



Linear-Time Dynamics Using Lagrange Multipliers

David Baraff
CMU

Presentation by: Elif Tosun



Outline

- Introduction
- Motivation
- Lagrange Multiplier Formulation
- Approach
- Sparse Solution
- Auxiliary Constraints
- Results

Introduction

Problem: “Forward simulation with constraints”

Properties:

- Sparse constraints for articulated figures
n bodies, $O(n)$ constraints
- Nearly or completely acyclic systems - robot arms, humans...

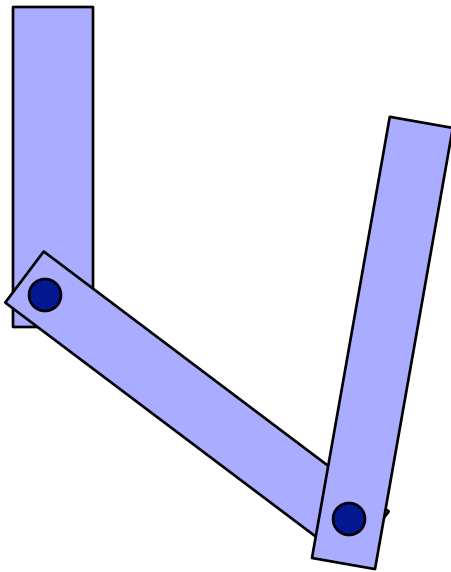
Methods:

7. *Reduced Coordinate*: Reduce the # of coordinates needed to describe system's state
8. *Constraint Forces*: Introduce additional forces into system to maintain constraints (*)



Very Basic Example

- Given: n bodies



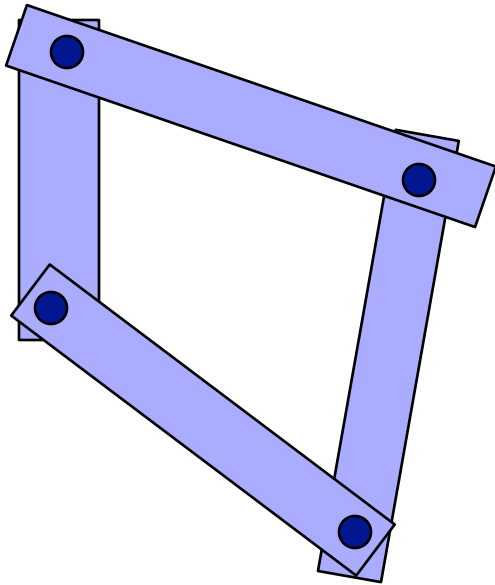
- Primary Constraints
 - $n-1$
 - Each between 2 bodies
 - Acyclic
 - Can be non-holonomic (can be velocity dependent)
 - Must be equality

Example

- Given: n bodies

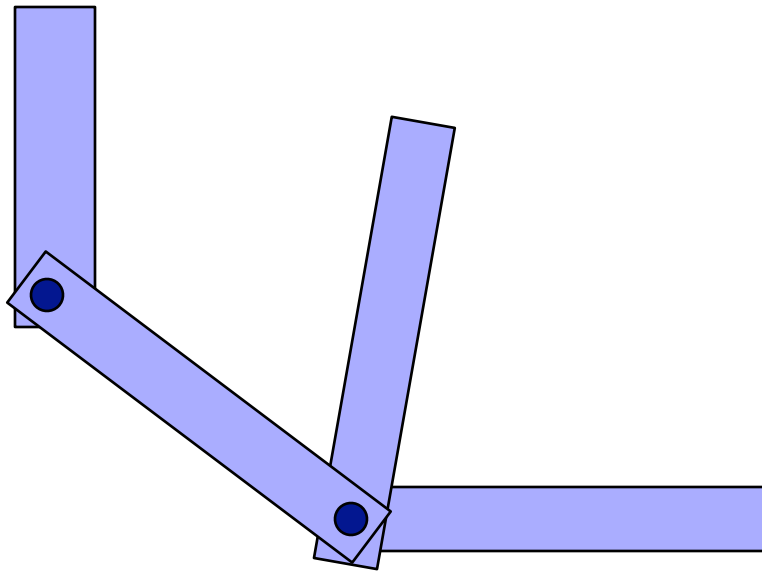
- Auxiliary Constraints

- Closed loops



Example

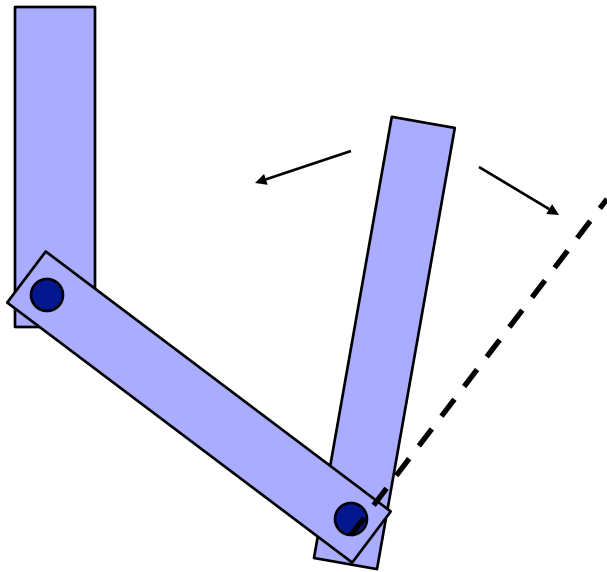
- Given: n bodies



- Auxiliary Constraints
 - Closed loops
 - Between 3+ bodies

Example

- Given: n bodies



- Auxiliary Constraints
 - Closed loops
 - Between 3+ bodies
 - Inequality Constraints

Overview of method

- Direct (non-iterative) method
- Constraints can be of various dimensions
- Bodies need not be rigid
- Uses a very simple sparse-matrix technique
- No steep learning curve
- Easy to implement
- Auxiliary method can be used with reduced coordinate methods too!

Motivation

- Given: a system with m d.o.f (called "*maximal coordinates*")
+ set of constraints that remove c d.o.f.

Reduced Coordinate Methods:

Method: Parameterize remaining $n=m-c$ d.o.f. using reduced set of n coordinates (called "*generalized coordinates*")

Cons:

- parameterization very hard
- if one can be found - need $O(n^3)$ time needed for acceleration computation.

Pros :

- loop-free articulated bodies - $O(n)$ time achievable
- Eliminates drifting problem in multiplier methods (need constraint stabilization)
- “may” run faster due to larger time steps taken by integrator

Motivation

Lagrange Multiplier Methods:

Method:

- use the maximal coordinates (m d.o.f.) + *new* constraint forces.
- Basis for constraint forces known apriori.
- Lagrange multipliers: vector of scalar coordinates for linear combination

Pros :

- Allow an arbitrary set of constraints to be combined.
- Allow/encourage highly modular knowledge/software design (bodies, constraints, geometry)
- Handle non-holonomic constraints(e.g. velocity-dependent)
- No need for parameterization

Cons:

- solving a $O(n) \times O(n)$ system.

BUT: This method takes linear time!

Lagrange Multiplier Formulation

- Bodies: $\mathbf{M}\dot{\mathbf{v}} = \mathbf{F}$
- Constraints: $\mathbf{j}_{i1}\dot{\mathbf{v}}_1 + \cdots + \mathbf{j}_{ik}\dot{\mathbf{v}}_k + \cdots + \mathbf{j}_{in}\dot{\mathbf{v}}_n + \mathbf{c}_i = \mathbf{0}$
 - Linear condition on the acceleration
 - For primary: Only 2 \mathbf{j}_{ik} will be non-zero for constraint i .
 - All constraints: $\mathbf{J}\dot{\mathbf{v}} + \mathbf{c} = \mathbf{0}$
 - “Workless force”

$$\mathbf{F}_i^c = \begin{pmatrix} \mathbf{j}_{i1}^T \\ \vdots \\ \mathbf{j}_{in}^T \end{pmatrix} \lambda_i \quad \Rightarrow \quad \mathbf{F}^c = \mathbf{J}^T \boldsymbol{\lambda}$$

Want to find $\boldsymbol{\lambda}$ s.t. constraint force + ext force produce motion that also satisfies constraints

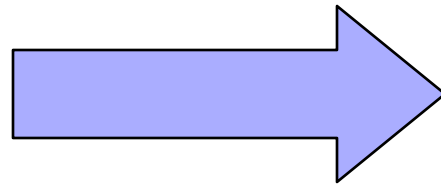
Lagrange Multiplier Formulation

$$\mathbf{M}\dot{\mathbf{v}} = \mathbf{F}$$

$$\mathbf{M}\dot{\mathbf{v}} = \mathbf{F}^c = \mathbf{J}^T \boldsymbol{\lambda}$$

$$\dot{\mathbf{v}} = \underbrace{\mathbf{M}^{-1} \mathbf{J}^T \boldsymbol{\lambda}}$$

$$\mathbf{J}\dot{\mathbf{v}} + \mathbf{c} = \mathbf{0}$$



$$\mathbf{J}\mathbf{M}^{-1} \mathbf{J}^T \boldsymbol{\lambda} = \mathbf{c}$$

Formulation

- Block-matrix formulation
- Block dimensions: based on body and constraint dimensions:
 - Body dim: # d.o.f. when unconstrained
 - Constraint dim: # of d.o.f. it removes
- Let p = largest dimension among bodies.
Block operations take const time.

Approach

Newton's
Law

$$\mathbf{M}\dot{\mathbf{v}} = \mathbf{J}^T \boldsymbol{\lambda} + \mathbf{F}^{\text{ext}}$$

$$\dot{\mathbf{v}} = \mathbf{M}^{-1} \mathbf{J}^T \boldsymbol{\lambda} + \mathbf{M}^{-1} \mathbf{F}^{\text{ext}}$$

constraints

$$\mathbf{J}\dot{\mathbf{v}} + \mathbf{c} = \mathbf{0}$$

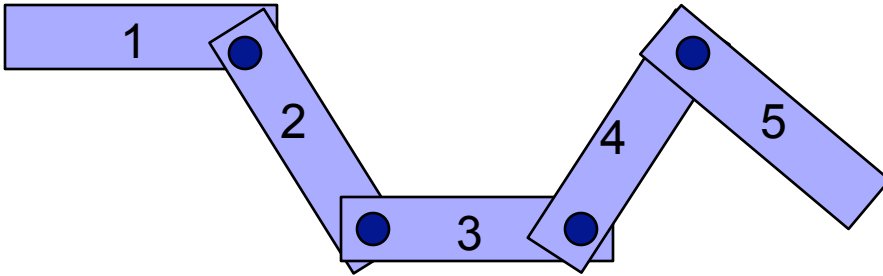
$$\mathbf{J}(\mathbf{M}^{-1} \mathbf{J}^T \boldsymbol{\lambda} + \mathbf{M}^{-1} \mathbf{F}^{\text{ext}}) + \mathbf{c} = \mathbf{0}$$

$$\mathbf{A} = \mathbf{J}\mathbf{M}^{-1} \mathbf{J}^T \quad \text{and} \quad \mathbf{b} = -(\mathbf{J}\mathbf{M}^{-1} \mathbf{F}^{\text{ext}} + \mathbf{c})$$

□ Now all we need is to solve is: $\mathbf{A} \mathbf{l} = \mathbf{b}$

$A \mathbf{l} = \mathbf{b}$

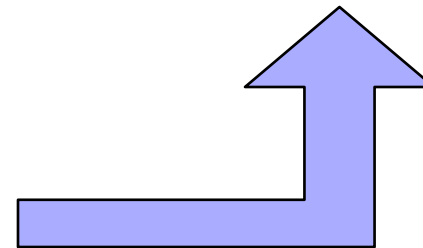
- if A not too large : Cholesky
- if Serial chain - Banded Cholesky – can take $O(n)$



$JM^{-1}J^T =$	A	A	0	0
	A	A	A	0
	0	A	A	A
	0	0	A	A

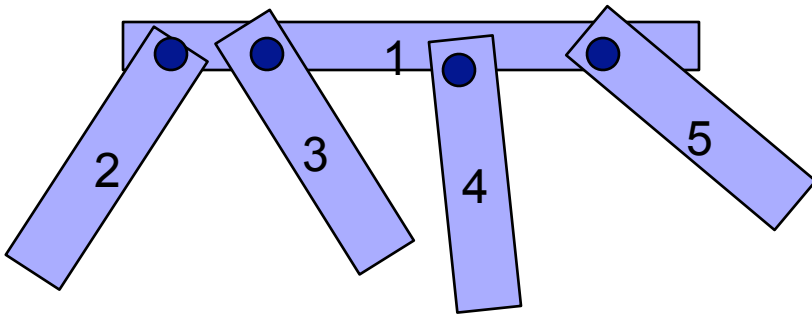
$J =$	X	X			
		X	X		
			X	X	
				X	X

$M =$	Y				
		Y			
			Y		
				Y	
					Y



$A I = b$

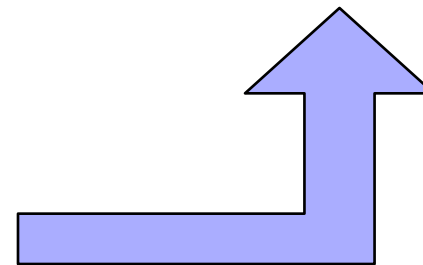
- If branched – A is completely dense... $O(n^3)$



$JM^{-1}J^T =$	A	A	A	A
	A	A	A	A
	A	A	A	A
	A	A	A	A

$J =$	X	X			
	X		X		
	X			X	
	X				X

$M =$	Y				
		Y			
			Y		
				Y	
					Y



Need a sparse formulation!

Sparse Formulation

$$\underbrace{\begin{pmatrix} \mathbf{M} & -\mathbf{J}^T \\ -\mathbf{J} & \mathbf{0} \end{pmatrix}}_{\mathbf{H}} \begin{pmatrix} \mathbf{y} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ -\mathbf{b} \end{pmatrix}$$

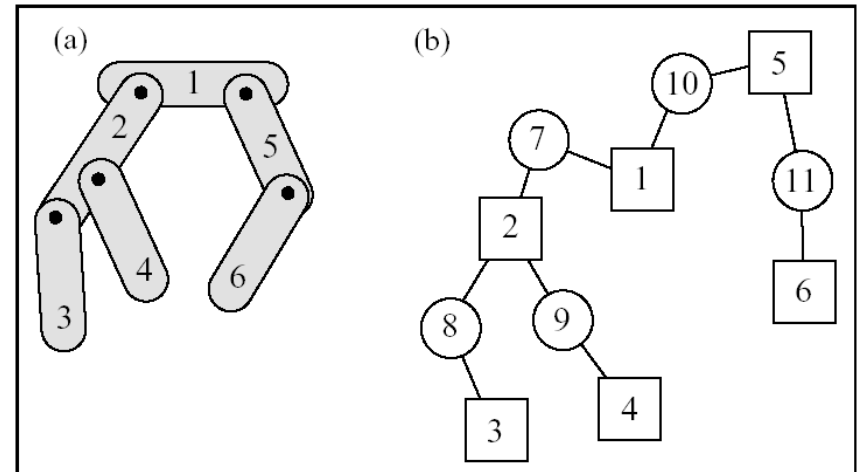
$$\mathbf{M}\mathbf{y} - \mathbf{J}^T\boldsymbol{\lambda} = \mathbf{0} \rightarrow \mathbf{y} = \mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda}$$

$$-\mathbf{J}\mathbf{y} = -\mathbf{b} \rightarrow \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda} = \mathbf{b} \rightarrow \mathbf{A}\boldsymbol{\lambda} = \mathbf{b}$$

- Robotics/mech.eng literature
- H is always sparse
- Can be solved in $O(n)$

Non-Linear Solution

- Consider graph of H:



- Where the matrix looks like:

- Factor $H=LDL^T$ ($O(n^3)$)
- Then solve ($O(n^2)$)
 $H=LDL^T x=(0; -b)$

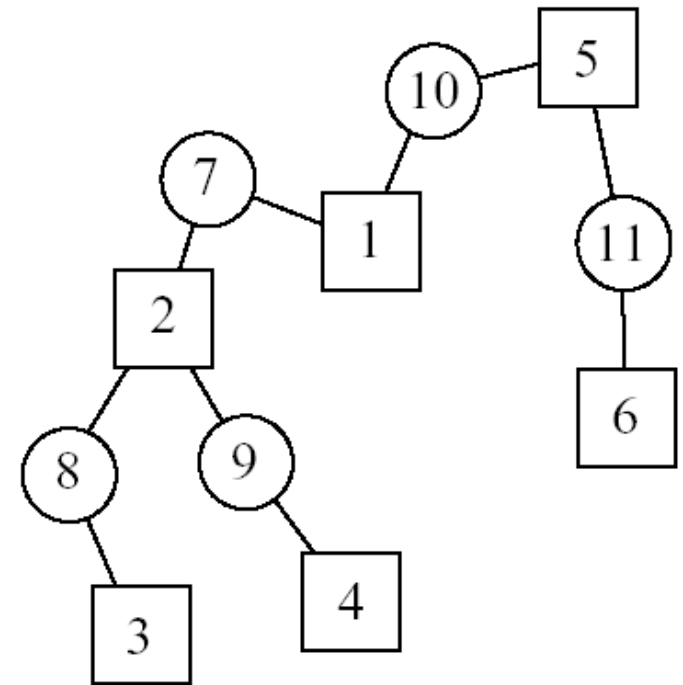
$$H = \begin{pmatrix} M_1 & 0 & 0 & 0 & 0 & 0 & j_{11}^T & 0 & 0 & j_{41}^T & 0 \\ 0 & M_2 & 0 & 0 & 0 & 0 & j_{12}^T & j_{22}^T & j_{32}^T & 0 & 0 \\ 0 & 0 & M_3 & 0 & 0 & 0 & 0 & j_{23}^T & 0 & 0 & 0 \\ 0 & 0 & 0 & M_4 & 0 & 0 & 0 & 0 & j_{34}^T & 0 & 0 \\ 0 & 0 & 0 & 0 & M_5 & 0 & 0 & 0 & 0 & j_{45}^T & j_{55}^T \\ 0 & 0 & 0 & 0 & 0 & M_6 & 0 & 0 & 0 & 0 & j_{56}^T \\ j_{11} & j_{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & j_{22} & j_{23} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & j_{32} & 0 & j_{34} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ j_{41} & 0 & 0 & 0 & j_{45} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & j_{55} & j_{56} & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Linear Solution

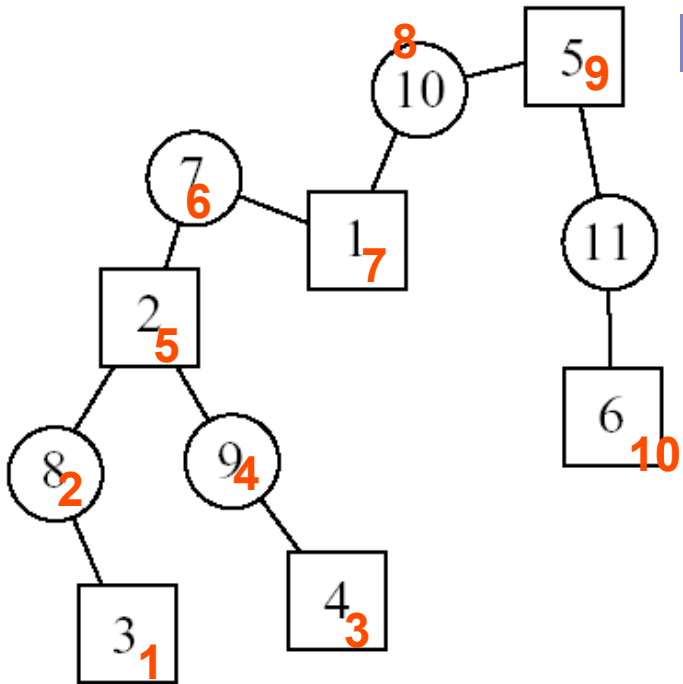
■ Sparse Matrix Theory:

“A matrix whose graph is acyclic possesses a **perfect elimination order**”

→ H can be reordered s.t. when factored the matrix L will be just as sparse as H.



$$\mathbf{H} = \begin{pmatrix} \mathbf{M}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{j}_{11}^T & \mathbf{0} & \mathbf{0} & \mathbf{j}_{41}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{j}_{12}^T & \mathbf{j}_{22}^T & \mathbf{j}_{32}^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M}_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{j}_{23}^T & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{M}_4 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{j}_{34}^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{M}_5 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{j}_{45}^T & \mathbf{j}_{55}^T \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{M}_6 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{j}_{56}^T \\ \mathbf{j}_{11} & \mathbf{j}_{12} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{j}_{22} & \mathbf{j}_{23} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{j}_{32} & \mathbf{0} & \mathbf{j}_{34} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{j}_{41} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{j}_{45} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{j}_{55} & \mathbf{j}_{56} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}$$



$$\begin{pmatrix} \mathbf{M}_3 & \mathbf{j}_{23}^T & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{j}_{23} & 0 & 0 & 0 & \mathbf{j}_{22} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{M}_4 & \mathbf{j}_{34}^T & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{j}_{34} & 0 & \mathbf{j}_{32} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{j}_{22}^T & 0 & \mathbf{j}_{32}^T & \mathbf{M}_2 & \mathbf{j}_{12}^T & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{j}_{12} & 0 & \mathbf{j}_{11} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{j}_{11}^T & \mathbf{M}_1 & \mathbf{j}_{41}^T & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{j}_{41} & 0 & \mathbf{j}_{45} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{j}_{45}^T & \mathbf{M}_5 & 0 & \mathbf{j}_{55}^T \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{M}_6 & \mathbf{j}_{56}^T \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{j}_{55} & \mathbf{j}_{56} & 0 \end{pmatrix}$$

Ordering

- Graph is a rooted tree: parent child relationship between every edge.
- Every node's index greater than its children's indices. (DFS)

Bookkeeping (child/parent relations)

- In each row only one non-zero block every occurs to the right of the diagonal (at most 1 parent/node)
- Eliminates one inner for loop in each algorithm

L can be computed in O(n) time and LDLtx=(b) can be solved in O(n) time.

$$\dot{\mathbf{v}} = \mathbf{M}^{-1} (\mathbf{J}^T \boldsymbol{\lambda} + \mathbf{F}^{\text{ext}})$$

Auxiliary Constraints

Idea:

- While computing multipliers for auxiliary constraints “anticipate” responses of primary constraints due to auxiliary forces.
- Once auxiliary constraints are computed, add primary constraints into system, which won't violate auxiliary constraints due to “anticipation”

Notation:

- For auxiliary constraints
 - $K \sim J$
 - $m \sim l$

Full Algorithm

1. Formulate sparse matrix H for primary constraints, and factor

$O(n)$

2. Given F^{ext} ,

■ Solve $\mathbf{A} \mathbf{l}_1 = -(\mathbf{J}\mathbf{M}^{-1}\mathbf{F}^{\text{ext}} + \mathbf{c})$ for \mathbf{l}_1

■ Compute primary constraint force due to F^{ext} :
 $\mathbf{J}^T \mathbf{l}_1$

■ Compute system's acceleration without auxiliary constraints: $\mathbf{v}^{\text{aux}} = \mathbf{M}^{-1}(\mathbf{J}^T \mathbf{l}_1 + \mathbf{F}^{\text{ext}})$.

$O(n)$

$$\mathbf{J}^a \dot{\mathbf{v}} + \mathbf{c}^a = \mathbf{a}$$

$$\mathbf{J}^a (\mathbf{M}^{-1} (\mathbf{F}^{\text{resp}} + \mathbf{k}_i) \mathbf{m} + \dot{\mathbf{v}}^{\text{aux}}) + \mathbf{c}^a = \mathbf{a}.$$

3. For each of k auxiliary constraints

- Solve $\mathbf{A} \mathbf{l}_m = -(\mathbf{J} \mathbf{M}^{-1} \mathbf{k}_m)$
- Compute response force by primary to constraint \mathbf{k}_m
 $\mathbf{F}^{\text{resp}} = \mathbf{J}^T \mathbf{l}_m$
- Fill coefficient matrix

$O(kn)$

4. Solve for auxiliary multipliers \mathbf{m}

$O(k^3)$

5. Given \mathbf{K}_m (the auxiliary constraint force)

$$\text{Solve } \mathbf{A} \mathbf{l}_{\text{final}} = -(\mathbf{J}\mathbf{M}^{-1}(\mathbf{K}_m + \mathbf{F}^{\text{ext}}) + \mathbf{c}) \quad O(n)$$

Compute primary constraints response to $\mathbf{K}_m + \mathbf{F}^{\text{ext}}$

$$\text{Total constraint force} = \mathbf{K}_m + \mathbf{J}^T \mathbf{l}_{\text{final}}$$

$$\text{Total external forces} = \mathbf{F}^{\text{ext}}$$

$$\text{NET FORCE acting} = \mathbf{K}_m + \mathbf{J}^T \mathbf{l}_{\text{final}} + \mathbf{F}^{\text{ext}}$$

6. Solve for net acceleration of system and move to next step.

$$\text{TOTAL RUNNING TIME: } O(n) + O(kn+k^3)$$

Results

- Run on SGI indigo 250Mhz, R4400 processors
 - 2D system : 54 primary constraints 7.75ms
 - 96 primary, 12 auxiliary ~25ms
 - 3D system : 96 primary, 3 auxiliary 18 ms
 - 127 primary: ~45ms
- Comparisons:
 - Baraff (Lagrange, $O(n^3)$)
 - For smaller systems 2 fold improvement
 - For bigger systems as big as a factor of 40
 - Schröder (Reduced Coordinate, $O(n)$)
 - Competitive + no need for SVD which causes ill-conditioning.
 - Bramble (Lagrange, Iterative, $O(n^2)$)
 - For smaller matrices competitive, for larger clearly faster