Solving under-constrained hyperelasticity without the null space

Leila Ghaffari, Toby Isaac, and Jed Brown

University of Colorado Boulder

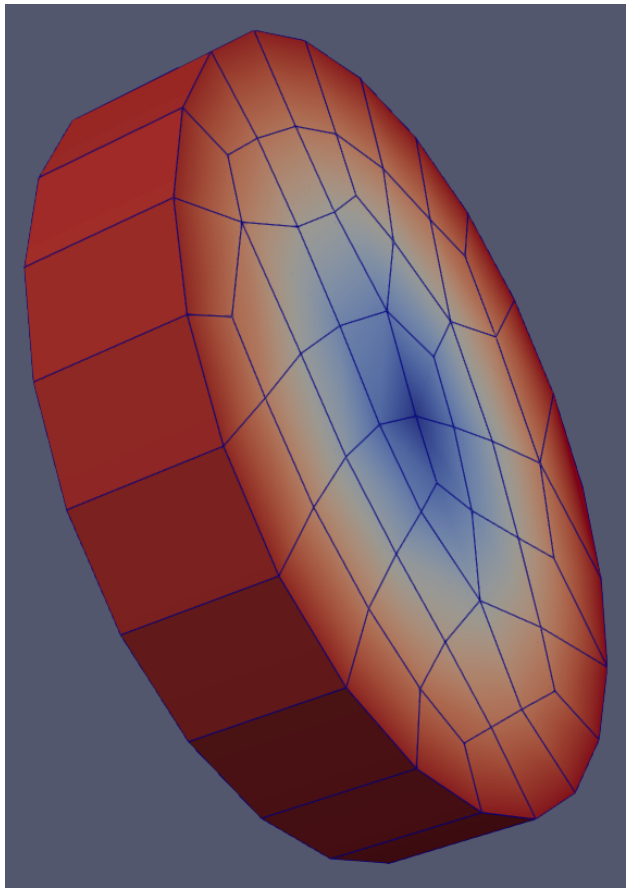
Leila.Ghaffari@colorado.edu

SIAM-PP24

March 6, 2024

Motivation & Goal

- Under-constrained hyperelasticity: free energy $f(\boldsymbol{x})$ has invariant motions
- Some motions (translations) are in the null space of the Hessian, but some (rotations) aren't
- How does this affect minimizing $f(\boldsymbol{u})$ using Newton-Krylov methods?



Our goal:

Robust Newton-Krylov for under-constrained hyperelasticity

Free Energy Functional & the Newton-Krylov Method

Free energy: $f(\boldsymbol{x})$

- $\boldsymbol{g}(\boldsymbol{x}_k) = \nabla f(\boldsymbol{x}_k) \longleftarrow$ gradient
- $H(\boldsymbol{x}_k) = \nabla^2 f(\boldsymbol{x}_k) \longleftarrow$ Hessian

Key Step: Solve $H_k \delta \boldsymbol{x}_k = -\boldsymbol{g}_k$ using Krylov method

Objective: Find $\boldsymbol{x}_*$ such that $f(\boldsymbol{x}_*)$ is local minimum

Important fact:

$$\text{if } g_k \perp \ker(H_k) \longrightarrow \delta \boldsymbol{x}_k = -H_k^{+} \boldsymbol{g}_k$$

Small Strain Invariants vs. Finite Strain Invariants

Small strain:

$$\boldsymbol{\varepsilon}(\boldsymbol{u}) = \frac{1}{2}\left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right)$$

The rigid body modes are in the null space of the symmetric gradient.

Finite strain:

$$\boldsymbol{e}(\boldsymbol{u}) = \frac{1}{2}\left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T + \underbrace{\nabla \boldsymbol{u} \nabla \boldsymbol{u}^T}_{\text{nonlinear}}\right)$$

The rigid body modes are **NOT** in the null space of the Green strain tensor.

Free Energy Restrictions: Objectivity & Invariance!

Coordinates of N particles of a free body without Dirichlet BCs:

$$\boldsymbol{x} = (x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_N, y_N, z_N)^T$$

Translational Invariants:

$\boldsymbol{t}_x = (1, 0, 0, 1, 0, 0, \ldots, 1, 0, 0)^T$

$\boldsymbol{t}_y = (0, 1, 0, 0, 1, 0, \ldots, 0, 1, 0)^T$

$\boldsymbol{t}_z = (0, 0, 1, 0, 0, 1, \ldots, 0, 0, 1)^T$

$$f(\boldsymbol{x} + \boldsymbol{t}) = f(\boldsymbol{x})$$

- is *subspace*:
  $\boldsymbol{t} \in \ker(H(\boldsymbol{x})), \boldsymbol{g} \perp \boldsymbol{t}$
- *doesn't* affect Krylov iteration

Rotational Invariants:

$\boldsymbol{q}$: an element in $SO(3)$
$\boldsymbol{R}(\boldsymbol{q}) = (I_N \otimes R(\boldsymbol{q}))$: rotation matrix
$\boldsymbol{o}(\boldsymbol{q}, \boldsymbol{x}) = \boldsymbol{R}(\boldsymbol{q})\boldsymbol{x}$: orbit function

$\boldsymbol{r}_x = (0, -z_1, y_1, 0, -z_2, y_2, \ldots, 0, -z_n, y_n)^T$

$\boldsymbol{r}_y = (z_1, 0, -x_1, z_2, 0, -x_2, \ldots, z_n, 0, -x_n)^T$

$\boldsymbol{r}_z = (-y_1, x_1, 0, -y_2, x_2, 0, \ldots, -y_n, x_n, 0)^T$

* $\boldsymbol{r}_x$, $\boldsymbol{r}_y$, and $\boldsymbol{r}_z$ are the tangents of the orbit at $\boldsymbol{x}$.

$$f(\boldsymbol{o}(\boldsymbol{q}, \boldsymbol{x})) = f(\boldsymbol{x})$$

- is *curved*: $\boldsymbol{r} \notin \ker(H(\boldsymbol{x}))$
- *does* affect Krylov iteration

Plain Newton-Krylov & the Rotations!

An Implementation in PETSc:

Rigid body modes:

$$Z_{m\times 6} = [\boldsymbol{t}_x|\boldsymbol{t}_y|\boldsymbol{t}_z|\boldsymbol{r}_x|\boldsymbol{r}_y|\boldsymbol{r}_z]$$

$$Y_{m\times 6} = A_{m\times m} Z_{m\times 6}$$

$$D_{6\times 6} = Y^T Y$$

$$D = U\sigma^2 V^T$$

$\rightarrow \hat{V}$: corresponding right singular vectors of the zero singular values

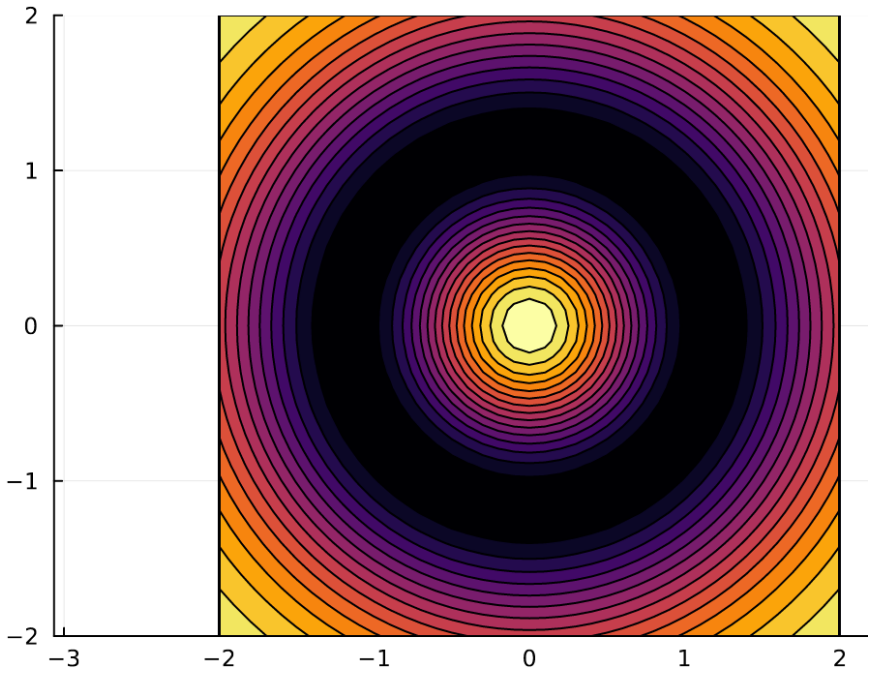
The desired (?) null space:

$$\hat{Z} = Z\hat{V}$$

```
newton_ad(x₀, strain_energy); # reproduced in Julia
```

```
Iteration 1: energy 0.5402974677121607
             gradient norm 1.4105813620865468
             hessian rank 21
             effective condition number 61.58224740081209
             step length 1.0
Iteration 2: energy 0.005247987808287236
             gradient norm 0.11625451629789378
             hessian rank 21
             effective condition number 777.3217296711898
             step length 1.0
Iteration 3: energy 5.415541871298145e-7
             gradient norm 0.0015862456717649644
             hessian rank 21
             effective condition number 20594.65732797073
             step length 1.0
Iteration 4: energy 6.893112972106431e-15
             gradient norm 1.284767074447696e-7
             hessian rank 21
             effective condition number 1.8820447105892365e9
             step length 1.0
Iteration 5: energy 2.220446049250321e-16
             gradient norm 2.748665926407727e-15
             hessian rank 18
             effective condition number 15.000000000000001
```

Re-parameterize Rotations

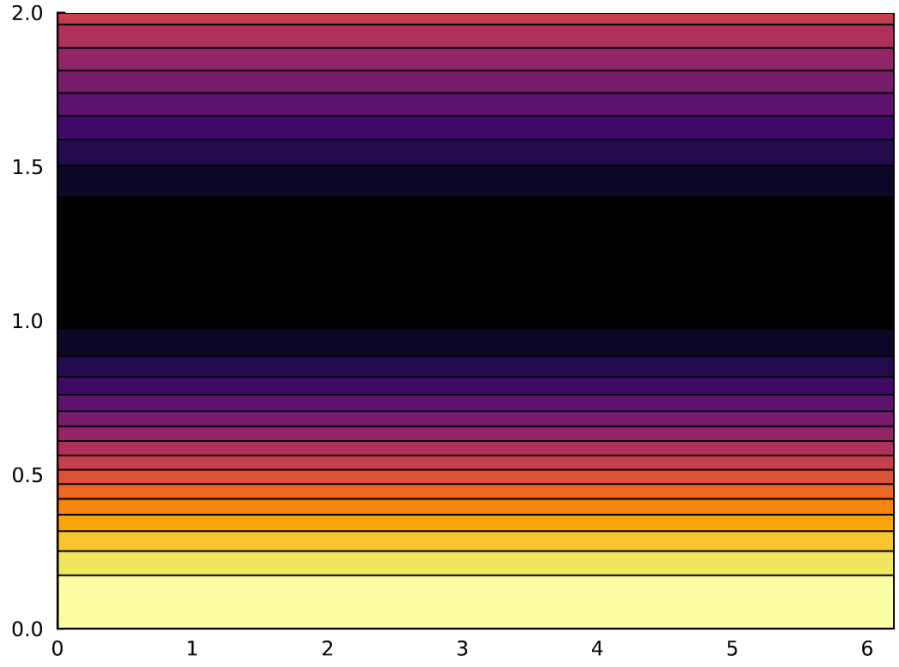


[36]:

```
H = ForwardDiff.hessian(f, randn(2))
rank(H)
```

[36]:

```
2
```



[38]:

```
Ĥ = ForwardDiff.hessian(f̂, randn(2))
rank(Ĥ)
```

[38]:

```
1
```

Project out the Rotational Modes

U: Orthonormal basis spanning $\partial_{\boldsymbol{q}} o\,(\boldsymbol{q}, \boldsymbol{x}) : \boldsymbol{q} \in SO(3)$

Modified Hessian: $\tilde{H} = (I - UU^T)H(I - UU^T)$

Rotation-corrected Newton Krylov

It's easy to implement, but does it affect convergence?

If there is a neighborhood $\mathcal{N}$ where:

- H is Lipschitz
- $\tilde{H}$ is Lipschitz up to unitary transformation: $\min \tilde{H}(\boldsymbol{x})Q - Q\tilde{H}(\boldsymbol{y}) : Q^T Q = I \lesssim |\boldsymbol{x} - \boldsymbol{y}|$
- The curvature of orbits is bounded: $|D^2 o\,(\boldsymbol{q}, \boldsymbol{x})| \leq C$

Then the iteration $x_{k+1} = x_k - \tilde{H}(\boldsymbol{x}_k)^+ \boldsymbol{g}_k$ converges R-quadratically near a solution

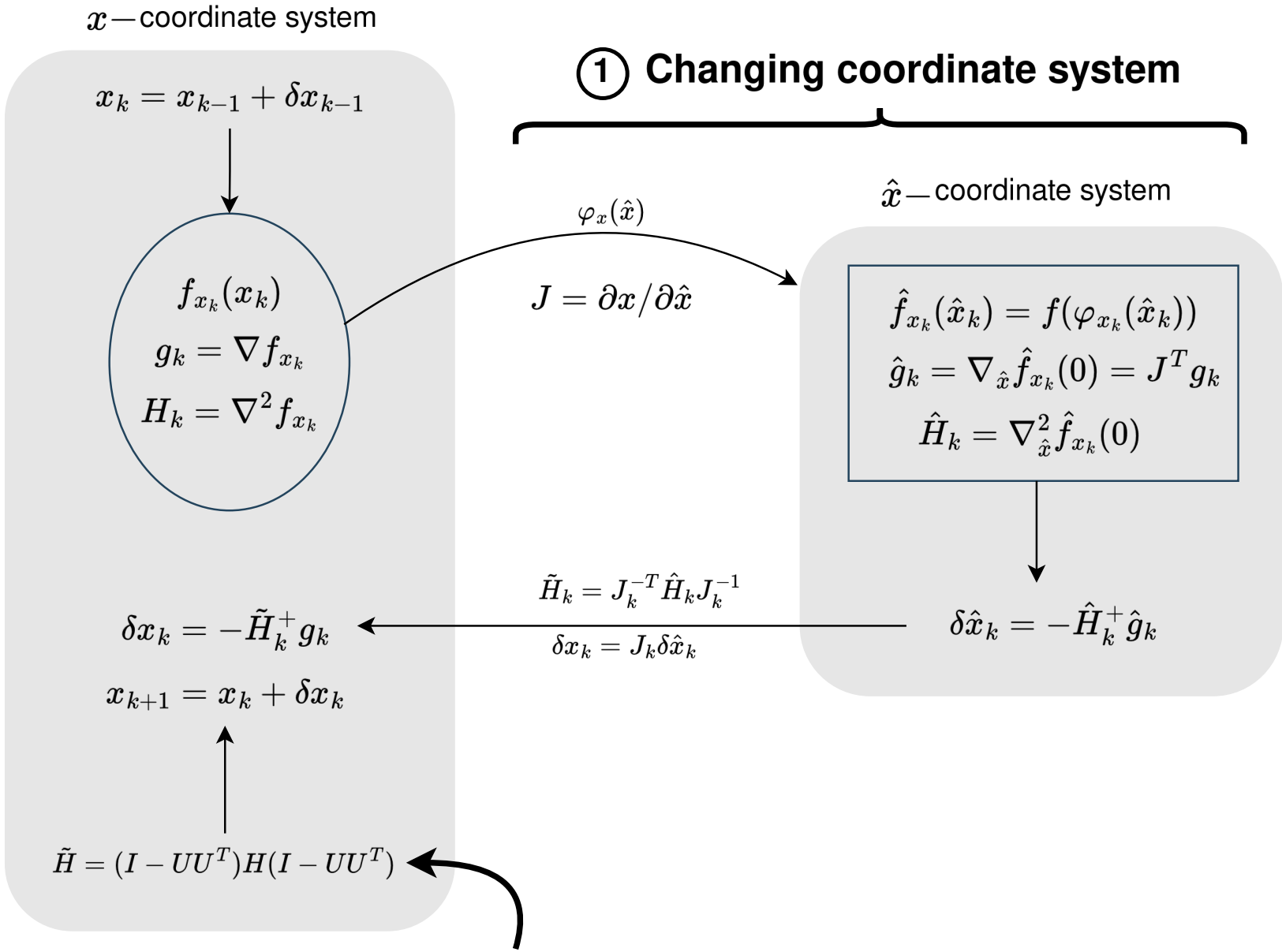
Plain vs Corrected Newton-Krylov

```
newton_ad(x₀, strain_energy);
```

```
Iteration 1: energy 0.5402974677121604
             gradient norm 1.4105813620865468
             hessian rank 21
             effective condition number 61.582247400812136
             step length 1.0
Iteration 2: energy 0.005247987808286681
             gradient norm 0.11625451629789381
             hessian rank 21
             effective condition number 777.3217296712144
             step length 1.0
Iteration 3: energy 5.415541876854953e-7
             gradient norm 0.0015862456717675895
             hessian rank 21
             effective condition number 20594.65732787524
             step length 1.0
Iteration 4: energy 6.44902375026225e-15
             gradient norm 1.2847656622972564e-7
             hessian rank 21
             effective condition number 1.8820448808005939e9
             step length 1.0
Iteration 5: energy -1.221245327087671e-15
             gradient norm 3.2984620808806433e-15
             hessian rank 18
             effective condition number 15.00000000000001
```

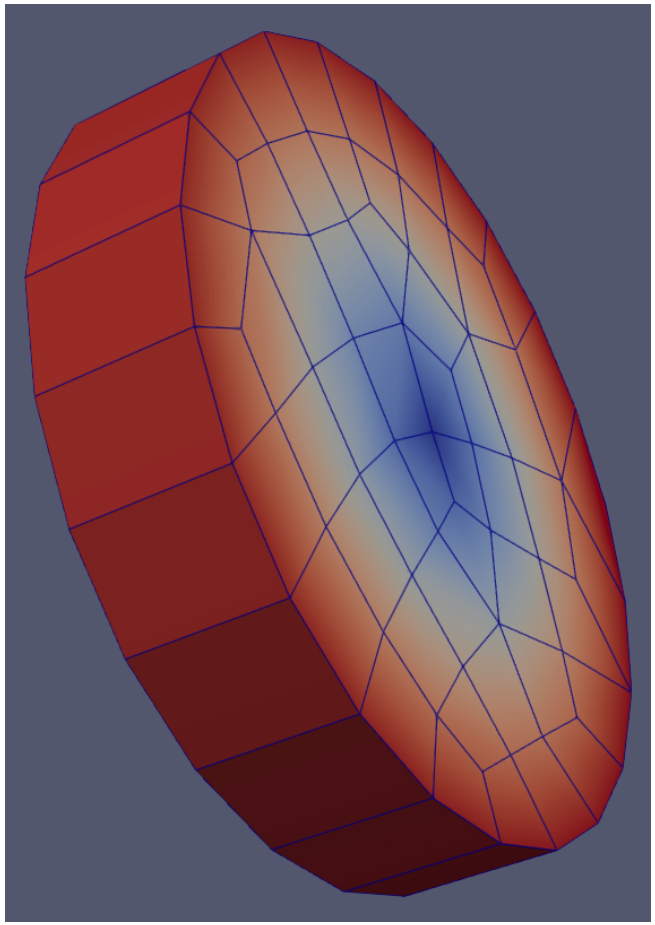
```
newton(x₀, strain_energy, x -> ForwardDiff.gradient(strain_energy, x),
       x -> rigid_body_corrected_hessian(strain_energy, x));
```

```
Iteration 1: energy 0.5402974677121604
             gradient norm 1.4105813620865468
             hessian rank 18
             effective condition number 15.55394648873718
             step length 1.0
Iteration 2: energy 0.004959838476031193
             gradient norm 0.11654294393222203
             hessian rank 18
             effective condition number 15.20556531877761
             step length 1.0
Iteration 3: energy 6.684727159775428e-7
             gradient norm 0.001280378513351747
             hessian rank 18
             effective condition number 15.002477767222615
             step length 1.0
Iteration 4: energy 9.788756571327347e-15
             gradient norm 2.0455627721105363e-7
             hessian rank 18
             effective condition number 14.999999934516481
             step length 1.0
Iteration 5: energy 1.1102230246251573e-15
             gradient norm 3.3453785797687322e-15
             hessian rank 18
             effective condition number 15.000000000000023
```


x — coordinate system
$x_k = x_{k-1} + \delta x_{k-1}$
$f_{x_k}(x_k)$
$g_k = \nabla f_{x_k}$
$H_k = \nabla^2 f_{x_k}$
$\varphi_x(\hat{x})$
$J = \partial x / \partial \hat{x}$
1 Changing coordinate system
$\hat{x}$ — coordinate system
$\hat{f}_{x_k}(\hat{x}_k) = f(\varphi_{x_k}(\hat{x}_k))$
$\hat{g}_k = \nabla_{\hat{x}} \hat{f}_{x_k}(0) = J^T g_k$
$\hat{H}_k = \nabla^2_{\hat{x}} \hat{f}_{x_k}(0)$
$\delta \hat{x}_k = -\hat{H}_k^+ \hat{g}_k$
$\tilde{H}_k = J_k^{-T} \hat{H}_k J_k^{-1}$
$\delta x_k = J_k \delta \hat{x}_k$
$\delta x_k = -\tilde{H}_k^+ g_k$
$x_{k+1} = x_k + \delta x_k$
$\tilde{H} = (I - UU^T) H (I - UU^T)$
2 Projecting out rotational modes

What if the body is partially constrained?

- Rotation-corrected Newton should only project out tangents of invariant rotations
- Can the invariant rotations be determined automatically?



What if the set of invariant rotations is unknown?

A rotation $\boldsymbol{q}$ is invariant if

$$f(\boldsymbol{o}(\epsilon\boldsymbol{q}, \boldsymbol{x})) = f(\boldsymbol{x}) \quad \text{for all} \epsilon$$

First order condition:

$$\partial_{\boldsymbol{q}} f(\boldsymbol{o}(\epsilon\boldsymbol{q}, \boldsymbol{x})) = 0 \Rightarrow g(\boldsymbol{x})^T V_{\boldsymbol{o}}(\boldsymbol{x})\boldsymbol{q} = 0 \text{where, } \partial_{\boldsymbol{q}}\boldsymbol{o}(0, \boldsymbol{x}) = V_{\boldsymbol{o}}(\boldsymbol{x})\boldsymbol{q}$$

Second order condition:

$$\partial^2_{\boldsymbol{q}} f(\boldsymbol{o}(\epsilon\boldsymbol{q}, \boldsymbol{x})) = 0 \Rightarrow u^T[\nabla_{\boldsymbol{x}} V_{\boldsymbol{o}}(\boldsymbol{x}) : \boldsymbol{g}(\boldsymbol{x})]\boldsymbol{q} + \boldsymbol{u}^T H(\boldsymbol{x}) V_{\boldsymbol{o}}(\boldsymbol{x})\boldsymbol{q} = 0$$

Therefore $\boldsymbol{q}$ is an invariant rotation if:

$$\begin{bmatrix} HV_{\boldsymbol{o}} + [\nabla_{\boldsymbol{x}} V_{\boldsymbol{o}}(\boldsymbol{x}) : \boldsymbol{g}(\boldsymbol{x})] \\ \boldsymbol{g}^T V_{\boldsymbol{o}} \end{bmatrix} \boldsymbol{q} = 0$$

Determining Invariant Rotations

If the orbit function $o(\boldsymbol{q}, \boldsymbol{x})$ of all rotations is known, the subset of invariant rotations can be determined at the cost (per Newton iteration) of

- 3 Hessian-vector products
- One tall-skinny $\mathbb{R}^{n \times 3}$ QR factorization

Outlook

- Projecting out the whole rigid body mode from the Hessian
- Implementation in PETSc and our solid mechanics software (**Ratel**: https://gitlab.com/micromorph/ratel)