We previously focused on the time integration scheme

$$x^{n+1} = x^n + \frac{dt}{2}\{v^n + v^{n+1}\} \quad (1)$$

$$v^{n+1} = v^n + dt\, M^{-1} f(x^n, v^{n+1}) \quad (2)$$

In order to solve this equation, we substituted

$$f(x,v) = \underbrace{g}_{\text{Gravity, or other external forces}} + \underbrace{f^{el}(x)}_{\text{elastic (Hookean) spring forces}} + \underbrace{f^{d}(x,v)}_{\text{Damping forces}}$$

and observed that the damping forces (as defined) can be written in the form

$$f^d(x,v) = G(x) \cdot v$$

(Furthermore, G is a symmetric matrix, and also negative definite, i.e $-G$ is positive definite)

With these assumptions, equation (2) becomes:

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

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

Lastly, to preserve the symmetry of the matrix on the left-hand-side we multiply everything from the left with M:

$$\Rightarrow \underbrace{\left\{M - dt\,G(x^n)\right\}}_{\text{Symmetric \& positive definite.}} v^{n+1} = Mv^n + dt\left\{g + f^{el}(x^n)\right\} \quad (3)$$

Equation (3) can be solved using the Conjugate Gradients algorithm, to yield v^{n+1}. This is then substituted into equation (1) to obtain x^{n+1}.

Before moving forward, we demonstrate yet another equivalent approach for solving eq. (2): We introduce the "velocity change" δv, defined as: $\delta v := v^{n+1} - v^n$, or $v^{n+1} = v^n + \delta v$

Substitute into (2):

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

$$\left\{I - dt\,M^{-1}G(x^n)\right\}\delta v = dt\,M^{-1}\underbrace{\left\{g + f^{el}(x^n) + G(x^n)v^n\right\}}_{= f^{total}(x^n, v^n)}$$

$$\Rightarrow \left\{M - dt\,G(x^n)\right\}\delta v = dt\,f(x^n, v^n)$$

$$\Rightarrow \left\{\frac{1}{dt}M - G(x^n)\right\}\delta v = f(x^n, v^n). \quad (4)$$

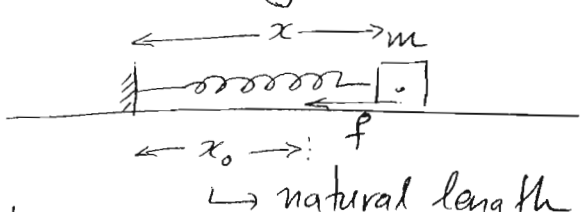
Equation (4) is also solvable with CG, as (3) is, but may be somewhat more resilient to loss of precision & cancellation, when δv is <u>small</u> (e.g. when an object comes to rest).

The aforementioned technique requires that $dt < \frac{2\,b\,l_0}{k}$ (at least) for stability. It is tempting to try and avoid any restrictions on dt, by using a technique such as Backward Euler:

$$\left.\begin{aligned} x^{n+1} &= x^n + dt\, v^{n+1} \\ v^{n+1} &= v^n + dt\, M^{-1} f(x^{n+1}, v^{n+1}) \end{aligned}\right\} \quad (*)$$

Although applying system $(*)$ to compute x^{n+1} & v^{n+1} would eliminate the timestep restriction, solving $(*)$ is complicated since $f(x,v)$ has a nonlinear dependence on x. Contrast that with v, which only contributes a linear term $f^d = G(x^n)\, v^{n+1}$

Temporarily, let us examine what this method would look like for a single spring, in 1 dimension:



$$f(x,v) = f^{el}(x) + f^{d}(x,v)$$
$$= -k(x - x_0) - bv$$

Thus

$$x^{n+1} = x^n + dt\, v^{n+1} \quad (5)$$

$$v^{n+1} = v^n + \frac{dt}{m}\left(-k(x^{n+1} - x_0) - b v^{n+1}\right) \quad (6)$$

From equations (5), (6) we can annihilate one of x^{n+1} or v^{n+1} to make the solution easier. Either choice works. We will try here to annihilate v^{n+1} leaving behind an equation for x^{n+1} alone. Just as before, we will also define

$$\delta x := x^{n+1} - x^n \quad \text{or} \quad x^{n+1} = x^n + \delta x$$

$$\text{Eq (5)} \Rightarrow x^{n+1} - x^n = dt\, v^{n+1} \Rightarrow \delta x = dt\, v^{n+1} \Rightarrow v^{n+1} = \frac{1}{dt}\delta x$$

$$\text{Eq (6)} \Rightarrow \frac{1}{dt}\delta x = v^n + \frac{dt}{m}\left\{-k\left(x^n + \delta x - x_0\right) - b\,\frac{1}{dt}\delta x\right\}$$

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

$\Rightarrow$ Multiply everything $\frac{m}{dt}\times$:

$$\left(\frac{m}{dt^2} + \frac{b}{dt} + k\right)\delta x = \frac{m}{dt}v^n - k(x^n - x_0)$$

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

Solving for δx yields $x^{n+1} = x^n + \delta x$

and $v^{n+1} = \frac{1}{dt}\delta x$

Looking more closely at this manipulation, the reason it was successful was that the force

$$f(x,v) = -k(x-x_0) - bv$$

was <u>linear</u>, both in x & v (affine, actually).

This is <u>not</u> the case, however, with our 3D spring forces

$$f = -k\left(\frac{l}{l_0} - 1\right)\frac{x_1 - x_2}{l}$$

$$= -k\left(\frac{\|x_1 - x_2\|_2}{l_0} - 1\right)\frac{x_1 - x_2}{\|x_1 - x_2\|_2}$$

squares, square roots etc.

(What we would have needed, for linearity, would have looked like $f = A \cdot x + b$ ...)

In the interest of easy solvability we will make a certain <u>approximation</u> and replace

$$f(x^{n+1}, v^{n+1}) \longleftarrow f^{\text{modified}}(x^{n+1}, v^{n+1})$$

$$= Ax^{n+1} + Bv^{n+1} + C$$

which will allow us to mimic the previous solution process.

Of course, we are giving up some physical fidelity in doing so, but we are using Backward Euler to be able to take <u>large</u> dt's, which is a similar conscious compromise of accuracy vs. cost.

We will perform this approximation for every component of force. For example for the damping force we will do the following:

$$f^d(x^{n+1}, v^{n+1}) = G(x^{n+1})v^{n+1} \approx G(x^n)v^{n+1}$$

nonlinear! (in x^{n+1} & v^{n+1})

↳ linear!

The elastic forces are significantly more challenging. Let's look at them, 1 spring at a time

$$f_1(x_1^{n+1}, x_2^{n+1}) = -k\left(\frac{\|x_1^{n+1} - x_2^{n+1}\|}{l_0}\right)\frac{x_1^{n+1} - x_2^{n+1}}{\|x_1^{n+1} - x_2^{n+1}\|} \overset{?}{\approx} M_1^? x_1 + M_2^? x_2 + b^?$$

$$f_2(x_1^{n+1}, x_2^{n+1}) = -f_1$$

Matters are complicated because f_1, f_2 are <u>vector-valued</u> functions of <u>vector-valued</u> arguments x_1 & x_2. If this was just a real-valued function of a single variable, one way for constructing such an approximation would be suggested by the Taylor series

$$f(x) = f(x_*) + f'(x_*)\cdot(x - x_*) + f''(x_*)\frac{(x-x_*)^2}{2!} + \cdots$$

Thus we can take $f(x) \approx f(x_*) + f'(x_*)(x - x_*)$

$$= \underbrace{f(x_*) - f'(x_*)x_*}_{\text{const.}} + \underbrace{f'(x_*)}_{\text{const.}}\cdot x = ax + b$$

There is in fact a vector-equivalent version of this theorem. If $f : \mathbb{R}^n \to \mathbb{R}^n$

$$x \mapsto f(x)$$

then:

$$f(x) \approx f(x_*) + Jf(x_*)(x - x_*) \qquad (7)$$

$f(x) \in \mathbb{R}^n$, $f(x_*) \in \mathbb{R}^n$, $Jf(x_*)$ is $n \times n$, $(x - x_*) \in \mathbb{R}^n$

if $f = (f_1, f_2, f_3, \ldots, f_n)$ and $x = (x_1, x_2 \ldots, x_n)$

the Jacobian Jf is defined as follows:

$$Jf = \begin{pmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \cdots & \partial f_1/\partial x_n \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \cdots & \partial f_2/\partial x_n \\ \vdots & & \vdots \\ \partial f_n/\partial x_1 & \cdots & \partial f_n/\partial x_n \end{pmatrix}$$

and $Jf(x_*)$ means that all partial derivatives are evaluated at $x = x_*$.

We will go ahead and apply this approach to the spring force function $f_1(x_1, x_2) = -k\left(\frac{\ell}{\ell_0} - 1\right)\frac{x_1 - x_2}{\ell}$

Here $x : (x_1, x_2)$
6D 3D 3D

In fact we will use the following specific values for x & x_*:

$x \leftarrow x^{n+1}$

$x_* \leftarrow x^n$

Thus, we seek an approximation of the form:

$$f_1(x_1^{n+1}, x_2^{n+1}) \approx f(x_1^n, x_2^n) + Jf_1(x_1^n, x_2^n)(x_1^{n+1} - x_1^n, x_2^{n+1} - x_2^n)$$

$$= Jf_1(x_1^n, x_2^n)(\delta x_1, \delta x_2)$$

Based on the prior definition

$$\delta x_1 = x_1^{n+1} - x_1^n$$
$$\delta x_2 = x_2^{n+1} - x_2^n$$

The matrix $Jf_1(x^n)$ can be computed by (painful...) differentiation of the force definition. We omit the algebra here, and summarize the result as follows:

The entire product $Jf_1(x^n) \cdot (\delta x)$ is equal to:

$$Jf_1(x^n)\,\delta x = \left\{ -k\left(\frac{1}{l_0} - \frac{1}{l(x^n)}\right)\left(I - n(x^n)n(x^n)^T\right) - \frac{k}{l_0} n(x^n)n(x^n) \right\} \cdot (\delta x_1 - \delta x_2)$$

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

Notes: $l(x^n) = \|x_1^n - x_2^n\|_2$

$$n(x^n) = \frac{x_1^n - x_2^n}{\|x_1^n - x_2^n\|}$$

Since $f_2 = -f_1$ we can write the force approximation for <u>both</u> forces as

$$\left.\begin{aligned} f_1(x^{n+1}) &\approx f_1(x^n) - K^s(x^n)(\delta x_1 - \delta x_2) \\ f_2(x^{n+1}) &\approx f_2(x^n) + K^s(x^n)(\delta x_1 - \delta x_2) \end{aligned}\right\} \Rightarrow$$

$$\underbrace{\begin{bmatrix} f_1(x^{n+1}) \\ f_2(x^{n+1}) \end{bmatrix}}_{f_{ij}(x^{n+1})} \approx \begin{bmatrix} f_1(x^n) \\ f_2(x^n) \end{bmatrix} + \underbrace{\begin{bmatrix} -K^s(x^n) & K^s(x^n) \\ K^s(x^n) & -K^s(x^n) \end{bmatrix}}_{K_{ij}(x^n)} \begin{bmatrix} \delta x_1 \\ \delta x_2 \end{bmatrix}$$

or $$f_{ij}(x^{n+1}) \approx f_{ij}(x^n) + K_{ij}(x^n) \cdot \delta x$$

By adding the contribution of <u>all</u> springs, we arrive at the "global" approximation

$$\boxed{f^{el}(x^{n+1}) = f^{el}(x^n) + K(x^n) \cdot \delta x}$$

(Compare with $f^d(x, v) = G(x) \cdot v$)

A note on naming:

$$\delta f := \underbrace{f(x^{n+1}) - f(x^n)}_{\text{"Real" force difference}} \approx \underbrace{K(x^n)\,\delta x = K(x^n)(x^{n+1} - x^n)}_{\text{Force "differential"}}$$

This gives rise to our final approximation, for all forces

$$f^{\text{total}}(x^{n+1}, v^{n+1}) \approx \underbrace{g}_{\text{Actual gravity}} + \underbrace{f^{el}(x^n) + K(x^n)(x^{n+1} - x^n)}_{\text{"Linearized" spring force}} + \underbrace{G(x^n)\,v^{n+1}}_{\text{Damping.}}$$

By using these approximations, Backward Euler becomes:

$$x^{n+1} = x^n + dt\, v^{n+1} \Rightarrow \boxed{v^{n+1} = \frac{1}{dt}\delta x}$$

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

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

$\Rightarrow$

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

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

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

Process:

→ Form right-hand side

→ Solve for δx (with CG?)

→ Compute $x^{n+1} = x^n + \delta x$

→ Compute $v^{n+1} = \frac{1}{dt} \delta x$

Implementation notes:

→ For CG, we need to provide boundary conditions (a.k.a. constrained values) for any components of δx that are kinematically specified. If a particle x_p is moved on a pre-defined trajectory, we need to set $\delta x_p = x_p^{n+1} - x_p^n$ (x_p^{n+1}: known)

→ Due to the approximations we make, this scheme may not achieve the 100% of stability guarantees that B.E. promises. (It generally is quite robust, though)

→ As before, we never really compute the matrix that is fed into CG explicitly. All we need is some procedure that, given a vector of values, e.g. w, computes: