Last Time:

- Floating-base dynamics in maximal coordinates
- Variational integrators " "

Today:

- "Fast" dynamics algorithms

What does "Fast" mean?

- Classical Lagrangian dynamics requires $O(n^3)$ floating-point operations to compute accelerations (DoF)

$$M(q)\dot{v} + C(q,v) = F$$

$$\Rightarrow \dot{v} = \underbrace{M^{-1}(q)}(F - C(q,v))$$

Solving an $n \times n$ linear system is $O(n^3)$

- This result holds for discrete-time dynamics with RK methods etc. (just changes the constant in front of n^3)

- A naive approach to maximal-coordinate dynamics is even worse:

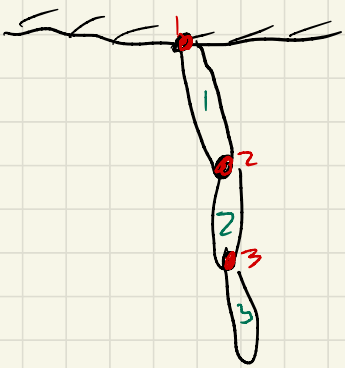
$$\left[\begin{array}{c|c} M & C^T \\ \hline C & 0 \end{array}\right] \left[\begin{array}{c} x \\ \hline \lambda \end{array}\right] = \left[\begin{array}{c} d \\ \hline c \end{array}\right]$$

$O((N_{bodies} + N_{constraints})^3)$

- "Fast" generally refers to algorithms that can beat the naive $O(n^3)$ complexity.

Sparsity Structure:

- Let's look closer at a multi-link pendulum:



- The maximal-coordinate KKT system has the following structure:

$$\begin{bmatrix} M_1 & & & C_1^{1T} & C_1^{2T} & \\ & M_2 & & & C_2^{2T} & C_2^{3T} \\ & & M_3 & & & C_3^{3T} \\ C_1^1 & & & & & \\ C_1^2 & C_2^2 & & & \mathbf{0} & \\ & C_2^3 & C_3^3 & & & \end{bmatrix} \begin{bmatrix} \dot{V}_1 \\ \dot{V}_2 \\ \dot{V}_3 \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

C_j^i ← constraint, ↖ link

Constraints:

$$d(q) = 0 \Rightarrow \frac{\partial d}{\partial q}\dot{q} = 0 \Rightarrow \underbrace{\frac{\partial d}{\partial q}}_{C_j^i}\dot{q} + \underbrace{\frac{d}{dt}\left(\frac{\partial d}{\partial q}\right)}_{-d}\dot{q} = 0$$

- Block structure repeats for any number of links.

- The key to building fast algorithms is exploiting this sparsity.

- Let's look at the dynamics of the last link:

$$M_3 \dot{v}_3 + C_3^{3T} \lambda_3 = f_3$$

$$\Rightarrow \dot{v}_3 = \underbrace{-M_3^{-1} C_3^{3T}}_{W_3} \lambda_3 + \underbrace{M_3^{-1} f_3}_{w_3}$$

$$= W_3 \lambda_3 + w_3$$

- Let's plug this into the last constraint:

$$C_2^3 \dot{v}_2 + C_3^3 \dot{v}_3 = c_3$$

$$\Rightarrow C_2^3 \dot{v}_2 + C_3^3 W_3 \lambda_3 + C_3^3 w_3 = c_3$$

$$\Rightarrow \lambda_3 = \underbrace{-(C_3^3 W_3)^{-1} C_2^3}_{S_3} \dot{v}_2 + \underbrace{(C_3^3 W_3)^{-1} [c_3 - C_3^3 w_3]}_{s_3}$$

$$= S_3 \dot{v}_2 + s_3$$

- Now move up to the dynamics for link 2:

$$M_2 \dot{v}_2 + C_2^{2T} \lambda_2 + C_2^{3T} \lambda_3 = f_3$$

- Plug in our previous equation for λ_3:

$$M_2\dot{v}_2 + C_2^{2T}\lambda_2 + C_2^{3T}(S_3\dot{v}_2 + s_3) = f_3$$

$$\Rightarrow \dot{v}_2 = \underbrace{-(M_2 + C_2^{3T}S_3)^{-1}C_2^{2T}}_{W_2}\lambda_2 + \underbrace{(M_2 + C_2^{3T}S_3)^{-1}(f_2 - C_2^{3T}s_3)}_{w_2}$$

$$= W_2\lambda_2 + w_2$$

- Plug into the next constraint equation:

$$C_1^2\dot{v}_1 + C_2^2\dot{v}_2 = c_2$$

$$\Rightarrow C_1^2\dot{v}_1 + C_2^2[W_2\lambda_2 + w_2] = c_2$$

$$\Rightarrow \lambda_2 = \underbrace{-(C_2^2W_2)^{-1}C_1^2}_{S_2}\dot{v}_1 + \underbrace{(C_2^2W_2)^{-1}(c_2 - C_2^2w_2)}_{s_2}$$

$$= S_2\dot{v}_1 + s_2$$

- Now we have a recursion:

$$W_n = -(M_n + C_n^{n+1T}S_{n+1})^{-1}C_n^{nT}$$

$$w_n = (M_n + C_n^{n+1T}S_{n+1})^{-1}(f_n - C_n^{n+1T}s_{n+1})$$

$$S_n = -(C_n^nW_n)^{-1}C_{n-1}^n$$

$$s_n = (C_n^nW_n)^{-1}(c_n - C_n^nw_n)$$

- Once we get to link/joint 1, we can solve for λ_1:

$$C_1^T \dot{V}_1 = C_1$$

plug in $\dot{V}_1 = W_1 \lambda_1 + w_1$

$$\Rightarrow C_1^T W_1 \lambda_1 + C_1^T w_1 = C_1$$

$$\Rightarrow \lambda_1 = (C_1^T W_1)^{-1} (C_1 - C_1^T w_1)$$

- Now we can set up a forward recursion for all of the $\dot{V}_n$ and λ_n:

$$\dot{V}_1 = W_1 \lambda_1 + w_1$$

$$\lambda_2 = S_2 \dot{V}_1 + s_2$$

$$\vdots$$

$$\dot{V}_n = W_n \lambda_n + w_n$$

$$\lambda_{n+1} = S_n \dot{V}_n + w_n$$

- What is the complexity of our new algorithm?

$$W_n = -\underbrace{(M_n + C_n^{n+1T} S_{n+1})^{-1}}\, C_n^{nT}$$

rigid-body mass matrix
6×6 in 3D
3×3 in 2D

$$S_n = -\underbrace{(C_n^n W_n)^{-1}}\, C_{n-1}^n$$

Joint constraint
5×5 in 3D
2×2 in 2D

- These linear systems are all fixed size (and they're small / fast to solve).

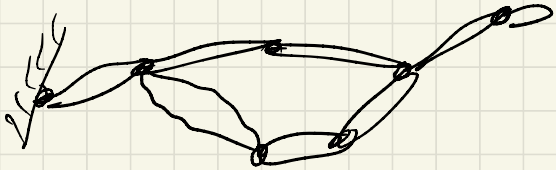
- We solve n of these for an n-link pendulum:

$$\Rightarrow O((6^3 + 5^3)n) = O(n)$$

Linear complexity!

- It is generally possible to achieve $O(n)$ complexity for any systems that are "open kinematic chains":

- Closed loops are problematic:



$$C = \begin{bmatrix} C_1^1 & & & & & & \\ C_1^2 & C_2^2 & & & & & \\ & C_3^2 & C_3^3 & & & & \\ & & C_4^3 & C_4^4 & & & \\ & & & C_5^4 & C_5^5 & & \\ C_1^6 & & & & & C_5^6 & \end{bmatrix}$$

no longer banded
$\Rightarrow$ breaks recursive factorization