Companion Notes for `landau_hermite_jax.py`: Fast Landau–Hermite Collisions via SOE→MPO/TT (JAX-first)

(auto-generated companion for the standalone script)

February 3, 2026

Abstract

This document explains the numerics, implementation, and test suite used by the standalone script `landau_hermite_jax.py`. The script implements a fast evaluation of the spatially homogeneous Landau (Fokker–Planck) collision operator in a 3D tensor-product Hermite basis, using a separable quadrature ("SOE" style separation), tensor-product operator factorizations ("MPO" viewpoint), and optional tensor-train (TT) compression (currently used in the NumPy debug path; the JAX path relies on JIT+vectorized contractions).

The goal is pedagogic completeness: what the code computes, how it is normalized, how it is time-stepped, what "linearized" means in this context, why the tests are designed the way they are, and how to interpret the produced figures (Fig. 1/Fig. 2 panels and the test plots).

Contents

1 Quickstart and reproducibility

1.1 What the script produces

Running `python landau_hermite_jax.py` (defaults) writes:

- `Fig1_panel.pdf` and `Fig1_panel.png`: a 2×2 diagnostic panel for a one-species relaxation problem and a two-species equilibration problem.
- `Fig2_panel.pdf` and `Fig2_panel.png`: same dynamics/parameters as Fig. 1, but with a different, strongly non-Maxwellian initial condition (IC) designed to remain nonnegative on diagnostic grids.
- Optionally (if `--run_tests`), a timestamped directory under `tests_landau_hermite/run_YYYYmmdd_HHMMSS/` containing plots, CSV, and JSON summaries. The latest run is also linked as `tests_landau_hermite/latest`.

1.2 Backend

The script has two backends:

- **JAX backend** (`--backend jax`, default): uses `jax.jit`, `vmap`/`einsum`, and `lax.scan` time stepping for performance and scalability.
- **NumPy backend** (`--backend numpy`): for debugging and for certain explicit checks.

Both backends use the *same* precomputed coefficient tables and the same fast SOE→MPO contraction structure; the difference is whether contractions are executed with NumPy or JAX/XLA.

1.3 CLI options

The script self-documents its CLI via `-h`. For convenience we include the exact help text:

```
usage: landau_hermite_jax.py [-h] [--backend {jax,numpy}] [--nmax NMAX]
                             [--Q Q] [--maxK MAXK]
                             [--integrator {rk2,ssprk3,rk4}]
                             [--linearized {on,off}]
                             [--linearized_method {tangent,matrix}]
                             [--progress_chunks PROGRESS_CHUNKS] [--quiet]
                             [--use_tt] [--tt_tol TT_TOL] [--tt_rmax TT_RMAX]
                             [--dt_1sp DT_1SP] [--dt_2sp DT_2SP]
                             [--tmax_1sp TMAX_1SP] [--tmax_2sp TMAX_2SP]
                             [--steps_1sp STEPS_1SP] [--steps_2sp STEPS_2SP]
                             [--fig1_ic {prl_m2,twostream}]
                             [--no_enforce_nonneg_ic] [--amp1 AMP1]
                             [--dT2 DT2] [--outprefix_fig1 OUTPREFIX_FIG1]
                             [--outprefix_fig2 OUTPREFIX_FIG2]
                             [--outprefix OUTPREFIX] [--skip_fig1]
                             [--skip_fig2] [--fig2_strength FIG2_STRENGTH]
                             [--seed SEED] [--entropy_nx ENTROPY_NX]
                             [--entropy_xlim ENTROPY_XLIM] [--run_tests]
                             [--tests_outdir TESTS_OUTDIR]
                             [--tests_nmax_list TESTS_NMAX_LIST]
                             [--tests_reps_rhs TESTS_REPS_RHS]
                             [--tests_reps_bench TESTS_REPS_BENCH]
                             [--tests_steps TESTS_STEPS] [--tests_dt TESTS_DT]
                             [--tests_entropy_nx TESTS_ENTROPY_NX]
                             [--tests_entropy_xlim TESTS_ENTROPY_XLIM]
                             [--tests_Q_sweep TESTS_Q_SWEEP]
                             [--tests_maxK_sweep TESTS_MAXK_SWEEP]
```

```
                            [--tests_max_numpy_nmax TESTS_MAX_NUMPY_NMAX]

Standalone fast SOE→MPO/TT Landau-Hermite (JAX-first): Fig1 + Fig2 panels.

options:
  -h, --help            show this help message and exit
  --backend {jax,numpy}
  --nmax NMAX
  --Q Q
  --maxK MAXK
  --integrator {rk2,ssprk3,rk4}
  --linearized {on,off}
                        Include linearized (Maxwellian-background) overlays.
  --linearized_method {tangent,matrix}
                        How to compute linearized evolution (default: matrix-
                        free tangent linear).
  --progress_chunks PROGRESS_CHUNKS
                        If >0, run JAX time stepping in fixed-size chunks and
                        print progress per chunk.
  --quiet               Reduce prints.
  --use_tt              Use optional TT/MPO contraction in NumPy backend
                        (debug/scaling).
  --tt_tol TT_TOL
  --tt_rmax TT_RMAX
  --dt_1sp DT_1SP
  --dt_2sp DT_2SP
  --tmax_1sp TMAX_1SP
  --tmax_2sp TMAX_2SP
  --steps_1sp STEPS_1SP
  --steps_2sp STEPS_2SP
  --fig1_ic {prl_m2,twostream}
                        Fig1 IC family for the 1-species case.
  --no_enforce_nonneg_ic
                        Disable IC nonnegativity enforcement on diagnostic
                        grids (not recommended at low nmax).
  --amp1 AMP1           Fig1 1sp control: for prl_m2 it's the 2nd-moment
                        anisotropy amplitude; for twostream it's the stream
                        separation u in vx/vth.
  --dT2 DT2             Target temperature excess for the hot species in 2sp
                        ICs (positivity-safe).
  --outprefix_fig1 OUTPREFIX_FIG1
  --outprefix_fig2 OUTPREFIX_FIG2
  --outprefix OUTPREFIX
                        (compat) Alias for --outprefix_fig1; if
                        --outprefix_fig2 is default, set it to
                        OUTPREFIX+'_Fig2'.
  --skip_fig1
  --skip_fig2
  --fig2_strength FIG2_STRENGTH
                        Scales the positive polynomial distortion used for the
                        Fig2 initial condition (larger => farther from
                        Maxwellian).
  --seed SEED
  --entropy_nx ENTROPY_NX
```

```
--entropy_xlim ENTROPY_XLIM
--run_tests           Run internal correctness/performance tests and write
                      plots into tests_landau_hermite/.
--tests_outdir TESTS_OUTDIR
--tests_nmax_list TESTS_NMAX_LIST
--tests_reps_rhs TESTS_REPS_RHS
--tests_reps_bench TESTS_REPS_BENCH
--tests_steps TESTS_STEPS
--tests_dt TESTS_DT
--tests_entropy_nx TESTS_ENTROPY_NX
                      Entropy grid resolution used in --run_tests physics
                      checks (smaller is faster).
--tests_entropy_xlim TESTS_ENTROPY_XLIM
                      Entropy grid half-width used in --run_tests physics
                      checks.
--tests_Q_sweep TESTS_Q_SWEEP
                      Comma-separated Q values for convergence sweeps in
                      --run_tests.
--tests_maxK_sweep TESTS_MAXK_SWEEP
                      Comma-separated maxK values for convergence sweeps in
                      --run_tests.
--tests_max_numpy_nmax TESTS_MAX_NUMPY_NMAX
                      In --run_tests, only do NumPy-vs-JAX comparisons up to
                      this nmax (NumPy gets slow for larger nmax).
```

2 Physical model: Landau collision operator

2.1 Continuous operator (context)

For species a colliding with species b, the Landau collision operator can be written (in one common form) as

$$Q_{ab}(f_a, f_b)(\boldsymbol{v}) = \nabla_{\boldsymbol{v}} \cdot \int_{\mathbb{R}^3} \mathbf{U}(\boldsymbol{v} - \boldsymbol{v}') \left[f_b(\boldsymbol{v}') \nabla_{\boldsymbol{v}} f_a(\boldsymbol{v}) - f_a(\boldsymbol{v}) \nabla_{\boldsymbol{v}'} f_b(\boldsymbol{v}') \right] \mathrm{d}\boldsymbol{v}', \tag{1}$$

where $\mathbf{U}$ is the Landau tensor kernel (Coulomb interaction case). This operator is:

- **bilinear** in (f_a, f_b),
- **conservative**: it preserves total number, total momentum, and total energy (appropriate combinations across species),
- **entropy producing**: for the *nonlinear* operator, relative entropy to a local Maxwellian is non-increasing in time (H-theorem).

Standard references include Landau (1936), Rosenbluth–MacDonald–Judd (1957), and modern expositions such as Helander & Sigmar (2002) and Villani (2002).

2.2 What the script discretizes

The code discretizes *velocity* in a tensor-product Hermite basis and evolves the corresponding expansion coefficients in time for spatially homogeneous relaxation. All spatial dependence is absent: the ODE is in coefficient space.

3 Normalization, basis, and coefficient tensor representation

3.1 Thermal normalization

Each species has a reference thermal speed v_{th} and mass m. The code uses normalized velocity coordinates

$$x = v_x/v_{\mathrm{th}}, \qquad y = v_y/v_{\mathrm{th}}, \qquad z = v_z/v_{\mathrm{th}}. \tag{2}$$

The convention used throughout is

$$v_{\mathrm{th}} = \sqrt{\frac{2T_{\mathrm{eq}}}{m}} \quad \Longleftrightarrow \quad T_{\mathrm{eq}} = \frac{m v_{\mathrm{th}}^2}{2}, \tag{3}$$

where T_{eq} is the equilibrium temperature corresponding to the chosen normalization.

3.2 Hermite basis used in the code

The script uses a Hermite–Gaussian basis in each dimension:

$$\psi_n(x) = \frac{1}{\sqrt{\pi}} \frac{H_n(x)}{\sqrt{2^n n!}}\, e^{-x^2}, \tag{4}$$

where H_n is the physicists' Hermite polynomial. The code implements ψ_n in `psi_1d`. This is a standard choice for Hermite spectral methods in kinetic theory; it provides an efficient ladder structure for derivatives and velocity multiplications.

3.3 Tensor coefficient representation

For a truncation parameter $n_{\max}$, define $p = n_{\max} + 1$. The coefficient tensor is

$$f[\alpha, \beta, \gamma], \quad \alpha, \beta, \gamma \in \{0, \ldots, n_{\max}\}, \tag{5}$$

so that

$$f(x, y, z) \approx \sum_{\alpha,\beta,\gamma=0}^{n_{\max}} f[\alpha, \beta, \gamma]\, \psi_\alpha(x) \psi_\beta(y) \psi_\gamma(z). \tag{6}$$

The code keeps this 3D "cube" representation (shape (p, p, p)) throughout to avoid flattening overhead and to make tensor contractions explicit and JIT-friendly.

3.4 Moment diagnostics and invariants

From the Hermite coefficients, the script extracts:

- density n,
- momentum $\boldsymbol{P}$,
- total kinetic energy W,
- temperature $T = \frac{2}{3}\frac{W}{n}$,
- anisotropy measure $A = (T_z - T_x)/T_{\mathrm{avg}}$.

These are computed from low-order coefficients (see `invariants_from_tensor` and `temperature_components_hat_f`). Conservation of $(n, \boldsymbol{P}, W)$ (or total invariants for 2 species) is one of the most stringent correctness checks for collision operators.

4 Fast collision evaluation: SOE→MPO/TT

4.1 Why naive evaluation is expensive

In a generic coefficient formulation, the bilinear operator has the form

$$\frac{\mathrm{d}f_k}{\mathrm{d}t} = \sum_{k_a,k_b} C^{ab}_{k,k_a,k_b}\, f_{a,k_a}\, f_{b,k_b}, \tag{7}$$

which is $O(N^3)$ if the dense tensor C^{ab} is formed and contracted directly (with $N = p^3$). Even storing C^{ab} is prohibitive.

The script avoids forming C^{ab} and instead uses a structured evaluation that exploits separability of the underlying integrals and the tensor-product basis.

4.2 SOE / separable quadrature idea

The coefficient formulas in this method contain factors of the form $\frac{1}{2A+1}$ (with A an integer index arising from the Coulomb kernel moments). The key separability identity is:

$$\frac{1}{2A+1} = \int_0^1 s^{2A}\,\mathrm{d}s. \tag{8}$$

Approximating the integral using Gauss–Legendre quadrature on $[0,1]$ gives:

$$\frac{1}{2A+1} \approx \sum_{q=1}^{Q} w_q\, s_q^{2A}, \tag{9}$$

which is a *sum of separable terms* across dimensions because A itself decomposes into a sum of 1D indices. In the code, `leggauss_01_np` constructs (s_q, w_q), and the resulting q-terms are handled via a small loop over q (either explicit loop in NumPy or vectorized/JAX loop).

This "SOE" terminology is used in the code base to mean "sum of separable terms" (often also called separated representations, sum-of-products, etc.). Related ideas are common in fast algorithms for integral operators (e.g. Beylkin & Monzón; Hackbusch & Khoromskij).

4.3 Separable Hankel structure

The algorithm also uses a Hankel-like dependence on combined indices $K = k'_p + k_b + \text{shift}$. In the code, this enters through the table

$$F_q[n] = \begin{cases} 0, & n \text{ odd}, \\ (-1)^{n/2}\, t_{n/2}\, s_q^n, & n \text{ even}, \end{cases} \tag{10}$$

where

$$t_n = \frac{\sqrt{(2n)!}}{2^n n!}. \tag{11}$$

See `t_factor_np` and `Fq_table_np`. The truncation parameter `maxK` limits the maximum Hankel index needed; the docstring in `build_model_tables_np` explains the rule of thumb (`maxK` comfortably above $\sim 3n_{\max}$).

4.4 MPO viewpoint: tensor-product operator contraction

After SOE separation, the operator can be evaluated through a sequence of *1D* mode products along the x, y, and z axes. In quantum many-body language this is a matrix-product operator (MPO); in numerical linear algebra it is a Kronecker-structured operator.

Concretely, the code precomputes (for each quadrature node q, tensor component indices i, j, and term type) three 1D matrices that act along each dimension. Applying the operator to a 3D tensor then reduces to three successive contractions (a Kronecker product action), implemented via `einsum`/`tensordot` and fused by XLA in the JAX path.

4.5 TT (tensor train) compression

For larger $n_{\max}$, some intermediate tensors can become costly. A tensor-train (TT) factorization approximates a 3D tensor $X \in \mathbb{R}^{p\times p\times p}$ by low-rank factors (TT-SVD). The script includes an optional TT rounding path (enabled via `--use_tt` in the NumPy backend) to reduce intermediate ranks and memory traffic when exploring higher $n_{\max}$.

TT/MPS references: Oseledets (2011), Hackbusch (2012), and related MPO/MPS reviews (Schollwöck 2011; Orús 2014).

4.6 Algorithmic walkthrough (as implemented)

This subsection summarizes the concrete data flow in `landau_hermite_jax.py`.

Inputs and shapes. Fix a truncation $n_{\max}$ and set $p = n_{\max} + 1$. Each distribution is a tensor $f \in \mathbb{R}^{p\times p\times p}$ in the Hermite basis.

Step 0: precompute tables (once per $(n_{\max}, Q, \texttt{maxK})$ and species pair). The function `build_model_tables_np` builds a container of dense arrays:

- Gauss–Legendre nodes/weights (s_q, w_q) on $[0, 1]$ for the separable quadrature.
- 1D mixing coefficients `a1d` encoding relative/COM transforms with the species-dependent angle $(\cos\theta, \sin\theta)$.
- A dense 1D coefficient table `P1D` that maps intermediate indices back to Hermite indices (a 1D "dual↔primal" transform used repeatedly in each dimension).
- The separable Hankel tables `Fq[q,K]` which implement the even-K Coulomb moment factors multiplied by s_q^K (see `Fq_table_np`).
- Preassembled 1D matrices `M_buildS[q,i,j,term,dim]` used to build auxiliary tensors S_1^{ij} and S_2^{ij} from the background distribution (next paragraph).

In the JAX backend these arrays are transferred to device once and treated as constants by JIT.

Step 1: build auxiliary tensors from f_b. For a cross-collision $Q_{ab}(f_a, f_b)$, the algorithm first constructs a small set of intermediate tensors (denoted S_1^{ij} and S_2^{ij} in the code) that play the role of Rosenbluth-potential moments in this Hermite formulation. Each S_ℓ^{ij} has shape (p', p', p') where $p' = 2n_{\max} + 2$ (the intermediate index range required by the Hankel structure). The construction is a sum over quadrature nodes q of Kronecker-structured applications:

$$S_\ell^{ij} \approx \sum_{q=1}^{Q} w_q \left(M^{(x)}_{q,i,j,\ell} \otimes M^{(y)}_{q,i,j,\ell} \otimes M^{(z)}_{q,i,j,\ell} \right) f_b, \tag{12}$$

implemented via three successive 1D contractions. This avoids forming any dense 6D objects.

Step 2: apply S to f_a to produce the RHS. Given S_1, S_2 (from f_b), the RHS for f_a is assembled by applying another Kronecker-structured mapping and then projecting back to the Hermite coefficient cube using the 1D `P1D` table along each axis. This is the "MPO" viewpoint: the full operator factorizes over dimensions, so applying it is a sequence of mode products rather than a giant tensor contraction.

Optional: TT compression (NumPy path). When `--use_tt` is enabled in the NumPy backend, selected intermediate tensors may be TT-rounded to reduce ranks. The JAX backend currently relies primarily on XLA fusion rather than TT-SVD, because SVD-based compression is relatively expensive on CPU for the small-to-moderate p used in the main figures.

Complexity. The dominant cost scales roughly like $O(Q\,p^4)$ for the structured contractions (with a small constant), rather than $O(p^9)$ for a dense bilinear tensor.

5 Nonlinear vs linearized evolution

5.1 Nonlinear operator

The main ODE uses the full bilinear operator:

$$\text{1sp:}\quad \frac{\mathrm{d}f}{\mathrm{d}t} = Q_{11}(f,f), \qquad \text{2sp:}\quad \frac{\mathrm{d}f_a}{\mathrm{d}t} = Q_{ab}(f_a,f_b), \quad \frac{\mathrm{d}f_b}{\mathrm{d}t} = Q_{ba}(f_b,f_a). \tag{13}$$

5.2 Linearization about a Maxwellian

Let M denote a Maxwellian equilibrium (in the *fixed* normalized coordinates for each species). For a perturbation h with $f = M + h$ and $\|h\| \ll 1$,

$$Q(M+h, M+h) = Q(M,M) + Q(h,M) + Q(M,h) + O(h^2). \tag{14}$$

Since $Q(M,M) = 0$, the linearized operator is

$$L(h) = Q(h,M) + Q(M,h). \tag{15}$$

For two species, the linearized Jacobian couples δf_a and δf_b through the mixed terms.

5.3 Entropy vs free energy for linearized dynamics

The nonlinear Landau operator satisfies an H-theorem: relative entropy to the *instantaneous* local Maxwellian decreases. The script measures this via a grid-based KL divergence:

$$\mathcal{D}(t) = \int f \log\!\left(\frac{f}{M_{\text{local}}}\right) \mathrm{d}v. \tag{16}$$

However, this KL functional is *not* expected to be monotone for tangent-linear evolution (and can become ill-defined if the linearized solution produces slight negative values due to truncation).

For linearized evolution, a standard Lyapunov functional is the quadratic "free energy" (second variation of entropy at a Maxwellian):

$$\mathcal{F}(t; M) = \int \frac{(f-M)^2}{M}\, \mathrm{d}v. \tag{17}$$

The script therefore plots $\mathcal{D}/\mathcal{D}_0$ for nonlinear curves and $\mathcal{F}/\mathcal{F}_0$ for linearized curves in the main panels, and it tests monotonicity for each in `--run_tests`.

6 Time stepping and JAX implementation

6.1 Integrator

The script uses explicit SSPRK3 (Shu–Osher) by default, with options for RK2 and RK4. SSPRK3 offers a good stability/accuracy compromise for dissipative problems while using only 3 RHS evaluations per step. See Shu & Osher (1988) and Gottlieb–Shu–Tadmor (2001).

6.2 JAX performance strategy

Key performance choices:

- Enable float64 (`jax_enable_x64`) for accurate invariants and Maxwellian fixed-point checks.
- Treat all precomputed tables as constants to JIT (single device transfer).
- Use `lax.scan` for the time loop so the full stepper is compiled once.
- Keep shapes static (fixed $p = n_{\max} + 1$ and fixed Q per run).
- Avoid Python loops over tensor indices; rely on vectorized contractions.

These are typical "best practices" for XLA-based array programming systems.

7 Initial conditions used in Fig. 1 and Fig. 2

7.1 Fig. 1: one-species IC (two-stream-like, positivity-safe)

The default Fig. 1 one-species IC is designed to be qualitatively "two-stream-like" without any negative values. It is constructed as an even mixture of two shifted Maxwellians in v_x:

$$f(x, y, z) \;\propto\; \frac{1}{2}\Big(M(x-u) + M(x+u)\Big)\, M(y)\, M(z), \tag{18}$$

with u set by `--amp1` (interpreted as stream separation for `--fig1_ic twostream`). The code projects the 1D x dependence onto the truncated $\{\psi_n\}$ basis and optionally scales the non-Maxwellian component to enforce nonnegativity on diagnostic grids.

7.2 Fig. 1: two-species IC (positivity-safe hot/equilibrium)

Representing an extremely cold Maxwellian at low $n_{\max}$ in a fixed normalized coordinate can lead to Gibbs-like oscillations and negative reconstructed f. To avoid this, the two-species IC used in this standalone script heats one species using a positive isotropic polynomial distortion of the equilibrium Maxwellian (exactly representable at low order), while keeping the other species near equilibrium. The parameter `--dT2` sets the target temperature excess for the hot species.

7.3 Fig. 2: far-from-Maxwellian ICs

Fig. 2 uses a positive polynomial distortion (including quartic terms when $n_{\max} \geq 4$) to create strongly non-Maxwellian shapes while maintaining nonnegativity on diagnostic grids. The strength is controlled by `--fig2_strength`. The two-species Fig. 2 IC also tunes the temperature separation in a positivity-safe way.

Table 1: Test sweep summary (JAX backend). Reported times are steady-state. Conservation rates are for cross-collision totals $d/dt\,(n, P, W)$ evaluated on a random near-Maxwellian state.

$n_{\max}$	$N = (n_{\max}{+}1)^3$	t_{self} [ms]	t_{cross} [ms]	t_{int} [s]	$\|RHS(M)\|/\|M\|$ (cross)	$\|dW_{\text{tot}}/dt\|$
2	27	0.093	0.177	0.014	4.23e-16	1.63e-15
3	64	0.280	0.556	0.044	4.35e-16	8.50e-12
4	125	0.584	1.141	0.092	5.98e-16	4.78e-11
5	216	0.854	1.711	0.133	6.00e-16	3.26e-10

8 Figures produced by the script

8.1 Main panels

8.2 Test suite summary and plots

The `--run_tests` mode runs a sweep over $n_{\max}$, checks Maxwellian fixed points, cross-invariant rates, finite-difference consistency of the linearization (when feasible), and short-run performance. It also performs physics-style checks of monotonicity (nonlinear KL entropy and linear quadratic free energy) and generates convergence sweeps in $(Q, \texttt{maxK})$.

Interpreting the test outputs. The tests are designed around standard invariants and stability properties of the Landau operator:

- **Maxwellian fixed point**: $RHS(M) \approx 0$ is a strict algebraic property of the collision operator. In floating point, the residual should be near roundoff if the discretization/tables are correct.
- **Conservation**: for cross-collisions, Q_{ab} and Q_{ba} exchange momentum/energy but must conserve totals. The test plots $|d/dt\,(n_{\text{tot}}, P_{\text{tot}}, W_{\text{tot}})|$ computed directly from the RHS.
- **H-theorem diagnostics**: nonlinear KL entropy to the instantaneous local Maxwellian should be non-increasing (up to truncation/time-discretization). Linearized dynamics should instead be monitored with quadratic free energy about the fixed Maxwellian background.
- **Linearization correctness**: two complementary checks are used: (i) finite-difference self-consistency (NumPy, small $n_{\max}$) and (ii) small-perturbation agreement between nonlinear and tangent-linear time traces (JAX).
- **Performance**: steady-state RHS timing (after compilation) and a short integration benchmark.

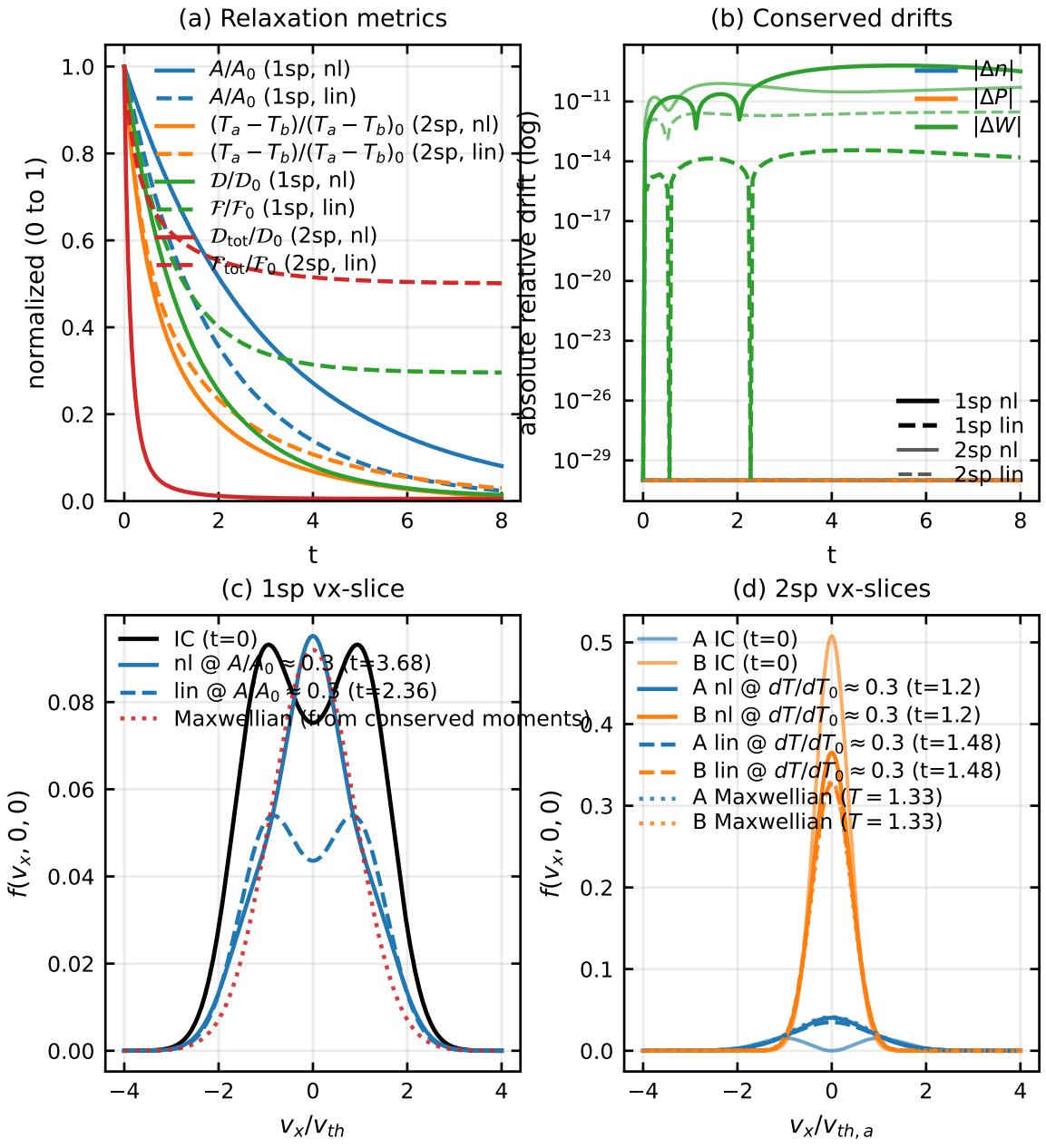

Figure 1: Fig. 1 panel produced by `landau_hermite_jax.py`.

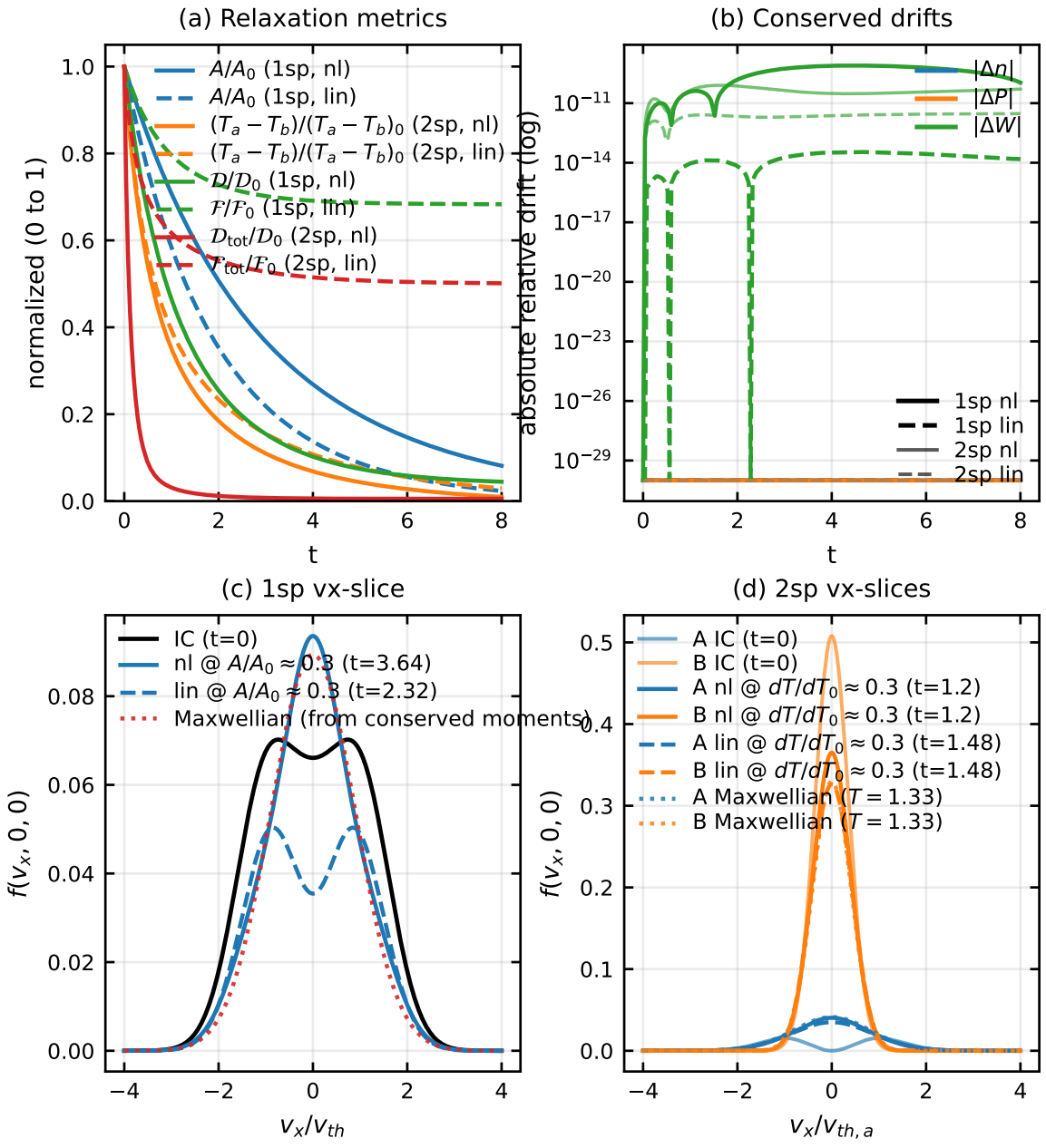

Figure 2: Fig. 2 panel produced by `landau_hermite_jax.py`.

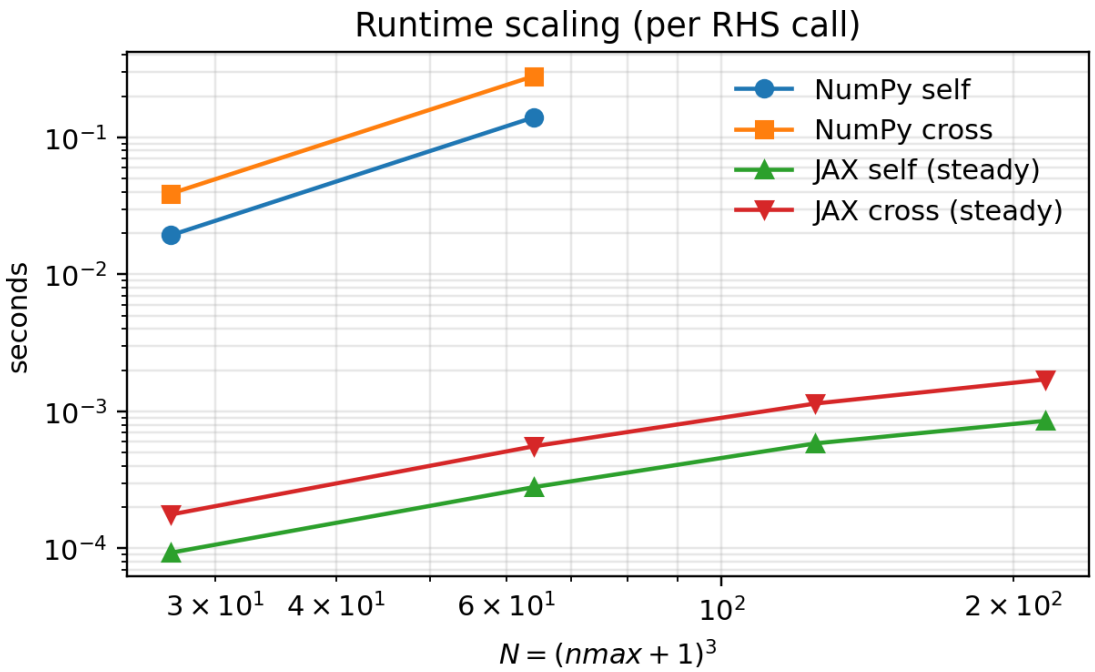

Figure 3: Per-RHS-call runtime scaling across $n_{\max}$ (from the latest test run).

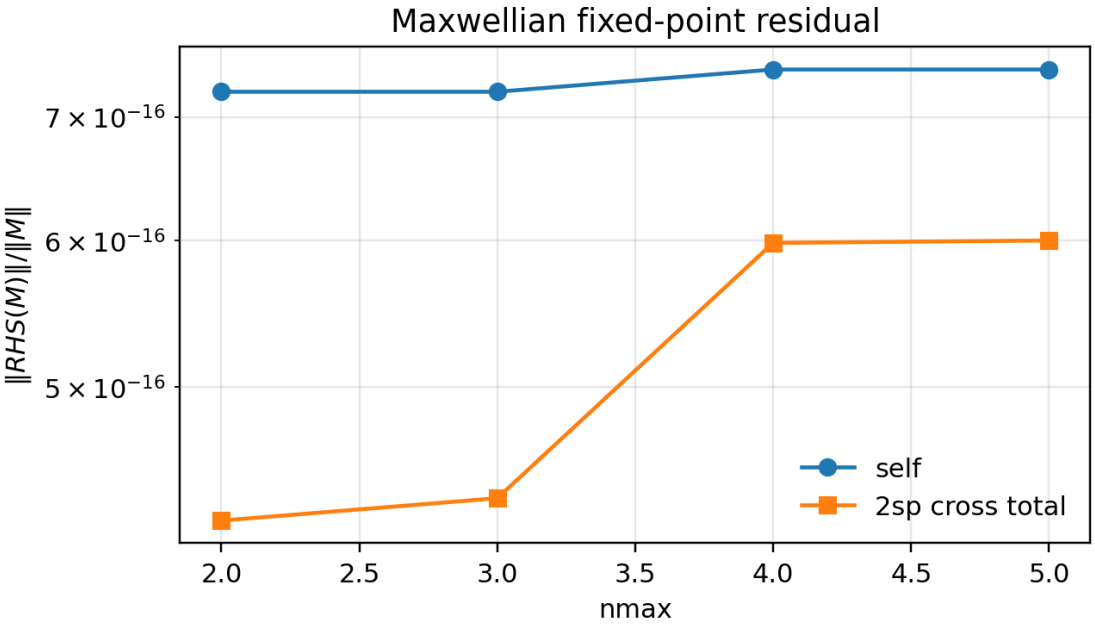

Figure 4: Maxwellian fixed-point residual $\|RHS(M)\|/\|M\|$ across $n_{\max}$.

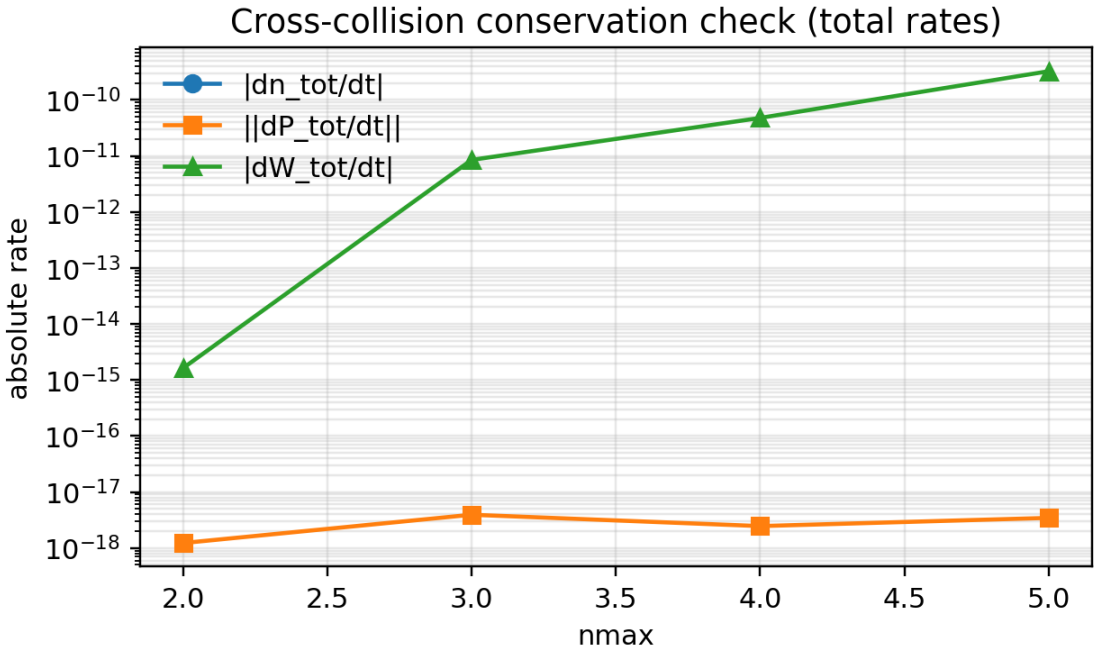

Figure 5: Cross-collision conservation-rate check: total invariant rates computed from the RHS on a random near-Maxwellian state.

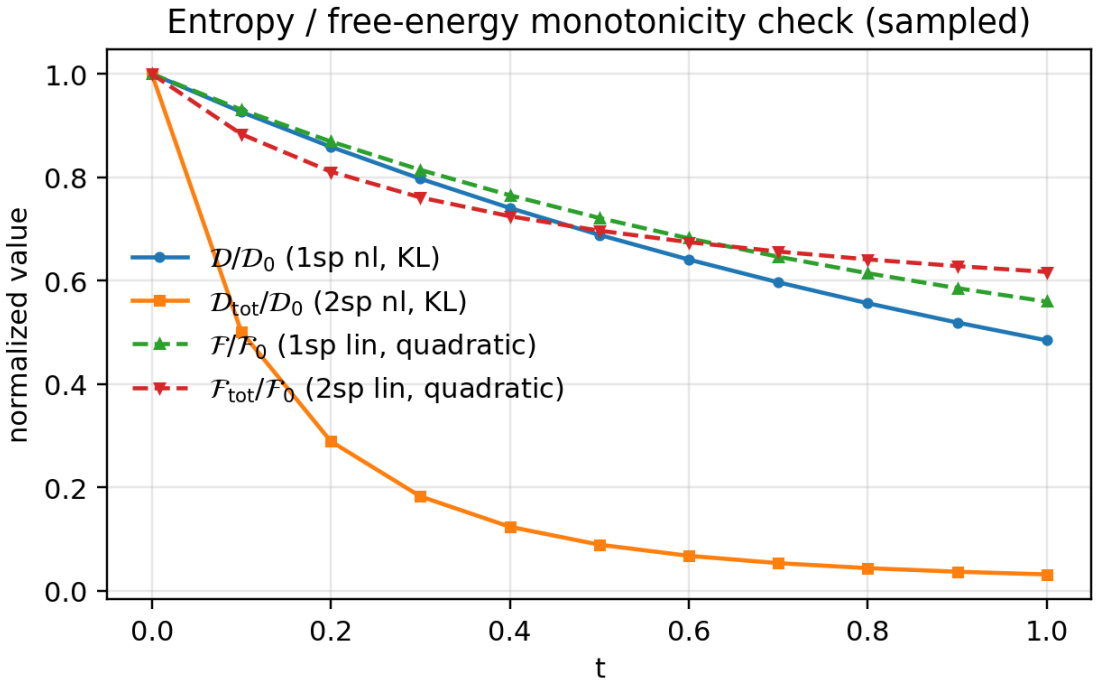

Figure 6: Physics-style monotonicity check: nonlinear KL entropy to instantaneous local Maxwellians (solid) and linearized quadratic free energy about fixed Maxwellians (dashed).

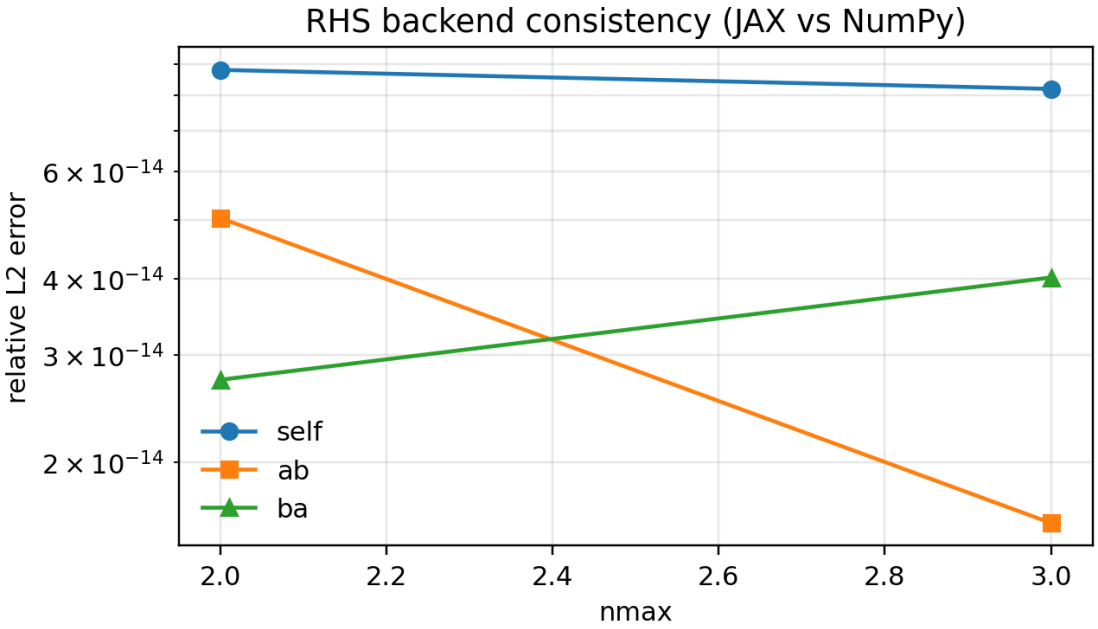

Figure 7: Backend consistency between NumPy and JAX for small $n_{\max}$ where NumPy comparisons are enabled.

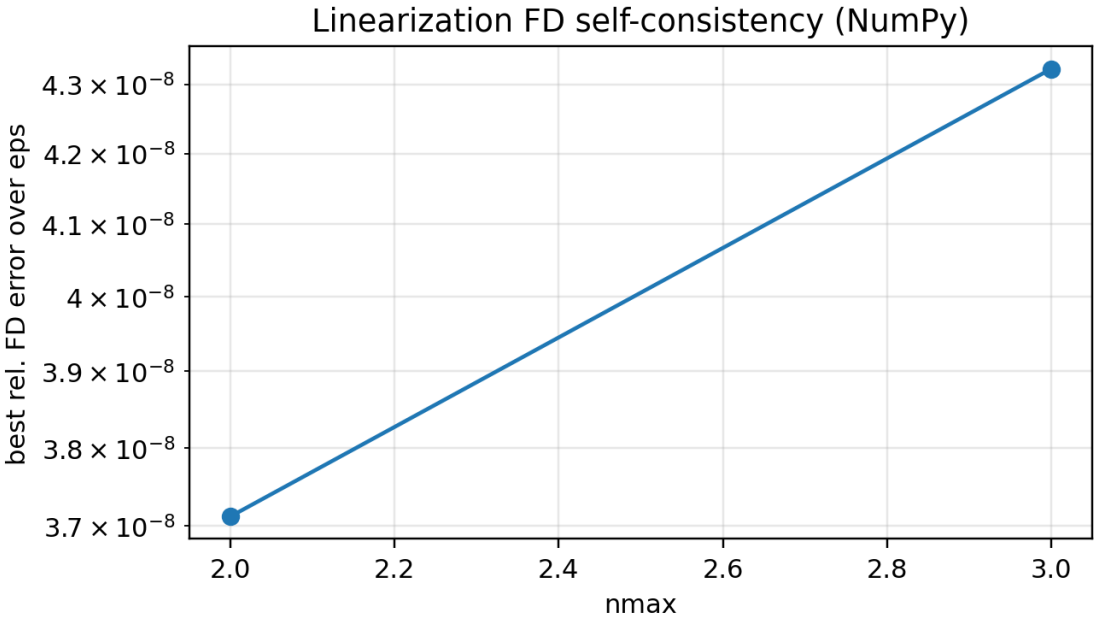

Figure 8: Finite-difference self-consistency check of the linearization (NumPy, where enabled).

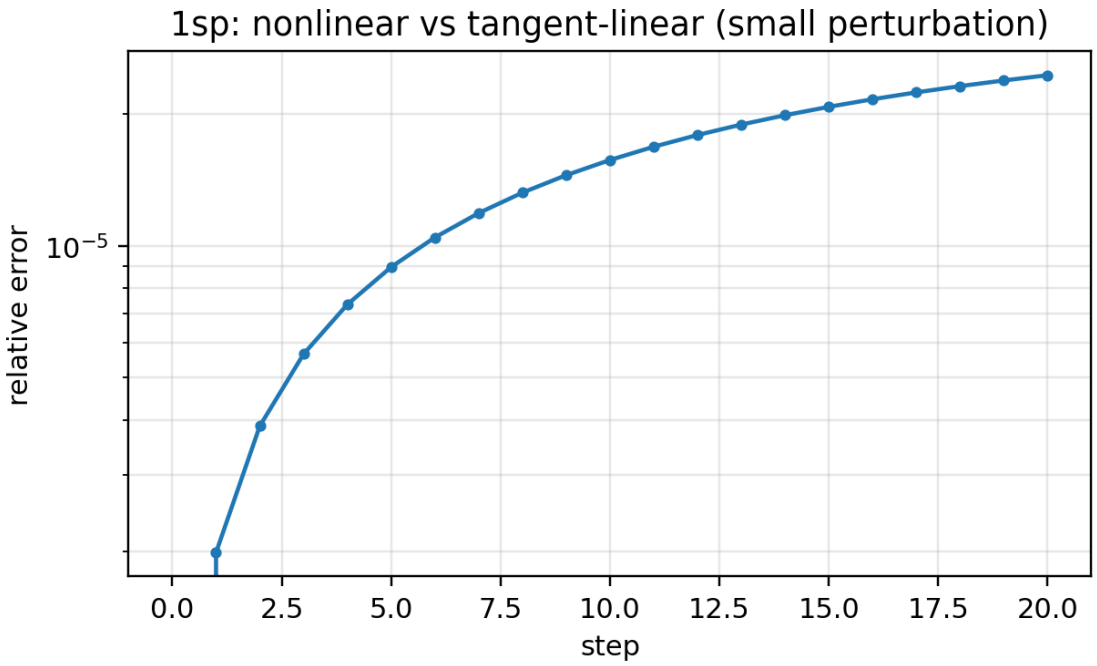

Figure 9: Small-perturbation check: nonlinear vs tangent-linear for the 1-species problem.

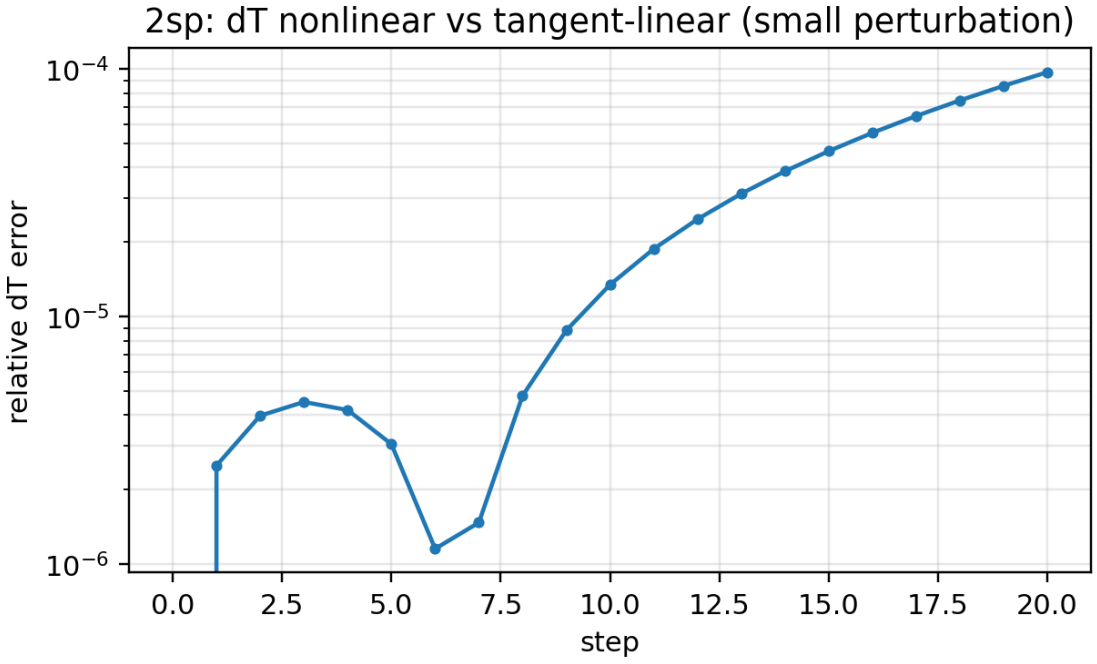

Figure 10: Small-perturbation check: nonlinear vs tangent-linear for the 2-species temperature-exchange metric.

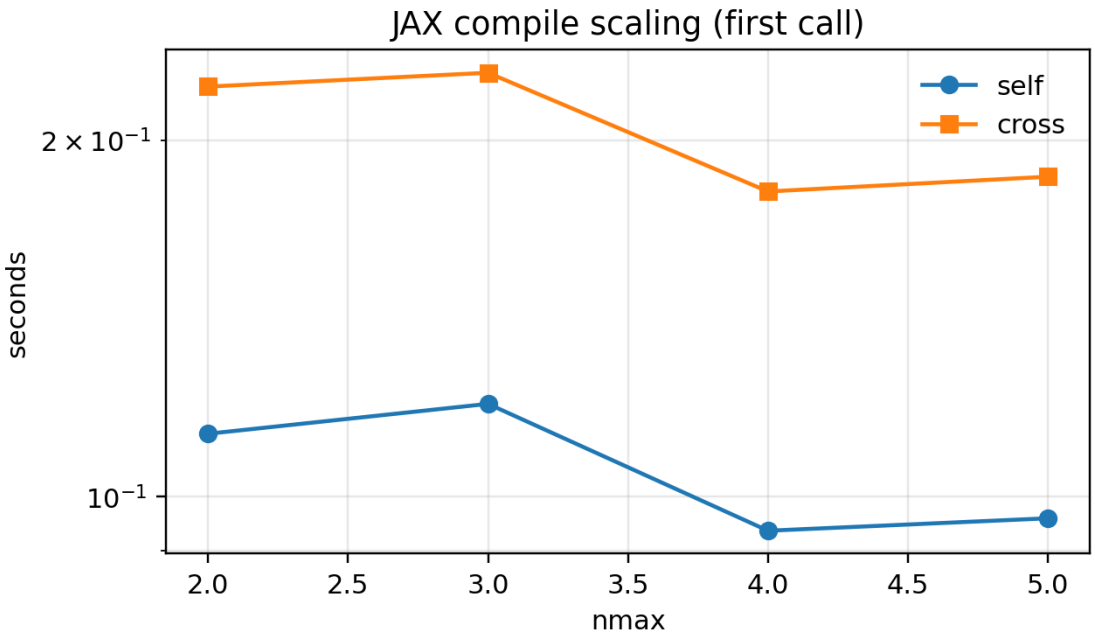

Figure 11: JAX compile-time scaling across $n_{\max}$ (first call).

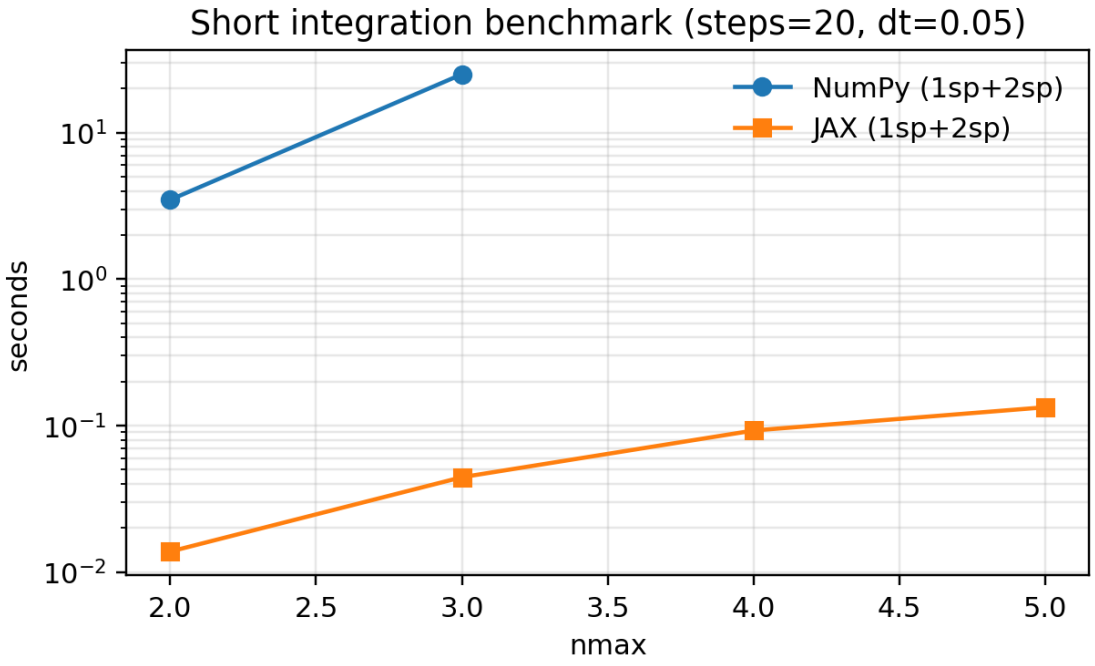

Figure 12: Short integration benchmark (NumPy vs JAX where enabled).

9 Interpreting results and limitations

9.1 Positivity

Hermite spectral truncations do *not* guarantee pointwise positivity of the reconstructed distribution function, even if the underlying physical solution is nonnegative. This is a well-known spectral/Gibbs phenomenon. To keep the plotted slices physically meaningful, the default Fig. 1 one-species IC is built from a manifestly nonnegative two-stream mixture and is optionally "damped" in high modes to remain nonnegative on diagnostic grids.

9.2 Monotonicity

For the nonlinear operator, $\mathcal{D}(t)$ (KL divergence to the instantaneous local Maxwellian) is the referee-proof diagnostic. For the linearized evolution, the correct monotone quantity is the quadratic free energy $\mathcal{F}(t; M)$ about the fixed Maxwellian background, not KL to a time-dependent Maxwellian.

9.3 Truncation and parameter choices

The separation accuracy is controlled by:

- $n_{\max}$: Hermite resolution (spectral truncation),
- Q: quadrature nodes in the SOE separation,
- `maxK`: maximum Hankel index in the Coulomb moment tables.

The test suite includes Q/`maxK` sweeps and Maxwellian fixed-point residuals to help select safe defaults.

10 References (selected)

References

[1] L. D. Landau, "Kinetic equation for the Coulomb effect," *Physikalische Zeitschrift der Sowjetunion* (1936).

[2] M. N. Rosenbluth, W. M. MacDonald, and D. L. Judd, "Fokker–Planck equation for an inverse-square force," *Physical Review* **107** (1957).

[3] P. Helander and D. J. Sigmar, *Collisional Transport in Magnetized Plasmas*, Cambridge Univ. Press (2002).

[4] C. Villani, "A review of mathematical topics in collisional kinetic theory," in *Handbook of Mathematical Fluid Dynamics*, Vol. I (2002).

[5] C.-W. Shu and S. Osher, "Efficient implementation of essentially non-oscillatory shock-capturing schemes," *Journal of Computational Physics* **77** (1988).

[6] S. Gottlieb, C.-W. Shu, and E. Tadmor, "Strong stability-preserving high-order time discretization methods," *SIAM Review* **43** (2001).

[7] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing* **33** (2011).

[8] W. Hackbusch, *Tensor Spaces and Numerical Tensor Calculus*, Springer (2012).

[9] U. Schollwöck, "The density-matrix renormalization group in the age of matrix product states," *Annals of Physics* **326** (2011).

[10] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states," *Annals of Physics* **349** (2014).

[11] G. H. Golub and J. H. Welsch, "Calculation of Gauss quadrature rules," *Mathematics of Computation* **23** (1969).

[12] J. P. Boyd, *Chebyshev and Fourier Spectral Methods*, 2nd ed., Dover (2001). (Background on spectral truncation/oscillations.)

[13] H. Grad, "On the kinetic theory of rarefied gases," *Communications on Pure and Applied Mathematics* **2** (1949).

[14] G. Beylkin and L. Monzón, "Approximation by exponential sums revisited," *Applied and Computational Harmonic Analysis* (2005).

[15] J. Bradbury et al., "JAX: composable transformations of Python+NumPy programs," (2018). https://github.com/jax-ml/jax