Contents

CubeFit: Disk-backed spectral HyperCube + global Kaczmarz fit

One-liner: Build a disk-backed spectral HyperCube `models[s,c,p,l]` and run a global Kaczmarz NNLS fit to recover one non-negative weight vector `x[C*P]` that explains every spaxel spectrum as a mixture of templates convolved with per-spaxel LOSVDs, rebinned to the observed grid.

TL;DR quick start

```
from hdf5_manager import H5Manager as H5M
from hypercube_builder import build_hypercube
from pipeline_runner import PipelineRunner

# 1) Populate HDF5 (single writer)
mgr = H5M("cube.h5", tem_pix=teLL, obs_pix=spLL)
mgr.set_velocity_grid(vbins)  # /VelPix
mgr.populate_from_arrays(
    losvd=nApHists,           # (S,V,C)
    datacube=laGrid,          # (L,S) or (S,L) accepted; stored as (S,L)
    templates=lnGrid,         # (T, nZ, nAge, nAlpha) → (P,T)
    mask=spmask,              # optional, length L
    xpix=X_pix, ypix=Y_pix,   # optional per-pixel coords (nPix,)
    binnum=bin_idx,           # optional pixel→bin map (nPix,) in [0..S-1]
)
mgr.ensure_rebin_and_resample()  # builds /R_T, /TemplatesFFT(_R), etc.

# 2) Build HyperCube (single writer; uncompressed for speed)
build_hypercube("cube.h5", S_chunk=128, C_chunk=1, P_chunk=360,
                n_workers=<~cores>, compression=None)

# 3) Live prefit snapshot (read-only; SWMR-safe)
H5M.live_prefit_snapshot_from_models("cube.h5",
                                     out_png="prefit_overlay.png")
```

```python
# 4) Global Kaczmarz fit (RAM-bounded, single process + BLAS threads)
runner = PipelineRunner("cube.h5")
x, stats = runner.solve_all(
    epochs=E, pixels_per_aperture=K, lr= ,
    project_nonneg=True,
    orbit_weights=cWeights,   # per-component; used as ratio targets
    blas_threads=8,           # keep modest (memory-bandwidth bound)
    verbose=True,
)

# 5) (Optional) Reconstruct model cube ŷ(s, :) = A(s)^T x
from kz_fitSpec import reconstruct_model_cube_single
reconstruct_model_cube_single("cube.h5", x, blas_threads=8)
```

Canonical on-disk layout (HDF5)

```
/DataCube           (S, L) float64   # observed spectra
/Templates          (P, T) float32   # flattened from (T, nZ, nAge, nAlpha)
/LOSVD              (S, V, C) float32
/TemPix             (T,)   float64   # log- template grid
/ObsPix             (L,)   float64   # log- observed grid
/VelPix             (V,)   float64   # km/s LOSVD bin centers
/R_T                (T, L) float32   # dense rebin operator (templates→observed)
/RebinMatrix        (L, T) float32   # transpose cache (if emitted)
/TemplatesFFT       (P, T//2+1) complex64
/TemplatesFFT_R     (P, T//2+1) complex64
/Mask               (L,)   bool      # optional

/HyperCube/models   (S, C, P, L) float32   # chunked; convolved & rebinned
/HyperCube/_done    (Sgrid, Cgrid, Pgrid) bool/int   # tile bitmap

# NEW per-pixel metadata (optional)
# image-plane coordinates and mapping to spectral bins
/XPix               (nPix,) float64
/YPix               (nPix,) float64
/BinNum             (nPix,) int32   # values in [0..S-1]

# Fitting
/X_global           (C*P,) float64  # latest committed global solution
/Fit/
  x_latest          (C*P,) float64  # live checkpoint
  x_best            (C*P,) float64  # best-so-far by validation RMSE
  x_ring            (R, C*P) float64  # ring buffer (R=cfg.ring_size)
  x_vis_ring        (R, C*P) float32  # optional for dashboards
  metrics/
    epoch, time_sec, train_rmse_ewma, val_rmse, delta_x_rel,
    nnz, l1_norm, l2_norm, mass_err_L1, mass_err_Linf
  sample/
    spaxels         (Ns,)  int32
    val_pixels      (Kval,) int32
```

Root attrs:

- `dims` — JSON with `{nSpat,nLSpec,nTSpec,nVel,nComp,nPop}`
- `template_meta_json` — original template axis sizes/order
- `guard.*`, `guard_info_json` — convolution padding/coverage info

Golden rules (to avoid old pitfalls)

- **Convolve on template grid, then rebin.** Edge effects governed by template coverage vs. max LOSVD width on log- ; guard margins tracked in `guard_info`.
- **Single writer; many readers.** Builder is sole writer; diagnostics/solver read with `swmr=True` where appropriate. Live plotting/snapshots are SWMR-safe.
- **Float32 on disk; float64 in math.** Store models as f32; compute in f64.
- **Dense `R_T` is (T, L).** If you see (L, T), transpose before use.

Boot sequence

1. **Populate HDF5**

```
mgr = H5M(h5_path, tem_pix=teLL, obs_pix=spLL)
mgr.set_velocity_grid(vbins)  # /VelPix

# NEW: optional per-pixel metadata (for imaging overlays, mapping, etc.)
mgr.populate_from_arrays(
    losvd=nApHists, datacube=laGrid, templates=lnGrid, mask=spmask,
    xpix=X_pix, ypix=Y_pix, binnum=bin_idx
)

mgr.ensure_rebin_and_resample()
```

2. **Build HyperCube** (single writer)

```
build_hypercube(
    h5_path,
    S_chunk=128, C_chunk=1, P_chunk=360,  # compute & storage tiles
    n_workers=<~cores>,
    compression=None,                     # fastest; compress after analysis
)
```

3. **Sanity snapshot (SWMR, non-blocking)**

```
H5M.live_prefit_snapshot_from_models(h5_path,
                                     out_png="prefit_live.png")
```

4. **Global Kaczmarz fit (RAM-bounded)**

```
runner = PipelineRunner(h5_path)
x, stats = runner.solve_all(
    epochs=E, pixels_per_aperture=K, lr= ,
    project_nonneg=True, orbit_weights=cWeights,
    blas_threads=8, verbose=True,
)
```

5. **(Optional) Compress**

```
H5M.compress_hypercube_inplace(h5_path, codec="gzip",
                               level=2, shuffle=True)
# or repack to a new file
```

Live diagnostics

- **SWMR snapshot:** `H5M.live_prefit_snapshot_from_models(...)`
  - TL: observed spectra (mask shaded; unmasked drives y-limits)
  - TR: templates rebinned (pre-conv)
  - BL: LOSVD + kernel sampled on log- offsets
  - BR: stored models for several diverse (s,c) pairs
- **Tile status**

```
H5M.print_hypercube_done_status(h5_path)
# prints shape, chunks, N_done / N_total tiles
```

Global Kaczmarz (NNLS) — what actually runs

Data updates: for each selected pixel in a spaxel, promote the column `A[:, ]` to f64 and do a standard Kaczmarz step on `x` (f64). No per-column prior scaling.

Scale-invariant ratio penalty (optional): let `s_c = Σ_p x_{c,p}`. With anchor `i` (usually the largest prior weight), stochastically enforce

```
s_c − (w_c / w_i) * s_i  0
```

with small step size and low frequency. This nudges per-component **ratios** to match `orbit_weights` (length C) while allowing global flux to float.

Key cfg knobs (with defaults):

- `epochs`: 1
- `pixels_per_aperture`: 256 (K rows per spaxel per epoch)
- `lr`: 0.25
- `project_nonneg`: True
- `row_order`: ”random” | ”sequential”
- `blas_threads`: e.g., 4–8 (memory-bandwidth bound)
- **Ratios:**
  `ratio_use=None` (auto on if weights provided), `ratio_prob=0.02`,
  `ratio_eta=None` (defaults to `0.1*lr`), `ratio_anchor="auto"`,
  `ratio_min_weight` interpreted **relative to max(w)**, `ratio_batch=2`.

Warm-start, resume, and smart seeding

- **Resume priority (automatic):** `/Fit/x_best` → `/X_global` → `x0` (user-supplied). If nothing to resume and no `x0`, a fast **streaming Jacobi** initializer builds a nonnegative seed:

```
d += a a;  b += a*y        over a small Ns×K subset
x0 = clip(b / (d +  ), 0)
```

- **Still supports direct warm-start:** pass `x0` to `solve_all(...)` to override resume/seed logic.

Fit tracking & live dashboard (NEW architecture)

The previous single-process tracker caused HDF5 mode/SWMR conflicts. The tracker is now **asynchronous**:

- A dedicated **writer process** owns the only append handle to the HDF5.
- The solver/main process never opens the file for writing; it only sends small messages (RMSE EWMA, periodic checkpoints).
- Compatible with `live_fit_dashboard` reading the same `/Fit/*` paths.

Public API (unchanged):

```
from fit_tracker import FitTracker, TrackerConfig

tracker = FitTracker(h5_path, cfg=TrackerConfig(
    metrics_interval_sec=300, val_interval_sec=3600, ckpt_interval_sec=1800,
    Ns=16, Kval=32, diag_seed=12345, ring_size=96
))
tracker.set_orbit_weights(cWeights)   # optional; improves mass metrics

# During solve:
tracker.on_batch(rmse)                # once per spaxel (EWMA)
tracker.maybe_save(x, epoch=ep)       # gated; sends metrics/ckpt/val
...
tracker.close()                       # optional; auto on exit
```

What it writes:

- `/Fit/x_latest`, `/Fit/x_best` (+ ring buffers for time series)
- `/Fit/metrics/*` appended rows
- `/Fit/sample/*` fixed validation sample (Ns spaxels × Kval pixels)

Reconstructing the model cube $\hat{y}$ = A x

Fast, memory-bounded, single process + BLAS threads:

```
from kz_fitSpec import reconstruct_model_cube_single

reconstruct_model_cube_single(
    h5_path="cube.h5", x_global=x,
    array_name="ModelCube",   # writes (S,L) float64
    blas_threads=8,
    S_tile=None,              # default: align to HDF5 S_chunk
    max_bytes_per_slab=6_000_000_000,  # ~6 GiB budget
)
```

What it does:

- Reads slabs `(dS, C, Pb, L)` in **float32**.
- Performs `einsum('dcpl,cp->dl', slab, x_cp[:,p0:p1], out=tmp[:dS,:])`.
- Accumulates into `Y_tile[:dS, :]` and writes `out[s0:s1, :]`.
- Avoids the **last-tile broadcast bug** by always using `(dS, :)` views.
- Uses `np.float64` accumulation for numerical stability; no big f64 copy of the slab is made (promotion happens through `out=`).

Skip recompute safely: if you stamp a digest/attrs on `ModelCube`, you can detect it and skip. Minimal check:

```
with open_h5(h5, "reader") as f:
    if "/ModelCube" in f and f["/ModelCube"].shape == (S, L):
        print("ModelCube present - skipping reconstruction.")
    else:
        reconstruct_model_cube_single(h5, x)
```

Performance knobs & resource guidance

Hypercube build: `S_chunk`, `P_chunk`, `n_workers`

- Start `S_chunk` 128, `P_chunk` 256-360, `C_chunk = 1`.
- Uncompressed models are fastest; compress at the end.

Solver throughput / memory:

- Prefer **moderate BLAS threads** (e.g., 4–8) over 48; the solver is memory-bandwidth bound.
- Use `pixels_per_aperture` (K) to cap per-spaxel rows (e.g., 128–384).
- On clusters, export:

```
export OMP_NUM_THREADS=8
export MKL_NUM_THREADS=$OMP_NUM_THREADS
export OPENBLAS_NUM_THREADS=$OMP_NUM_THREADS
export NUMEXPR_MAX_THREADS=$OMP_NUM_THREADS
```

Reconstruction (`A x`):

- Single process + BLAS threads avoids HDF5 lock contention and OS page-cache explosions seen with many processes.
- `max_bytes_per_slab` and `S_tile`/`P_block` auto-tune to keep peak RSS flat; a 6 GiB budget with `blas_threads 8` typically peaks under ~60 GiB.

Troubleshooting

- **"Missing /TemplatesFFT; call populate_from_arrays first."**
  You're on the lazy-FFT path. Run `mgr.ensure_rebin_and_resample()`.
- **"Object dtype has no native HDF5 equivalent."**
  Don't write dicts to attrs. Use `H5M._write_guard_attrs` (JSON + scalars).
- **RuntimeError: Missing /HyperCube/models after build.**
  Verify builder completed and `_done` shows 100%; re-open with `swmr=True`.
- **Convolved models look wrong / negative.**
  Ensure `conv → rebin` (not vice-versa); `R_T` orientation is `(T,L)`. Check `H5M.diagnose_rebin_edges(...)` → `Safe=True`. Increase template padding if `Safe=False`.
- **Row L mismatch (e.g., 680 vs 724).**
  Masked effective length vs. full L. Reader and solver must apply the same mask consistently.
- **HDF5 open errors during training (locks/SWMR).**
  The new `FitTracker` isolates all writes in a separate process. Don't open the base file for writing in the solver process.
- **NumPy 2.0 string attrs.**
  Use bytes (`b"..."` or `np.bytes_("...")`) instead of `np.string_`.

- **Illegal instruction on compute nodes.**
  If you see `SIGILL` from OpenBLAS/MKL, constrain the CPU target: `OPENBLAS_CORETYPE=NEHALEM`, `MKL_DEBUG_CPU_TYPE=5`.

Minimal REPL checks

```
import h5py, numpy as np
with h5py.File(h5_path, "r", swmr=True) as f:
    M = f["/HyperCube/models"]; S,C,P,L = M.shape
    print("models", M.shape, "chunks", M.chunks)
    done = np.asarray(f["/HyperCube/_done"], bool)
    print("tiles done:", done.sum(), "/", done.size)
    for s,c in [(0,0), (S//3, C//4), (S-1, C-1)]:
        slab = np.asarray(M[s,c,:,:], np.float32)  # (P,L)
        print((s,c), "max|model|=", float(np.max(np.abs(slab))))
```

API notes & changes (recent)

- `populate_from_arrays(...)` now optionally accepts:
  - `xpix`, `ypix` — per-pixel image-plane coordinates → `/XPix`,`/YPix`
  - `binnum` — pixel→bin map (ints in `[0..S-1]`) → `/BinNum`
- **Kaczmarz solver:**
  - Keeps `x0` warm-start; resume priority is handled in the runner.
  - Ratio constraints use **per-component** `orbit_weights` (length C) and operate on `s_c = Σ_p x_{c,p}`; ratios do **not** normalize `w`.
  - Added hooks used by the tracker (`on_epoch_end`, optional batch RMSE).
- **FitTracker:** fully reworked as a separate **writer process**; same public API; writes `/Fit/*` in the base HDF5; NumPy-2.0 clean.
- **Reconstruction:** `reconstruct_model_cube_single(...)` now uses `(dS,:)` views for `out=` and accumulation (fixes last-tile broadcast), promotes f32→f64 via `out=` (no large f64 slab copy).

Edge-effects checklist

- `H5M.diagnose_rebin_edges(h5_path) → Safe? True`
- Template span with guard: `tem_lo/tem_hi` beyond `obs_lo/obs_hi` by `Kguard_px * dlog`.
- LOSVD max velocity consistent with kernel support used in builder. If not, increase `safety_pad_px` during `populate_from_arrays`.

File roles (where to look)

- `hdf5_manager.py` — all I/O, grids, rebin, FFT caches, guard/diagnostics, compression helpers, SWMR-safe live plotting. Now writes `/XPix`,`/YPix`, `/BinNum` when provided.
- `hypercube_builder.py` — writes `/HyperCube/models + _done` (single writer).
- `hypercube_reader.py` — streams spaxel slabs; mask-aware; returns f32→f64 views to solver.
- `kaczmarz_solver.py` — global Kaczmarz loop; ratio rows; projects to non-neg if requested; callback hooks for tracking.
- `fit_tracker.py` — **async** writer-process tracker; compatible with live dashboard.
- `pipeline_runner.py` — glues reader solver; resume priority; smart seed; tracker wiring; exposes `solve_all`.

- `kz_fitSpec.py` — end-to-end script; includes prefit diagnostics and `reconstruct_model_cube_single(...)`.

License

(Insert your license.)

Happy fitting. Keep the writer single, the readers many, the math float64, and the page cache tame.