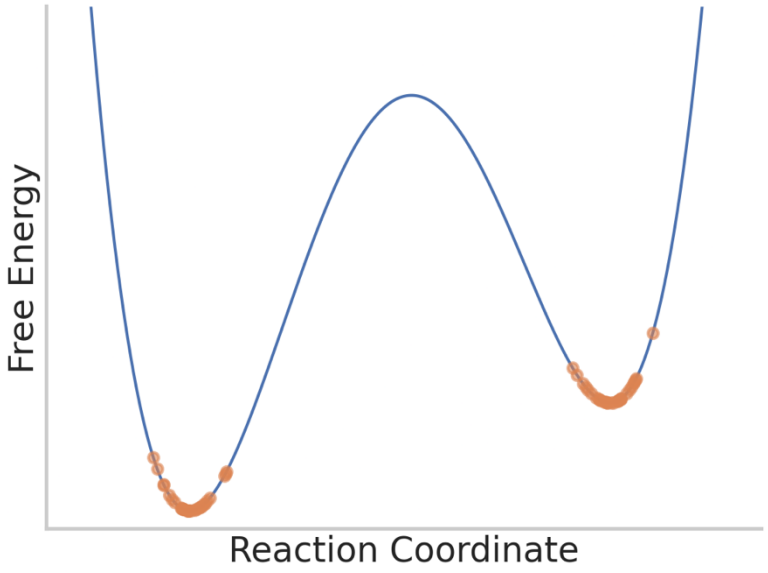
Finding Transition States with HIP

Gentles Ascent Dynamics (GAD)
and
Eigenvalue Descent
By Memo

With transition states we can approximate reaction rates

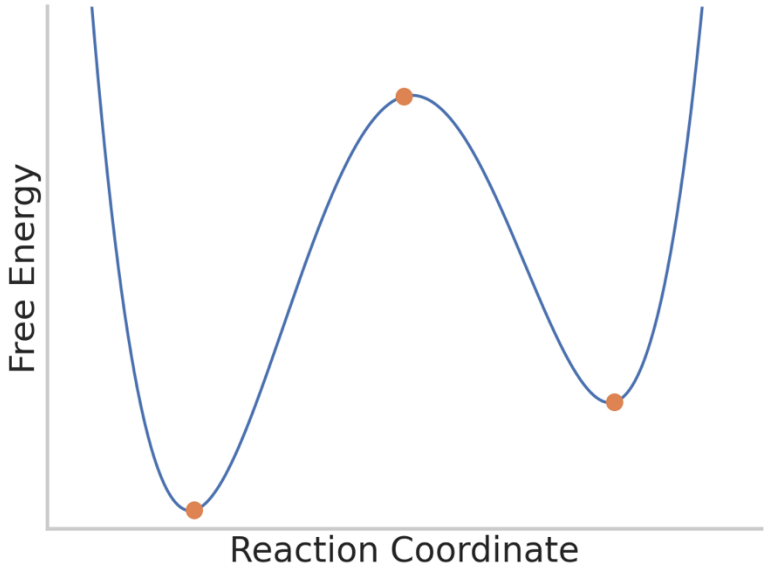
Free Energy Difference

no reaction rates



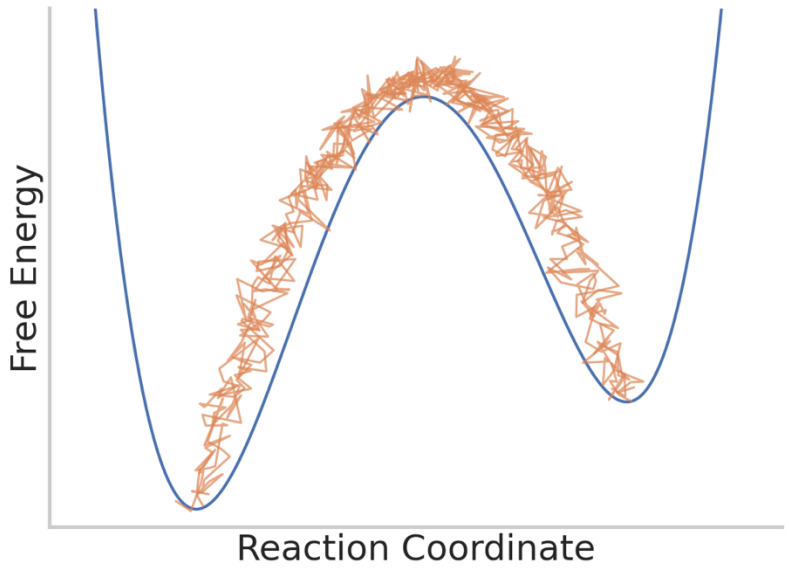
Transition State Theory

approximate reaction rates

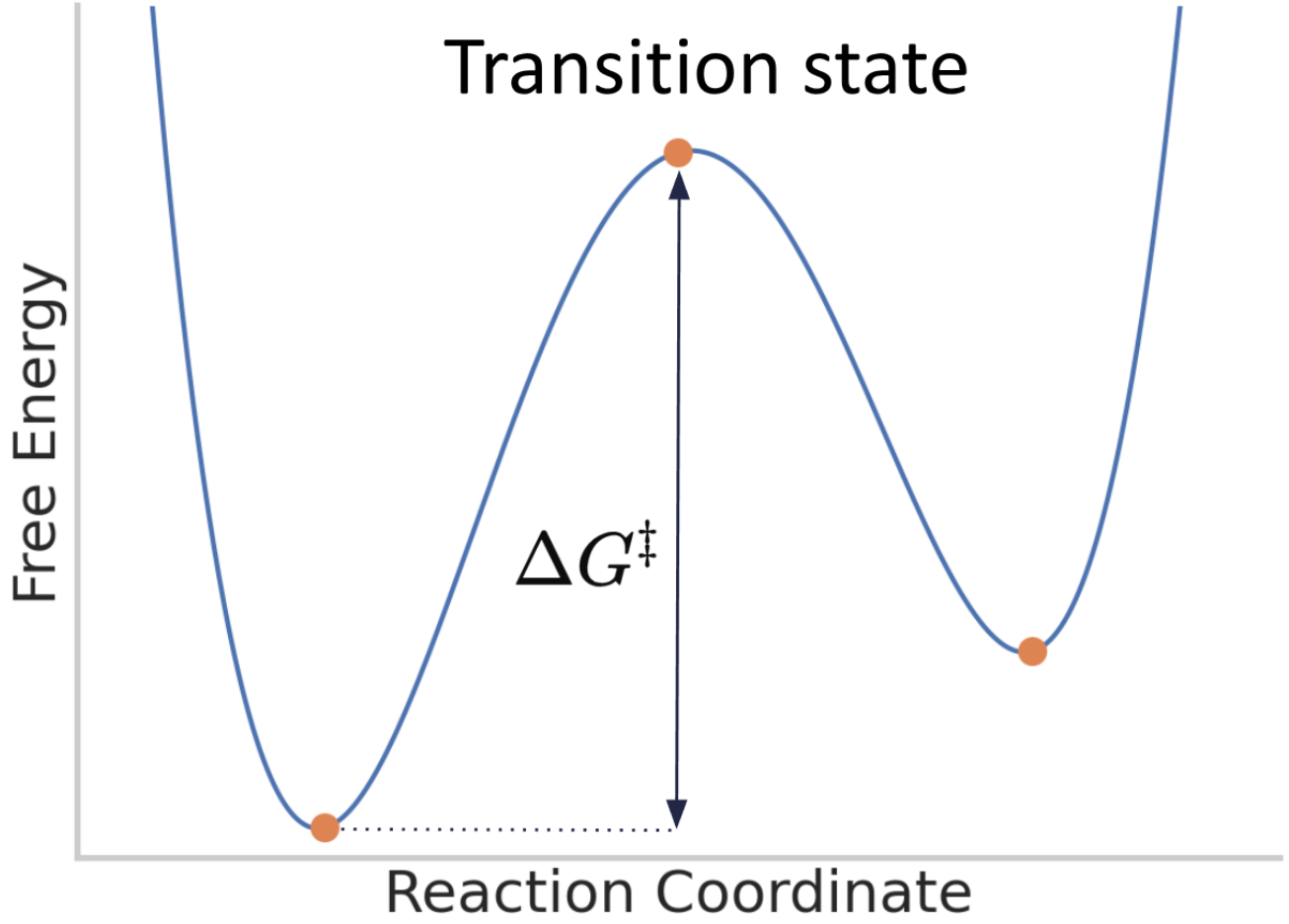


Transition Path Sampling

true reaction rates



With transition states we can approximate reaction rates



Eyring equation:

$$k(T) = \frac{k_B T}{h} \exp\left(-\frac{\Delta G^{\ddagger}}{RT}\right)$$

We can identify transition states using the smallest two eigenvalues

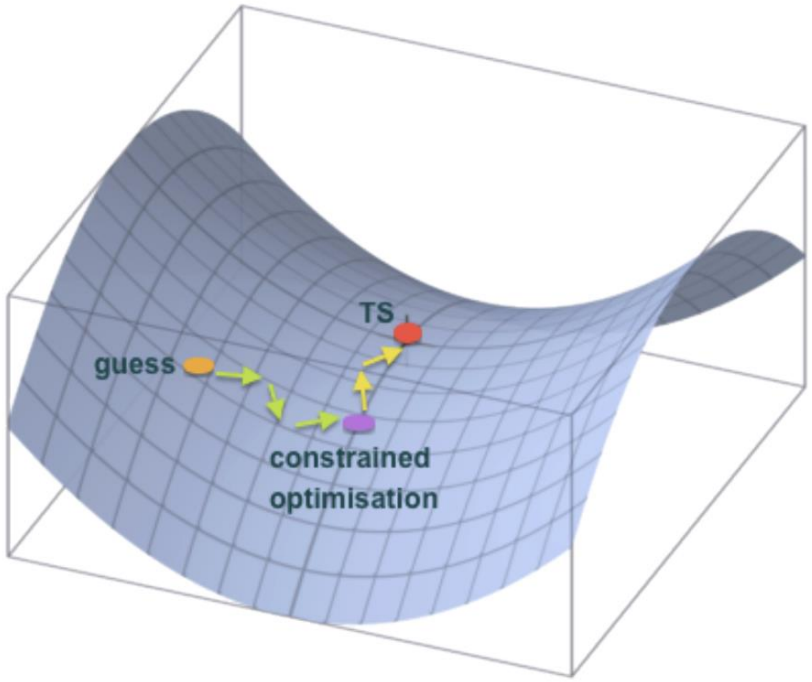
Transition State = Saddle point in PES

Characterized by: $\lambda_0 < 0$, $\lambda_1 > 0$, $\lambda_2 > 0$, ...

Product $\lambda_0 \times \lambda_1 < 0$ indicates TS

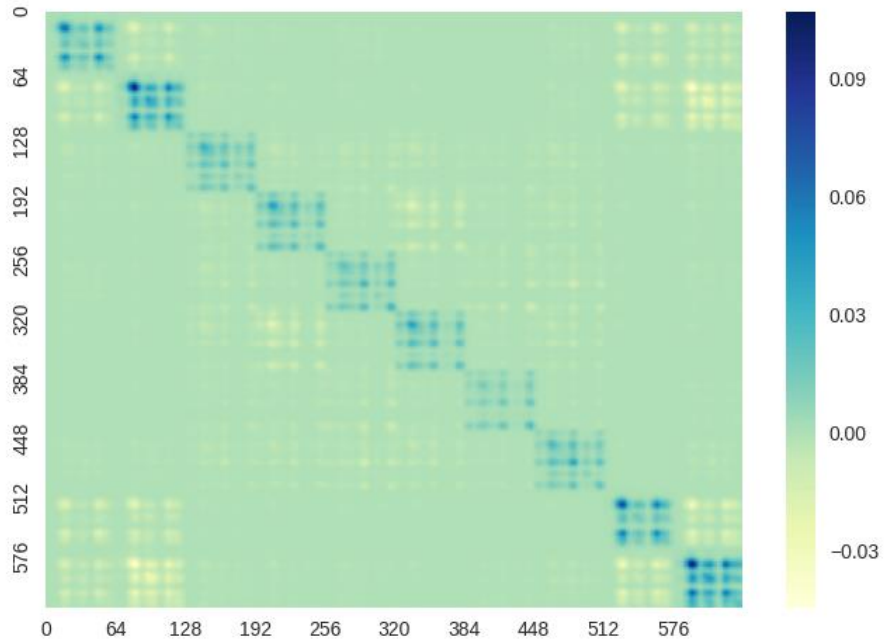
Eigvals, eigvecs = torch.eigh(H_f)

Eigvals = [λ_0 , λ_1, ...] # λ_0 , λ_1 are the smallest two eigenvalues



HIP predicts Hessians directly from atomic positions

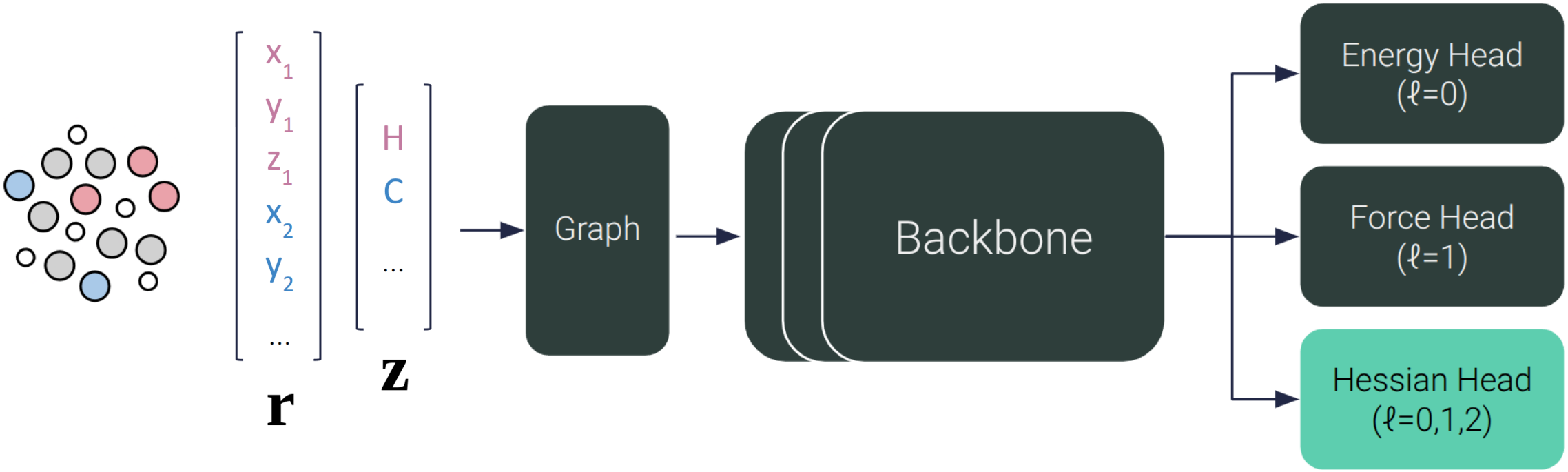
- HIP: Machine Learning Interatomic Potential
- Input: Atomic positions (x, y, z)
- Output: Full Hessians ($\nabla^2 V$)
- Key: No autodiff needed
- Enables direct saddle point analysis



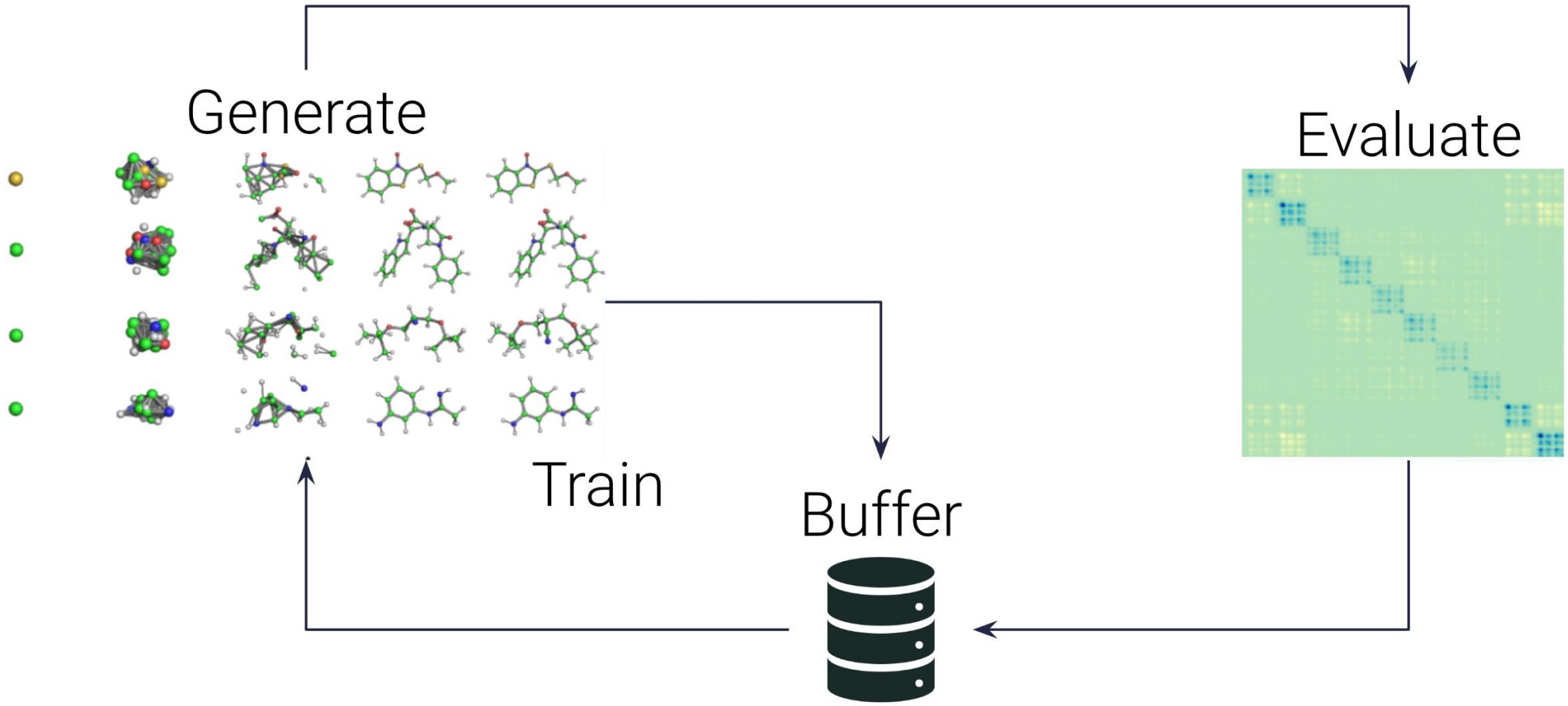
$$H_f(x,y) = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}$$

Eigvals, eigvecs = torch.eigh(H_f)
Eigvals = [λ_0 , λ_1, ...] # λ_0 , λ_1 are the smallest two eigenvalues

HIP predicts the Hessian from atom positions



We will train a generative model to generate transition states



We search for transition states using Gentlest Ascent Dynamics (GAD)

GAD equation:

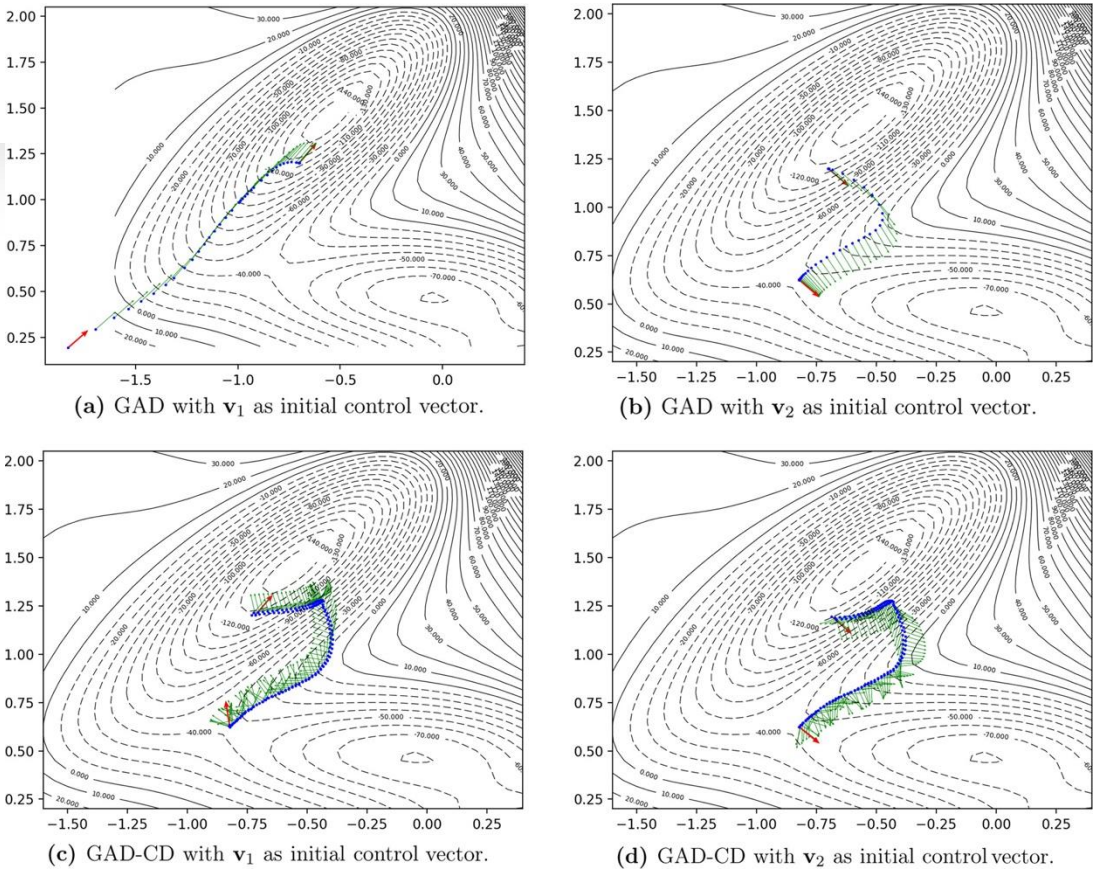
$$\mathbf{x}' = -\nabla V(x) + 2(\nabla V, \mathbf{v_0})\mathbf{v_0} \quad [1]$$

V(x) = Energy scalar

$\mathbf{v_0}$ = eigenvector of smallest eigenvalue

|| GAD step || = || **Force** ||

*First order Integration (Euler)



What GAD trajectories roughly look like

GAD-Euler

[1] W. E and X. Zhou, "The gentlest ascent dynamics," Nonlinearity, vol. 24, no. 6, pp. 1831–1842, May 2011, doi: https://doi.org/10.1088/0951-7715/24/6/008.

91.3% of trajectories starting from reactants found transition states

Experiment A:
GAD eq'n with Euler Integation (GAD-Euler)

Ending Geometries	Percentage
Minimum	2.8%
Transition State	**91.3%**
Unstable	3.8%
Higher Order Unstable	2.1%

dt = 0.005Å, n_samples = 300, max_steps = 500

Average steps to convergence & average time per sample (RTX 2070):
<<Euler: **25.1 steps** to TS**, 50s per sample**>>
<<RK45: **7.3 steps** to TS**, 64s per sample** >>

GAD-Euler

Typical graph that ended at Transition State

(91.3% of samples)

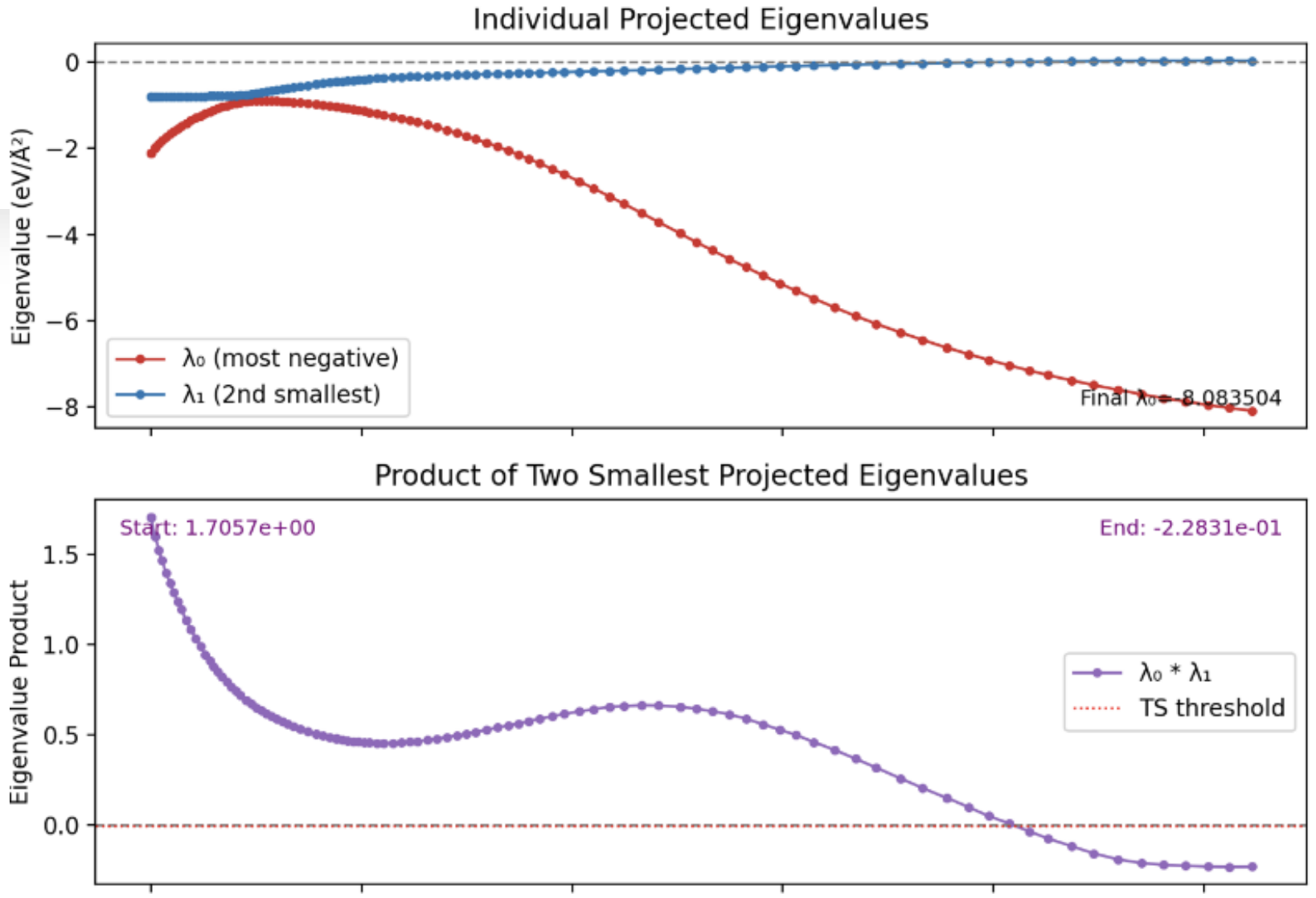


Figure 2: Trajectory of $C_2H_3N_3O_2$ starting, slowly converging to a TS

GAD-Euler

The samples that didn't reach TSs had two failure modes:

Stuck 1: 5.9% end up unstable

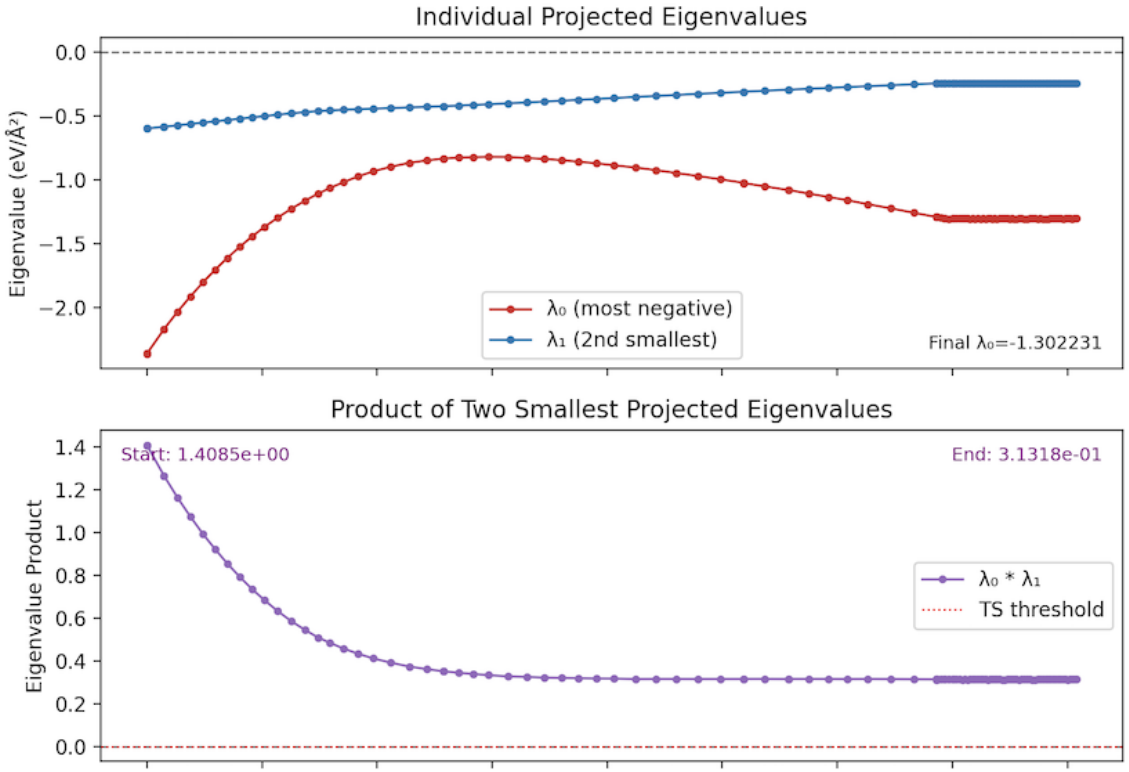


Figure 3: Trajectory of $C_2H_3N_3O_2$, unable to escape an unstable geometry

Stuck 2: 2.8% end up unstable

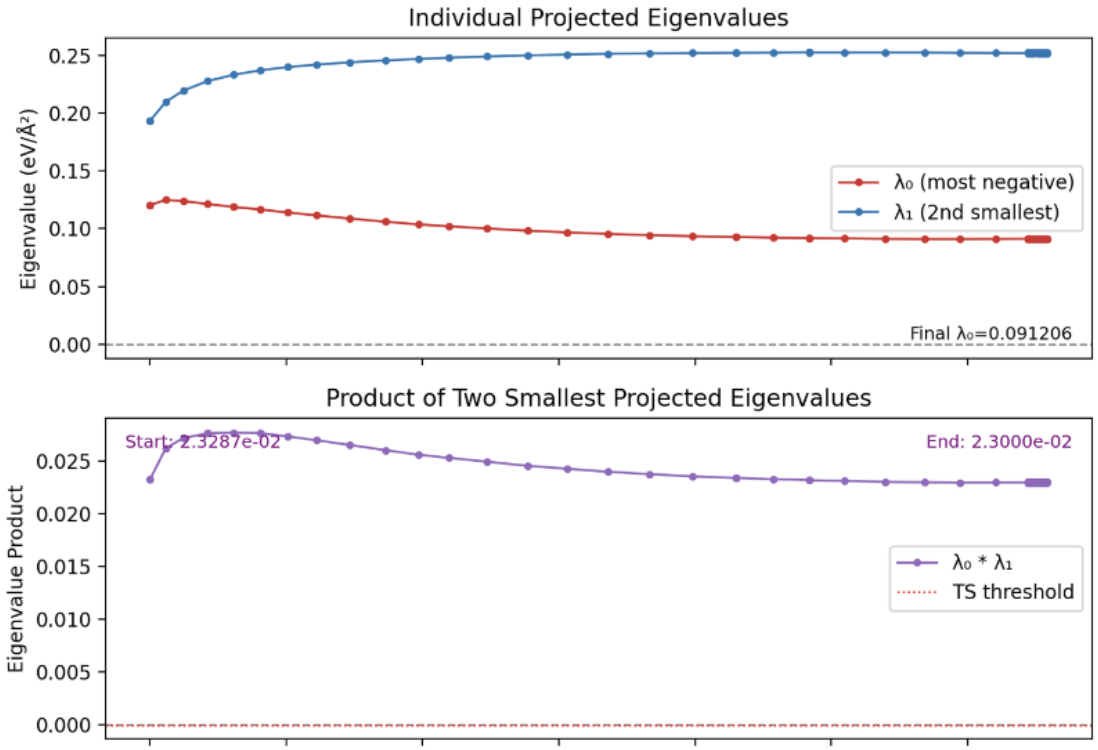


Figure 3: Trajectory of $C_2H_3N_3O_2$, unable to escape an energy minimum

GAD-Euler

We've confirmed GAD works. Can we try to directly optimize λ_0 and λ_1?

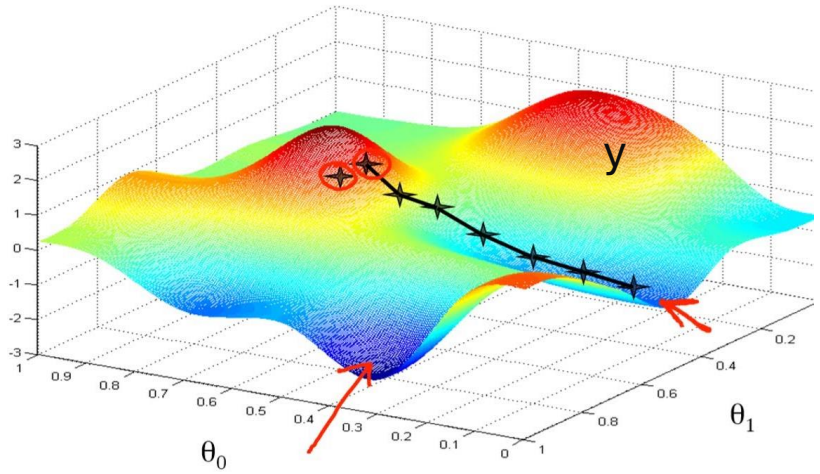
Core idea:

Directly find gradients of λ_0, λ_1 , and just optimize towards a Transition State ($\lambda_0 * \lambda_1$).

Benefits:

- Much simpler algorithm
- Could lead to more direct paths than GAD
- Could be faster than GAD

V(x,y)



The most direct way of approaching our goal (TS) is to run a gradient descent on the TS condition directly

TS Condition:

negative $\lambda_0 * \lambda_1$

Gradient descent on:

$\lambda_0 * \lambda_1$ until it converges to negative

Gradient path: loss → eigenvalues → projected Hessian → raw Hessian → HIP model → coordinates

The most direct way of approaching our goal (TS) is to run a gradient descent on the TS condition directly


```
for step in range(n_steps):
    # Compute Hessian and eigenvalues
    hessian = model.get_hessian(coords)
    hessian_proj = project_out_rotations_translations(hessian, coords)
    eigenvalues = torch.linalg.eigvalsh(hessian_proj)

    # Loss: product of two smallest eigenvalues
    λ0, λ1 = eigenvalues[0], eigenvalues[1]
    loss = λ0 * λ1  # Negative at TS (λ0 < 0, λ1 > 0)

    # Gradient descent
    grad = torch.autograd.grad(loss, coords)[0]
    coords = coords - lr * grad

    # Early stop if product is sufficiently negative
    if loss < -1e-4:
        break
```

92.7% of samples converged (starting from reactant position)

Eigenvalue Product Descent

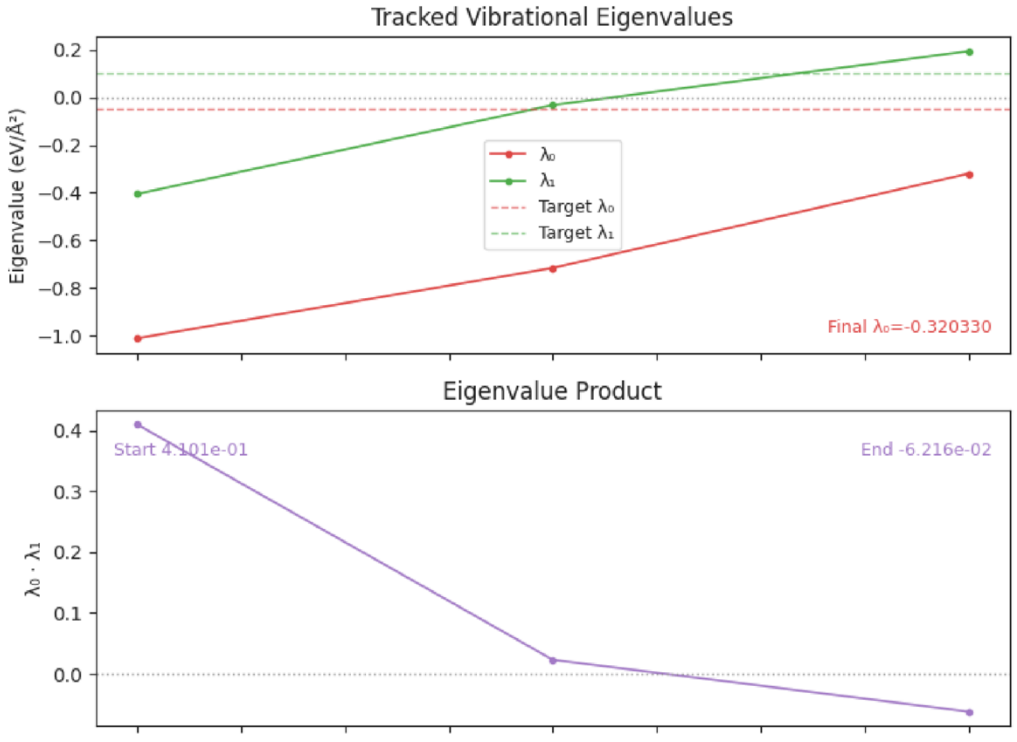
Experiment B:
Eigenvalue product descent with Euler stepping

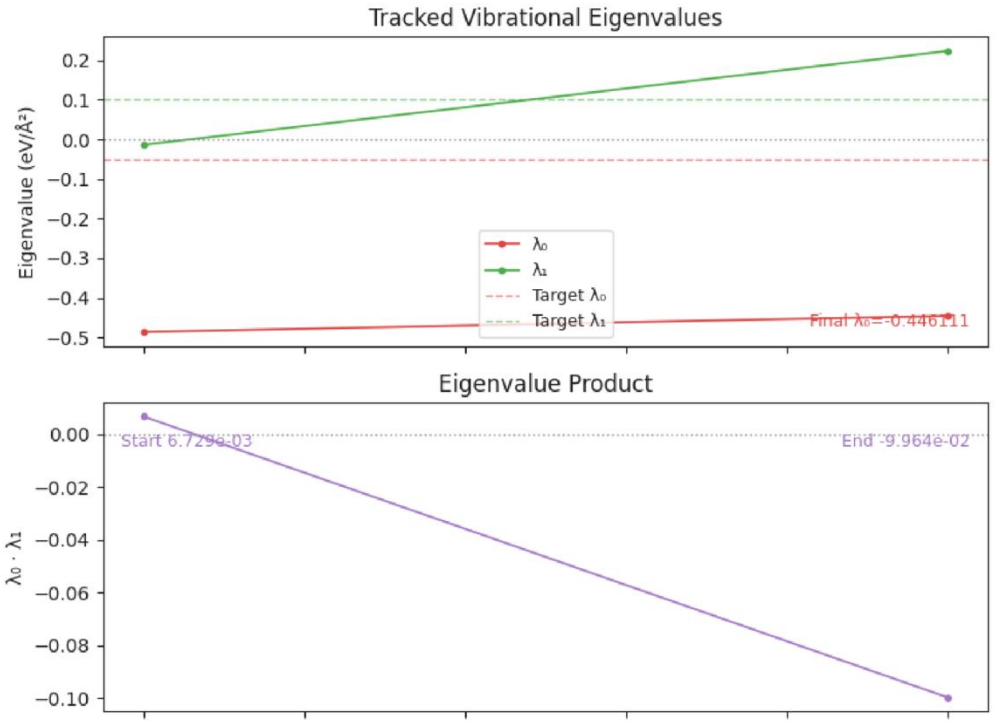
Loss Function	TS Success Rate (%)	Avg λ_0 (eV/Å²)	Avg λ_1 (eV/Å²)	Avg $\lambda_0 \times \lambda_1$
Product Descent	**92.7**	-0.2405	0.0763	-8.260e-03

dt = 0.01 Å, n_samples = 300, max_steps = 500

The samples converged almost immediately (within 1-3 steps)

Eigenvalue Product Descent

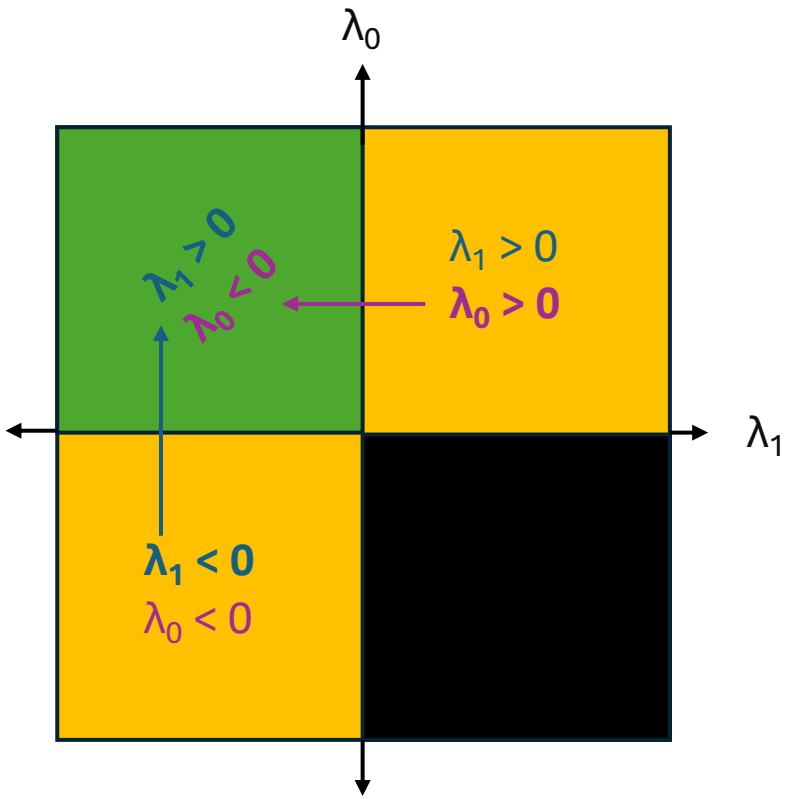




Average steps to convergence & average time per sample (RTX 2070):
<<Euler: **25.1 steps** to TS**, 50s per sample**>>
<<RK45: **7.3 steps** to TS**, 64s per sample** >>
<<EigProd: **2.3 steps** to TS**, 3.2s per sample** >>

What if there is an EVEN MORE direct way of approaching transition states

Only change λ_0 OR λ_1 one at a time, depending on the quadrant you are in



What if there is an EVEN MORE direct way of approaching transition states

Direct Descent

```python
for step in range(n_steps):
    # Compute Hessian and eigenvalues
    hessian = model.get_hessian(coords)
    hessian_proj = project_out_rotations_translations(hessian, coords)
    eigenvalues = torch.linalg.eigvalsh(hessian_proj)

    n_negative = (eigenvalues < 0).sum()
    λ0 = eigenvalues[0]

    # Adaptive loss based on current saddle order
    if n_negative == 0:
        # No negatives → push λ0 below target
        loss = (λ0 - neg_target) ** 2
    elif n_negative == 1:
        # Exactly one negative → done!
        loss = 0.0
        break
    else:
        # Too many negatives → push extras positive
        loss = ((pos_floor - eigenvalues[1:]) ** 2).sum()

    # Gradient descent
    grad = torch.autograd.grad(loss, coords)[0]
    coords = coords - lr * grad
```

Direct Descent Performed better, and was slightly faster

Direct Descent

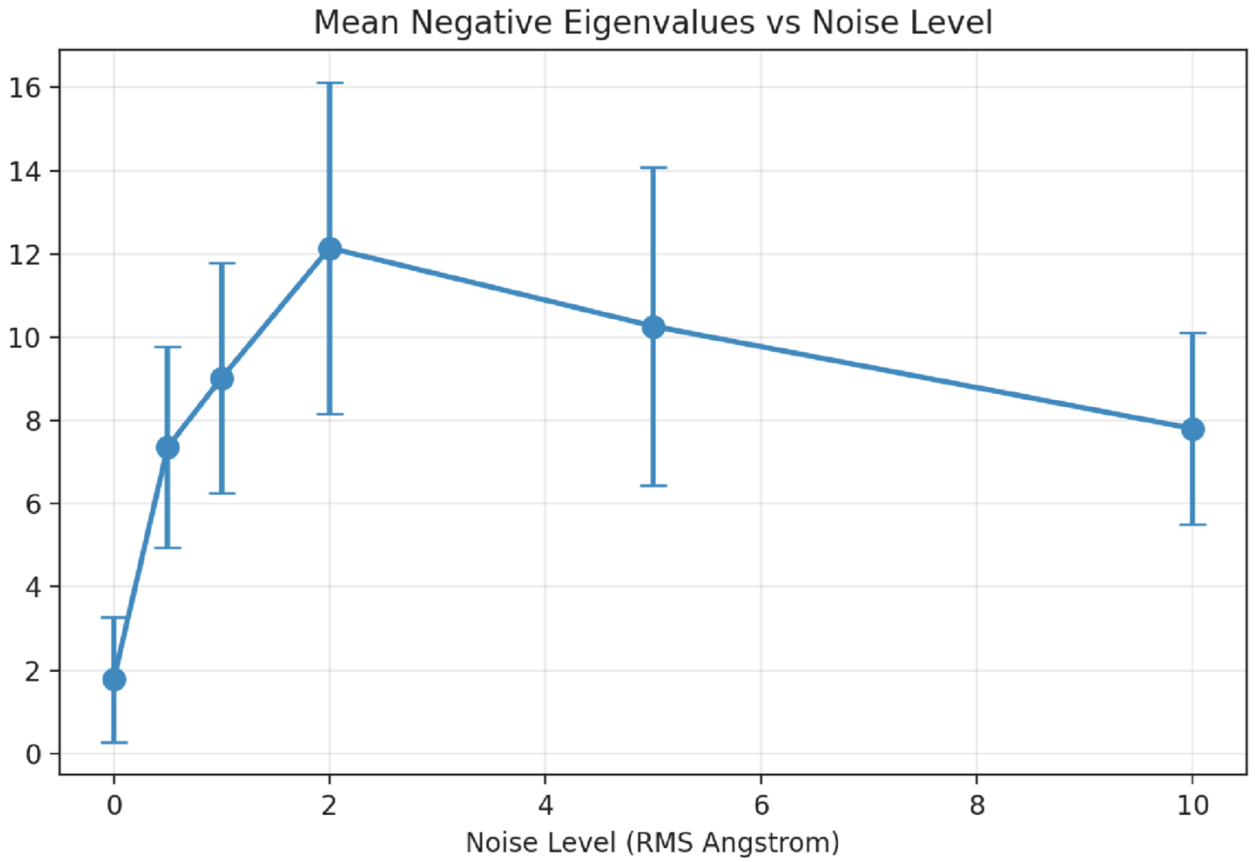
Loss Function Comparison (n=287, 500 steps)

TS Success = Converged to exactly 1 negative eigenvalue

Loss Function	TS Success Rate (%)	Avg λ_0 (eV/Å²)	Avg λ_1 (eV/Å²)	Avg $\lambda_0 \times \lambda_1$
Direct Descent	**99.7**	-0.1528	0.1241	-1.733e-02
Product Descent	**92.7**	-0.2405	0.0763	-8.260e-03

Average steps to convergence & average time per sample (RTX 2070):
<<Euler: **25.1 steps** to TS**, 50s per sample**>>
<<RK45: **7.3 steps** to TS**, 64s per sample** >>
<<EigProd: **2.3 steps** to TS**, 3.2s per sample** >>
<<Direct: **1.8 steps** to TS**, 3.1s per sample** >>

So far, we've started from reactants. But the generative model will start from random geometries.

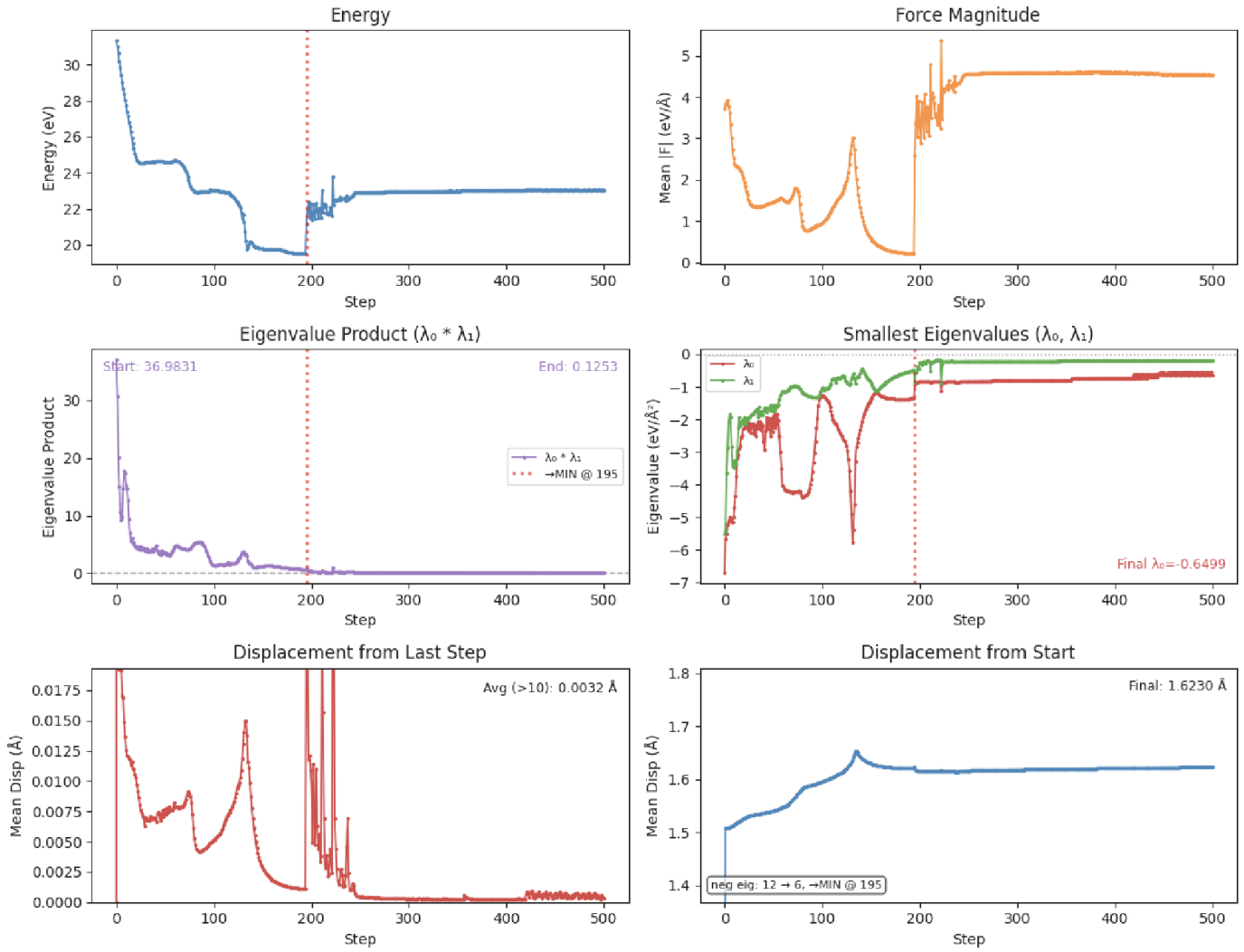


- We want random starts with higher # of imaginary frequencies, as that is exactly what the generative model will produce.

How many negative eigenvalues we get based on how much noise we add

Pure GAD almost always plateaued at order 4-8 negative eigenvalues (13% convergence rate to TS)

Noisy Experiments – GAD-Euler

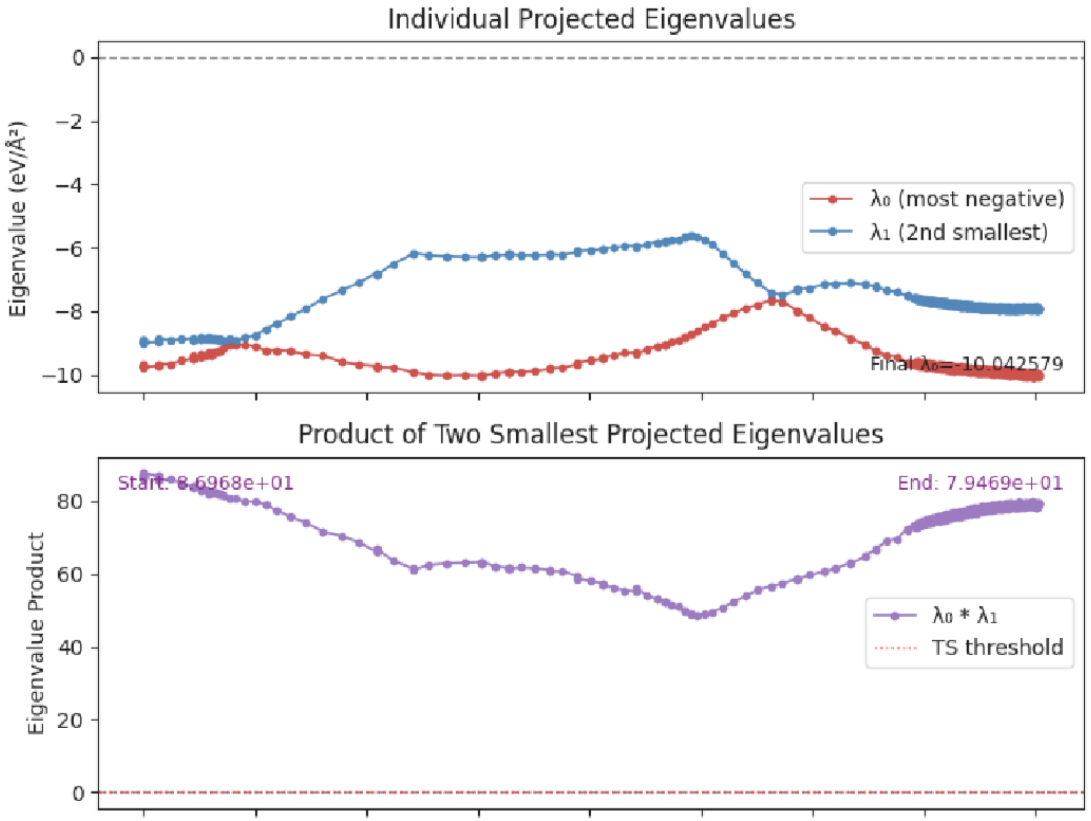


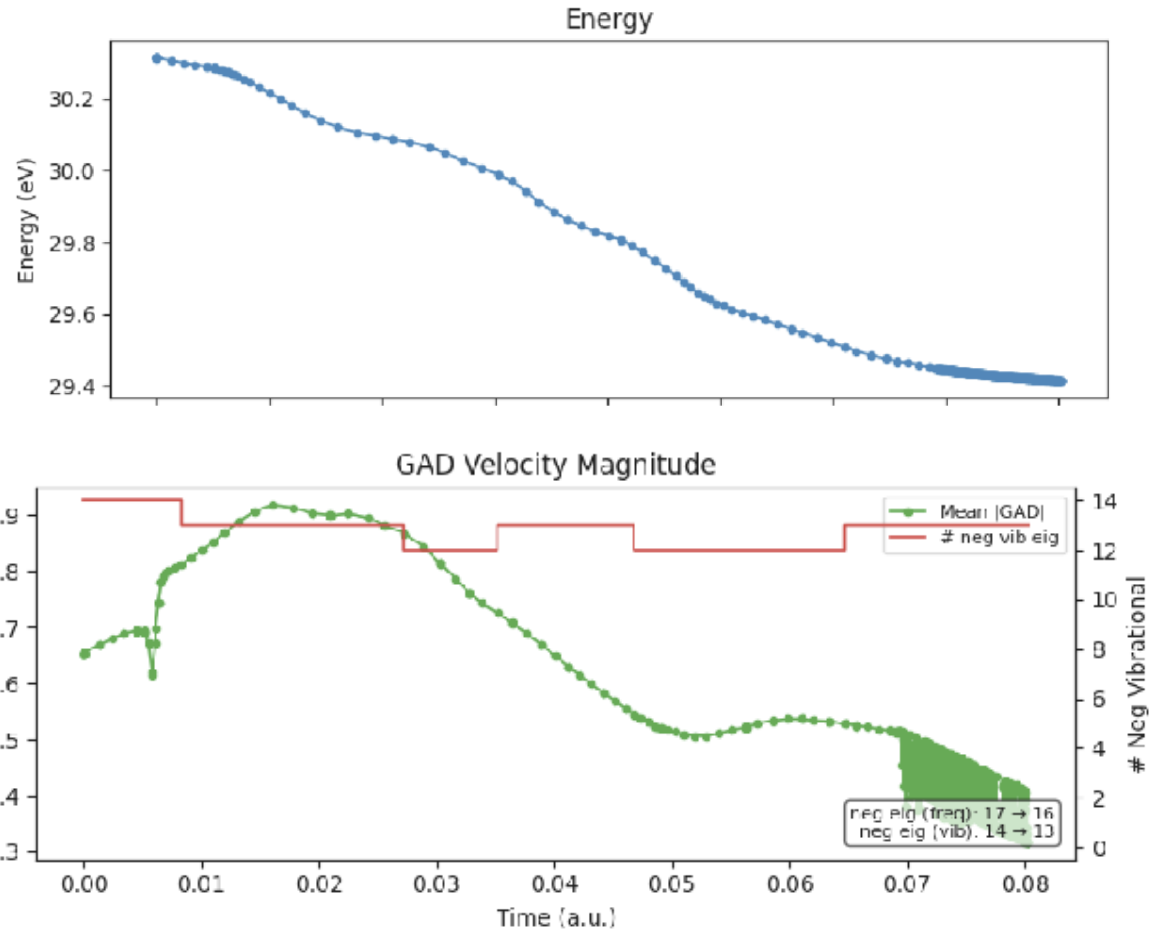
Typical GAD trajectory, which gets stuck

GAD-RK45 also plateaued, but sometimes oscillated (example 1) (<5% convergence rate to TS)

Noisy Experiments – GAD RK45

17 negative eigenvalues to 16

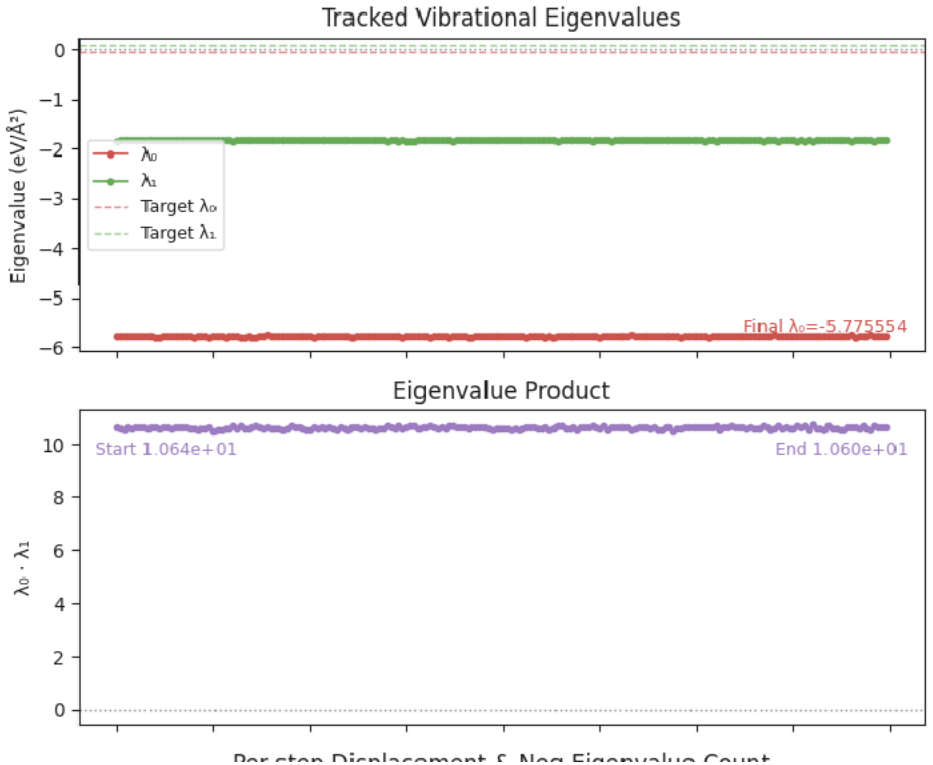




Direct Descent Methods had almost no effect on stability, and none converged

Noisy Experiments – Direct Descent

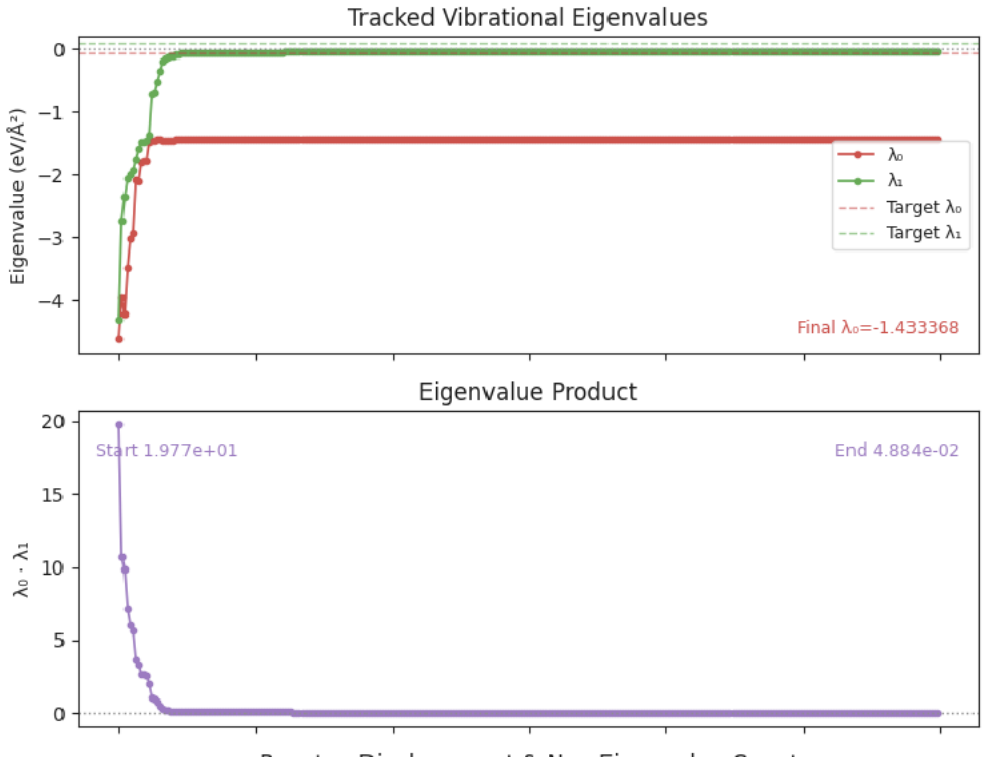
Nearly all graphs resemble the following:

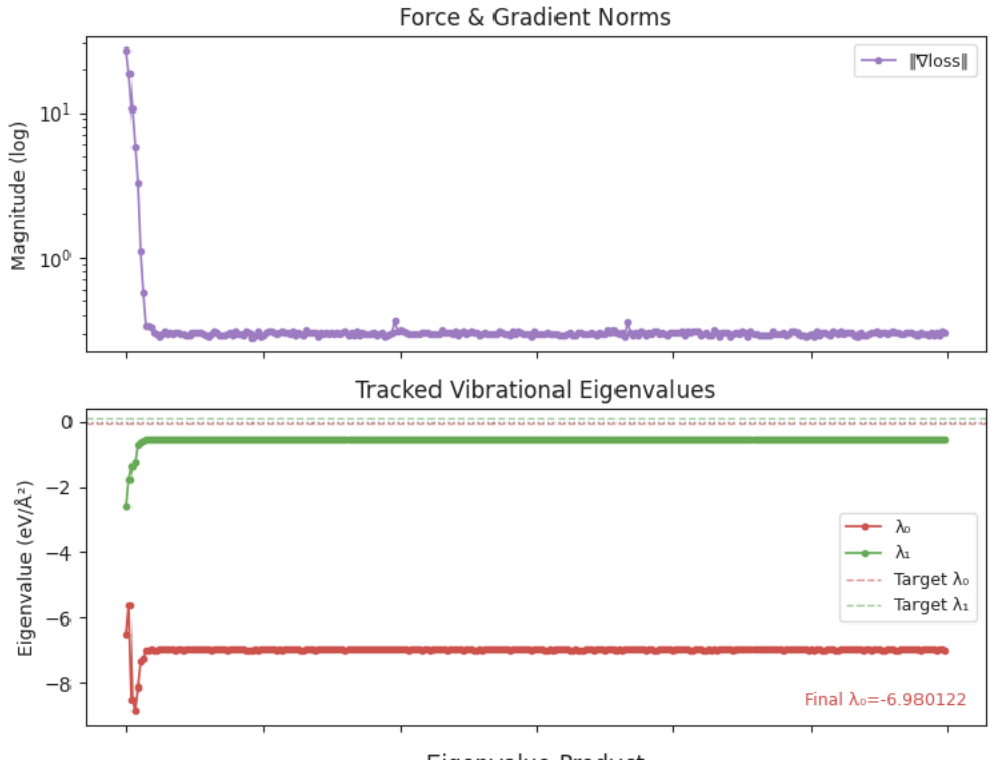


Product Descent very quickly reached plateaus, <5% converged

Noisy Experiments – Product Descent

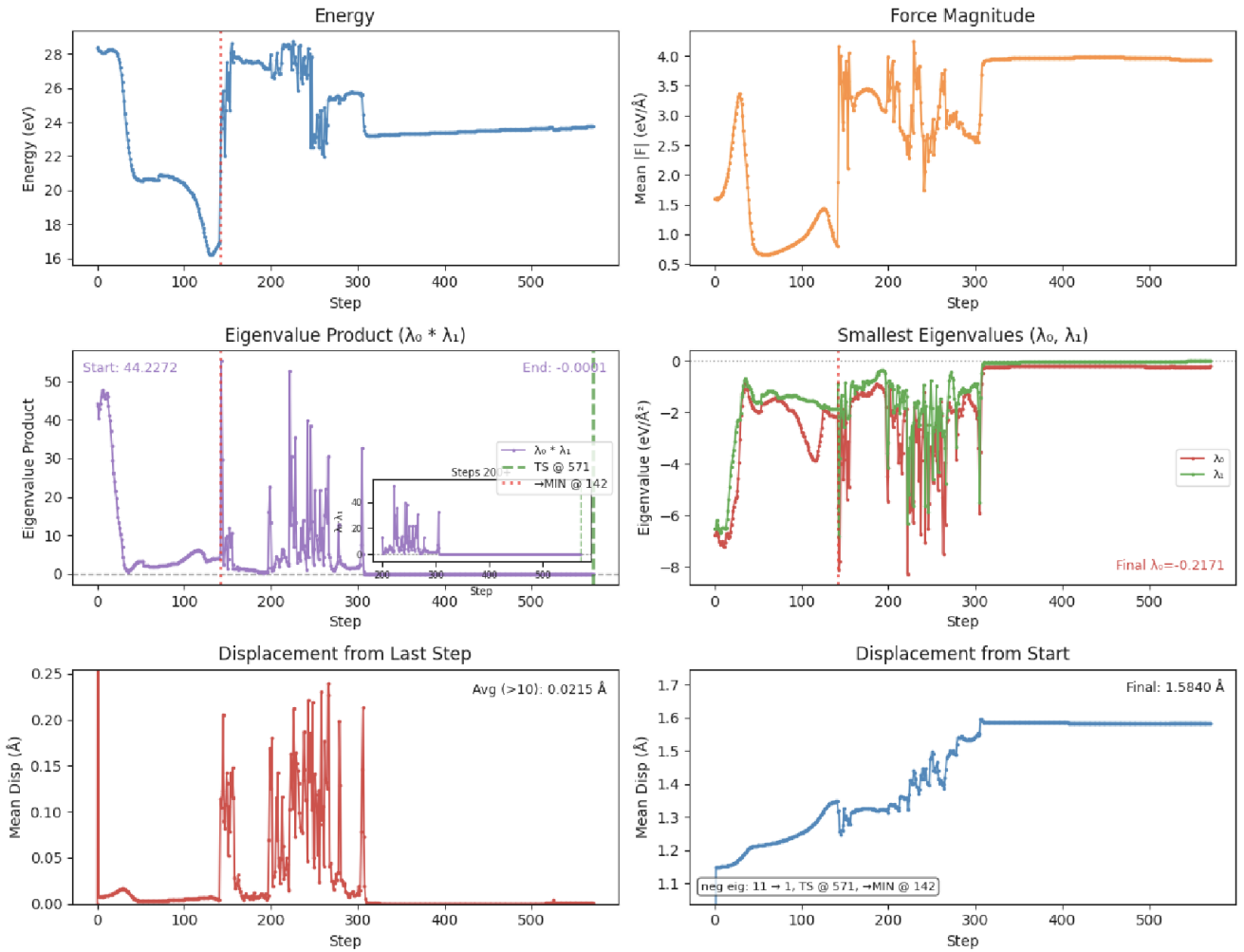
Nearly all graphs resemble the following:





While many hybrid methods didn't work, GAD + Product Descent improved performance! (~33% Convergence) (Converged Example)

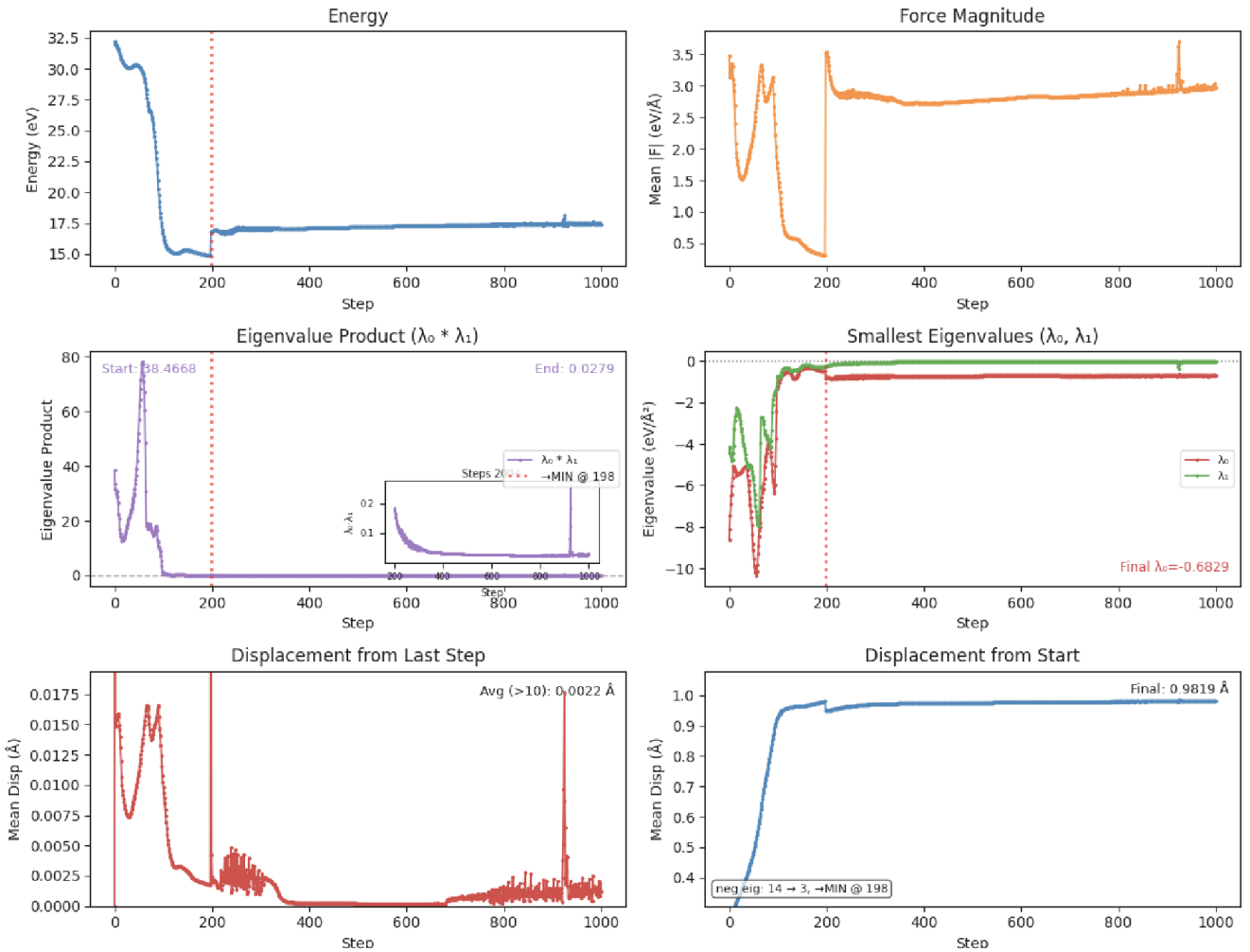
Noisy Experiments – Hybrid Method




```
#    --eig-descent-lr 0.01
#    --eig-descent-max-step 0.5
#    --stop-at-ts
#    --n-steps 1000
#    --dt 0.005
```

While many hybrid methods didn't work, GAD + Product Descent improved performance! (~33% Convergence) (Stalled example)

Noisy Experiments – Hybrid Method




```
#    --eig-descent-lr 0.01
#    --eig-descent-max-step 0.5
#    --stop-at-ts
#    --n-steps 1000
#    --dt 0.005
```

Quick note about GAD vs. Eigenvalue Descent

- GAD Cannot Escape Reactant Geometries (Direct Descent is still king).

```
[Transition Distribution]
  0neg-to-0neg: 5 samples
  1neg-to-1neg: 28 samples
  2neg-to-1neg: 49 samples
  3neg-to-1neg: 18 samples
```

New analytical forcefield: SCINE DFTB0

- Doesn’t preserve gradients (

```
[Transition Distribution]
  0neg-to-0neg: 5 samples
  1neg-to-1neg: 28 samples
  2neg-to-1neg: 49 samples
  3neg-to-1neg: 18 samples
```

HIP Enables Efficient Transition State Discovery

- Confirmed that GAD is a viable tool for TS search.

- Direct eigenvalue descent with access to gradients is a better approach
- (expected, as it's literally optimizing our criteria, and the paths are not extremely complicated.

- Can only achieve ~33% convergence on noisy geometries

December Experiments

GAD Algorithm Improvements for Transition State Search

From Noisy Geometries to Reliable Convergence

Overview

1. **New Forcefield Calculator** (SCINE DFB0)
2. **L-BFGS Energy Minimizer** — Reduce saddle order first
3. **Plateau Detection (Kicking)** — Adaptive step sizes
4. **Higher-Order GAD** — Multi-mode escape
5. **Mode Tracking + Trust Radius** — Final solution (92% / 71%)

SCINE Sparrow is a different level of theory than T1x and HIP

Key Insight

HIP and SCINE forcefields disagree significantly on the number of negative eigenvalues, especially for reactant geometries.

https://github.com/qcscine/sparrow

Table 1: Summary of Mean Negative Eigenvalues by Geometry and Calculator ($n = 100$)

Geometry	HIP Mean	SCINE Mean	Agreement
midpoint_rt	3.55	4.84	23 (23.0%)
reactant	0.00	1.80	5 (5.0%)
product	0.10	2.18	10 (10.0%)
midpoint_rt_noise1A	9.55	11.36	11 (11.0%)
midpoint_rt_noise2A	13.42	10.07	10 (10.0%)

Key Observations:

- **95% disagreement** on reactant geometries
- SCINE only sees 5% of reactants that HIP sees
- **Fix:** Apply climbing algorithm when minima detected

(In any other geometry, HIP and SCINE are quite consistent.)

Like HIP, starting from reactants, SCINE can achieve 100% convergence using GAD with some simple tweaks

Key Insight

Pure GAD achieves **100% convergence on non-minima** samples. The challenge is making it work on noisy geometries with many negative eigenvalues.

Overall Statistics (Pure GAD on TS geometries)

λ_0:	-0.074139 ± 0.079542 eV/Å^2
λ_1:	0.044880 ± 0.062035 eV/Å^2
$\lambda_0 \cdot \lambda_1$:	$-5.443389\text{e-}03 \pm 7.901079\text{e-}03$
Steps taken:	18.2 ± 28.8
Steps to TS:	18.2 ± 28.8
Final time:	0.132 ± 0.182 s

Transition Distribution

```
0neg-to-1neg:  5 samples
1neg-to-1neg:  11 samples
2neg-to-1neg:  14 samples
```

While these didn't converge automatically with GAD, a simple kick was applied to get them out of minima.

Noisy Geometry Experiments

Where we were:

- Previous attempts either go in wrong directions, or plateau
- Cannot escape highly unstable regions

3 – Plateau Detection (Kicking)

Instead of directly using our algorithms, what if we just minimize the energy *(and possibly fine-tune later using our algorithms)?*

Key Insight

Idea: Since GAD converges 100% from non-minima, first reduce the saddle order using L-BFGS energy minimization, then apply GAD.

The Algorithm

1. Project forces into vibrational subspace (mass-weighted Eckart projection)
2. Use projected forces as gradients for L-BFGS
3. Check eigenvalue count periodically
4. Stop when target saddle order reached (e.g., 1 negative eigenvalue)

3 L-BFGS Energy Minimizer

While the order of instability is reduced in most samples, they plateau, and there are many who have very little effect on instability.

3 L-BFGS Energy Minimizer

SCINE Results

10/100 Convergence

- Pretty good reduction of order
- 2.5s–10s per sample
- Good reduction if starting above 10 eigenvalues
- Many plateau at ∼8 negative eigenvalues

HIP Results

0/100 Convergence

- Number of neg-eigs plateau
- Energy changes become negligible
- Really good reduction of order
- 0.8s–4.5s per sample

Conclusion: L-BFGS alone is insufficient. Need a more elegant approach.

4 – Plateau Detection (Kicking)

GAD is the most elegant way forward, but it plateaus (stepsize becomes tiny). What if we kick it in the GAD direction by a big step?

Key Insight

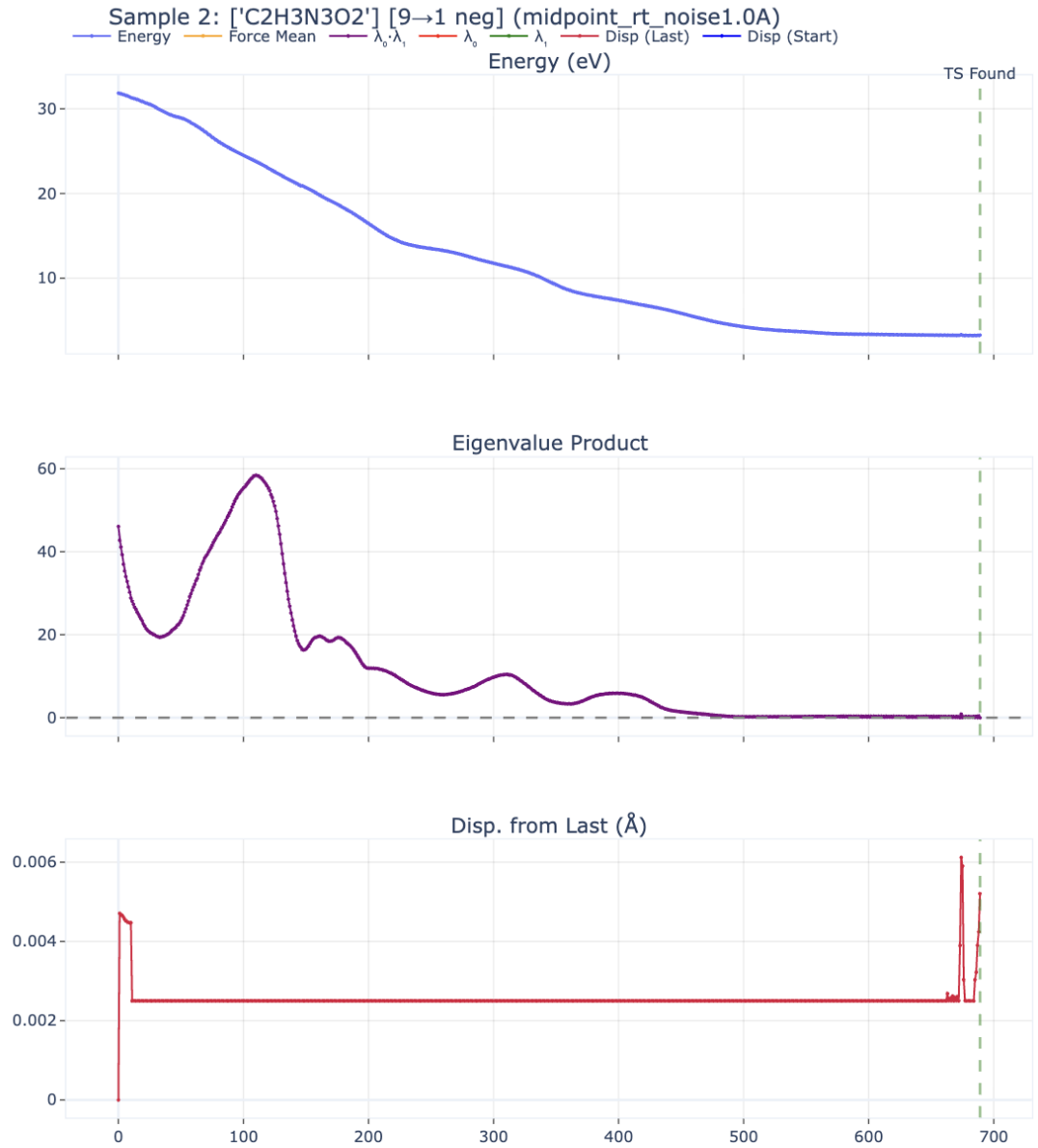
During GAD optimization, the algorithm can get "stuck" at high-index saddle points.
Solution: Monitor negative eigenvalue count and adaptively adjust step size.

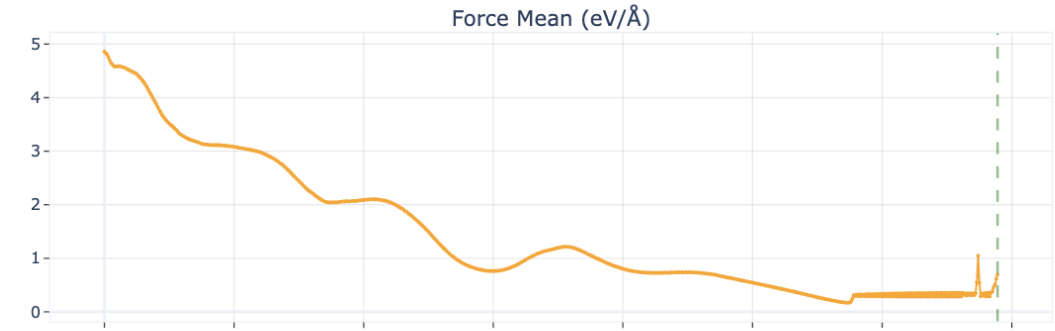
4.1 The Algorithm

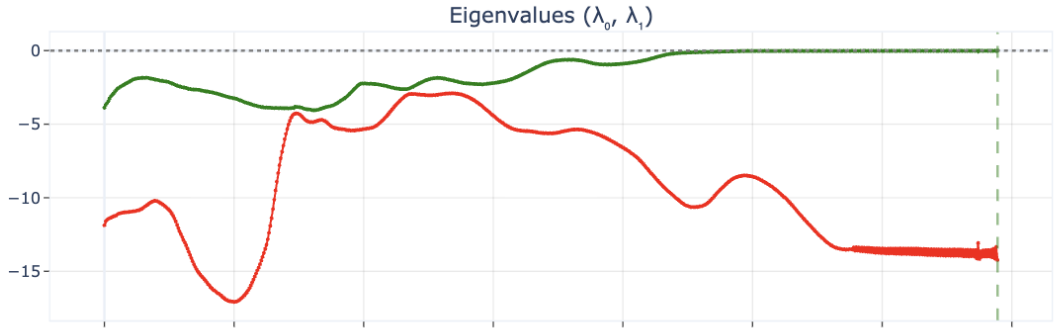
1. Track `best_neg_vib` (lowest saddle order seen)
2. **If improved:** Reset step size to base `dt`
3. **If worsened:** Shrink step size (more careful steps)
4. **If unchanged for** `patience` **steps:** Boost step size (kick!)
5. Enforce minimum step size floor (500 μÅ)

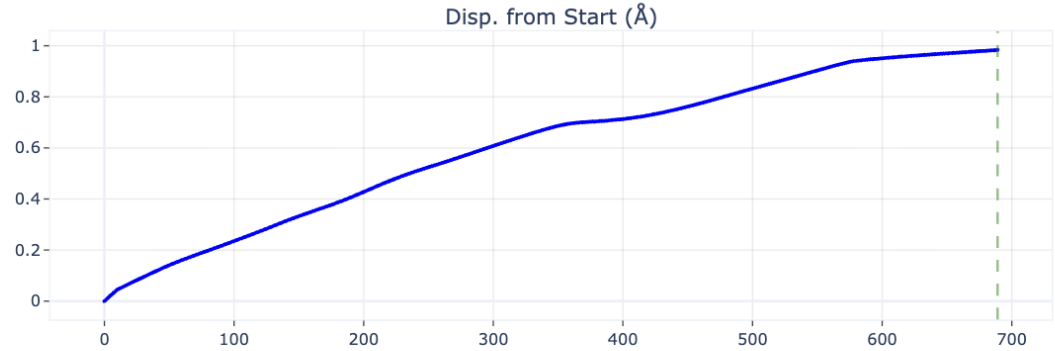
4 – Plateau Detection (Kicking)

Successful sample that was saved by step size floor + kicking



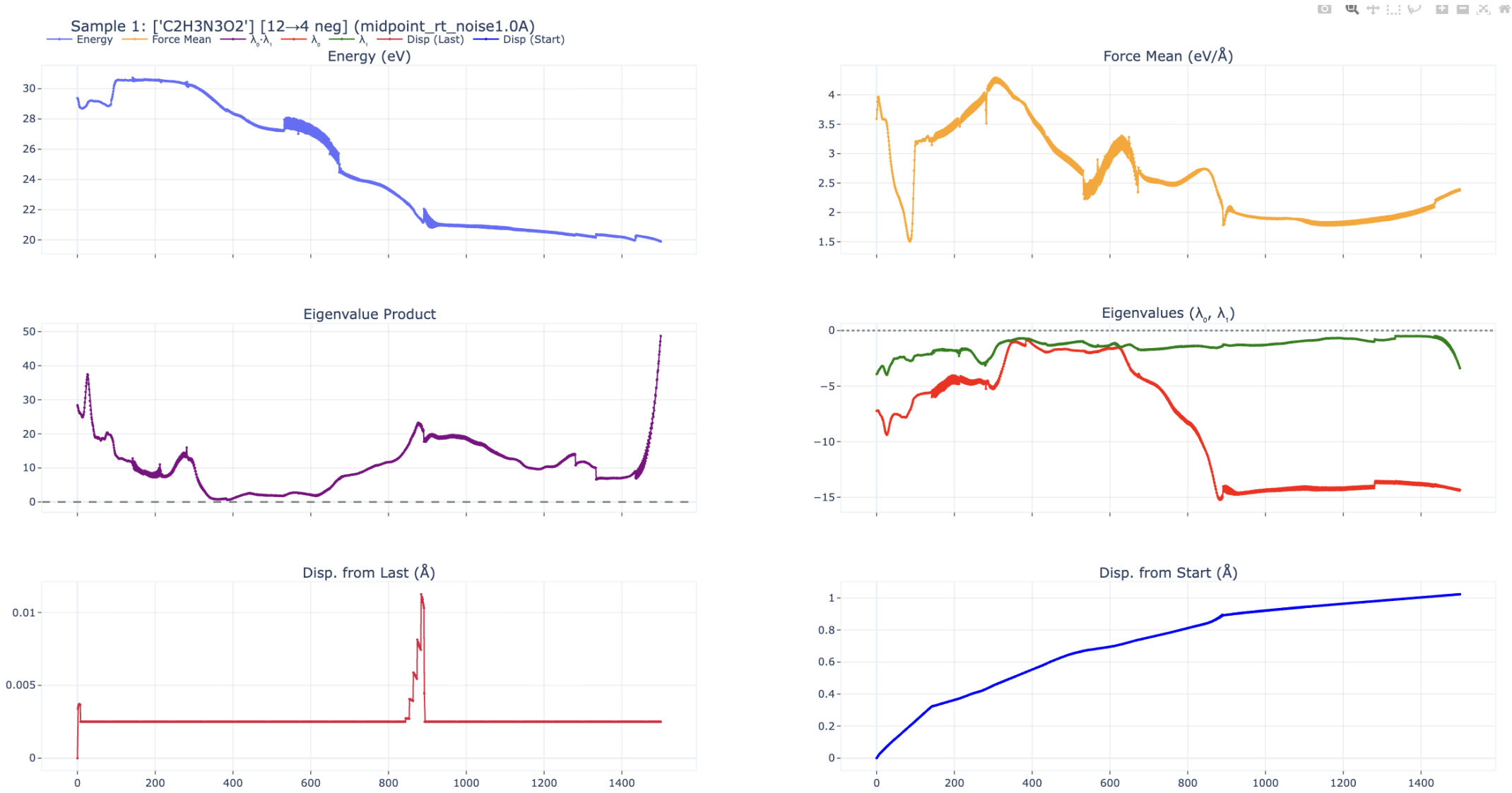






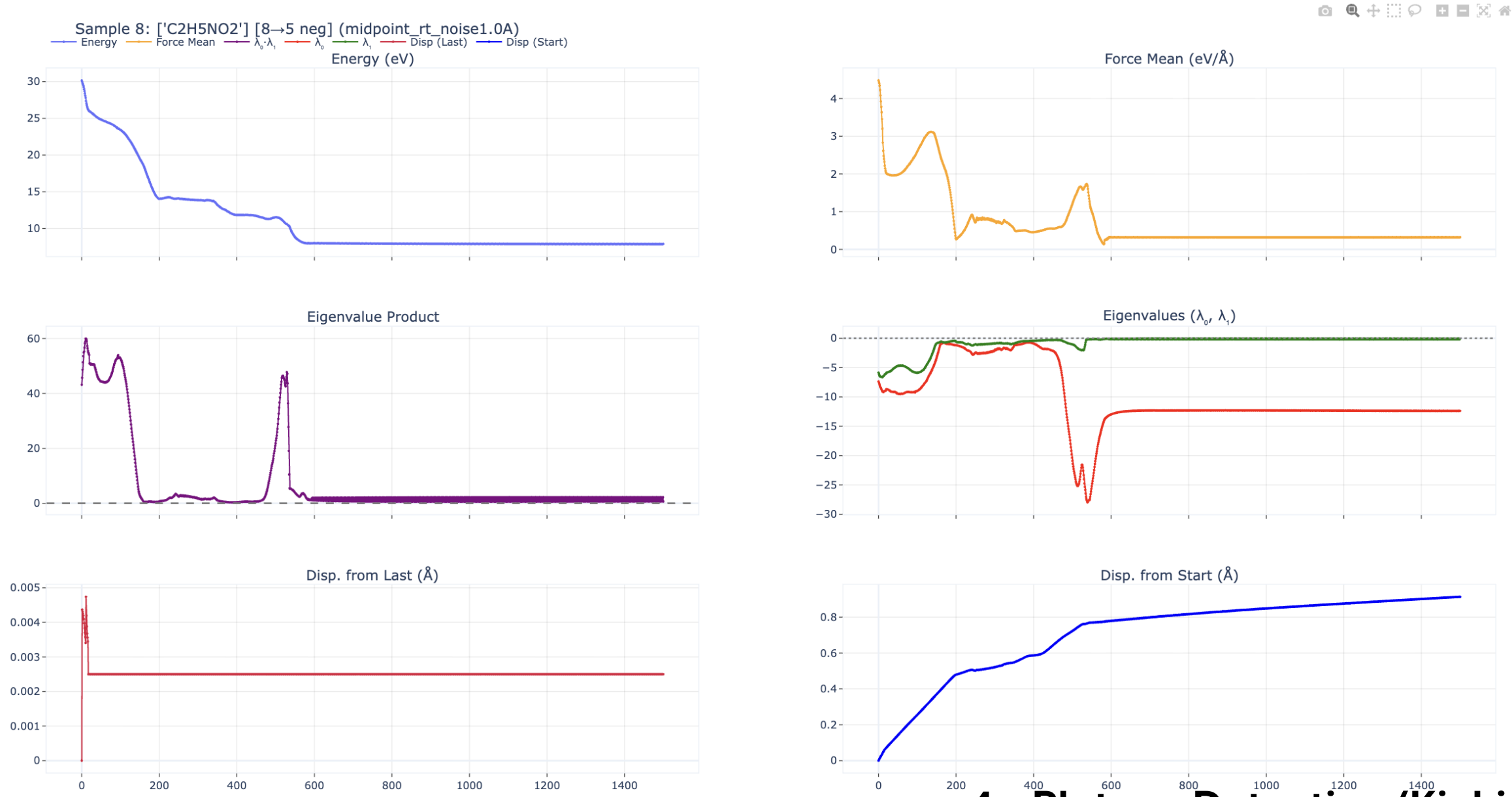
4 – Plateau Detection (Kicking)

Sample that diverged because of kicking



4 – Plateau Detection (Kicking)

Sample that started oscillating as it stepped with size stepsize floor



4 – Plateau Detection (Kicking)

Even though this is a very naïve algorithm (forcing GAD to take big steps), it had 40% convergence, much higher than before.

SCINE Results

40% Convergence

λ_0:	-3.49 ± 3.19 eV/Å^2
λ_1:	-0.21 ± 0.38 eV/Å^2
Steps:	1134.5 ± 471.4
Time:	9.92 ± 4.29 s

HIP Results

40% Convergence

λ_0:	-10.85 ± 4.49 eV/Å^2
λ_1:	-0.43 ± 1.01 eV/Å^2
Steps:	1092.0 ± 516.8
Time:	75.70 ± 35.91 s

Typical Failure Modes:

- Start at floor displacement, good movement until ~600 steps, then oscillates (~7 neg)
- Some graphs with ranges 12neg–4neg: Explodes at ~1300 steps

4 – Plateau Detection (Kicking)

5 Higher-Order GAD (Multi-Mode Escape)

(Smarter kicks)

GAD only considers the smallest eigenvector. What if we explore higher orders?

Key Insight

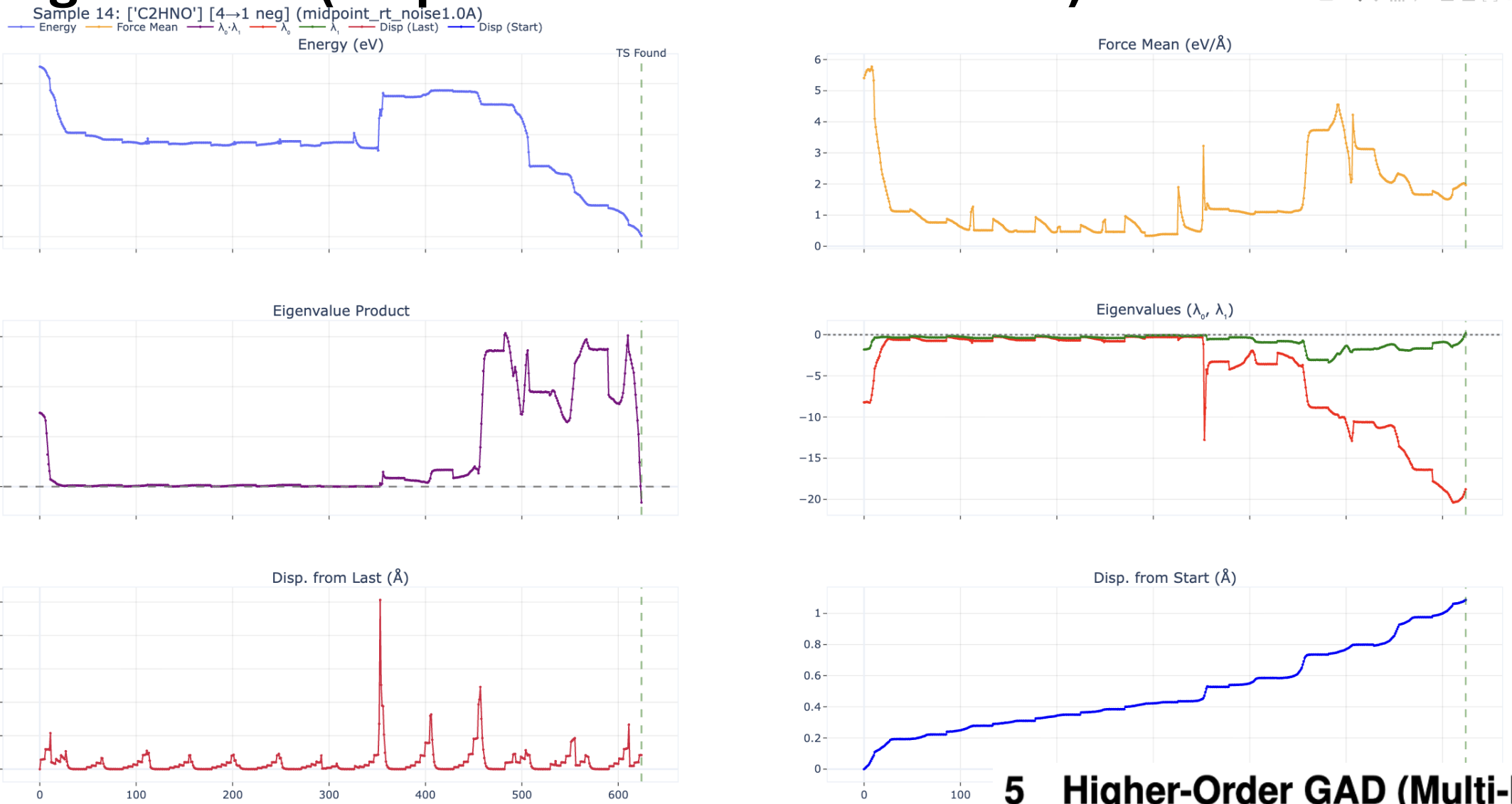
The “plateaus” didn’t make sense because the geometry was still unstable. **Discovery:** The eigenvalue being minimized was converging (force norm becoming tiny), but other negative modes remained.
Solution: When stuck at a high-index saddle, perturb along $\mathbf{v}_2$ (second eigenvector) to escape, then resume GAD.

1. Run GAD until displacement becomes tiny (plateau detected)
2. If converged to order-1 saddle: success
3. If stuck at order > 1: Perturb along v2 (escape perturbation)
4. Resume GAD from perturbed geometry
5. Repeat until order-1 or max cycles

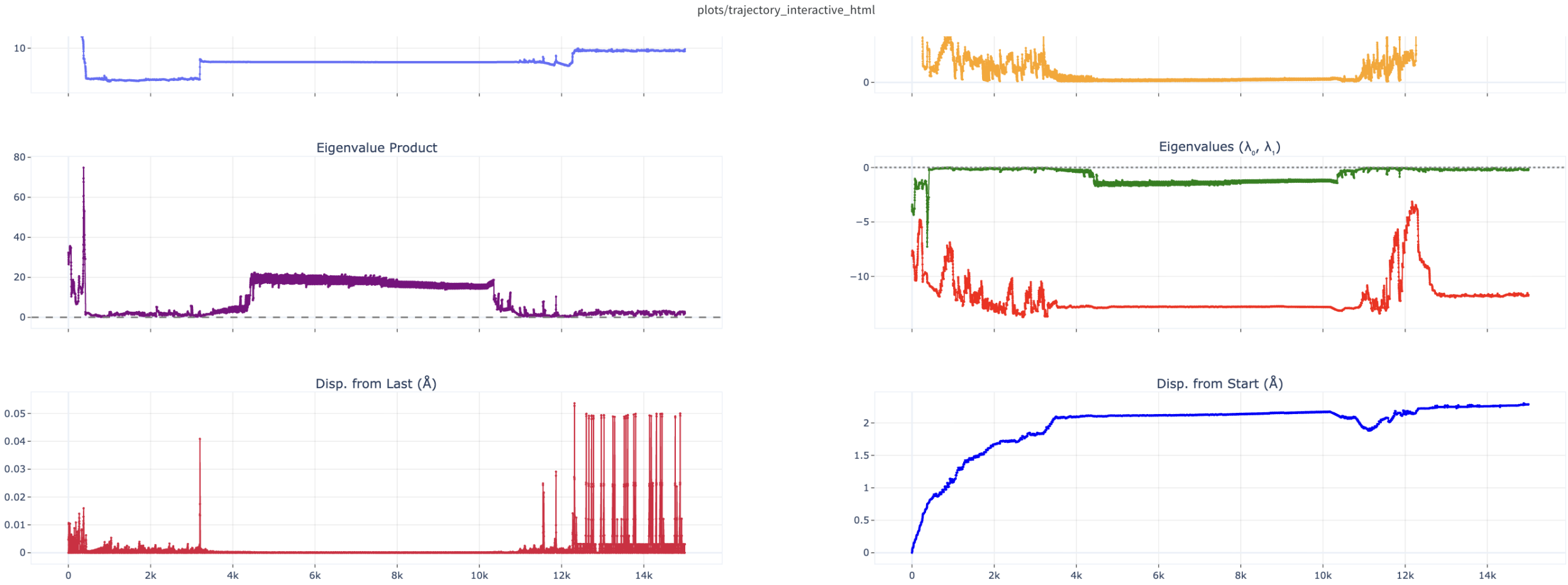
reached

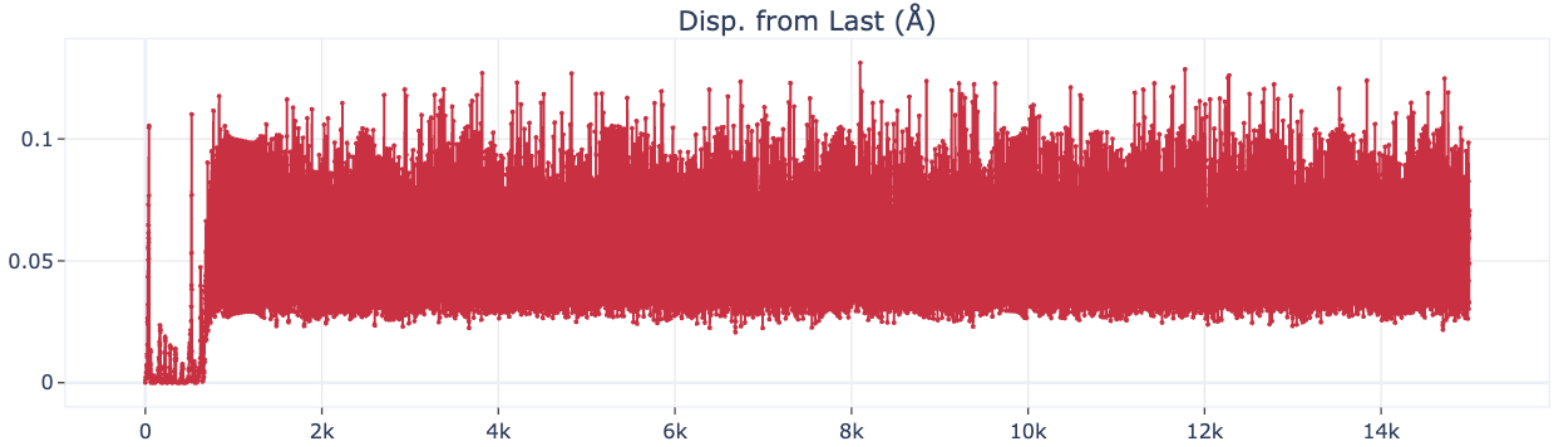
5 Higher-Order GAD (Multi-Mode Escape)

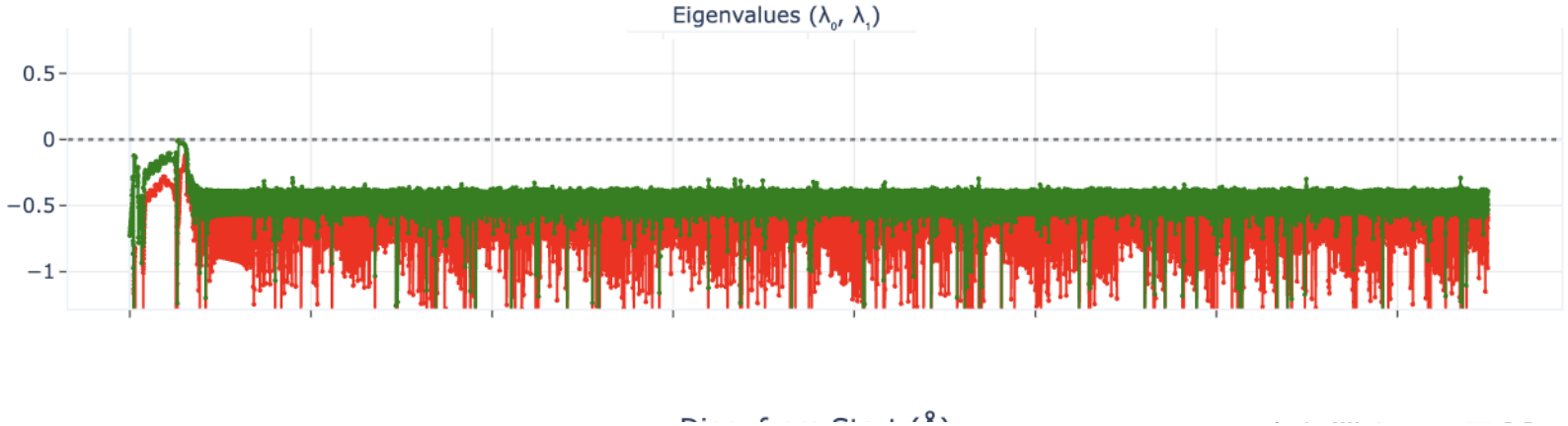
Look at the bottom left, we are constantly applying kicks as soon as the force direction becomes orthogonal to the smallest eigenvector (step size becomes too small).

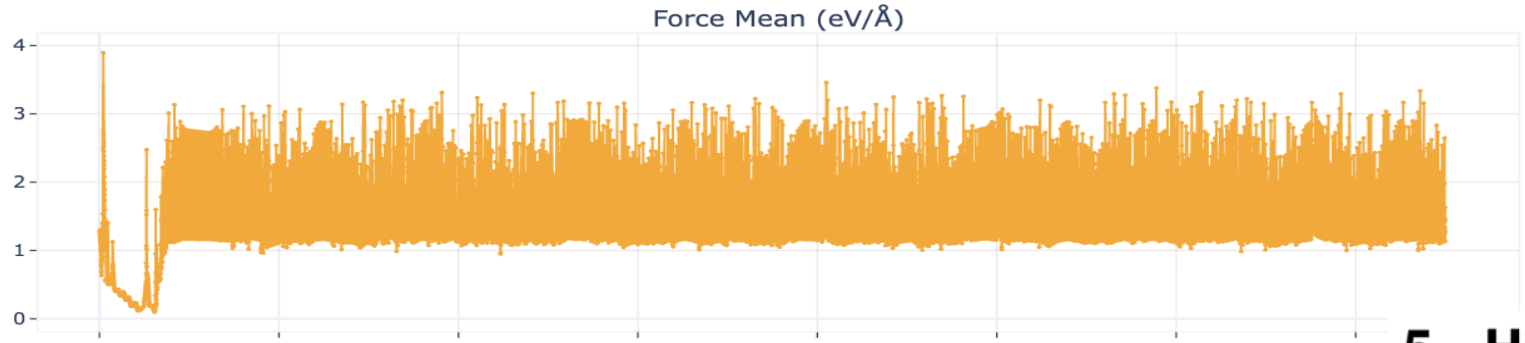


In this example, the trajectory just keeps oscillating. Hypothesize that this is because the “lowest eigenvalue” identity is being switched, because we’re not tracking modes.









A closer look at the displacement, eigenvalues, and force mean that just keeps oscillating

Even though we're still at 40% convergence, trajectories look much better (less plateaus, and getting much closer to convergence)

HIP Results

40% Convergence

Same comments as SCINE.

HIP Errors observed:
`edge_vec_0_distance` errors ($\sim 10^{-5}$)

λ_0:	-4.02 ± 4.18 eV/Å^2
λ_1:	-0.71 ± 1.20 eV/Å^2
Steps:	344.1 ± 175.6
Time:	24.19 ± 12.30 s

SCINE Results

40% Convergence

Same as kicking—but **different failure mode**!

Before: Moving in wrong directions (naive)
Now: 2–3 smallest eigenvalues oscillating with each other

λ_0:	-5.60 ± 9.81 eV/Å^2
λ_1:	-0.25 ± 0.52 eV/Å^2
Steps:	365.7 ± 82.3
Time:	2.84 ± 0.55 s

Two tweaks to try and get multi-mode GAD to work:

(based on observations from trajectories)

Key Insight

Two critical improvements that enable robust convergence:

1. **Mode Tracking:** When eigenvalues are close or degenerate, eigenvector ordering can swap between steps, causing oscillations. Track the eigenvector with maximum overlap to previous step.
2. **Trust Radius:** Limit maximum displacement per step to prevent "explosions" into unphysical regions.

Combined Effect:

- Mode tracking eliminates oscillations from eigenvector reordering
- Trust radius prevents geometry explosions after escape perturbations
- Together: stable traversal through high-dimensional saddle landscape

SCINE Results

92%

λ_0:	-4.02 ± 5.77 eV/Å^2
λ_1:	-0.006 ± 0.07 eV/Å^2
Steps:	3934.9 ± 5032.0
Time:	35.67 ± 45.53 s

HIP Results

71%

λ_0:	-5.22 ± 5.43 eV/Å^2
λ_1:	-0.30 ± 0.84 eV/Å^2
Steps:	3858.8 ± 5636.1
Time:	305.75 ± 390.02 s

Sample that was previously stuck in an infinite oscillation loop, now converging