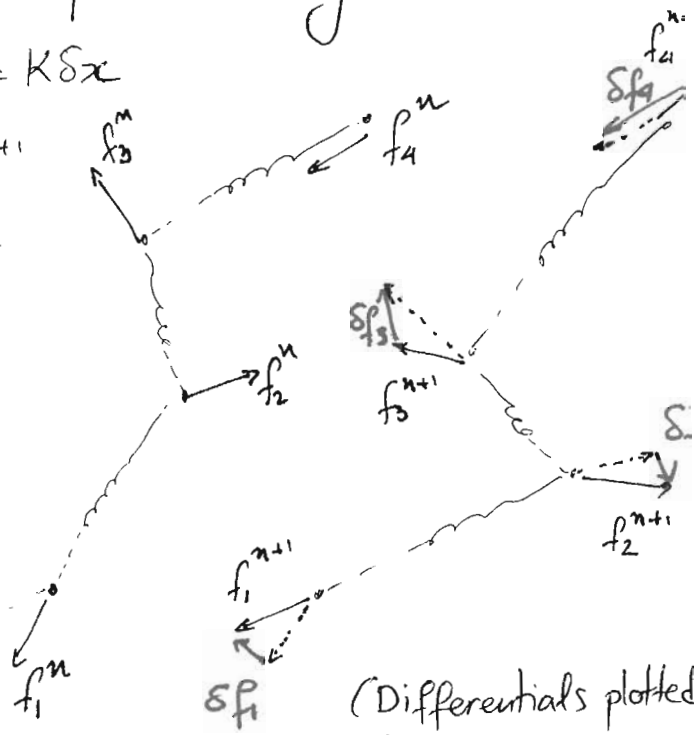
Compare:

$f^d = G \cdot v \Rightarrow$ Damping forces are a linear function of velocities

$\delta f = K \cdot \delta x \Rightarrow$ Spring force differentials are a linear function of the changes in position

Instead of writing down a formula for K (yet) we focus on explaining <u>how</u>, given a position change δx, we compute the force differential $\delta f = K\delta x$

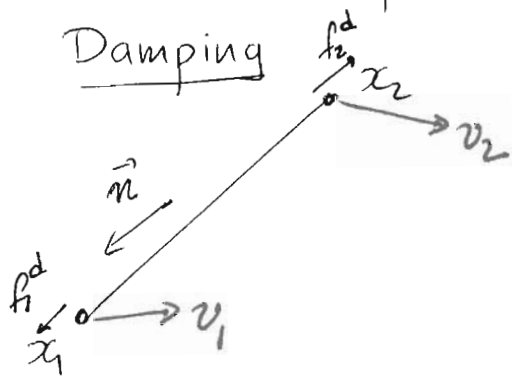




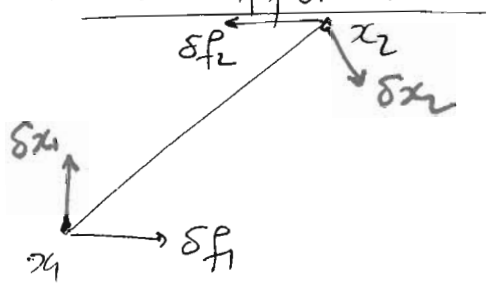
(Differentials plotted as differences, since they approximate them)

The force differential, for each particle, is the result of the contribution from each spring (compare: the damping forces are accumulated from the contribution of each spring

Damping



Force differential



$$f_1^d = -b\, n \cdot n^T (v_1 - v_2)$$

$$f_2^d = -f_1^d$$

$$= b\, n n^T (v_1 - v_2)$$

$$\delta f_1 = -K^s(x_1, x_2)(\delta x_1 - \delta x_2)$$

$$\delta f_2 = -\delta f_1$$

$$= K^s(x_1, x_2)(\delta x_1 - \delta x_2)$$

Add_Damping_Force (x, v, f^d)

Foreach spring (i,j):

$$n = \frac{x_i - x_j}{\|x_i - x_j\|}$$

$$f_i^d \mathrel{+}= -bnn^T(v_i - v_j)$$

$$f_j^d \mathrel{+}= bnn^T(v_i - v_j)$$

Add_Force_Differential $(x, \delta x, \delta f)$

Foreach spring (i,j)

$$K^s = K^s(x_i, x_j) \quad \text{[TBD]}$$

$$\delta f_i \mathrel{+}= -K^s(\delta x_i - \delta x_j)$$

$$\delta f_j \mathrel{+}= K^s(\delta x_i - \delta x_j)$$

- CG is applied to solve system $A\delta x = b$, with

$$A = \frac{1}{dt^2}M - \frac{1}{dt}G(x^n) - K$$

$$b = \frac{1}{dt}Mv^n + \left\{g + f^{el}(x^n)\right\} \Rightarrow \text{Direct computation, easy!}$$

All we need to do to use a general purpose CG routine, is explain how, given an input vector $\underline{w}$, we can compute the product $A \cdot w$

$$Aw = \left[\frac{1}{dt^2}M - \frac{1}{dt}G(x^n) - K\right]w$$

$$= \frac{1}{dt^2}Mw \rightsquigarrow \text{Easy!}$$

$$- \frac{1}{dt}G(x^n)w \rightsquigarrow \text{Interpret } w \text{ "as } v\text{"}, \text{ call Add\_Damping}(x, w, f^d)$$
$$\Rightarrow f^d = G(x^n)w !$$

$$- Kw \rightsquigarrow \text{Interpret } w \text{ "as } \delta x\text{"}, \text{ call Add\_Differential}(x, w, \delta f)$$
$$\Rightarrow \delta f = Kw !$$

□ Definiteness For CG to work we need the matrix

$$A = \frac{1}{dt^2} M - \frac{1}{dt} G - K$$ to be positive definite

↳ ???

↳ Guaranteed positive definite

↳ Diagonal, with positive entries (=eigenvalues) ⇒ ok!

The following can be shown:

⇒ If every matrix $K_s(x_i, x_j)$ is positive definite, then the global matrix $(-K)$ is positive definite, too (but, it is also possible that some K^s are not +ive definite, yet the global one manages to be).

$$K_s(x_i, x_j) = \underbrace{k\left(\frac{1}{l_0} - \frac{1}{l}\right)(I - nn^T)}_{\text{Positive definite only when } l > l_0!} + \underbrace{\frac{k}{l_0} nn^T}_{\text{Always positive definite}}$$

Note: $l = \|x_i - x_j\|$

$$n = \frac{x_i - x_j}{\|x_i - x_j\|}$$

Thus, if we use this formula as is, we may risk the CG matrix losing definiteness ⇒ CG will not work! (or go unstable).

3 options

I. Do nothing special, hope for the best

→ Matrix $-K$, although not guaranteed to be definite, is averaged together with positive definite matrices $\frac{1}{dt^2}M$ & $-\frac{1}{dt}G$. Adding these terms does have a chance to boost the eigenvalues of the sum to positive values, and guarantee definiteness. This is certainly the case when $dt << 1$, since the division $\frac{1}{dt^2}M$ & $-\frac{1}{dt}G$ makes these terms even stronger. However if we are forced to take too small steps to avoid visible instability, this would defy the purpose of using Backward Euler.

II. Throw away the "negative definite" fraction of K, on a case-by-case basis

The term of K with the potential of being negative definite is $k\left(\frac{l}{l_0}-\frac{1}{l_0}\right)(I-nn^T)$, and does in fact become -ive definite when $l < l_0$. We could set

$$K^s = \begin{cases} k\left(\frac{l}{l_0}-\frac{1}{l_0}\right)(I-nn^T)+\frac{k}{l_0}nn^T, & \text{when } l > l_0 \text{ (spring extended)} \\ \frac{k}{l_0}nn^T, & \text{when } l < l_0 \text{ (spring compressed).} \end{cases}$$

Of course, this is yet another simplification/approximation that

moves us farther away from the "proper" application of Backward Euler. However, this will drastically improve the robustness of CG as a solver and is generally a very viable compromise.

III. Always throw away the "potentially -ive definite" part of K

i.e. set $K^S = \frac{k}{l_0} nn^T$ ALWAYS

Downside: Yet another approximation. May require just a few extra CG iterations to match option #2 in quality.

Upside: Easy. Always robust. Also, with this definition of K^S, note the similarity between damping forces & spring force differentials

Add_Damping

Foreach spring (i,j):

$$f_i^d \mathrel{+}= -bnn^T(v_i - v_j)$$

$$f_j^d \mathrel{+}= bnn^T(v_i - v_j)$$

Add_Differential

Foreach spring (i,j):

$$\delta f_i \mathrel{+}= -\frac{k}{l_0} nn^T(\delta x_i - \delta x_j)$$

$$\delta f_j \mathrel{+}= \frac{k}{l_0} nn^T(\delta x_i - \delta x_j)$$

Essentially we can have just 1 function call do both of these operations, by simply specifying the appropriate constants...

Boundary (Constrained) Values

When solving $A\,\delta x = b$ using CG, we need to provide information for those δx_i's that correspond to constrained (i.e. kinematically predetermined) particles.

Specifically, in PhysBAM's CG implementation:

- Project() : Needs to zero-out elements corresponding to constrained particles, as always
- Set_Boundary_Conditions(δx):

  Needs to set those elements corresponding to kinematic particles to their <u>known</u> <u>δx value</u> (i.e. position change)

For each constrained p_i

$$\delta x_i \leftarrow x_i^{n+1} - x_i^{n}$$

(x_i^{n+1}: known!)