膜宇宙論 条件14/eta検証 解析スクリプト全文カタログ

2026年4月11日午後セッション / 坂口 忍
スクリプト 6 本、総行数 3,585 行

目次・スクリプト一覧

#	ファイル名	目的	結果	行数
1	sparc_kappa_estimation.py	kappa=const 推定	X級（否定）	723
2	sparc_kappa_nonlinear.py	非線形8モデル系統テスト	M3有意（B-級）	743
3	sparc_kappa_circularity.py	循環参照5段階検証	真の信号確認	511
4	sparc_M3_M6_independence.py	M3/M6独立性7段階	M6冗長	607
5	epsilon_c_derivation.py	epsilon_c理論導出	X級（不成功）	531
6	sparc_eta_derivation.py	eta第一原理導出	部分的成功	470

検証パイプラインの流れ: Script 1（kappa=const否定）-> Script 2（非線形M3有意）-> Script 3（循環参照テスト通過）-> Script 4（M6冗長判明）-> Script 5（epsilon_c導出不成功）-> Script 6（eta導出部分的成功）

実行環境: ローカル Windows + Claude Code（VS Code拡張）。コマンド: uv run --with scipy --with matplotlib python [script]。データ: SPARC sparc_gc.csv + Rotmod_LTG/（175銀河の回転曲線）。Script 5 のみデータ不要（純粋理論計算）。

Script 1: sparc_kappa_estimation.py

目的

kappa=const（線形膜剛性）モデルの推定

膜ラグランジアン L = U(epsilon;c) + kappa(d epsilon/dr)^2 の kappa を定数として推定する。gc = gc_deep x (1 + kappa x E_grad/E_local) を log 空間で最小二乗フィットし、bootstrap CI (N=2000)、普遍性検証（kappa_i vs 銀河パラメータ）、hR 偏相関改善、循環参照チェック（シャッフルテスト）を実施。

結果

X級（否定）: kappa x median(ratio) = -0.006 (0.6%補正)、dAIC = +1.1 (悪化)、kappa_i vs log(hR) で rho=-0.474（普遍定数でない）。線形パラメータ化は膜の非線形応答を捉えられない。

実行方法

```
uv run --with scipy --with matplotlib python sparc_kappa_estimation.py
```

入力データ: sparc_gc.csv + Rotmod_LTG/*.dat
出力: kappa_estimation_results.png + 標準出力

ソースコード全文

```
#!/usr/bin/env python3
"""
sparc_kappa_estimation.py
=========================
□□□□□ κ □□□

□□□: gc = gc_deep × (1 + κ × E_grad/E_local)
  - gc_deep = deep-MOND limit gc (x&lt;1 □□□□□□)
  - ε(r) = sqrt(gN(r) / gc_deep)
  - E_grad = mean[(dε/dr)²]  (□□□□□radial bin□□)
  - E_local = mean[ε²]
  - ratio = E_grad / E_local

□□□□:
  (1) κ □□□□□ + bootstrap CI
  (2) κ □□□□□□□□□□ (□□ vs □□□□□□□)
  (3) κ □□□□□□ (□ bending rigidity)
  (4) gc_corrected □□ hR □□□□□□□□
  (5) □□13%□□□□□□

□□: uv run --with scipy --with matplotlib python sparc_kappa_estimation.py
"""

import os, sys, glob
import numpy as np
from scipy.optimize import minimize_scalar, minimize
from scipy.stats import spearmanr, pearsonr
import warnings
warnings.filterwarnings('ignore')

# ============================================================
# 0. □□□□
# ============================================================
BASE = r"D:\□□□□□□\□□□□□□\□□□□□\□□□□□□\□□□□"
ROTMOD = os.path.join(BASE, "Rotmod_LTG")
GC_CSV = os.path.join(BASE, "sparc_gc.csv")

a0 = 1.2e-10  # m/s^2, MOND□□□□□□□

# ============================================================
# 1. sparc_gc.csv □□□□
# ============================================================
def load_gc_csv():
    """sparc_gc.csv □□□□□□gc□□□□ (m/s^2□□□□□)"""
    import csv
    data = {}
    with open(GC_CSV, 'r') as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = row.get('galaxy') or row.get('Galaxy') or row.get('name')
            if name is None:
                # □□□□□□□□
                for k in row:
                    if 'gal' in k.lower() or 'name' in k.lower():
                        name = row[k]
                        break
            if name is None:
                continue
            name = name.strip()
```

```
            # gc□
            gc_val = None
            for k in ['gc', 'gc_obs', 'g_c', 'gc_a0']:
                if k in row and row[k]:
                    try:
                        gc_val = float(row[k])
                        break
                    except:
                        pass
            if gc_val is None:
                continue

            # vflat, hR
            vflat = None
            for k in ['vflat', 'Vflat', 'v_flat']:
                if k in row and row[k]:
                    try:
                        vflat = float(row[k])
                        break
                    except:
                        pass
            hR = None
            for k in ['hR', 'Reff', 'h_R', 'hr']:
                if k in row and row[k]:
                    try:
                        hR = float(row[k])
                        break
                    except:
                        pass
            Yd = None
            for k in ['Yd', 'yd', 'Y_d', 'SPS']:
                if k in row and row[k]:
                    try:
                        Yd = float(row[k])
                        break
                    except:
                        pass

            data[name] = {'gc': gc_val, 'vflat': vflat, 'hR': hR, 'Yd': Yd}

    # gc □ a0□□□m/s^2□□□□□□□
    gc_vals = [d['gc'] for d in data.values()]
    median_gc = np.median(gc_vals)
    if median_gc &lt; 1e-5:
        # □□ m/s^2 □□
        gc_unit = 'mps2'
    else:
        # a0 □□ → m/s^2 □□□
        gc_unit = 'a0'
        for name in data:
            data[name]['gc'] *= a0

    print(f"sparc_gc.csv: {len(data)} galaxies loaded (gc unit detected: {gc_unit})")
    return data

# ============================================================
# 2. Rotmod_LTG □□□□□□□□□
# ============================================================
def load_rotcurve(galaxy_name):
    """
    Rotmod_LTG/&lt;galaxy&gt;_rotmod.dat □□□□□
    □: Rad(kpc), Vobs, errV, Vgas, Vdisk, Vbul, SBdisk, SBbul
    """
    fname = os.path.join(ROTMOD, f"{galaxy_name}_rotmod.dat")
    if not os.path.exists(fname):
        return None

    rad, vobs, vgas, vdisk, vbul = [], [], [], [], []
    with open(fname, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith('#'):
                continue
            parts = line.split()
            if len(parts) &lt; 6:
                continue
            try:
                r = float(parts[0])
                vo = float(parts[1])
                vg = float(parts[3])
                vd = float(parts[4])
                vb = float(parts[5])
                rad.append(r)
                vobs.append(vo)
                vgas.append(vg)
                vdisk.append(vd)
                vbul.append(vb)
            except:
                continue

    if len(rad) &lt; 5:
```

```
        return None

    return {
        'r': np.array(rad),       # kpc
        'vobs': np.array(vobs),   # km/s
        'vgas': np.array(vgas),   # km/s
        'vdisk': np.array(vdisk), # km/s (sqrt(Yd) ▫▫▫)
        'vbul': np.array(vbul),   # km/s
    }

# ============================================================
# 3. gN ▫▫ + ε(r) + E_grad/E_local
# ============================================================
def compute_strain_quantities(rc, gc_deep, Yd=0.5):
    """
    ▫▫▫▫▫▫▫:
      gN(r) = (Yd*Vdisk² + Vgas² + Vbul²) / r   (▫▫▫ in m/s²)
      ε(r) = sqrt(gN(r) / gc_deep)
      E_grad = mean[(dε/dr)²]
      E_local = mean[ε²]
      ratio = E_grad / E_local
    """
    r_kpc = rc['r']
    r_m = r_kpc * 3.086e19  # kpc -&gt; m

    # gN = V_bar² / r (centripetal acceleration from baryons)
    v_bar2 = Yd * rc['vdisk']**2 + rc['vgas']**2 + rc['vbul']**2  # (km/s)²
    v_bar2_mps2 = v_bar2 * 1e6  # -&gt; (m/s)²

    # gN in m/s²
    gN = v_bar2_mps2 / r_m

    # gc_deep ▫ 0 ▫▫▫▫▫▫▫▫▫▫
    if gc_deep &lt;= 0:
        return None

    # ε(r) = sqrt(gN / gc_deep), gN &lt; 0 ▫▫▫▫▫
    gN_pos = np.maximum(gN, 0)
    ratio_gN = gN_pos / gc_deep
    epsilon = np.sqrt(ratio_gN)

    # ε &gt; 1 ▫▫▫▫▫▫ Newtonian ▫▫ → ▫▫▫▫▫▫▫▫▫▫▫
    # ▫▫▫ deep-MOND ▫▫▫▫▫▫ ε &gt;&gt; 1 ▫▫▫▫

    # dε/dr (finite difference)
    if len(r_m) &lt; 3:
        return None

    # central difference (▫▫ forward/backward)
    deps_dr = np.zeros_like(epsilon)
    deps_dr[0] = (epsilon[1] - epsilon[0]) / (r_m[1] - r_m[0])
    deps_dr[-1] = (epsilon[-1] - epsilon[-2]) / (r_m[-1] - r_m[-2])
    for i in range(1, len(epsilon)-1):
        deps_dr[i] = (epsilon[i+1] - epsilon[i-1]) / (r_m[i+1] - r_m[i-1])

    E_grad = np.mean(deps_dr**2)
    E_local = np.mean(epsilon**2)

    if E_local == 0:
        return None

    return {
        'r_kpc': r_kpc,
        'gN': gN,
        'epsilon': epsilon,
        'deps_dr': deps_dr,
        'E_grad': E_grad,
        'E_local': E_local,
        'ratio': E_grad / E_local,
        'mean_epsilon': np.mean(epsilon),
        'max_epsilon': np.max(epsilon),
    }

# ============================================================
# 4. gc_deep ▫▫▫ (x&lt;1 ▫▫▫▫MOND▫▫)
# ============================================================
def compute_gc_deep(rc, Yd=0.5):
    """
    Deep-MOND limit: gobs = sqrt(gN * gc)
    → gc = gobs² / gN  (▫▫)
    x = gN/gc &lt; 1 ▫▫▫▫▫▫▫▫▫
    → median gc_deep
    """
    r_m = rc['r'] * 3.086e19
    v_bar2 = (Yd * rc['vdisk']**2 + rc['vgas']**2 + rc['vbul']**2) * 1e6
    v_obs2 = (rc['vobs'] * 1e3)**2

    gN = v_bar2 / r_m
    gobs = v_obs2 / r_m

    # gc_point = gobs² / gN (deep MOND)
    mask = gN &gt; 0
```

```
    gc_points = gobs[mask]**2 / gN[mask]

    if len(gc_points) &lt; 3:
        return None

    # x = gN / gc &lt; 1 □□□□ (iterative: □□□ median gc □□□)
    gc_med = np.median(gc_points)
    if gc_med &lt;= 0:
        return None

    x_vals = gN[mask] / gc_med
    deep_mask = x_vals &lt; 1.0

    if np.sum(deep_mask) &lt; 3:
        # x&lt;1 □□□□□ → □□ median □□□
        return gc_med

    return np.median(gc_points[deep_mask])

# ============================================================
# 5. □□□□□
# ============================================================
def main():
    print("=" * 70)
    print("K□□□□□□□□□")
    print("=" * 70)

    # --- □□□□□□□ ---
    gc_data = load_gc_csv()

    # --- □□□□□ ---
    results = []

    galaxies = sorted(gc_data.keys())
    n_loaded = 0
    n_skipped = 0

    for gname in galaxies:
        gd = gc_data[gname]
        gc_obs = gd['gc']  # m/s^2
        vflat = gd.get('vflat')
        hR = gd.get('hR')
        Yd = gd.get('Yd') or 0.5

        if gc_obs &lt;= 0 or vflat is None or hR is None:
            n_skipped += 1
            continue
        if vflat &lt;= 0 or hR &lt;= 0:
            n_skipped += 1
            continue

        rc = load_rotcurve(gname)
        if rc is None:
            n_skipped += 1
            continue

        # gc_deep
        gc_deep = compute_gc_deep(rc, Yd=Yd)
        if gc_deep is None or gc_deep &lt;= 0:
            n_skipped += 1
            continue

        # □□□
        sq = compute_strain_quantities(rc, gc_deep, Yd=Yd)
        if sq is None:
            n_skipped += 1
            continue

        results.append({
            'name': gname,
            'gc_obs': gc_obs,
            'gc_deep': gc_deep,
            'vflat': vflat,
            'hR': hR,
            'Yd': Yd,
            'ratio': sq['ratio'],
            'E_grad': sq['E_grad'],
            'E_local': sq['E_local'],
            'mean_eps': sq['mean_epsilon'],
            'n_points': len(rc['r']),
            'rmax_hR': rc['r'][-1] / hR if hR &gt; 0 else 0,
        })
        n_loaded += 1

    print(f"\nLoaded: {n_loaded}, Skipped: {n_skipped}")

    if n_loaded &lt; 20:
        print("ERROR: □□□□□□□□□□□□□□□□□□□□□□□□")
        sys.exit(1)

    # --- numpy□□□ ---
    gc_obs = np.array([r['gc_obs'] for r in results])
```

```
    gc_deep = np.array([r['gc_deep'] for r in results])
    ratio = np.array([r['ratio'] for r in results])
    vflat = np.array([r['vflat'] for r in results])
    hR = np.array([r['hR'] for r in results])
    Yd = np.array([r['Yd'] for r in results])
    rmax_hR = np.array([r['rmax_hR'] for r in results])
    names = [r['name'] for r in results]

    # ============================================================
    # (1) κ □□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("(1) κ □□□□□")
    print("=" * 70)

    # □□□: gc_obs = gc_deep * (1 + κ * ratio)
    # log□□: log(gc_obs) = log(gc_deep) + log(1 + κ * ratio)
    # → □□□□: minimize Σ[log(gc_obs/gc_deep) - log(1 + κ*ratio)]²

    log_excess = np.log10(gc_obs / gc_deep)

    # □□ gc_obs/gc_deep □□□□□
    med_excess = np.median(gc_obs / gc_deep)
    print(f"gc_obs/gc_deep: median={med_excess:.3f}, "
          f"mean={np.mean(gc_obs/gc_deep):.3f}, "
          f"std(log)={np.std(log_excess):.3f} dex")
    print(f"ratio (E_grad/E_local): median={np.median(ratio):.4e}, "
          f"range=[{np.min(ratio):.4e}, {np.max(ratio):.4e}]")

    def chi2_kappa(kappa):
        model = np.log10(1 + kappa * ratio)
        return np.sum((log_excess - model)**2)

    # grid search for initial value
    kappa_grid = np.linspace(-1e20, 1e20, 1000)
    # ratio □□□□□□□□□□□□□□□□□□□□
    ratio_scale = np.median(ratio)
    if ratio_scale &gt; 0:
        kappa_max = 10.0 / ratio_scale
    else:
        kappa_max = 1e20

    kappa_grid = np.linspace(-kappa_max, kappa_max, 2000)
    chi2_vals = [chi2_kappa(k) for k in kappa_grid]
    k_init = kappa_grid[np.argmin(chi2_vals)]

    # optimize
    from scipy.optimize import minimize_scalar
    res = minimize_scalar(chi2_kappa, bounds=(-kappa_max*2, kappa_max*2), method='bounded')
    kappa_best = res.x
    chi2_best = res.fun

    # null model (κ=0)
    chi2_null = np.sum(log_excess**2)

    N = len(gc_obs)
    dof = N - 1
    dAIC = chi2_best - chi2_null + 2  # +1 parameter

    print(f"\nκ_best = {kappa_best:.6e}")
    print(f"ratio_scale (median) = {ratio_scale:.6e}")
    print(f"κ * median(ratio) = {kappa_best * ratio_scale:.4f}")
    print(f"chi2(κ=0) = {chi2_null:.3f}, chi2(κ_best) = {chi2_best:.3f}")
    print(f"dAIC = {dAIC:.1f} (negative = κ□□□□□)")

    # R² improvement
    SS_tot = chi2_null
    SS_res = chi2_best
    R2 = 1 - SS_res / SS_tot
    print(f"R²(κ□□) = {R2:.4f}")

    # gc_corrected
    gc_corrected = gc_deep * (1 + kappa_best * ratio)
    residual_corrected = np.log10(gc_obs / gc_corrected)
    residual_uncorrected = log_excess
    print(f"residual std: uncorrected={np.std(residual_uncorrected):.4f} dex, "
          f"corrected={np.std(residual_corrected):.4f} dex")

    # ============================================================
    # (1b) Bootstrap CI for κ
    # ============================================================
    print("\n--- Bootstrap CI (N=2000) ---")
    np.random.seed(42)
    kappa_boot = []
    for _ in range(2000):
        idx = np.random.randint(0, N, N)
        le = log_excess[idx]
        ra = ratio[idx]
        def chi2_b(k):
            return np.sum((le - np.log10(np.maximum(1 + k * ra, 1e-10)))**2)
        rb = minimize_scalar(chi2_b, bounds=(-kappa_max*2, kappa_max*2), method='bounded')
        kappa_boot.append(rb.x)
```

```
428 |
429 |     kappa_boot = np.array(kappa_boot)
430 |     ci_lo, ci_hi = np.percentile(kappa_boot, [2.5, 97.5])
431 |     print(f"κ = {kappa_best:.4e}, 95%CI = [{ci_lo:.4e}, {ci_hi:.4e}]")
432 |     print(f"CI includes 0? {'YES' if ci_lo &lt;= 0 &lt;= ci_hi else 'NO'}")
433 |
434 |     # ============================================================
435 |     # (2) κ □□□□□□□□□□
436 |     # ============================================================
437 |     print("\n" + "=" * 70)
438 |     print("(2) κ □□□□□□")
439 |     print("=" * 70)
440 |
441 |     # □□□□□ κ_i = (gc_obs/gc_deep - 1) / ratio
442 |     mask_ratio = ratio &gt; 0
443 |     kappa_individual = np.full(N, np.nan)
444 |     kappa_individual[mask_ratio] = (gc_obs[mask_ratio] / gc_deep[mask_ratio] - 1) / ratio[mask_ratio]
445 |
446 |     valid = ~np.isnan(kappa_individual) &amp;amp; np.isfinite(kappa_individual)
447 |     # outlier clip (|κ_i| &gt; 100 * median)
448 |     ki_valid = kappa_individual[valid]
449 |     med_ki = np.median(ki_valid)
450 |     mad_ki = np.median(np.abs(ki_valid - med_ki))
451 |     clip = np.abs(ki_valid - med_ki) &lt; 10 * (mad_ki + 1e-30)
452 |     ki_clipped = ki_valid[clip]
453 |
454 |     print(f"□□κ_i: N={len(ki_clipped)}, median={np.median(ki_clipped):.4e}, "
455 |           f"MAD={mad_ki:.4e}, CV={mad_ki/(abs(med_ki)+1e-30):.2f}")
456 |
457 |     # κ_i vs galaxy parameters
458 |     # (ratio &gt; 0 □□ clip □□□□□□□□□□)
459 |     idx_valid = np.where(valid)[0]
460 |     idx_valid = idx_valid[clip]
461 |
462 |     params_test = {
463 |         'log(vflat)': np.log10(vflat[idx_valid]),
464 |         'log(hR)': np.log10(hR[idx_valid]),
465 |         'log(Yd)': np.log10(np.maximum(Yd[idx_valid], 0.01)),
466 |         'rmax/hR': rmax_hR[idx_valid],
467 |         'log(gc_deep)': np.log10(gc_deep[idx_valid]),
468 |     }
469 |
470 |     print("\nκ_i vs □□□□□□□ (Spearman):")
471 |     for pname, pvals in params_test.items():
472 |         finite = np.isfinite(pvals) &amp;amp; np.isfinite(ki_clipped)
473 |         if np.sum(finite) &lt; 10:
474 |             continue
475 |         rho, p = spearmanr(pvals[finite], ki_clipped[finite])
476 |         sig = "***" if p &lt; 0.001 else "**" if p &lt; 0.01 else "*" if p &lt; 0.05 else ""
477 |         print(f"  {pname:15s}: rho={rho:+.3f}, p={p:.4f} {sig}")
478 |
479 |     # ============================================================
480 |     # (2b) Two-parameter model: κ = κ0 + κ1 * log(param)
481 |     # ============================================================
482 |     print("\n--- κ □□□□□□□ ---")
483 |
484 |     for dep_name, dep_vals in [('log_vflat', np.log10(vflat)),
485 |                                ('log_hR', np.log10(hR)),
486 |                                ('log_Yd', np.log10(np.maximum(Yd, 0.01)))]:
487 |         def chi2_2p(params):
488 |             k0, k1 = params
489 |             kappa_var = k0 + k1 * dep_vals
490 |             model = np.log10(np.maximum(1 + kappa_var * ratio, 1e-10))
491 |             return np.sum((log_excess - model)**2)
492 |
493 |         r2p = minimize(chi2_2p, [kappa_best, 0], method='Nelder-Mead')
494 |         dAIC_2p = r2p.fun - chi2_best + 2  # +1 extra param
495 |         print(f"  κ = κ0 + κ1*{dep_name}: κ0={r2p.x[0]:.4e}, κ1={r2p.x[1]:.4e}, "
496 |               f"dAIC vs const-κ = {dAIC_2p:+.1f}")
497 |
498 |     # ============================================================
499 |     # (3) hR □□□□□□□□
500 |     # ============================================================
501 |     print("\n" + "=" * 70)
502 |     print("(3) hR □□□□□□")
503 |     print("=" * 70)
504 |
505 |     log_gc_obs = np.log10(gc_obs)
506 |     log_gc_deep = np.log10(gc_deep)
507 |     log_gc_corr = np.log10(np.maximum(gc_corrected, 1e-30))
508 |     log_vflat = np.log10(vflat)
509 |     log_hR = np.log10(hR)
510 |
511 |     # □□□: ρ(gc, hR | vflat)
512 |     from numpy.polynomial.polynomial import polyfit
513 |
514 |     def partial_corr(x, y, z):
515 |         """ρ(x,y|z) by residualization"""
516 |         cx = np.polyfit(z, x, 1)
517 |         cy = np.polyfit(z, y, 1)
518 |         rx = x - np.polyval(cx, z)
519 |         ry = y - np.polyval(cy, z)
```

```
        return spearmanr(rx, ry)

    rho_obs, p_obs = partial_corr(log_gc_obs, log_hR, log_vflat)
    rho_deep, p_deep = partial_corr(log_gc_deep, log_hR, log_vflat)
    rho_corr, p_corr = partial_corr(log_gc_corr, log_hR, log_vflat)

    print(f"ρ(gc_obs, hR | vflat)       = {rho_obs:+.3f} (p={p_obs:.4f})")
    print(f"ρ(gc_deep, hR | vflat)      = {rho_deep:+.3f} (p={p_deep:.4f})")
    print(f"ρ(gc_corrected, hR | vflat) = {rho_corr:+.3f} (p={p_corr:.4f})")
    print(f"□□: |ρ| {abs(rho_obs):.3f} → {abs(rho_corr):.3f} "
          f"({(1-abs(rho_corr)/abs(rho_obs))*100:.1f}% reduction)")

    # x_outer □□□
    log_ratio = np.log10(np.maximum(ratio, 1e-30))

    def partial_corr_multi(target, y, controls):
        """□□□□□□□□□□"""
        from numpy.linalg import lstsq
        X = np.column_stack(controls)
        X = np.column_stack([X, np.ones(len(X))])
        ct, _, _, _ = lstsq(X, target, rcond=None)
        cy, _, _, _ = lstsq(X, y, rcond=None)
        rt = target - X @ ct
        ry = y - X @ cy
        return spearmanr(rt, ry)

    rho_full, p_full = partial_corr_multi(
        log_gc_corr, log_hR, [log_vflat, rmax_hR])
    print(f"ρ(gc_corrected, hR | vflat, rmax/hR) = {rho_full:+.3f} (p={p_full:.4f})")

    # ============================================================
    # (4) □□□□□□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("(4) □□□□□□□□")
    print("=" * 70)

    # ratio □ hR □□□
    rho_rh, p_rh = spearmanr(ratio, hR)
    print(f"ρ(ratio, hR) = {rho_rh:+.3f} (p={p_rh:.4f})")

    # ratio □ gc_obs □□□
    rho_rg, p_rg = spearmanr(ratio, gc_obs)
    print(f"ρ(ratio, gc_obs) = {rho_rg:+.3f} (p={p_rg:.4f})")

    # ratio □ hR □□□□□□□?
    # E_grad/E_local □ hR □□□□□□□ (ε(r) = sqrt(gN/gc_deep))
    # □□□ gN □ 1/r □□□□□□□□□□□□□□□
    rho_rh_vf, p_rh_vf = partial_corr(log_ratio, log_hR, log_vflat)
    print(f"ρ(ratio, hR | vflat) = {rho_rh_vf:+.3f} (p={p_rh_vf:.4f})")
    print("(E_grad/E_local □ hR □□□□□□□□□□□□□□□□)")

    # □□□□□: ratio □□□□□□
    np.random.seed(42)
    rho_shuffle = []
    for _ in range(1000):
        ratio_s = np.random.permutation(ratio)
        gc_s = gc_deep * (1 + kappa_best * ratio_s)
        log_gc_s = np.log10(np.maximum(gc_s, 1e-30))
        rho_s, _ = partial_corr(log_gc_s, log_hR, log_vflat)
        rho_shuffle.append(rho_s)
    rho_shuffle = np.array(rho_shuffle)
    pctile = np.mean(rho_shuffle &lt;= rho_corr) * 100
    print(f"\n□□□□□□□□: real ρ={rho_corr:+.3f}, "
          f"shuffle mean={np.mean(rho_shuffle):+.3f}, "
          f"percentile={pctile:.1f}%")

    # ============================================================
    # (5) □□□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("(5) κ □□□□□□")
    print("=" * 70)

    print(f"""
□□□□□□□□: L[ε] = U(ε;c) + κ(dε/dr)²

κ_best = {kappa_best:.4e}
κ * median(ratio) = {kappa_best * ratio_scale:.4f}
  → □□□□□□□ {abs(kappa_best * ratio_scale)*100:.1f}% □ gc □□

□□□□□:
  - κ &gt; 0: □□□□ gc □□□□□□□□□□□ → □□□□□□□□□□□□
  - κ &lt; 0: □□□□ gc □□□□□□□□□□□□□□□□□□□□□□□□
  - |κ×ratio| &lt;&lt; 1: □□□□□□□□□□□□

□□ bending rigidity □□□□:
  D_bend = κ × h_membrane³ / 12(1-ν²)  (□□□□)
  □□□ h_membrane □□□□□□□□□ν □□□□□□
""")

    # ============================================================
```

```
    # (6) □□13%□□□□
    # ============================================================
    print("=" * 70)
    print("(6) □□13%□□□□□□")
    print("=" * 70)

    # hR□□□□□□ (gc_obs □□□)
    # raw → |vflat → |vflat,x → |vflat,x,ratio → κ□□□
    rho_raw, _ = spearmanr(log_gc_obs, log_hR)

    print(f"\nhR□□□□□□□□□:")
    print(f"  raw:                          ρ = {rho_raw:+.3f}")
    print(f"  |vflat (gc_obs):              ρ = {rho_obs:+.3f}")
    print(f"  |vflat (gc_corrected):        ρ = {rho_corr:+.3f}")

    if abs(rho_obs) &gt; 0:
        reduction_pct = (1 - abs(rho_corr) / abs(rho_obs)) * 100
        print(f"  κ□□□□□□□: {reduction_pct:.1f}%")

        # □□□□□: 69%(x_outer) + 18%(ratio) + 13%(□□)
        # κ□□□ ratio □□□□ → 18%□□□□
        print(f"\n  □□: κ□□□ E_grad/E_local (18%, B-□) □□□□□□")
        print(f"  κ□□□□□□□□□□□□□□□□□□□□□□□□□□□□")

    # ============================================================
    # (7) □□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("□□□□")
    print("=" * 70)

    universal = "YES" if (ci_lo &lt;= 0 &lt;= ci_hi) or abs(dAIC) &lt; 2 else "MAYBE"

    print(f"""
κ_best = {kappa_best:.4e}
95% CI = [{ci_lo:.4e}, {ci_hi:.4e}]
CI includes 0: {'YES' if ci_lo &lt;= 0 &lt;= ci_hi else 'NO'}
dAIC vs κ=0: {dAIC:.1f}
R² improvement: {R2:.4f}
hR□□□□□: |ρ| {abs(rho_obs):.3f} → {abs(rho_corr):.3f}
□□□□□: (□□□□□□□□□□□□□)

□□□□□□:
  - κ &gt; 0 □□ CI excludes 0 → □□□□□□□□□□□
  - κ ~ 0 (CI includes 0) → □□□□□□□□□□□□□□□□□□
  - □□□□□□□ → κ = κ(morphology) □□□□□□
""")

    # ============================================================
    # (8) □□□□
    # ============================================================
    try:
        import matplotlib
        matplotlib.use('Agg')
        import matplotlib.pyplot as plt

        fig, axes = plt.subplots(2, 2, figsize=(12, 10))

        # (a) gc_obs vs gc_corrected
        ax = axes[0, 0]
        ax.scatter(np.log10(gc_obs), np.log10(gc_corrected), s=10, alpha=0.5, c='steelblue')
        lim = [np.min(np.log10(gc_obs))-0.1, np.max(np.log10(gc_obs))+0.1]
        ax.plot(lim, lim, 'k--', lw=1)
        ax.set_xlabel('log10(gc_obs)')
        ax.set_ylabel('log10(gc_corrected)')
        ax.set_title(f'gc_obs vs gc_corrected (kappa={kappa_best:.2e})')

        # (b) κ_i distribution
        ax = axes[0, 1]
        ax.hist(ki_clipped, bins=40, alpha=0.7, color='steelblue', edgecolor='white')
        ax.axvline(kappa_best, color='red', lw=2, label=f'best={kappa_best:.2e}')
        ax.axvline(0, color='grey', ls='--', lw=1)
        ax.set_xlabel('kappa_i (individual)')
        ax.set_ylabel('count')
        ax.set_title('Distribution of per-galaxy kappa')
        ax.legend()

        # (c) ratio vs log_excess
        ax = axes[1, 0]
        ax.scatter(ratio, log_excess, s=10, alpha=0.5, c='steelblue')
        r_sorted = np.sort(ratio)
        ax.plot(r_sorted, np.log10(1 + kappa_best * r_sorted), 'r-', lw=2,
                label=f'kappa={kappa_best:.2e}')
        ax.set_xlabel('E_grad / E_local')
        ax.set_ylabel('log10(gc_obs / gc_deep)')
        ax.set_title('Strain gradient ratio vs gc excess')
        ax.legend()

        # (d) bootstrap κ
        ax = axes[1, 1]
        ax.hist(kappa_boot, bins=50, alpha=0.7, color='steelblue', edgecolor='white')
        ax.axvline(kappa_best, color='red', lw=2, label=f'best={kappa_best:.2e}')
```

```
        ax.axvline(ci_lo, color='orange', ls='--', lw=1, label=f'95%CI')
        ax.axvline(ci_hi, color='orange', ls='--', lw=1)
        ax.axvline(0, color='grey', ls='--', lw=1)
        ax.set_xlabel('kappa (bootstrap)')
        ax.set_ylabel('count')
        ax.set_title('Bootstrap distribution of kappa')
        ax.legend()

        plt.tight_layout()
        figpath = os.path.join(BASE, 'kappa_estimation_results.png')
        plt.savefig(figpath, dpi=150)
        print(f"\n□□□□: {figpath}")

    except Exception as e:
        print(f"\n□□□□□□□: {e}")

    print("\n□□□")

if __name__ == '__main__':
    main()
```

Script 2: sparc_kappa_nonlinear.py

目的

条件14 非線形膜剛性モデル8種の系統的テスト

kappa=const の否定を受け、8種の非線形モデルを AICc + LOO-CV + シャッフルで系統評価。M1: kappa=const（再現）、M2: kappa0 x ratio^n、M3: epsilon閾値（塑性/弾性二相）、M4: epsilon-modulated ratio、M5: 塑性指標 /、M6: u_strain/E_kin、M7: 勾配非一様性、M8: epsilon曲率 d^2 epsilon/dr^2。過適合制御は AICc（小サンプル補正）+ LOO-CV（全銀河）+ シャッフル順列（1000回）の3重。

結果

M3（epsilon閾値, dAICc=-5.1, LOO=0.351）が最良。epsilon_c~=0.072, beta=-0.23。M6（dAICc=-5.7）、M8（dAICc=-5.6）も有意だが、後に M3 と冗長と判明。M1/M4/M5/M7 は不支持。

実行方法

```
uv run --with scipy --with matplotlib python sparc_kappa_nonlinear.py
```

入力データ: sparc_gc.csv + Rotmod_LTG/*.dat
出力: kappa_nonlinear_results.png + 標準出力

ソースコード全文

```
#!/usr/bin/env python3
"""
sparc_kappa_nonlinear.py
========================
□□14 □□□□: □□□□□□□□□□□□□□□□

□□:
  - κ=const □□□ (dAIC=+1.1, 0.6%□□, □□□□□□□)
  - E_grad/E_local ratio □ 18% (B-□) □□□□□
  - □□: ratio → κ □□□□□□□□□□□

□□: κ □□□□□□□□□□□□□ ε □□□□□
  → □□/□□□□□□□ (□□14) □□□□□□□□

□□□□□□:
  M0: gc = gc_deep                              (null, 0 param)
  M1: gc = gc_deep × (1 + κ₀ × ratio)           (const κ, 1 param) [□□]
  M2: gc = gc_deep × (1 + κ₀ × ratio^n)         (power-law, 2 param)
  M3: gc = gc_deep × f(ε_max)                    (ε□□, 2 param)
  M4: gc = gc_deep × (1 + κ₀ × ratio × g(ε))    (ε-modulated, 2 param)
  M5: gc = gc_deep × (1 + κ₀ × Φ_plastic)       (□□□□, 1 param)
  M6: gc = gc_deep × h(E_strain/E_kinetic)       (□□-□□□, 1-2 param)

□□□□□:
  - AICc (□□□□□□□AIC)
  - LOO-CV (leave-one-out cross-validation)
  - Shuffle permutation (1000□)
  - hR□□□□□□□□

□□: uv run --with scipy --with matplotlib python sparc_kappa_nonlinear.py
"""

import os, sys
import numpy as np
from scipy.optimize import minimize, minimize_scalar
from scipy.stats import spearmanr
import warnings
warnings.filterwarnings('ignore')

# ============================================================
# 0. □□□□□
# ============================================================
BASE = r"D:\□□□□□□\□□□□□□\□□□□□\□□□□□□□\□□□□"
ROTMOD = os.path.join(BASE, "Rotmod_LTG")
GC_CSV = os.path.join(BASE, "sparc_gc.csv")
a0 = 1.2e-10  # m/s^2

# ============================================================
# 1. □□□□□□□ (□□□□□□□□□)
# ============================================================
def load_gc_csv():
    import csv
    data = {}
    with open(GC_CSV, 'r') as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = None
```

```
            for k in row:
                if 'gal' in k.lower() or 'name' in k.lower():
                    name = row[k].strip(); break
            if not name: continue
            gc_val = None
            for k in ['gc', 'gc_obs', 'g_c', 'gc_a0']:
                if k in row and row[k]:
                    try: gc_val = float(row[k]); break
                    except: pass
            if gc_val is None: continue
            vflat, hR, Yd = None, None, None
            for k in ['vflat','Vflat','v_flat']:
                if k in row and row[k]:
                    try: vflat = float(row[k]); break
                    except: pass
            for k in ['hR','Reff','h_R','hr']:
                if k in row and row[k]:
                    try: hR = float(row[k]); break
                    except: pass
            for k in ['Yd','yd','Y_d','SPS']:
                if k in row and row[k]:
                    try: Yd = float(row[k]); break
                    except: pass
            data[name] = {'gc': gc_val, 'vflat': vflat, 'hR': hR, 'Yd': Yd}
    gc_vals = [d['gc'] for d in data.values()]
    if np.median(gc_vals) &gt; 1e-5:
        for name in data: data[name]['gc'] *= a0
    print(f"sparc_gc.csv: {len(data)} galaxies")
    return data

def load_rotcurve(galaxy_name):
    fname = os.path.join(ROTMOD, f"{galaxy_name}_rotmod.dat")
    if not os.path.exists(fname): return None
    rad, vobs, vgas, vdisk, vbul = [], [], [], [], []
    with open(fname, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith('#'): continue
            parts = line.split()
            if len(parts) &lt; 6: continue
            try:
                rad.append(float(parts[0]))
                vobs.append(float(parts[1]))
                vgas.append(float(parts[3]))
                vdisk.append(float(parts[4]))
                vbul.append(float(parts[5]))
            except: continue
    if len(rad) &lt; 5: return None
    return {k: np.array(v) for k, v in
            zip(['r','vobs','vgas','vdisk','vbul'], [rad,vobs,vgas,vdisk,vbul])}

def compute_gc_deep(rc, Yd=0.5):
    r_m = rc['r'] * 3.086e19
    v_bar2 = (Yd * rc['vdisk']**2 + rc['vgas']**2 + rc['vbul']**2) * 1e6
    v_obs2 = (rc['vobs'] * 1e3)**2
    gN = v_bar2 / r_m
    gobs = v_obs2 / r_m
    mask = gN &gt; 0
    gc_pts = gobs[mask]**2 / gN[mask]
    if len(gc_pts) &lt; 3: return None
    gc_med = np.median(gc_pts)
    if gc_med &lt;= 0: return None
    x = gN[mask] / gc_med
    deep = x &lt; 1.0
    return np.median(gc_pts[deep]) if np.sum(deep) &gt;= 3 else gc_med

# ============================================================
# 2. □□□□□□□□
# ============================================================
def compute_extended_strain(rc, gc_deep, Yd=0.5):
    """
    □□□□ radial profile □□□□□□□□□□□□
    """
    r_kpc = rc['r']
    r_m = r_kpc * 3.086e19
    v_bar2 = (Yd * rc['vdisk']**2 + rc['vgas']**2 + rc['vbul']**2) * 1e6
    v_obs2 = (rc['vobs'] * 1e3)**2
    gN = np.maximum(v_bar2 / r_m, 0)
    gobs = v_obs2 / r_m

    epsilon = np.sqrt(gN / gc_deep)

    # dε/dr
    deps_dr = np.zeros_like(epsilon)
    deps_dr[0] = (epsilon[1] - epsilon[0]) / (r_m[1] - r_m[0])
    deps_dr[-1] = (epsilon[-1] - epsilon[-2]) / (r_m[-1] - r_m[-2])
    for i in range(1, len(epsilon)-1):
        deps_dr[i] = (epsilon[i+1] - epsilon[i-1]) / (r_m[i+1] - r_m[i-1])

    E_grad = np.mean(deps_dr**2)
    E_local = np.mean(epsilon**2)
    ratio = E_grad / E_local if E_local &gt; 0 else 0
```

```
    # --- □□□□ ---
    eps_max = np.max(epsilon)
    eps_mean = np.mean(epsilon)
    eps_outer = np.mean(epsilon[-max(1, len(epsilon)//3):])  # □□1/3□□□

    # □□□□: ε &gt; ε_c (□□) □□□□□
    # ε_c □□□□□□□□□

    # □□□□□□□□□ u_strain □ ε² (□□) or ε³ (□□□)
    u_strain_elastic = np.mean(epsilon**2)
    u_strain_cubic = np.mean(epsilon**3)

    # □□□□□□□ proxy: v_obs²
    E_kin = np.mean(v_obs2)  # (m/s)²

    # □□-□□□
    strain_kin_ratio = u_strain_elastic / (E_kin / (3.086e19)**2) if E_kin &gt; 0 else 0

    # ε□□□□□□□: std(dε/dr) / mean(|dε/dr|)
    grad_nonuniform = np.std(deps_dr) / (np.mean(np.abs(deps_dr)) + 1e-50)

    # ε profile □□□: d²ε/dr²
    if len(epsilon) &gt;= 5:
        d2eps = np.gradient(np.gradient(epsilon, r_m), r_m)
        curvature = np.mean(d2eps**2)
    else:
        curvature = 0

    return {
        'ratio': ratio,
        'E_grad': E_grad,
        'E_local': E_local,
        'eps_max': eps_max,
        'eps_mean': eps_mean,
        'eps_outer': eps_outer,
        'epsilon_profile': epsilon,
        'u_elastic': u_strain_elastic,
        'u_cubic': u_strain_cubic,
        'strain_kin': strain_kin_ratio,
        'grad_nonuniform': grad_nonuniform,
        'curvature': curvature,
        'n_points': len(epsilon),
    }

# ============================================================
# 3. □□□□□□□□
# ============================================================
def build_dataset():
    gc_data = load_gc_csv()
    results = []
    for gname in sorted(gc_data.keys()):
        gd = gc_data[gname]
        gc_obs = gd['gc']
        vflat, hR, Yd = gd.get('vflat'), gd.get('hR'), gd.get('Yd') or 0.5
        if gc_obs &lt;= 0 or not vflat or not hR or vflat &lt;= 0 or hR &lt;= 0:
            continue
        rc = load_rotcurve(gname)
        if rc is None: continue
        gc_deep = compute_gc_deep(rc, Yd=Yd)
        if gc_deep is None or gc_deep &lt;= 0: continue
        sq = compute_extended_strain(rc, gc_deep, Yd=Yd)
        if sq is None or sq['ratio'] == 0: continue
        results.append({
            'name': gname, 'gc_obs': gc_obs, 'gc_deep': gc_deep,
            'vflat': vflat, 'hR': hR, 'Yd': Yd,
            **{k: sq[k] for k in sq if k != 'epsilon_profile'},
        })
    print(f"Dataset: {len(results)} galaxies\n")
    return results

# ============================================================
# 4. □□□□□ + □□□□□□□
# ============================================================
def aicc(chi2, n, k):
    """Small-sample corrected AIC"""
    aic = chi2 + 2*k
    if n - k - 1 &gt; 0:
        return aic + 2*k*(k+1) / (n - k - 1)
    return aic

def loo_cv(log_excess, predictor_func, param_fitter, N):
    """Leave-one-out cross-validation: returns mean squared error"""
    errors = []
    for i in range(N):
        mask = np.ones(N, dtype=bool)
        mask[i] = False
        params = param_fitter(mask)
        if params is None:
            continue
        pred = predictor_func(i, params)
        errors.append((log_excess[i] - pred)**2)
```

```
    return np.mean(errors) if errors else 999

class ModelTester:
    def __init__(self, data):
        self.N = len(data)
        self.gc_obs = np.array([d['gc_obs'] for d in data])
        self.gc_deep = np.array([d['gc_deep'] for d in data])
        self.log_excess = np.log10(self.gc_obs / self.gc_deep)
        self.ratio = np.array([d['ratio'] for d in data])
        self.eps_max = np.array([d['eps_max'] for d in data])
        self.eps_mean = np.array([d['eps_mean'] for d in data])
        self.eps_outer = np.array([d['eps_outer'] for d in data])
        self.u_elastic = np.array([d['u_elastic'] for d in data])
        self.u_cubic = np.array([d['u_cubic'] for d in data])
        self.strain_kin = np.array([d['strain_kin'] for d in data])
        self.grad_nonuniform = np.array([d['grad_nonuniform'] for d in data])
        self.curvature = np.array([d['curvature'] for d in data])
        self.vflat = np.array([d['vflat'] for d in data])
        self.hR = np.array([d['hR'] for d in data])
        self.rmax_hR = np.array([d.get('rmax_hR', 0) for d in data])
        self.names = [d['name'] for d in data]

        # null chi2
        self.chi2_null = np.sum(self.log_excess**2)
        self.mse_null = np.mean(self.log_excess**2)

        print(f"log(gc_obs/gc_deep): mean={np.mean(self.log_excess):.4f}, "
              f"std={np.std(self.log_excess):.4f}")

    def test_model(self, name, n_params, fit_func, predict_func,
                   shuffle_test=True, loo=True):
        """
        ▫▫▫▫▫▫▫▫
        fit_func(mask=None) -&gt; params  (mask=None▫▫▫▫▫)
        predict_func(params) -&gt; array of model predictions (log10 scale)
        """
        print(f"\n{'—'*60}")
        print(f"Model: {name} ({n_params} params)")
        print(f"{'—'*60}")

        # Fit on all data
        params = fit_func()
        if params is None:
            print("  FIT FAILED")
            return None

        pred = predict_func(params)
        resid = self.log_excess - pred
        chi2 = np.sum(resid**2)
        R2 = 1 - chi2 / self.chi2_null
        _aicc = aicc(chi2, self.N, n_params)
        _aicc_null = aicc(self.chi2_null, self.N, 0)
        dAICc = _aicc - _aicc_null

        print(f"  params: {params}")
        print(f"  chi2={chi2:.3f}, R2={R2:.4f}, dAICc={dAICc:+.1f}")
        print(f"  resid std={np.std(resid):.4f} (null={np.std(self.log_excess):.4f})")

        # hR partial correlation
        log_vf = np.log10(self.vflat)
        log_hR = np.log10(self.hR)
        gc_model = self.gc_deep * 10**pred
        log_gc_m = np.log10(np.maximum(gc_model, 1e-30))

        def pcorr(x, y, z):
            cx = np.polyfit(z, x, 1)
            cy = np.polyfit(z, y, 1)
            return spearmanr(x - np.polyval(cx, z), y - np.polyval(cy, z))

        rho_obs, _ = pcorr(np.log10(self.gc_obs), log_hR, log_vf)
        rho_model, p_model = pcorr(log_gc_m, log_hR, log_vf)

        print(f"  hR▫▫▫: gc_obs={rho_obs:+.3f}, model={rho_model:+.3f} "
              f"(▫▫{(1-abs(rho_model)/abs(rho_obs))*100:.1f}%)")

        # Shuffle test
        if shuffle_test:
            np.random.seed(42)
            better = 0
            for _ in range(1000):
                idx = np.random.permutation(self.N)
                le_s = self.log_excess[idx]
                chi2_s = np.sum((le_s - pred)**2)
                if chi2_s &lt; chi2:
                    better += 1
            print(f"  Shuffle: {better}/1000 better (p={better/1000:.3f})")

        # LOO-CV
        if loo and self.N &lt;= 200:
            loo_errors = []
            for i in range(self.N):
                mask = np.ones(self.N, dtype=bool)
```

```
                mask[i] = False
                p_loo = fit_func(mask)
                if p_loo is None: continue
                pred_i = predict_func(p_loo, idx=i)
                loo_errors.append((self.log_excess[i] - pred_i)**2)
            mse_loo = np.mean(loo_errors)
            print(f"  LOO-CV MSE={mse_loo:.6f} (null={self.mse_null:.6f}, "
                  f"ratio={mse_loo/self.mse_null:.4f})")

        return {
            'name': name, 'n_params': n_params, 'chi2': chi2,
            'R2': R2, 'dAICc': dAICc, 'rho_hR': rho_model,
            'params': params, 'resid_std': np.std(resid),
        }

    # ============================================================
    # M1: κ const (□□)
    # ============================================================
    def test_M1(self):
        ratio = self.ratio
        le = self.log_excess
        def fit(mask=None):
            r, e = (ratio, le) if mask is None else (ratio[mask], le[mask])
            def chi2(k):
                return np.sum((e - np.log10(np.maximum(1+k*r, 1e-10)))**2)
            rsc = np.median(r)
            if rsc &gt; 0:
                km = 5.0/rsc
            else:
                km = 1e40
            res = minimize_scalar(chi2, bounds=(-km, km), method='bounded')
            return [res.x]
        def pred(params, idx=None):
            k = params[0]
            r = ratio if idx is None else np.array([ratio[idx]])
            return np.log10(np.maximum(1 + k*r, 1e-10))
        return self.test_model("M1: κ=const", 1, fit, pred)

    # ============================================================
    # M2: κ × ratio^n (power-law□□□)
    # ============================================================
    def test_M2(self):
        ratio = self.ratio
        le = self.log_excess
        log_ratio = np.log10(np.maximum(ratio, 1e-50))

        def fit(mask=None):
            r, e = (ratio, le) if mask is None else (ratio[mask], le[mask])
            lr = np.log10(np.maximum(r, 1e-50))
            def chi2(p):
                k0, n = p
                val = k0 * np.abs(r)**n * np.sign(r)
                return np.sum((e - np.log10(np.maximum(1 + val, 1e-10)))**2)
            best = None
            for n0 in [0.3, 0.5, 1.0, 1.5, 2.0]:
                rsc = np.median(r)
                k0_init = 0.01 / (np.median(np.abs(r))**n0 + 1e-50)
                for k_sign in [1, -1]:
                    res = minimize(chi2, [k_sign*k0_init, n0], method='Nelder-Mead',
                                   options={'maxiter': 5000, 'xatol': 1e-10})
                    if best is None or res.fun &lt; best.fun:
                        best = res
            return list(best.x) if best else None

        def pred(params, idx=None):
            k0, n = params
            r = ratio if idx is None else np.array([ratio[idx]])
            val = k0 * np.abs(r)**n * np.sign(r)
            return np.log10(np.maximum(1 + val, 1e-10))

        return self.test_model("M2: κ₀×ratio^n", 2, fit, pred)

    # ============================================================
    # M3: ε□□□□□ (□□/□□□□)
    #   gc = gc_deep × (1 + A × f_plastic)
    #   f_plastic = fraction of points with ε &gt; ε_c
    # ============================================================
    def test_M3(self):
        eps_profiles = self.eps_mean  # use per-galaxy ε_mean as proxy
        eps_max = self.eps_max
        le = self.log_excess

        # f_plastic(ε_c) = fraction with ε &gt; ε_c → use eps_max vs threshold
        # □□□: eps_max &gt; ε_c □□ "plastic-dominated"
        # gc = gc_deep × (ε_max / ε_c)^β  for ε_max &gt; ε_c, else gc = gc_deep

        def fit(mask=None):
            em = eps_max if mask is None else eps_max[mask]
            e = le if mask is None else le[mask]
            lem = np.log10(np.maximum(em, 1e-10))

            def chi2(p):
```

```
                log_ec, beta = p
                excess = np.maximum(lem - log_ec, 0)  # max(0, log(eps_max/eps_c))
                model = beta * excess
                return np.sum((e - model)**2)

            best = None
            for lec in np.linspace(np.percentile(lem, 10), np.percentile(lem, 90), 10):
                for b in [-1, -0.5, 0.5, 1]:
                    res = minimize(chi2, [lec, b], method='Nelder-Mead',
                                   options={'maxiter': 3000})
                    if best is None or res.fun &lt; best.fun:
                        best = res
            return list(best.x) if best else None

        def pred(params, idx=None):
            log_ec, beta = params
            em = eps_max if idx is None else np.array([eps_max[idx]])
            lem = np.log10(np.maximum(em, 1e-10))
            excess = np.maximum(lem - log_ec, 0)
            return beta * excess

        return self.test_model("M3: ε□□(□□/□□)", 2, fit, pred)

    # ============================================================
    # M4: ε-modulated ratio
    #   gc = gc_deep × (1 + κ₀ × ratio × (ε_mean/ε_ref)^m)
    # ============================================================
    def test_M4(self):
        ratio = self.ratio
        eps_mean = self.eps_mean
        le = self.log_excess

        def fit(mask=None):
            r = ratio if mask is None else ratio[mask]
            em = eps_mean if mask is None else eps_mean[mask]
            e = le if mask is None else le[mask]

            eps_ref = np.median(em)

            def chi2(p):
                k0, m = p
                mod = k0 * r * (em / eps_ref)**m
                return np.sum((e - np.log10(np.maximum(1 + mod, 1e-10)))**2)

            rsc = np.median(r)
            k_init = 0.01 / (rsc + 1e-50)
            best = None
            for m0 in [-2, -1, 0, 1, 2]:
                for ks in [1, -1]:
                    res = minimize(chi2, [ks*k_init, m0], method='Nelder-Mead',
                                   options={'maxiter': 5000})
                    if best is None or res.fun &lt; best.fun:
                        best = res
            return list(best.x) if best else None

        def pred(params, idx=None):
            k0, m = params
            r = ratio if idx is None else np.array([ratio[idx]])
            em = eps_mean if idx is None else np.array([eps_mean[idx]])
            eps_ref = np.median(self.eps_mean)
            mod = k0 * r * (em / eps_ref)**m
            return np.log10(np.maximum(1 + mod, 1e-10))

        return self.test_model("M4: κ₀×ratio×(ε/ε_ref)^m", 2, fit, pred)

    # ============================================================
    # M5: □□□□ (□□14□□□□□)
    #   □ε□□□□□□ gc □□□
    #   gc = gc_deep × (1 + A × f_high_strain)
    #   f_high_strain = mean[ε² × I(ε &gt; median_ε)]
    # ============================================================
    def test_M5(self):
        # □□□□ ε □□□□: u_cubic / u_elastic = &lt;ε³&gt;/&lt;ε²&gt; = &lt;ε&gt;_weighted
        # → □ε□□□□□
        plastic_indicator = self.u_cubic / (self.u_elastic + 1e-50)
        le = self.log_excess

        def fit(mask=None):
            pi = plastic_indicator if mask is None else plastic_indicator[mask]
            e = le if mask is None else le[mask]
            log_pi = np.log10(np.maximum(pi, 1e-10))

            def chi2(p):
                a, = p
                model = a * log_pi
                return np.sum((e - model)**2)

            # linear regression
            from numpy.linalg import lstsq
            X = log_pi.reshape(-1, 1)
            sol, _, _, _ = lstsq(X, e, rcond=None)
            return [sol[0]]
```

```
        def pred(params, idx=None):
            a = params[0]
            pi = plastic_indicator if idx is None else np.array([plastic_indicator[idx]])
            return a * np.log10(np.maximum(pi, 1e-10))

        return self.test_model("M5: ▫▫▫▫ &lt;ε³&gt;/&lt;ε²&gt;", 1, fit, pred)

    # ============================================================
    # M6: ▫▫▫▫▫▫▫ / ▫▫▫▫▫▫▫▫
    # ============================================================
    def test_M6(self):
        sk = self.strain_kin
        le = self.log_excess

        def fit(mask=None):
            s = sk if mask is None else sk[mask]
            e = le if mask is None else le[mask]
            log_s = np.log10(np.maximum(s, 1e-50))
            finite = np.isfinite(log_s)
            if np.sum(finite) &lt; 10: return None

            from numpy.linalg import lstsq
            X = log_s[finite].reshape(-1, 1)
            sol, _, _, _ = lstsq(X, e[finite], rcond=None)
            return [sol[0]]

        def pred(params, idx=None):
            a = params[0]
            s = sk if idx is None else np.array([sk[idx]])
            return a * np.log10(np.maximum(s, 1e-50))

        return self.test_model("M6: u_strain/E_kin", 1, fit, pred)

    # ============================================================
    # M7: ▫▫▫▫▫▫ (▫▫)
    #   κ ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫ gc ▫▫▫▫
    # ============================================================
    def test_M7(self):
        gn = self.grad_nonuniform
        le = self.log_excess

        def fit(mask=None):
            g = gn if mask is None else gn[mask]
            e = le if mask is None else le[mask]
            log_g = np.log10(np.maximum(g, 1e-10))
            finite = np.isfinite(log_g)
            if np.sum(finite) &lt; 10: return None
            from numpy.linalg import lstsq
            X = log_g[finite].reshape(-1, 1)
            sol, _, _, _ = lstsq(X, e[finite], rcond=None)
            return [sol[0]]

        def pred(params, idx=None):
            a = params[0]
            g = gn if idx is None else np.array([gn[idx]])
            return a * np.log10(np.maximum(g, 1e-10))

        return self.test_model("M7: ▫▫▫▫▫▫", 1, fit, pred)

    # ============================================================
    # M8: ε▫▫ (d²ε/dr²)
    # ============================================================
    def test_M8(self):
        curv = self.curvature
        le = self.log_excess

        def fit(mask=None):
            c = curv if mask is None else curv[mask]
            e = le if mask is None else le[mask]
            log_c = np.log10(np.maximum(c, 1e-80))
            finite = np.isfinite(log_c)
            if np.sum(finite) &lt; 10: return None
            from numpy.linalg import lstsq
            X = log_c[finite].reshape(-1, 1)
            sol, _, _, _ = lstsq(X, e[finite], rcond=None)
            return [sol[0]]

        def pred(params, idx=None):
            a = params[0]
            c = curv if idx is None else np.array([curv[idx]])
            return a * np.log10(np.maximum(c, 1e-80))

        return self.test_model("M8: ε▫▫ d²ε/dr²", 1, fit, pred)

# ============================================================
# 5. ▫▫▫
# ============================================================
def main():
    print("=" * 70)
    print("▫▫14 ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫")
    print("=" * 70)
```

```
|
|     data = build_dataset()
|     mt = ModelTester(data)
|
|     results = []
|     results.append(mt.test_M1())   # κ const (baseline)
|     results.append(mt.test_M2())   # κ₀ × ratio^n
|     results.append(mt.test_M3())   # ε□□
|     results.append(mt.test_M4())   # ε-modulated ratio
|     results.append(mt.test_M5())   # □□□□
|     results.append(mt.test_M6())   # u_strain/E_kin
|     results.append(mt.test_M7())   # □□□□□□
|     results.append(mt.test_M8())   # ε□□
|
|     results = [r for r in results if r is not None]
|
|     # ============================================================
|     # □□□
|     # ============================================================
|     print("\n" + "=" * 70)
|     print("□□□□□□□□□")
|     print("=" * 70)
|     print(f"{'Model':&lt;35s} {'k':&gt;2s} {'dAICc':&gt;7s} {'R²':&gt;7s} "
|           f"{'ρ(hR)':&gt;7s} {'std':&gt;7s}")
|     print("-" * 70)
|     print(f"{'M0: null (gc=gc_deep)':&lt;35s} {'0':&gt;2s} {'0.0':&gt;7s} {'0.0':&gt;7s} "
|           f"{'--':&gt;7s} {np.std(mt.log_excess):&gt;7.4f}")
|     for r in sorted(results, key=lambda x: x['dAICc']):
|         print(f"{r['name']:&lt;35s} {r['n_params']:&gt;2d} {r['dAICc']:&gt;+7.1f} "
|               f"{r['R2']:&gt;7.4f} {r['rho_hR']:&gt;+7.3f} {r['resid_std']:&gt;7.4f}")
|
|     # □□□□□
|     best = min(results, key=lambda x: x['dAICc'])
|     print(f"\n□□□□□: {best['name']} (dAICc={best['dAICc']:+.1f})")
|
|     if best['dAICc'] &gt;= 0:
|         print("→ null□□□ (gc = gc_deep) □□□□□□□□□□□□□□□")
|     elif best['dAICc'] &gt; -2:
|         print("→ □□□□□□□□□□□□□□□")
|     elif best['dAICc'] &gt; -6:
|         print("→ □□□□□□□□□□□□□□")
|     else:
|         print("→ □□□□□□□□□□□□□□")
|
|     # ============================================================
|     # □□
|     # ============================================================
|     print("\n" + "=" * 70)
|     print("□□□□□")
|     print("=" * 70)
|     print("""
| E_grad/E_local □□□18% (B-□) □□□□□□□□□□□□□
| □□8□□□□□□□□□□□□□□□□□□□□□□□□□□□□
|
| □□□□:
|   dAICc &lt; -6  → □□□□□□□□□□□□□
|   dAICc &lt; -2  → □□□□□□□□□□
|   dAICc &gt;= 0  → □□□□□□□□□□□□
|
| □□□□□□□□□□□□:
|   → E_grad/E_local □□□□□□□□□□□□□□□□
|   → □□14□□ρ_eff = u_strain/c²□□□□□□□□□□□□
|   → □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
|
| □□□□□□□□□□□□□:
|   → □□14□□□□□□□□□
|   → κ(ε) □□□□□□□□□□□□□□□□□
| """)
|
|     # ============================================================
|     # □
|     # ============================================================
|     try:
|         import matplotlib
|         matplotlib.use('Agg')
|         import matplotlib.pyplot as plt
|
|         fig, axes = plt.subplots(2, 2, figsize=(12, 10))
|         fig.suptitle('Non-linear membrane stiffness models', fontsize=14)
|
|         # (a) dAICc comparison
|         ax = axes[0, 0]
|         names = [r['name'].split(':')[0] for r in results]
|         daics = [r['dAICc'] for r in results]
|         colors = ['green' if d &lt; -6 else 'orange' if d &lt; -2 else 'red' for d in daics]
|         ax.barh(range(len(names)), daics, color=colors, alpha=0.7)
|         ax.set_yticks(range(len(names)))
|         ax.set_yticklabels(names, fontsize=8)
|         ax.axvline(0, color='black', ls='--', lw=1)
|         ax.axvline(-2, color='grey', ls=':', lw=1)
|         ax.set_xlabel('dAICc vs null')
|         ax.set_title('Model comparison')
```

```
        # (b) R² comparison
        ax = axes[0, 1]
        r2s = [r['R2'] for r in results]
        ax.barh(range(len(names)), r2s, color='steelblue', alpha=0.7)
        ax.set_yticks(range(len(names)))
        ax.set_yticklabels(names, fontsize=8)
        ax.set_xlabel('R²')
        ax.set_title('Variance explained')

        # (c) hR partial correlation
        ax = axes[1, 0]
        rhos = [abs(r['rho_hR']) for r in results]
        ax.barh(range(len(names)), rhos, color='coral', alpha=0.7)
        rho_obs_val = abs(results[0]['rho_hR']) if results else 0.35
        ax.axvline(rho_obs_val, color='black', ls='--', lw=1, label='gc_obs baseline')
        ax.set_yticks(range(len(names)))
        ax.set_yticklabels(names, fontsize=8)
        ax.set_xlabel('|ρ(gc_model, hR | vflat)|')
        ax.set_title('hR partial correlation (lower=better)')
        ax.legend(fontsize=8)

        # (d) scatter: best model prediction vs excess
        ax = axes[1, 1]
        if best:
            ax.scatter(mt.log_excess, np.zeros(mt.N), s=10, alpha=0.3, label='null residual')
            pred_best = best.get('params')
            ax.set_xlabel('log10(gc_obs/gc_deep)')
            ax.set_ylabel('model prediction')
            ax.set_title(f'Best model: {best["name"]}')

        plt.tight_layout()
        figpath = os.path.join(BASE, 'kappa_nonlinear_results.png')
        plt.savefig(figpath, dpi=150)
        print(f"\n□□□□: {figpath}")
    except Exception as e:
        print(f"\n□□□□□□□: {e}")

    print("\n□□□")

if __name__ == '__main__':
    main()
```

Script 3: sparc_kappa_circularity.py

目的

M3/M6/M8 の R^2=0.55 が gc_deep 共有ノイズのアーティファクトでないかの検証

epsilon = sqrt(gN/gc_deep) なので gc_deep が予測変数と応答変数の両方に共有される。gc_deep 推定ノイズが見かけの R^2 を生む可能性を 5 段階で検証。T1: Split-half gc_deep（データ点を2分割、独立に gc_deep 推定、交差使用）。T2: gc_deep シャッフル（銀河間でシャッフル、1000回）。T3: gc_obs 直接予測（gc_deep を介さない proxy で予測）。T4: gc_deep 撹乱（+/-0.05~0.5 dex ノイズ追加）。T5: log_excess vs log(strain_kin) の代数的関係分析。

結果

真の物理信号と確認。T1: split-half で R^2 が 93-96% 維持。T2: シャッフルで完全消失（mean=0.117, p=0.0000）。T3: 幾何平均法則を超える独立情報（dR^2=+0.119）。

実行方法

```
uv run --with scipy --with matplotlib python sparc_kappa_circularity.py
```

入力データ: sparc_gc.csv + Rotmod_LTG/*.dat
出力: 標準出力（図なし）

ソースコード全文

```
#!/usr/bin/env python3
"""
sparc_kappa_circularity.py
==========================
M3/M6/M8 □ R²=0.55 □□□□? gc_deep□□□□□□□□

□□□□□:
  □□□□: y = log(gc_obs / gc_deep)
  □□□□: x = f(ε), ε = sqrt(gN / gc_deep)
  → gc_deep □ y □□□□ x □□□□□□□□□
  → gc_deep □□□□□ y □ x □□□□□□□□ → □□□□R²

□□□:
  T1: Split-half gc_deep — □□□□□2□□□□□□gc_deep□□□
      □□□□gc_deep□□□□□□ → □□□□□□□ → □□R²
  T2: gc_deep □□□□□ — gc_deep □□□□□□□□□□
      gc_deep □□□□□□ → R² □□□□□□□□□
  T3: gc_obs □□□□ — y = log(gc_obs) □□□ (gc_deep □□□□□)
      → gc_deep □□□□□□□□□□□□
  T4: □□□□□ gc_deep — gc_deep □□□□□□ ±0.2 dex □□
      → R² □□□□ (□□□□□□□□□□□□)

□□: uv run --with scipy --with matplotlib python sparc_kappa_circularity.py
"""

import os, sys
import numpy as np
from scipy.optimize import minimize_scalar, minimize
from scipy.stats import spearmanr
import warnings
warnings.filterwarnings('ignore')

BASE = r"D:\□□□□□□\□□□□□□\□□□□□\□□□□□□□\□□□□"
ROTMOD = os.path.join(BASE, "Rotmod_LTG")
GC_CSV = os.path.join(BASE, "sparc_gc.csv")
a0 = 1.2e-10

# ============================================================
# □□□□□□□ (□□□□□□□□)
# ============================================================
def load_gc_csv():
    import csv
    data = {}
    with open(GC_CSV, 'r') as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = None
            for k in row:
                if 'gal' in k.lower() or 'name' in k.lower():
                    name = row[k].strip(); break
            if not name: continue
            gc_val = None
            for k in ['gc','gc_obs','g_c','gc_a0']:
                if k in row and row[k]:
                    try: gc_val = float(row[k]); break
                    except: pass
            if gc_val is None: continue
```

```
            vflat, hR, Yd = None, None, None
            for k in ['vflat','Vflat','v_flat']:
                if k in row and row[k]:
                    try: vflat = float(row[k]); break
                    except: pass
            for k in ['hR','Reff','h_R','hr']:
                if k in row and row[k]:
                    try: hR = float(row[k]); break
                    except: pass
            for k in ['Yd','yd','Y_d','SPS']:
                if k in row and row[k]:
                    try: Yd = float(row[k]); break
                    except: pass
            data[name] = {'gc': gc_val, 'vflat': vflat, 'hR': hR, 'Yd': Yd}
    gc_vals = [d['gc'] for d in data.values()]
    if np.median(gc_vals) &gt; 1e-5:
        for n in data: data[n]['gc'] *= a0
    return data

def load_rotcurve(galaxy_name):
    fname = os.path.join(ROTMOD, f"{galaxy_name}_rotmod.dat")
    if not os.path.exists(fname): return None
    cols = {k: [] for k in ['r','vobs','vgas','vdisk','vbul']}
    with open(fname, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith('#'): continue
            p = line.split()
            if len(p) &lt; 6: continue
            try:
                cols['r'].append(float(p[0]))
                cols['vobs'].append(float(p[1]))
                cols['vgas'].append(float(p[3]))
                cols['vdisk'].append(float(p[4]))
                cols['vbul'].append(float(p[5]))
            except: continue
    if len(cols['r']) &lt; 5: return None
    return {k: np.array(v) for k, v in cols.items()}

def compute_gc_deep_from_indices(rc, Yd, indices):
    """□□□□□□□□□□□□□□□□ gc_deep □□□"""
    r_m = rc['r'][indices] * 3.086e19
    v_bar2 = (Yd * rc['vdisk'][indices]**2 + rc['vgas'][indices]**2 + rc['vbul'][indices]**2) * 1e6
    v_obs2 = (rc['vobs'][indices] * 1e3)**2
    gN = v_bar2 / r_m
    gobs = v_obs2 / r_m
    mask = gN &gt; 0
    if np.sum(mask) &lt; 3: return None
    gc_pts = gobs[mask]**2 / gN[mask]
    gc_med = np.median(gc_pts)
    if gc_med &lt;= 0: return None
    x = gN[mask] / gc_med
    deep = x &lt; 1.0
    return np.median(gc_pts[deep]) if np.sum(deep) &gt;= 3 else gc_med

def compute_gc_deep(rc, Yd=0.5):
    return compute_gc_deep_from_indices(rc, Yd, np.arange(len(rc['r'])))

def compute_quantities(rc, gc_deep, Yd=0.5):
    """M6 (u_strain/E_kin) □ M3 (eps_max) □ M8 (curvature) □□□"""
    r_m = rc['r'] * 3.086e19
    v_bar2 = (Yd * rc['vdisk']**2 + rc['vgas']**2 + rc['vbul']**2) * 1e6
    v_obs2 = (rc['vobs'] * 1e3)**2
    gN = np.maximum(v_bar2 / r_m, 0)
    epsilon = np.sqrt(gN / gc_deep)

    u_elastic = np.mean(epsilon**2)
    E_kin = np.mean(v_obs2)
    strain_kin = u_elastic / (E_kin / (3.086e19)**2) if E_kin &gt; 0 else 0

    eps_max = np.max(epsilon)

    if len(epsilon) &gt;= 5:
        d2eps = np.gradient(np.gradient(epsilon, r_m), r_m)
        curvature = np.mean(d2eps**2)
    else:
        curvature = 0

    return {'strain_kin': strain_kin, 'eps_max': eps_max, 'curvature': curvature}

# ============================================================
# M6 □□□□□□
# ============================================================
def fit_M6(log_excess, log_sk):
    """M6: y = a * log(strain_kin) → 1-param OLS"""
    finite = np.isfinite(log_sk) &amp;amp; np.isfinite(log_excess)
    if np.sum(finite) &lt; 10: return None, 999
    from numpy.linalg import lstsq
    X = log_sk[finite].reshape(-1, 1)
    sol, _, _, _ = lstsq(X, log_excess[finite], rcond=None)
    pred = sol[0] * log_sk
    pred[~finite] = 0
```

```
    chi2 = np.sum((log_excess - pred)**2)
    return sol[0], chi2

def fit_M3(log_excess, log_eps_max):
    """M3: y = β * max(0, log(eps_max) - log(eps_c))"""
    finite = np.isfinite(log_eps_max)
    lem = log_eps_max[finite]
    e = log_excess[finite]
    best = None
    for lec in np.linspace(np.percentile(lem, 10), np.percentile(lem, 90), 20):
        for b in np.linspace(-1, 1, 20):
            excess = np.maximum(lem - lec, 0)
            chi2 = np.sum((e - b * excess)**2)
            if best is None or chi2 &lt; best[0]:
                best = (chi2, lec, b)
    if best is None: return None, None, 999
    # refine
    def chi2_f(p):
        exc = np.maximum(lem - p[0], 0)
        return np.sum((e - p[1] * exc)**2)
    res = minimize(chi2_f, [best[1], best[2]], method='Nelder-Mead')
    return res.x[0], res.x[1], res.fun

# ============================================================
# □□□
# ============================================================
def main():
    print("=" * 70)
    print("□□□□□□□: M3/M6/M8 □ R²=0.55 □□□□?")
    print("=" * 70)

    gc_data = load_gc_csv()

    # □□□□□
    galaxies = []
    for gname in sorted(gc_data.keys()):
        gd = gc_data[gname]
        gc_obs, vflat, hR = gd['gc'], gd.get('vflat'), gd.get('hR')
        Yd = gd.get('Yd') or 0.5
        if gc_obs &lt;= 0 or not vflat or not hR or vflat &lt;= 0 or hR &lt;= 0:
            continue
        rc = load_rotcurve(gname)
        if rc is None or len(rc['r']) &lt; 8: continue  # need enough points for split
        gc_deep = compute_gc_deep(rc, Yd)
        if gc_deep is None or gc_deep &lt;= 0: continue
        sq = compute_quantities(rc, gc_deep, Yd)
        if sq['strain_kin'] &lt;= 0: continue
        galaxies.append({
            'name': gname, 'gc_obs': gc_obs, 'gc_deep': gc_deep,
            'vflat': vflat, 'hR': hR, 'Yd': Yd, 'rc': rc, **sq
        })

    N = len(galaxies)
    print(f"Galaxies: {N}\n")

    gc_obs = np.array([g['gc_obs'] for g in galaxies])
    gc_deep = np.array([g['gc_deep'] for g in galaxies])
    strain_kin = np.array([g['strain_kin'] for g in galaxies])
    eps_max = np.array([g['eps_max'] for g in galaxies])
    curvature = np.array([g['curvature'] for g in galaxies])
    vflat = np.array([g['vflat'] for g in galaxies])
    hR = np.array([g['hR'] for g in galaxies])

    log_excess = np.log10(gc_obs / gc_deep)
    log_sk = np.log10(np.maximum(strain_kin, 1e-50))
    log_em = np.log10(np.maximum(eps_max, 1e-10))
    log_curv = np.log10(np.maximum(curvature, 1e-80))

    # --- Baseline ---
    chi2_null = np.sum(log_excess**2)
    a_m6, chi2_m6 = fit_M6(log_excess, log_sk)
    R2_m6 = 1 - chi2_m6/chi2_null
    print(f"Baseline M6: a={a_m6:.4f}, R²={R2_m6:.4f}")

    lec_m3, beta_m3, chi2_m3 = fit_M3(log_excess, log_em)
    R2_m3 = 1 - chi2_m3/chi2_null
    print(f"Baseline M3: ε_c={10**lec_m3:.4f}, β={beta_m3:.3f}, R²={R2_m3:.4f}")

    # ============================================================
    # T1: Split-half gc_deep (□□□□□)
    # ============================================================
    print("\n" + "=" * 70)
    print("T1: Split-half gc_deep")
    print("=" * 70)
    print("□□□□□□□□□□□/□□□□□□□□□gc_deep□□□")
    print("gc_deep_A □ ε □□□□gc_deep_B □ excess □□□ → □□□□□\n")

    np.random.seed(42)
    R2_split_m6 = []
    R2_split_m3 = []

    for trial in range(100):
```

```
        gc_deep_A = np.zeros(N)
        gc_deep_B = np.zeros(N)
        valid = np.ones(N, dtype=bool)

        for i, g in enumerate(galaxies):
            rc = g['rc']
            n_pts = len(rc['r'])
            perm = np.random.permutation(n_pts)
            half = n_pts // 2
            idxA, idxB = perm[:half], perm[half:]

            gA = compute_gc_deep_from_indices(rc, g['Yd'], idxA)
            gB = compute_gc_deep_from_indices(rc, g['Yd'], idxB)

            if gA is None or gB is None or gA &lt;= 0 or gB &lt;= 0:
                valid[i] = False
                continue
            gc_deep_A[i] = gA
            gc_deep_B[i] = gB

        if np.sum(valid) &lt; 50:
            continue

        # ε from gc_deep_A, excess from gc_deep_B
        # → □□□□□□
        le_B = np.log10(gc_obs[valid] / gc_deep_B[valid])

        # M6: strain_kin using gc_deep_A
        sk_A = []
        for i in np.where(valid)[0]:
            sq = compute_quantities(galaxies[i]['rc'], gc_deep_A[i], galaxies[i]['Yd'])
            sk_A.append(sq['strain_kin'])
        sk_A = np.array(sk_A)
        log_sk_A = np.log10(np.maximum(sk_A, 1e-50))

        a_s, chi2_s = fit_M6(le_B, log_sk_A)
        if a_s is not None:
            R2_s = 1 - chi2_s / np.sum(le_B**2)
            R2_split_m6.append(R2_s)

        # M3: eps_max using gc_deep_A
        em_A = []
        for i in np.where(valid)[0]:
            sq = compute_quantities(galaxies[i]['rc'], gc_deep_A[i], galaxies[i]['Yd'])
            em_A.append(sq['eps_max'])
        em_A = np.array(em_A)
        log_em_A = np.log10(np.maximum(em_A, 1e-10))

        _, _, chi2_s3 = fit_M3(le_B, log_em_A)
        R2_s3 = 1 - chi2_s3 / np.sum(le_B**2)
        R2_split_m3.append(R2_s3)

    R2_split_m6 = np.array(R2_split_m6)
    R2_split_m3 = np.array(R2_split_m3)

    print(f"M6 split-half R²: mean={np.mean(R2_split_m6):.4f} "
          f"(baseline={R2_m6:.4f}, ratio={np.mean(R2_split_m6)/R2_m6:.2f})")
    print(f"M3 split-half R²: mean={np.mean(R2_split_m3):.4f} "
          f"(baseline={R2_m3:.4f}, ratio={np.mean(R2_split_m3)/R2_m3:.2f})")

    if np.mean(R2_split_m6) / R2_m6 &lt; 0.3:
        print("□ M6: R² □ 70%+ □□ → gc_deep □□□□□□□□□")
    elif np.mean(R2_split_m6) / R2_m6 &lt; 0.7:
        print("□ M6: R² □ 30-70% □□ → □□□□□□□□□")
    else:
        print("□ M6: R² □□□□□ → □□□□□□□")

    # ============================================================
    # T2: gc_deep □□□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("T2: gc_deep □□□□□ (1000□)")
    print("=" * 70)

    np.random.seed(42)
    R2_shuffle_m6 = []
    for _ in range(1000):
        perm = np.random.permutation(N)
        gc_deep_s = gc_deep[perm]
        le_s = np.log10(gc_obs / gc_deep_s)

        sk_s = []
        for i in range(N):
            sq = compute_quantities(galaxies[i]['rc'], gc_deep_s[i], galaxies[i]['Yd'])
            sk_s.append(sq['strain_kin'])
        sk_s = np.array(sk_s)
        log_sk_s = np.log10(np.maximum(sk_s, 1e-50))

        a_s, chi2_s = fit_M6(le_s, log_sk_s)
        if a_s is not None:
            R2_shuffle_m6.append(1 - chi2_s / np.sum(le_s**2))

```

```
    R2_shuffle_m6 = np.array(R2_shuffle_m6)
    p_val = np.mean(R2_shuffle_m6 &gt;= R2_m6)
    print(f"M6 shuffle R²: mean={np.mean(R2_shuffle_m6):.4f}, "
          f"max={np.max(R2_shuffle_m6):.4f}")
    print(f"Real R²={R2_m6:.4f}, p={p_val:.4f}")

    if np.mean(R2_shuffle_m6) &gt; 0.3:
        print("□ □□□□□□□ R² □□□ → gc_deep □□□□□□□□□□")
    else:
        print("□ □□□□□□ R² □□ → □□□□□□□□")

    # ============================================================
    # T3: gc_obs □□□□ (gc_deep □□□)
    # ============================================================
    print("\n" + "=" * 70)
    print("T3: gc_obs □□□□ (gc_deep □□□□□)")
    print("=" * 70)

    # M6□□□ gc_deep □□□□□:
    # gN_mean = mean(V_bar²/r)  (gc_deep □□)
    # E_kin_mean = mean(V_obs²)
    # proxy_ratio = gN_mean / E_kin_mean  (□□□□□□□□□□□)

    log_gc_obs = np.log10(gc_obs)

    # gc_deep-free □□
    gN_mean_arr = []
    E_kin_arr = []
    for g in galaxies:
        rc = g['rc']
        r_m = rc['r'] * 3.086e19
        v_bar2 = (g['Yd'] * rc['vdisk']**2 + rc['vgas']**2 + rc['vbul']**2) * 1e6
        v_obs2 = (rc['vobs'] * 1e3)**2
        gN = np.maximum(v_bar2 / r_m, 0)
        gN_mean_arr.append(np.mean(gN))
        E_kin_arr.append(np.mean(v_obs2))

    gN_mean = np.array(gN_mean_arr)
    E_kin = np.array(E_kin_arr)

    # gc_deep-free proxy: gN_mean / E_kin (~ ε² □ proxy)
    proxy_nofree = gN_mean / (E_kin / (3.086e19)**2)
    log_proxy = np.log10(np.maximum(proxy_nofree, 1e-50))

    # gc_obs □ log_proxy □□□
    from numpy.linalg import lstsq
    finite = np.isfinite(log_proxy)
    X = np.column_stack([log_proxy[finite], np.ones(np.sum(finite))])
    sol, _, _, _ = lstsq(X, log_gc_obs[finite], rcond=None)
    pred_direct = sol[0] * log_proxy + sol[1]
    chi2_direct = np.sum((log_gc_obs - pred_direct)**2)
    chi2_null_direct = np.sum((log_gc_obs - np.mean(log_gc_obs))**2)
    R2_direct = 1 - chi2_direct / chi2_null_direct

    print(f"gc_obs ~ proxy (gc_deep-free): R²={R2_direct:.4f}")
    print(f"  (cf. M6 R²={R2_m6:.4f} uses gc_deep in both sides)")

    # gc_obs □ vflat²/hR □□□ (□□□□□□□ proxy)
    log_GSigma = np.log10(vflat**2 / hR)
    X2 = np.column_stack([log_GSigma, np.ones(N)])
    sol2, _, _, _ = lstsq(X2, log_gc_obs, rcond=None)
    pred_gs = sol2[0] * log_GSigma + sol2[1]
    R2_gs = 1 - np.sum((log_gc_obs - pred_gs)**2) / chi2_null_direct
    print(f"gc_obs ~ vflat²/hR (□□□□): R²={R2_gs:.4f}")

    # gc_obs □ gc_deep-free proxy + vflat²/hR □□□
    X3 = np.column_stack([log_proxy[finite], log_GSigma[finite], np.ones(np.sum(finite))])
    sol3, _, _, _ = lstsq(X3, log_gc_obs[finite], rcond=None)
    pred_comb = sol3[0]*log_proxy + sol3[1]*log_GSigma + sol3[2]
    R2_comb = 1 - np.sum((log_gc_obs - pred_comb)**2) / chi2_null_direct
    print(f"gc_obs ~ proxy + vflat²/hR: R²={R2_comb:.4f}")
    print(f"  proxy □□□ dR² = {R2_comb - R2_gs:.4f}")

    if R2_comb - R2_gs &lt; 0.02:
        print("□ proxy □ vflat²/hR □□□□□□□□□□□□□□□")
    else:
        print(f"□ proxy □ vflat²/hR □ +{(R2_comb-R2_gs)*100:.1f}% □□□□□□□□")

    # ============================================================
    # T4: gc_deep □□□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("T4: gc_deep □□□□□")
    print("=" * 70)

    np.random.seed(42)
    perturbations = [0.05, 0.1, 0.2, 0.3, 0.5]  # dex
    for sigma in perturbations:
        R2_perturbed = []
        for _ in range(100):
            noise = 10**(np.random.normal(0, sigma, N))
            gc_deep_p = gc_deep * noise
```

```
            le_p = np.log10(gc_obs / gc_deep_p)

            sk_p = []
            for i in range(N):
                sq = compute_quantities(galaxies[i]['rc'], gc_deep_p[i], galaxies[i]['Yd'])
                sk_p.append(sq['strain_kin'])
            sk_p = np.array(sk_p)
            log_sk_p = np.log10(np.maximum(sk_p, 1e-50))

            a_p, chi2_p = fit_M6(le_p, log_sk_p)
            if a_p is not None:
                R2_perturbed.append(1 - chi2_p / np.sum(le_p**2))

        R2_p = np.array(R2_perturbed)
        print(f"  sigma={sigma:.2f} dex: R²={np.mean(R2_p):.4f} "
              f"(baseline={R2_m6:.4f}, ratio={np.mean(R2_p)/R2_m6:.2f})")

    # ============================================================
    # T5: □□ — ε² = gN/gc_deep □ y = log(gc_obs/gc_deep) □
    #     □□□□□□□□□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("T5: □□□□□□□□")
    print("=" * 70)

    # M6 □□□:
    # y = a × log(strain_kin) = a × log(&lt;ε²&gt; / E_kin_scaled)
    # ε² = gN/gc_deep
    # y = log(gc_obs/gc_deep)
    #
    # deep MOND: gc_obs ≈ sqrt(gN × gc_deep_true) / something
    # → log(gc_obs/gc_deep) = log(gc_obs) - log(gc_deep)
    # → if gc_deep has error δ: y = y_true - δ
    # → ε² = gN/gc_deep = (gN/gc_deep_true) × 10^δ
    # → log(ε²) = log(ε²_true) + δ
    # → y = y_true - δ,  x = x_true + δ  → anti-correlation from δ

    # □□□□□: log_excess vs log(&lt;ε²&gt;) □□□
    log_eps2_mean = np.log10(np.array([g['strain_kin'] for g in galaxies]) *
                             E_kin / (3.086e19)**2 + 1e-50)
    # Actually &lt;ε²&gt; = u_elastic
    u_el = np.array([np.mean(np.sqrt(np.maximum(
        (g['Yd']*g['rc']['vdisk']**2 + g['rc']['vgas']**2 + g['rc']['vbul']**2)*1e6 /
        (g['rc']['r']*3.086e19) / g['gc_deep'], 0)))**2
        for g in galaxies])  # ε_mean² proxy

    # simpler: just compute correlation of excess with ε quantities
    rho_direct, p_direct = spearmanr(log_excess, np.log10(np.maximum(strain_kin, 1e-50)))
    print(f"ρ(log_excess, log_strain_kin) = {rho_direct:+.3f} (p={p_direct:.2e})")

    # If this is ~-0.7 or stronger, it's suspicious
    if abs(rho_direct) &gt; 0.6:
        print("□ □□□□□ → gc_deep □□□□□□□□□")
    elif abs(rho_direct) &gt; 0.3:
        print("□ □□□□□□ → □□□□□□□□□□□□□□□□□")
    else:
        print("□ □□□□ → gc_deep □□□□□□□□□□")

    # ============================================================
    # □□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("□□□□")
    print("=" * 70)
    print(f"""
□□□□□:
  T1 Split-half: M6 R² {np.mean(R2_split_m6):.4f} / {R2_m6:.4f} = {np.mean(R2_split_m6)/R2_m6:.2f}
  T1 Split-half: M3 R² {np.mean(R2_split_m3):.4f} / {R2_m3:.4f} = {np.mean(R2_split_m3)/R2_m3:.2f}
  T2 Shuffle: mean R²={np.mean(R2_shuffle_m6):.4f}, p={p_val:.4f}
  T3 Direct: gc_deep-free R²={R2_direct:.4f}, dR² over geometric mean = {R2_comb-R2_gs:.4f}
  T5 Correlation: ρ={rho_direct:+.3f}

□□□□:
  Split-half ratio &gt; 0.7 AND shuffle p &lt; 0.01 AND dR²(T3) &gt; 0.02
  → □□□□□□□

  Split-half ratio &lt; 0.3 OR shuffle R² &gt; 0.3
  → gc_deep □□□□□□□□□□□□□

  □□ → □□□□□□□□□□□□□□□□
""")

    print("□□□")

if __name__ == '__main__':
    main()
```

Script 4: sparc_M3_M6_independence.py

目的

M3（epsilon閾値）と M6（u_strain/E_kin）が相補的か冗長かの判定

7段階テスト: T1: M3残差 vs M6残差の相関。T2: M3+M6 結合モデル（3パラメータ）の dAICc。T2b: LOO-CV で結合モデルが単独より改善するか。T3: 増分 R^2（M3残差を M6 で予測 / M6残差を M3 で予測）。T4: hR 偏相関の段階的改善。T5: M3指標と M6指標の物理的独立性（vflat統制下の相関）。T6: M3/M6/M8 の三角関係。T7: シャッフル独立性テスト。

結果

M6 は M3 の冗長な代替指標。T3: M6 -> M3残差 R^2=0.0000（独立情報ゼロ）。T7: シャッフル p=0.76（偶然と区別不可）。T5: vflat 統制下で rho=0.90（ほぼ同一）。条件14の証拠は「3系列」から「M3の1系列のみ」に縮小。

実行方法

```
uv run --with scipy --with matplotlib python sparc_M3_M6_independence.py
```

入力データ: sparc_gc.csv + Rotmod_LTG/*.dat
出力: M3_M6_independence.png + 標準出力

ソースコード全文

```
#!/usr/bin/env python3
"""
sparc_M3_M6_independence.py
============================
M3 (ε□□) □ M6 (u_strain/E_kin) □□□□□□□□?

□□□:
  T1: M3□□ vs M6□□ □□□ → □□□□□□
  T2: M3+M6 combined model (3 param) → dAICc □□□□□□
  T3: M3□□□ M6 □□□ / M6□□□ M3 □□□ → □□R²
  T4: □□□□□□: M3□□□□□ M6 □ hR□□□□□
  T5: □□□□□□: M3□□□M6□□□□□□□

□□: uv run --with scipy --with matplotlib python sparc_M3_M6_independence.py
"""

import os, sys
import numpy as np
from scipy.optimize import minimize, minimize_scalar
from scipy.stats import spearmanr, pearsonr
import warnings
warnings.filterwarnings('ignore')

BASE = r"D:\□□□□□□\□□□□□□\□□□□□\□□□□□□□\□□□□"
ROTMOD = os.path.join(BASE, "Rotmod_LTG")
GC_CSV = os.path.join(BASE, "sparc_gc.csv")
a0 = 1.2e-10

# ============================================================
# □□□□□□□ (□□)
# ============================================================
def load_gc_csv():
    import csv
    data = {}
    with open(GC_CSV, 'r') as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = None
            for k in row:
                if 'gal' in k.lower() or 'name' in k.lower():
                    name = row[k].strip(); break
            if not name: continue
            gc_val = None
            for k in ['gc','gc_obs','g_c','gc_a0']:
                if k in row and row[k]:
                    try: gc_val = float(row[k]); break
                    except: pass
            if gc_val is None: continue
            vflat, hR, Yd = None, None, None
            for k in ['vflat','Vflat','v_flat']:
                if k in row and row[k]:
                    try: vflat = float(row[k]); break
                    except: pass
            for k in ['hR','Reff','h_R','hr']:
                if k in row and row[k]:
```

```
                    try: hR = float(row[k]); break
                    except: pass
            for k in ['Yd','yd','Y_d','SPS']:
                if k in row and row[k]:
                    try: Yd = float(row[k]); break
                    except: pass
            data[name] = {'gc': gc_val, 'vflat': vflat, 'hR': hR, 'Yd': Yd}
    gc_vals = [d['gc'] for d in data.values()]
    if np.median(gc_vals) &gt; 1e-5:
        for n in data: data[n]['gc'] *= a0
    return data

def load_rotcurve(gname):
    fname = os.path.join(ROTMOD, f"{gname}_rotmod.dat")
    if not os.path.exists(fname): return None
    cols = {k: [] for k in ['r','vobs','vgas','vdisk','vbul']}
    with open(fname, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith('#'): continue
            p = line.split()
            if len(p) &lt; 6: continue
            try:
                cols['r'].append(float(p[0]))
                cols['vobs'].append(float(p[1]))
                cols['vgas'].append(float(p[3]))
                cols['vdisk'].append(float(p[4]))
                cols['vbul'].append(float(p[5]))
            except: continue
    if len(cols['r']) &lt; 5: return None
    return {k: np.array(v) for k, v in cols.items()}

def compute_gc_deep(rc, Yd=0.5):
    r_m = rc['r'] * 3.086e19
    v_bar2 = (Yd * rc['vdisk']**2 + rc['vgas']**2 + rc['vbul']**2) * 1e6
    v_obs2 = (rc['vobs'] * 1e3)**2
    gN = v_bar2 / r_m
    gobs = v_obs2 / r_m
    mask = gN &gt; 0
    if np.sum(mask) &lt; 3: return None
    gc_pts = gobs[mask]**2 / gN[mask]
    gc_med = np.median(gc_pts)
    if gc_med &lt;= 0: return None
    x = gN[mask] / gc_med
    deep = x &lt; 1.0
    return np.median(gc_pts[deep]) if np.sum(deep) &gt;= 3 else gc_med

def compute_quantities(rc, gc_deep, Yd=0.5):
    r_m = rc['r'] * 3.086e19
    v_bar2 = (Yd * rc['vdisk']**2 + rc['vgas']**2 + rc['vbul']**2) * 1e6
    v_obs2 = (rc['vobs'] * 1e3)**2
    gN = np.maximum(v_bar2 / r_m, 0)
    epsilon = np.sqrt(gN / gc_deep)

    u_elastic = np.mean(epsilon**2)
    E_kin = np.mean(v_obs2)
    strain_kin = u_elastic / (E_kin / (3.086e19)**2) if E_kin &gt; 0 else 0
    eps_max = np.max(epsilon)

    if len(epsilon) &gt;= 5:
        d2eps = np.gradient(np.gradient(epsilon, r_m), r_m)
        curvature = np.mean(d2eps**2)
    else:
        curvature = 0

    # E_grad/E_local (ratio from previous sessions)
    deps_dr = np.zeros_like(epsilon)
    deps_dr[0] = (epsilon[1] - epsilon[0]) / (r_m[1] - r_m[0])
    deps_dr[-1] = (epsilon[-1] - epsilon[-2]) / (r_m[-1] - r_m[-2])
    for i in range(1, len(epsilon)-1):
        deps_dr[i] = (epsilon[i+1] - epsilon[i-1]) / (r_m[i+1] - r_m[i-1])
    E_grad = np.mean(deps_dr**2)
    E_local = np.mean(epsilon**2)
    ratio = E_grad / E_local if E_local &gt; 0 else 0

    return {
        'strain_kin': strain_kin, 'eps_max': eps_max, 'curvature': curvature,
        'ratio': ratio, 'eps_mean': np.mean(epsilon),
        'n_points': len(epsilon), 'rmax_hR': rc['r'][-1],
    }

# ============================================================
# □□□□□
# ============================================================
def build_dataset():
    gc_data = load_gc_csv()
    results = []
    for gname in sorted(gc_data.keys()):
        gd = gc_data[gname]
        gc_obs, vflat, hR = gd['gc'], gd.get('vflat'), gd.get('hR')
        Yd = gd.get('Yd') or 0.5
        if gc_obs &lt;= 0 or not vflat or not hR or vflat &lt;= 0 or hR &lt;= 0:
```

```
            continue
        rc = load_rotcurve(gname)
        if rc is None: continue
        gc_deep = compute_gc_deep(rc, Yd)
        if gc_deep is None or gc_deep &lt;= 0: continue
        sq = compute_quantities(rc, gc_deep, Yd)
        if sq['strain_kin'] &lt;= 0: continue
        results.append({
            'name': gname, 'gc_obs': gc_obs, 'gc_deep': gc_deep,
            'vflat': vflat, 'hR': hR, 'Yd': Yd,
            'rmax_hR': rc['r'][-1] / hR if hR &gt; 0 else 0,
            **{k: sq[k] for k in ['strain_kin','eps_max','curvature','ratio','eps_mean']},
        })
    print(f"Dataset: {len(results)} galaxies\n")
    return results

# ============================================================
# □□□□□□□□□
# ============================================================
def fit_M3(log_excess, log_eps_max, mask=None):
    """M3: y = beta * max(0, log(eps_max) - log_ec)"""
    if mask is None: mask = np.ones(len(log_excess), dtype=bool)
    lem = log_eps_max[mask]
    e = log_excess[mask]
    best = None
    for lec in np.linspace(np.percentile(lem, 5), np.percentile(lem, 95), 30):
        for b in np.linspace(-1, 1, 30):
            exc = np.maximum(lem - lec, 0)
            chi2 = np.sum((e - b * exc)**2)
            if best is None or chi2 &lt; best[0]:
                best = (chi2, lec, b)
    if best is None: return None
    res = minimize(lambda p: np.sum((e - p[1]*np.maximum(lem-p[0],0))**2),
                   [best[1], best[2]], method='Nelder-Mead')
    return res.x  # [log_ec, beta]

def pred_M3(params, log_eps_max):
    log_ec, beta = params
    return beta * np.maximum(log_eps_max - log_ec, 0)

def fit_M6(log_excess, log_sk, mask=None):
    """M6: y = a * log(strain_kin)"""
    if mask is None: mask = np.ones(len(log_excess), dtype=bool)
    finite = np.isfinite(log_sk[mask])
    from numpy.linalg import lstsq
    X = log_sk[mask][finite].reshape(-1, 1)
    sol, _, _, _ = lstsq(X, log_excess[mask][finite], rcond=None)
    return sol[0]  # a

def pred_M6(a, log_sk):
    return a * log_sk

# ============================================================
# AICc
# ============================================================
def aicc(chi2, n, k):
    aic = chi2 + 2*k
    if n - k - 1 &gt; 0:
        return aic + 2*k*(k+1) / (n - k - 1)
    return aic

def partial_corr(x, y, z):
    """rho(x,y|z)"""
    cx = np.polyfit(z, x, 1)
    cy = np.polyfit(z, y, 1)
    return spearmanr(x - np.polyval(cx, z), y - np.polyval(cy, z))

def partial_corr_multi(target, y, controls):
    from numpy.linalg import lstsq
    X = np.column_stack([*controls, np.ones(len(target))])
    ct, _, _, _ = lstsq(X, target, rcond=None)
    cy, _, _, _ = lstsq(X, y, rcond=None)
    return spearmanr(target - X @ ct, y - X @ cy)

# ============================================================
# □□□
# ============================================================
def main():
    print("=" * 70)
    print("M3 vs M6 □□□□□□")
    print("=" * 70)

    data = build_dataset()
    N = len(data)

    gc_obs = np.array([d['gc_obs'] for d in data])
    gc_deep = np.array([d['gc_deep'] for d in data])
    strain_kin = np.array([d['strain_kin'] for d in data])
    eps_max = np.array([d['eps_max'] for d in data])
    curvature = np.array([d['curvature'] for d in data])
    ratio = np.array([d['ratio'] for d in data])
    vflat = np.array([d['vflat'] for d in data])
```

```
    hR = np.array([d['hR'] for d in data])
    rmax_hR = np.array([d['rmax_hR'] for d in data])

    log_excess = np.log10(gc_obs / gc_deep)
    log_sk = np.log10(np.maximum(strain_kin, 1e-50))
    log_em = np.log10(np.maximum(eps_max, 1e-10))
    log_curv = np.log10(np.maximum(curvature, 1e-80))
    log_vf = np.log10(vflat)
    log_hR = np.log10(hR)

    chi2_null = np.sum(log_excess**2)

    # --- Fit individual models ---
    params_M3 = fit_M3(log_excess, log_em)
    pred_m3 = pred_M3(params_M3, log_em)
    resid_m3 = log_excess - pred_m3
    chi2_m3 = np.sum(resid_m3**2)

    a_M6 = fit_M6(log_excess, log_sk)
    pred_m6 = pred_M6(a_M6, log_sk)
    resid_m6 = log_excess - pred_m6
    chi2_m6 = np.sum(resid_m6**2)

    print(f"M3: log_ec={params_M3[0]:.3f} (eps_c={10**params_M3[0]:.4f}), "
          f"beta={params_M3[1]:.3f}, R2={1-chi2_m3/chi2_null:.4f}")
    print(f"M6: a={a_M6:.4f}, R2={1-chi2_m6/chi2_null:.4f}")

    # ============================================================
    # T1: □□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("T1: M3□□ vs M6□□ □□□")
    print("=" * 70)

    rho_resid, p_resid = spearmanr(resid_m3, resid_m6)
    rho_resid_p, p_resid_p = pearsonr(resid_m3, resid_m6)
    print(f"Spearman rho(resid_M3, resid_M6) = {rho_resid:+.4f} (p={p_resid:.2e})")
    print(f"Pearson  r(resid_M3, resid_M6)   = {rho_resid_p:+.4f} (p={p_resid_p:.2e})")

    if abs(rho_resid) &gt; 0.8:
        print("-&gt; □□□□: M3□M6□□□□□")
    elif abs(rho_resid) &gt; 0.5:
        print("-&gt; □□□□□□: □□□□□□□□□□□□□")
    else:
        print("-&gt; □□□□: M3□M6□□□□")

    # ============================================================
    # T2: Combined model M3+M6 (3 params)
    # ============================================================
    print("\n" + "=" * 70)
    print("T2: Combined model M3+M6 (3 params)")
    print("=" * 70)

    def fit_combined(le, lem, lsk, mask=None):
        if mask is None: mask = np.ones(len(le), dtype=bool)
        e = le[mask]
        em = lem[mask]
        sk = lsk[mask]

        def chi2_comb(p):
            log_ec, beta, a = p
            pred = beta * np.maximum(em - log_ec, 0) + a * sk
            return np.sum((e - pred)**2)

        best = None
        for lec_init in np.linspace(np.percentile(em, 10), np.percentile(em, 90), 10):
            for beta_init in [-0.3, -0.1, 0.1, 0.3]:
                for a_init in [-0.01, -0.005, 0.005, 0.01]:
                    res = minimize(chi2_comb, [lec_init, beta_init, a_init],
                                   method='Nelder-Mead', options={'maxiter': 5000})
                    if best is None or res.fun &lt; best.fun:
                        best = res
        return best.x, best.fun

    params_comb, chi2_comb = fit_combined(log_excess, log_em, log_sk)
    R2_comb = 1 - chi2_comb / chi2_null

    aicc_null = aicc(chi2_null, N, 0)
    aicc_m3 = aicc(chi2_m3, N, 2)
    aicc_m6 = aicc(chi2_m6, N, 1)
    aicc_comb = aicc(chi2_comb, N, 3)

    print(f"Combined: log_ec={params_comb[0]:.3f}, beta={params_comb[1]:.3f}, "
          f"a_M6={params_comb[2]:.4f}")
    print(f"\nModel comparison:")
    print(f"  {'Model':&lt;25s} {'k':&gt;3s} {'chi2':&gt;8s} {'R2':&gt;7s} {'AICc':&gt;8s} {'dAICc':&gt;8s}")
    print(f"  {'-'*60}")
    print(f"  {'M0: null':&lt;25s} {'0':&gt;3s} {chi2_null:&gt;8.2f} {'0.000':&gt;7s} "
          f"{aicc_null:&gt;8.1f} {'0.0':&gt;8s}")
    print(f"  {'M6: u_strain/E_kin':&lt;25s} {'1':&gt;3s} {chi2_m6:&gt;8.2f} "
          f"{1-chi2_m6/chi2_null:&gt;7.4f} {aicc_m6:&gt;8.1f} {aicc_m6-aicc_null:&gt;+8.1f}")
    print(f"  {'M3: eps threshold':&lt;25s} {'2':&gt;3s} {chi2_m3:&gt;8.2f} "
```

```
          f"{1-chi2_m3/chi2_null:&gt;7.4f} {aicc_m3:&gt;8.1f} {aicc_m3-aicc_null:&gt;+8.1f}")
    print(f"  {'M3+M6: combined':&lt;25s} {'3':&gt;3s} {chi2_comb:&gt;8.2f} "
          f"{R2_comb:&gt;7.4f} {aicc_comb:&gt;8.1f} {aicc_comb-aicc_null:&gt;+8.1f}")

    dAICc_comb_vs_m3 = aicc_comb - aicc_m3
    dAICc_comb_vs_m6 = aicc_comb - aicc_m6
    print(f"\n  M3+M6 vs M3 alone: dAICc = {dAICc_comb_vs_m3:+.1f}")
    print(f"  M3+M6 vs M6 alone: dAICc = {dAICc_comb_vs_m6:+.1f}")

    if dAICc_comb_vs_m3 &lt; -2 and dAICc_comb_vs_m6 &lt; -2:
        print("  -&gt; □□□□□□□□: M3□M6□□□□")
    elif dAICc_comb_vs_m3 &gt;= 0 and dAICc_comb_vs_m6 &gt;= 0:
        print("  -&gt; □□□□□□□□: □□□□□□")
    else:
        best_single = "M3" if chi2_m3 &lt; chi2_m6 else "M6"
        print(f"  -&gt; {best_single} □□□□□□□□□□□□")

    # ============================================================
    # T2b: LOO-CV for combined model
    # ============================================================
    print("\n--- LOO-CV for combined model ---")
    loo_errors_comb = []
    loo_errors_m3 = []
    loo_errors_m6 = []
    for i in range(N):
        mask = np.ones(N, dtype=bool)
        mask[i] = False

        # Combined
        try:
            p_c, _ = fit_combined(log_excess, log_em, log_sk, mask)
            pred_i = p_c[1] * max(log_em[i] - p_c[0], 0) + p_c[2] * log_sk[i]
            loo_errors_comb.append((log_excess[i] - pred_i)**2)
        except:
            pass

        # M3
        p3 = fit_M3(log_excess, log_em, mask)
        if p3 is not None:
            pred_i3 = pred_M3(p3, np.array([log_em[i]]))[0]
            loo_errors_m3.append((log_excess[i] - pred_i3)**2)

        # M6
        a6 = fit_M6(log_excess, log_sk, mask)
        pred_i6 = a6 * log_sk[i]
        loo_errors_m6.append((log_excess[i] - pred_i6)**2)

    mse_null = np.mean(log_excess**2)
    mse_comb = np.mean(loo_errors_comb) if loo_errors_comb else 999
    mse_m3 = np.mean(loo_errors_m3) if loo_errors_m3 else 999
    mse_m6 = np.mean(loo_errors_m6) if loo_errors_m6 else 999

    print(f"  LOO-CV MSE: null={mse_null:.5f}, M6={mse_m6:.5f}, "
          f"M3={mse_m3:.5f}, M3+M6={mse_comb:.5f}")
    print(f"  LOO ratio:  M6={mse_m6/mse_null:.3f}, M3={mse_m3/mse_null:.3f}, "
          f"M3+M6={mse_comb/mse_null:.3f}")

    if mse_comb &lt; min(mse_m3, mse_m6) * 0.95:
        print("  -&gt; Combined □ 5%+ □□: □□□□□□□")
    else:
        print("  -&gt; Combined □□□□□□: □□□□□□")

    # ============================================================
    # T3: □□R² (□□□□)
    # ============================================================
    print("\n" + "=" * 70)
    print("T3: □□ R2 □□□")
    print("=" * 70)

    # M3□□□M6□□□□□
    from numpy.linalg import lstsq
    X_m6 = log_sk.reshape(-1, 1)
    sol_m6on3, _, _, _ = lstsq(X_m6, resid_m3, rcond=None)
    pred_m6on3 = sol_m6on3[0] * log_sk
    R2_m6on3 = 1 - np.sum((resid_m3 - pred_m6on3)**2) / np.sum(resid_m3**2)

    # M6□□□M3□□□□□
    exc_m3 = np.maximum(log_em - params_M3[0], 0)
    X_m3 = exc_m3.reshape(-1, 1)
    sol_m3on6, _, _, _ = lstsq(X_m3, resid_m6, rcond=None)
    pred_m3on6 = sol_m3on6[0] * exc_m3
    R2_m3on6 = 1 - np.sum((resid_m6 - pred_m3on6)**2) / np.sum(resid_m6**2)

    print(f"M6 on M3 residuals: R2 = {R2_m6on3:.4f} "
          f"({'M6 adds info' if R2_m6on3 &gt; 0.05 else 'negligible'})")
    print(f"M3 on M6 residuals: R2 = {R2_m3on6:.4f} "
          f"({'M3 adds info' if R2_m3on6 &gt; 0.05 else 'negligible'})")

    # ============================================================
    # T4: hR□□□□□□□□
    # ============================================================
    print("\n" + "=" * 70)
```

```
    print("T4: hR□□□□□□□□□")
    print("=" * 70)

    gc_m3 = gc_deep * 10**pred_m3
    gc_m6 = gc_deep * 10**pred_m6
    gc_comb = gc_deep * 10**(params_comb[1]*np.maximum(log_em-params_comb[0],0)
                             + params_comb[2]*log_sk)

    log_gc_obs = np.log10(gc_obs)
    log_gc_m3 = np.log10(np.maximum(gc_m3, 1e-30))
    log_gc_m6 = np.log10(np.maximum(gc_m6, 1e-30))
    log_gc_comb = np.log10(np.maximum(gc_comb, 1e-30))

    rho_obs, _ = partial_corr(log_gc_obs, log_hR, log_vf)
    rho_m3, _ = partial_corr(log_gc_m3, log_hR, log_vf)
    rho_m6, _ = partial_corr(log_gc_m6, log_hR, log_vf)
    rho_comb, _ = partial_corr(log_gc_comb, log_hR, log_vf)

    print(f"  rho(gc, hR | vflat):")
    print(f"    gc_obs:       {rho_obs:+.4f}")
    print(f"    gc_M3:        {rho_m3:+.4f} (□□ {(1-abs(rho_m3)/abs(rho_obs))*100:.1f}%)")
    print(f"    gc_M6:        {rho_m6:+.4f} (□□ {(1-abs(rho_m6)/abs(rho_obs))*100:.1f}%)")
    print(f"    gc_M3+M6:     {rho_comb:+.4f} (□□ {(1-abs(rho_comb)/abs(rho_obs))*100:.1f}%)")

    # + rmax/hR □□
    rho_obs2, _ = partial_corr_multi(log_gc_obs, log_hR, [log_vf, rmax_hR])
    rho_comb2, _ = partial_corr_multi(log_gc_comb, log_hR, [log_vf, rmax_hR])
    print(f"\n  rho(gc, hR | vflat, rmax/hR):")
    print(f"    gc_obs:       {rho_obs2:+.4f}")
    print(f"    gc_M3+M6:     {rho_comb2:+.4f} (□□ {(1-abs(rho_comb2)/abs(rho_obs2))*100:.1f}%)")

    # ============================================================
    # T5: □□□□□□ — □□□□□□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("T5: M3□□ vs M6□□ □□□□□□□")
    print("=" * 70)

    rho_indicators, p_ind = spearmanr(log_em, log_sk)
    print(f"  rho(log_eps_max, log_strain_kin) = {rho_indicators:+.4f} (p={p_ind:.2e})")

    # M3 indicator = max(0, log_em - log_ec)
    m3_indicator = np.maximum(log_em - params_M3[0], 0)
    rho_m3_m6, p_m3_m6 = spearmanr(m3_indicator, log_sk)
    print(f"  rho(M3_indicator, log_strain_kin) = {rho_m3_m6:+.4f} (p={p_m3_m6:.2e})")

    # □□□: vflat□□□
    rho_ind_vf, p_ind_vf = partial_corr(log_em, log_sk, log_vf)
    print(f"  rho(log_eps_max, log_strain_kin | vflat) = {rho_ind_vf:+.4f} (p={p_ind_vf:.2e})")

    if abs(rho_indicators) &gt; 0.8:
        print("  -&gt; □□□□□□□□: M3□M6□□□□□□□□□")
    elif abs(rho_indicators) &gt; 0.5:
        print("  -&gt; □□□□□: □□□□□□□□□□□□□□")
    else:
        print("  -&gt; □□□: □□□□□□□□□")

    # ============================================================
    # T6: M8 (curvature) □□□□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("T6: M3/M6/M8 □□□□□")
    print("=" * 70)

    rho_36, _ = spearmanr(log_em, log_sk)
    rho_38, _ = spearmanr(log_em, log_curv)
    rho_68, _ = spearmanr(log_sk, log_curv)
    print(f"  rho(M3, M6) = {rho_36:+.3f}")
    print(f"  rho(M3, M8) = {rho_38:+.3f}")
    print(f"  rho(M6, M8) = {rho_68:+.3f}")

    if abs(rho_68) &gt; 0.9:
        print("  -&gt; M6□M8□□□□□□□□")
    if abs(rho_36) &lt; abs(rho_68):
        print("  -&gt; M3□M6/M8□□□□□□□□□")

    # ============================================================
    # T7: □□□□□□□□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("T7: □□□□□□□□□□")
    print("=" * 70)
    print("M3□□□□□□□□M6□□□□□□□□□R2□□□□□□")

    np.random.seed(42)
    R2_shuffle = []
    for _ in range(1000):
        sk_s = np.random.permutation(log_sk)
        sol_s, _, _, _ = lstsq(sk_s.reshape(-1,1), resid_m3, rcond=None)
        pred_s = sol_s[0] * sk_s
        R2_s = 1 - np.sum((resid_m3 - pred_s)**2) / np.sum(resid_m3**2)
        R2_shuffle.append(R2_s)
```

```
|
|     R2_shuffle = np.array(R2_shuffle)
|     p_shuffle = np.mean(R2_shuffle &gt;= R2_m6on3)
|     print(f"  Real R2(M6 on M3 residuals) = {R2_m6on3:.4f}")
|     print(f"  Shuffle: mean={np.mean(R2_shuffle):.4f}, max={np.max(R2_shuffle):.4f}")
|     print(f"  p = {p_shuffle:.4f}")
|     if p_shuffle &lt; 0.01:
|         print("  -&gt; M6□M3□□□□□□□□□□□□□□□□")
|     else:
|         print("  -&gt; M6□M3□□□□□□□□□□□□□□□□□□□")
|
|     # ============================================================
|     # □□□□
|     # ============================================================
|     print("\n" + "=" * 70)
|     print("□□□□")
|     print("=" * 70)
|
|     print(f"""
| □□□□□:
|   T1 □□□□:       rho = {rho_resid:+.4f}
|   T2 Combined dAICc: vs M3 = {dAICc_comb_vs_m3:+.1f}, vs M6 = {dAICc_comb_vs_m6:+.1f}
|   T2b LOO-CV ratio:  M3={mse_m3/mse_null:.3f}, M6={mse_m6/mse_null:.3f}, Comb={mse_comb/mse_null:.3f}
|   T3 □□R2:         M6-&gt;M3□□ = {R2_m6on3:.4f}, M3-&gt;M6□□ = {R2_m3on6:.4f}
|   T4 hR□□□:       obs={rho_obs:+.4f}, M3={rho_m3:+.4f}, M6={rho_m6:+.4f}, Comb={rho_comb:+.4f}
|   T5 □□□□:       rho(M3,M6) = {rho_indicators:+.4f}
|   T6 □□□□:       M6-M8 = {rho_68:+.3f}
|   T7 □□□□□:     p = {p_shuffle:.4f}
|
| □□□□:
|   □□□: □□□□ &lt; 0.5, Combined dAICc &lt; -2 vs both, □□R2 &gt; 0.05, p(T7) &lt; 0.01
|   □□:   □□□□ &gt; 0.8, Combined dAICc &gt;= 0 vs best, □□R2 &lt; 0.02
|   □□:   □□
| """)
|
|     # ============================================================
|     # □
|     # ============================================================
|     try:
|         import matplotlib
|         matplotlib.use('Agg')
|         import matplotlib.pyplot as plt
|
|         fig, axes = plt.subplots(2, 2, figsize=(12, 10))
|         fig.suptitle('M3 vs M6 Independence Test', fontsize=14)
|
|         # (a) M3 residual vs M6 residual
|         ax = axes[0, 0]
|         ax.scatter(resid_m3, resid_m6, s=10, alpha=0.5, c='steelblue')
|         ax.set_xlabel('M3 residual')
|         ax.set_ylabel('M6 residual')
|         ax.set_title(f'Residual correlation (rho={rho_resid:+.3f})')
|         lim = max(abs(resid_m3).max(), abs(resid_m6).max()) * 1.1
|         ax.set_xlim(-lim, lim)
|         ax.set_ylim(-lim, lim)
|         ax.plot([-lim, lim], [-lim, lim], 'k--', lw=0.5)
|
|         # (b) M3 indicator vs M6 indicator
|         ax = axes[0, 1]
|         ax.scatter(log_em, log_sk, s=10, alpha=0.5, c='steelblue')
|         ax.set_xlabel('log(eps_max) [M3]')
|         ax.set_ylabel('log(strain_kin) [M6]')
|         ax.set_title(f'Indicator correlation (rho={rho_indicators:+.3f})')
|
|         # (c) hR partial correlation comparison
|         ax = axes[1, 0]
|         labels = ['gc_obs', 'M3', 'M6', 'M3+M6']
|         rhos = [abs(rho_obs), abs(rho_m3), abs(rho_m6), abs(rho_comb)]
|         colors_bar = ['grey', 'steelblue', 'coral', 'green']
|         ax.bar(labels, rhos, color=colors_bar, alpha=0.7)
|         ax.set_ylabel('|rho(gc, hR | vflat)|')
|         ax.set_title('hR partial correlation (lower = better)')
|
|         # (d) LOO-CV comparison
|         ax = axes[1, 1]
|         labels2 = ['null', 'M6', 'M3', 'M3+M6']
|         mses = [mse_null, mse_m6, mse_m3, mse_comb]
|         ax.bar(labels2, mses, color=['grey', 'coral', 'steelblue', 'green'], alpha=0.7)
|         ax.set_ylabel('LOO-CV MSE')
|         ax.set_title('Cross-validation (lower = better)')
|
|         plt.tight_layout()
|         figpath = os.path.join(BASE, 'M3_M6_independence.png')
|         plt.savefig(figpath, dpi=150)
|         print(f"\nFigure saved: {figpath}")
|     except Exception as e:
|         print(f"\nFigure error: {e}")
|
|     print("\n□□□")
|
| if __name__ == '__main__':
|     main()
```

Script 5: epsilon_c_derivation.py

目的

M3 の epsilon_c ˜= 0.072 を U(epsilon;c) の構造から理論的に導出する試み

11仮説を系統テスト: H1: U 変曲点 U''=0、H2: 非調和応答開始、H3: 降伏点、H4: Duffing 非線形パラメータ、H5: MOND 補間展開、H6: 有限ディスク端効果、H7: c 分布、H8: 理論定数照合、H9: x_c=1/200、H10: 遷移構造、H11: Taylor 弾性限界。データ不要の純粋理論計算（コンテナ内実行）。

結果

全て不成功（X級）。根本原因: epsilon_proxy = sqrt(gN/gc) と epsilon_fold の非線形写像により、U(epsilon;c) の全特異点が epsilon_proxy ˜= 0 に帰着する。H9 のみ保留: 1/(10 sqrt(2)) = 0.0707 vs 観測 0.072（差 1.8%）だが理論的根拠なし。epsilon_c は経験的パラメータとして記録。

実行方法

```
python epsilon_c_derivation.py□□□□□□ or □□□□□□□□□□
```

入力データ: なし（純粋理論計算）
出力: epsilon_c_analysis.png + 標準出力

ソースコード全文

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
epsilon_c_derivation.py
=======================
M3 □ ε_c ≈ 0.072 □ U(ε;c) □□□□□□□□□□□?

ε_proxy = sqrt(gN/gc_deep) □□□□U(ε;c) □ ε_fold □□□□□□
□□□□: x = gN/gc = (c-1+ε_fold²)/(1-ε_fold)
→ ε_proxy = sqrt(x) = sqrt((c-1+ε_fold²)/(1-ε_fold))

□□:
  H1: U □□□□ U''=0
  H2: □□□□□□□□ (anharmonic/harmonic ratio)
  H3: □□-□□□□□□□□
  H4: free energy landscape □□□□□□□□
  H5: MOND □□□□□□□□□□□□□□□
  H6: □□□□□□ + □□□□□□□□
"""

import numpy as np
from scipy.optimize import brentq, minimize_scalar
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 1. U(ε;c) □□□□□
# ============================================================
def U(eps, c):
    """Free energy"""
    return -eps - eps**2/2 - c * np.log(1 - eps)

def U1(eps, c):
    """U' = dU/dε"""
    return -1 - eps + c / (1 - eps)

def U2(eps, c):
    """U'' = d²U/dε²"""
    return -1 + c / (1 - eps)**2

def U3(eps, c):
    """U''' = d³U/dε³"""
    return 2 * c / (1 - eps)**3

def U4(eps, c):
    """U'''' = d⁴U/dε⁴"""
    return 6 * c / (1 - eps)**4

# □□□□: U'(ε_eq) = 0 → ε_eq
def eps_eq(c):
    """U'(ε;c) = 0 □□"""
    if c &gt;= 1:
        return 0.0
    # -1 - ε + c/(1-ε) = 0 → ε² - ε + (c-1) = 0... no
    # Actually: c/(1-ε) = 1+ε → c = (1-ε)(1+ε) = 1-ε²
    # → ε = sqrt(1-c)
```

```
    return np.sqrt(max(1 - c, 0))

# x = gN/gc from equilibrium
def x_from_eps_fold(eps_f, c):
    """gN/gc = (c - 1 + ε²)/(1 - ε)"""
    return (c - 1 + eps_f**2) / (1 - eps_f)

# ε_proxy = sqrt(x)
def eps_proxy_from_fold(eps_f, c):
    x = x_from_eps_fold(eps_f, c)
    return np.sqrt(max(x, 0))

# ε_fold from ε_proxy (inverse)
def eps_fold_from_proxy(eps_p, c):
    """x = eps_p² = (c-1+ε²)/(1-ε)
    → ε² + eps_p²ε + (c-1-eps_p²) = 0"""
    x = eps_p**2
    # ε² + xε - (1-c+x) = 0 ... let me redo
    # (c-1+ε²)/(1-ε) = x
    # c - 1 + ε² = x(1-ε) = x - xε
    # ε² + xε + (c-1-x) = 0
    a, b, cc2 = 1, x, (c - 1 - x)
    disc = b**2 - 4*a*cc2
    if disc &lt; 0:
        return None
    return (-b + np.sqrt(disc)) / (2*a)

print("=" * 70)
print("ε_c □□□□□□: U(ε;c) □□□□□□□□□□")
print("=" * 70)

# ============================================================
# 2. □□□□ U(ε;c) □□□
# ============================================================
print("\n--- □□□□ ---")
for c in [0.1, 0.24, 0.5, 0.8, 1.0]:
    eeq = eps_eq(c)
    u2 = U2(eeq, c) if eeq &lt; 0.99 else float('inf')
    print(f"  c={c:.2f}: ε_eq={eeq:.4f}, U''(ε_eq)={u2:.4f}")

# ============================================================
# H1: U''=0 □□□□
# ============================================================
print("\n" + "=" * 70)
print("H1: U □□□□ U''(ε;c) = 0")
print("=" * 70)
print("U''(ε;c) = -1 + c/(1-ε)²")
print("U'' = 0 → ε_infl = 1 - sqrt(c)")

for c in [0.1, 0.24, 0.5, 0.8, 1.0]:
    eps_infl = 1 - np.sqrt(c)
    eps_p = eps_proxy_from_fold(eps_infl, c) if eps_infl &gt; 0 else 0
    print(f"  c={c:.2f}: ε_fold_infl={eps_infl:.4f}, ε_proxy_infl={eps_p:.4f}")

print("\n  → □□□□ ε_proxy □ c □□□□□□□0.072 □□□□□□□H1 □□□")

# ============================================================
# H2: □□□□ |U'''δε| / |U''| = 1 □□□
# ============================================================
print("\n" + "=" * 70)
print("H2: □□□□□□□□")
print("=" * 70)
print("ε_eq □□□□□ δε □ |U'''(ε_eq)δε/U''(ε_eq)| = 1 □□□ δε")

for c in [0.1, 0.24, 0.5, 0.8, 1.0]:
    eeq = eps_eq(c)
    if eeq &gt;= 0.99: continue
    u2v = U2(eeq, c)
    u3v = U3(eeq, c)
    if abs(u3v) &gt; 0:
        delta_eps = abs(u2v / u3v)
        # □□□□□□□ ε_fold
        eps_anh = eeq - delta_eps  # ε &lt; ε_eq □□
        if eps_anh &gt; 0:
            eps_p = eps_proxy_from_fold(eps_anh, c)
        else:
            eps_p = None
        ep_str = f"{eps_p:.4f}" if eps_p is not None else "N/A"
        print(f"  c={c:.2f}: eps_eq={eeq:.4f}, delta_eps={delta_eps:.4f}, "
              f"eps_fold_anh={eps_anh:.4f}, eps_proxy={ep_str}")

# ============================================================
# H3: □□-□□□□□□□□
# ============================================================
print("\n" + "=" * 70)
print("H3: □□-□□□□□□□□ (stress-strain yield)")
print("=" * 70)
print("σ(ε) = U'(ε;c) □□□□□□ |σ_max| □□□")
print("dσ/dε = U''(ε;c) = 0 → ε_yield = 1 - sqrt(c) (= H1□□□)")
print("→ H1□□□□□□□□□□□□□□□")

# □□□□□□□□□□□99%□□□□□□□□□□□□□□
```

```
print("\n  ▫▫▫▫: σ(ε)/σ_linear ▫▫▫▫10%▫▫▫▫")
for c in [0.1, 0.24, 0.5, 0.8, 1.0]:
    eeq = eps_eq(c)
    if eeq &gt;= 0.99: continue
    u2v = U2(eeq, c)

    # σ_linear(δε) = U''(ε_eq) × δε
    # σ_actual(δε) = U'(ε_eq + δε) - U'(ε_eq) = U'(ε_eq + δε)  [since U'(ε_eq)=0]
    # deviation = |σ_actual - σ_linear| / |σ_linear| = 10%

    for delta in np.linspace(0.001, 0.5, 1000):
        eps_test = eeq - delta  # going toward smaller ε (larger gN)
        if eps_test &lt;= 0.01: break
        sigma_lin = u2v * (-delta)
        sigma_act = U1(eps_test, c)
        if abs(sigma_lin) &gt; 0:
            dev = abs(sigma_act - sigma_lin) / abs(sigma_lin)
            if dev &gt; 0.1:
                x_yield = x_from_eps_fold(eps_test, c)
                eps_p_yield = np.sqrt(max(x_yield, 0))
                print(f"  c={c:.2f}: 10%▫▫ at δε={delta:.4f}, "
                      f"ε_fold={eps_test:.4f}, x={x_yield:.4f}, ε_proxy={eps_p_yield:.4f}")
                break

# ============================================================
# H4: U'''²/(U''×U'''') = 1 (Duffing▫▫▫▫▫▫▫▫)
# ============================================================
print("\n" + "=" * 70)
print("H4: Duffing▫▫▫▫▫▫▫▫ γ = U'''²/(U''×U'''')")
print("=" * 70)

for c in [0.1, 0.24, 0.5, 0.8, 1.0]:
    eeq = eps_eq(c)
    if eeq &gt;= 0.99: continue
    u2v = U2(eeq, c)
    u3v = U3(eeq, c)
    u4v = U4(eeq, c)
    gamma = u3v**2 / (u2v * u4v) if u2v * u4v != 0 else 0
    print(f"  c={c:.2f}: γ(ε_eq) = {gamma:.4f}")

print("  → γ ▫ c ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫")

# ============================================================
# H5: MOND▫▫▫3▫▫▫
# ============================================================
print("\n" + "=" * 70)
print("H5: MOND▫▫▫▫▫▫▫▫▫▫")
print("=" * 70)
print("g_eff/gc = (x + sqrt(x² + 4x))/2 ▫ x=ε_proxy² ▫▫▫")
print("→ g_eff/gc ≈ sqrt(x) + x/(4sqrt(x)) - x²/(32x^(3/2)) + ...")
print("→ = sqrt(x)(1 + sqrt(x)/4 - x/(32) + ...)")
print("3▫▫/2▫▫ = 1 ▫▫▫ x:")
# 2nd order correction: x/4 at order sqrt(x)
# ratio of 2nd to 1st: sqrt(x)/4
# ratio = 1 → sqrt(x) = 4 → x = 16 (Newtonian regime, not relevant)
# ratio = 0.1 → sqrt(x) = 0.4 → x = 0.16

# More carefully:
# ν(x) = (1 + sqrt(1+4/x))/2  (MOND interpolation)
# For x &lt;&lt; 1: ν ≈ 1/sqrt(x) - 1/(2x) + ... so g_eff = gN × ν ≈ sqrt(gN×gc)
# The correction to deep-MOND:
# g_eff = sqrt(gN×gc) × (1 + sqrt(gN/gc)/4 + ...)
# The relative correction = sqrt(x)/4

# At ε_proxy = 0.072: x = 0.00518, sqrt(x) = 0.072
# correction = 0.072/4 = 0.018 = 1.8%

print(f"\n  ε_proxy=0.072 → x=0.00518")
print(f"  MOND 2▫▫▫ = sqrt(x)/4 = {0.072/4:.4f} = 1.8%")
print(f"  → ▫▫▫▫▫▫▫MOND▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫")

# ============================================================
# H6: ▫▫▫▫▫▫ — ▫▫▫▫▫▫▫▫▫▫
# ============================================================
print("\n" + "=" * 70)
print("H6: ▫▫▫▫▫▫▫▫▫")
print("=" * 70)
print("gN(r) = vflat²/r × Σ▫▫")
print("ε_proxy(r) = sqrt(gN(r)/gc)")
print("ε_proxy_c = 0.072 → gN/gc = 0.005")
print("→ ▫▫▫ r/hR ▫▫▫▫▫▫▫▫▫▫▫?")

# For exponential disk: gN ▫ exp(-r/hR) × ▫▫▫ / r
# At large r: gN → 0 monotonically
# gN(r)/gc ≈ (vflat²/r) × (r/hR)² × exp(-r/hR) / gc  (▫▫)
# The peak of gN is around r ~ 2.2 hR
# gN_peak / gc ≈ typical gc_obs/a0 ≈ 0.24 means gN_peak ≈ 0.24×a0×f
# where f depends on the galaxy

print("  ε_proxy_c = 0.072 ▫ gN/gc ≈ 0.005 ▫▫▫")
print("  ▫▫▫▫▫ (gc~0.24 a0) ▫▫▫▫▫ r/hR ~ 5-8 ▫▫▫▫▫")
print("  → ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫")
```

```
242 |
243 | # ============================================================
244 | # H7: ▫ c ▫▫▫▫▫▫▫▫
245 | # ============================================================
246 | print("\n" + "=" * 70)
247 | print("H7: c ▫▫▫▫▫▫▫▫▫▫ ▫")
248 | print("=" * 70)
249 |
250 | # ▫▫: gc/a0 ▫▫▫▫ = 0.24 → c_median = 0.24
251 | # gc/a0 ▫▫▫▫▫▫▫▫ ~1.1 dex scatter
252 | # c ▫▫▫: ~0.01 ~ 5
253 |
254 | # ε_eq(c) = sqrt(1-c) for c &lt; 1
255 | # ε_proxy at equilibrium = sqrt(x_eq) where x_eq = (c-1+ε_eq²)/(1-ε_eq)
256 | # At equilibrium: x_eq = 0 (since U'(ε_eq)=0 means gN/gc = 0 at equilibrium!)
257 | # Wait, that's the free equilibrium without external gN.
258 |
259 | # Actually, ε_eq is the equilibrium fold WITHOUT external gravitational field.
260 | # With gN present, ε shifts from ε_eq.
261 |
262 | # The question: at what x = gN/gc does the response become non-linear?
263 | # Small-x expansion around equilibrium:
264 | # x ≈ U''(ε_eq;c)/1 × δε + (1/2)U'''(ε_eq;c)/...
265 | # This is getting complicated. Let me do it numerically.
266 |
267 | print("\n  ε_fold(ε_proxy, c) ▫▫▫▫▫▫▫▫▫▫▫:")
268 | print(f"  {'c':&gt;6s} {'ε_eq':&gt;8s} {'ε_fold(0.072)':&gt;14s} "
269 |       f"{'U2/U2_eq':&gt;10s} {'▫▫▫':&gt;8s}")
270 |
271 | results_h7 = []
272 | for c in np.linspace(0.05, 0.95, 19):
273 |     eeq = eps_eq(c)
274 |     ef = eps_fold_from_proxy(0.072, c)
275 |     if ef is None or ef &lt;= 0 or ef &gt;= 1:
276 |         continue
277 |     u2_eq = U2(eeq, c)
278 |     u2_at = U2(ef, c)
279 |     ratio = u2_at / u2_eq if u2_eq != 0 else 0
280 |
281 |     # ▫▫▫: (ε_eq - ε_fold) / ε_eq = how far from equilibrium
282 |     elastic_ratio = (eeq - ef) / eeq if eeq &gt; 0 else 0
283 |
284 |     print(f"  {c:6.3f} {eeq:8.4f} {ef:14.4f} {ratio:10.4f} {elastic_ratio:8.4f}")
285 |     results_h7.append((c, eeq, ef, ratio, elastic_ratio))
286 |
287 | # ============================================================
288 | # H8: ▫▫ 1/sqrt(6) ▫▫▫▫
289 | # ============================================================
290 | print("\n" + "=" * 70)
291 | print("H8: ▫▫▫▫▫▫▫▫▫ ▫▫")
292 | print("=" * 70)
293 |
294 | candidates = {
295 |     '1/sqrt(6) (Tc=sqrt(6)▫▫)': 1/np.sqrt(6),
296 |     '1/(4pi)': 1/(4*np.pi),
297 |     'sqrt(a0/c_light) dimensionless': np.sqrt(1.2e-10 / 3e8),  # ~2e-10, too small
298 |     '1/sqrt(Tc²) = 1/sqrt(6)': 1/np.sqrt(6),
299 |     '(1-1/e)/4': (1-1/np.e)/4,
300 |     'sqrt(1-Tc/3) = sqrt(1-sqrt(6)/3)': np.sqrt(max(1-np.sqrt(6)/3, 0)),
301 |     '1/sqrt(3×Tc) = 1/sqrt(3sqrt(6))': 1/np.sqrt(3*np.sqrt(6)),
302 |     'exp(-Tc) = exp(-sqrt(6))': np.exp(-np.sqrt(6)),
303 |     '1/(2Tc) = 1/(2sqrt(6))': 1/(2*np.sqrt(6)),
304 |     '1/Tc² = 1/6': 1/6,
305 |     'sqrt(1/Tc²) = 1/sqrt(6)': 1/np.sqrt(6),
306 |     '(Tc-2)/Tc = (sqrt(6)-2)/sqrt(6)': (np.sqrt(6)-2)/np.sqrt(6),
307 | }
308 |
309 | print(f"  ▫▫▫: ε_c = 0.072 +/- ?")
310 | print(f"\n  {'▫▫':40s} {'▫':&gt;10s} {'|▫|':&gt;10s}")
311 | print("  " + "-"*62)
312 | for name, val in sorted(candidates.items(), key=lambda x: abs(x[1]-0.072)):
313 |     diff = abs(val - 0.072)
314 |     marker = " ▫" if diff &lt; 0.01 else ""
315 |     print(f"  {name:40s} {val:10.5f} {diff:10.5f}{marker}")
316 |
317 | # ============================================================
318 | # H9: ▫▫▫ x_c = ε_c² ≈ 0.005 ▫▫▫
319 | # ============================================================
320 | print("\n" + "=" * 70)
321 | print("H9: x_c = ε_c² ≈ 0.005 ▫▫▫▫▫▫ ▫▫▫")
322 | print("=" * 70)
323 |
324 | print("""
325 |   ε_c² = 0.072² = 0.00518
326 |
327 |   ▫MOND▫▫▫2▫▫▫:
328 |     g_eff = sqrt(gN × gc) × [1 + (1/4)sqrt(gN/gc) - (1/32)(gN/gc) + ...]
329 |
330 |   x_c = 0.005 ▫▫▫▫:
331 |     1▫: sqrt(0.005)/4 = 0.018 = 1.8%
332 |
333 |   → ▫MOND▫▫▫2▫▫▫▫1-2%▫▫▫▫▫▫▫
```

```
|
|   ε_c² ≈ 1/200 ≈ 0.005
|   → ε_c ≈ 1/sqrt(200) = 1/(10sqrt(2)) ≈ 0.0707
| """)
|
| val_theory = 1/(10*np.sqrt(2))
| print(f"  □□□: 1/(10sqrt(2)) = {val_theory:.5f}")
| print(f"  □□□: ε_c ≈ 0.072")
| print(f"  □: {abs(0.072 - val_theory):.5f} ({abs(0.072-val_theory)/0.072*100:.1f}%)")
|
| # But 1/(10sqrt(2)) is ad hoc. Let's look deeper.
|
| # ============================================================
| # H10: □□□ gc_deep vs gc_obs □□□
| # ============================================================
| print("\n" + "=" * 70)
| print("H10: deep-MOND → full MOND □□□□□")
| print("=" * 70)
| print("""
|   gc_obs □ gc_deep × (□□□□) □□□□□□
|   M3 □□□□□: ε_max &gt; ε_c □ gc □ 23% □□□
|
|   ε_max = sqrt(gN_max / gc_deep) ≈ sqrt(vflat² / (hR × gc_deep))  [r~hR□□]
|
|   ε_max &gt; ε_c □□□:
|     gN_max / gc_deep &gt; ε_c²
|     → gN_max &gt; ε_c² × gc_deep
|
|   gc_deep ~ gc_obs (deep MOND) ~ η×sqrt(a0 × vflat²/hR) □□□:
|     gN_max ~ vflat²/hR (hR□□□□□□)
|     gc_deep ~ η×sqrt(a0 × vflat²/hR)
|
|     ε_max² = gN_max/gc_deep ~ (vflat²/hR) / (η×sqrt(a0 × vflat²/hR))
|            = sqrt(vflat²/hR) / (η×sqrt(a0))
|            = sqrt(G Sigma_0 / a0) / η
|
|   ε_max &gt; ε_c □□:
|     sqrt(G Sigma_0 / a0) &gt; η × ε_c
|     G Sigma_0 / a0 &gt; η² × ε_c²
|
|   η ~ 0.74, ε_c ~ 0.072 □□□□:
|     η² × ε_c² = 0.74² × 0.072² = 0.548 × 0.00518 = 0.00284
|     → G Sigma_0 &gt; 0.00284 × a0
|     → G Sigma_0 &gt; 3.4 × 10^-13 m/s²
| """)
|
| # This gives a surface density threshold
| GSigma_threshold = 0.74**2 * 0.072**2 * 1.2e-10
| print(f"  □□: G Sigma_0 &gt; {GSigma_threshold:.2e} m/s²")
| print(f"  vflat²/hR □□□□: vflat²/hR &gt; {GSigma_threshold:.2e} m/s²")
| print(f"  vflat ~ 100 km/s, hR ~ 3 kpc □□□:")
| vflat_test = 100e3  # m/s
| hR_test = 3 * 3.086e19  # m
| GSigma_test = vflat_test**2 / hR_test
| print(f"  G Sigma_0 = {GSigma_test:.2e} m/s² {'&gt;' if GSigma_test &gt; GSigma_threshold else '&lt;'} □□")
| print(f"  → □□□□□SPARC□□□□□□□□□")
|
| # ============================================================
| # H11: □□□□ Taylor□□□□□□□□□□
| # ============================================================
| print("\n" + "=" * 70)
| print("H11: U(ε;c) □ Taylor □□□□□□□□□□□□ □□□□")
| print("=" * 70)
|
| print("""
|   ε_fold = ε_eq - δε (□□□□□□□□δε □□□□□□□□)
|
|   U(ε_eq - δε; c) = U(ε_eq) + (1/2)U''(ε_eq)δε² - (1/6)U'''(ε_eq)δε³ + ...
|
|   □□□□□□= 3□□□2□□□10%□□:
|     |U'''δε/(3U'')| &lt; 0.1
|     → δε &lt; 0.3 |U''/U'''|
|
|   □□□□ + gN □□□□:
|     x = gN/gc = U''(ε_eq) × δε / [gc×(derivative mapping)]  (□□)
| """)
|
| print("\n  c □□□□□□ delta_eps_elastic □□□□□ eps_proxy:")
| header = "  {:&gt;6s} {:&gt;8s} {:&gt;12s} {:&gt;10s} {:&gt;14s} {:&gt;10s}".format(
|     "c", "eps_eq", "|U2/U3|", "delt_10%", "eps_fold_lim", "eps_proxy")
| print(header)
|
| for c in [0.10, 0.15, 0.20, 0.24, 0.30, 0.40, 0.50, 0.70, 0.90]:
|     eeq = eps_eq(c)
|     if eeq &gt;= 0.99: continue
|     u2v = U2(eeq, c)
|     u3v = U3(eeq, c)
|     ratio_u = abs(u2v / u3v) if u3v != 0 else 999
|     delta_10 = 0.3 * ratio_u
|     ef_lim = eeq - delta_10
|     if ef_lim &gt; 0.01 and ef_lim &lt; 0.99:
|         ep = eps_proxy_from_fold(ef_lim, c)
```

```
    else:
        ep = None
    ep_str = f"{ep:.4f}" if ep is not None else "N/A"
    print(f"  {c:6.3f} {eeq:8.4f} {ratio_u:12.4f} {delta_10:10.4f} "
          f"{ef_lim:14.4f} {ep_str:&gt;10s}")

# ============================================================
# □□□□
# ============================================================
print("\n" + "=" * 70)
print("□□□□")
print("=" * 70)

print("""
□□□□□□□□:

  H1: U□□□ → c□□□□□□□□□□
  H2: □□□□ → □□□□□□□□□□ε_proxy □□□□□□□□□□□□□
  H3: □□□ → H1□□□□□□□□
  H4: Duffing → □□□□□□□□□□□□
  H5: MOND□□ → 2□□□ 1.8% □□□□□□□□□
  H6: □□□□□□ → □□□□□□□□□□□□□□
  H7: c□□ → □□□□□□□□□□□□
  H8: □□□□□ → □□□□□□□□□□
  H9: x_c=1/200 → □□□□□□□□□□□□□□□□□□□
  H10: □□□□□ → □□□□□□□□□□□□□□□□□□□□□
  H11: Taylor□□□□□ → c□□□□□□□□□□□□□□

ε_c ≈ 0.072 □ U(ε;c) □□□□□□□□□□□□□□□
□□□□□□□□□□□□□

□□□□□□□□:
  (a) ε_c □ U(ε;c) □□□□□□□□□□□□□□□□□□□□□□□□
  (b) ε_c □□□□□□□□□□□□□□□□rmax/hR □□□□□□□□
  (c) ε_c □□□14□□□□□□□□U □□□□□□□□□□□□

□□: ε_c □□□□□□□□□□□□□□□□□□□
      □□□□□□ ε_c ≈ 0.07 +/- ? □□□□□□□□□□□□□□□□□
""")

# ============================================================
# □
# ============================================================
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# (a) U(ε;c) for various c
ax = axes[0, 0]
eps_range = np.linspace(0.01, 0.95, 200)
for c in [0.1, 0.24, 0.5, 0.8]:
    ax.plot(eps_range, U(eps_range, c), label=f'c={c}')
ax.set_xlabel('epsilon_fold')
ax.set_ylabel('U(epsilon;c)')
ax.set_title('Free energy U(epsilon;c)')
ax.legend()
ax.set_ylim(-5, 2)

# (b) U''(ε;c) — stiffness
ax = axes[0, 1]
for c in [0.1, 0.24, 0.5, 0.8]:
    ax.plot(eps_range, U2(eps_range, c), label=f'c={c}')
ax.axhline(0, color='grey', ls='--')
ax.set_xlabel('epsilon_fold')
ax.set_ylabel('U\'\'(epsilon;c)')
ax.set_title('Local stiffness')
ax.legend()
ax.set_ylim(-2, 20)

# (c) ε_proxy vs ε_fold for various c
ax = axes[1, 0]
for c in [0.1, 0.24, 0.5, 0.8]:
    eeq = eps_eq(c)
    ef_range = np.linspace(0.01, eeq-0.01, 100)
    ep = [eps_proxy_from_fold(ef, c) for ef in ef_range]
    ax.plot(ef_range, ep, label=f'c={c}')
ax.axhline(0.072, color='red', ls='--', lw=2, label='epsilon_c=0.072')
ax.set_xlabel('epsilon_fold')
ax.set_ylabel('epsilon_proxy = sqrt(gN/gc)')
ax.set_title('epsilon_proxy vs epsilon_fold')
ax.legend()

# (d) Non-linearity ratio
ax = axes[1, 1]
for c in [0.1, 0.24, 0.5, 0.8]:
    eeq = eps_eq(c)
    ef_range = np.linspace(0.01, eeq-0.01, 100)
    ratios = []
    ep_vals = []
    for ef in ef_range:
        ep = eps_proxy_from_fold(ef, c)
        if ep &gt; 0:
            nl = abs(U3(ef, c) * (eeq - ef) / U2(ef, c)) if U2(ef, c) != 0 else 0
            ratios.append(nl)
```

```
518 |             ep_vals.append(ep)
519 |     ax.plot(ep_vals, ratios, label=f'c={c}')
520 | ax.axvline(0.072, color='red', ls='--', lw=2)
521 | ax.axhline(1, color='grey', ls='--')
522 | ax.set_xlabel('epsilon_proxy')
523 | ax.set_ylabel('|U3 * delta_eps / U2|')
524 | ax.set_title('Non-linearity ratio')
525 | ax.legend()
526 | ax.set_ylim(0, 3)
527 |
528 | plt.tight_layout()
529 | plt.savefig('/home/claude/epsilon_c_analysis.png', dpi=150)
530 | print("\nFigure saved: epsilon_c_analysis.png")
531 | print("\n▫▫▫")
```

Script 6: sparc_eta_derivation.py

目的

eta の第一原理導出: 深MOND領域のガス分率仮説の検証

核心的洞察: 深MOND極限で d ln(gc)/d ln(Yd) = -(1 - f_gas_deep)。eta ~ Yd^(-0.41) の観測から f_gas_deep = 0.59 を予測。テスト項目: (1) SPARC 各銀河の深MOND領域で f_gas_deep を直接計算。(2) 銀河ごとの beta_predicted vs eta_obs。(3) eta_predicted vs eta_obs (per-galaxy vs fixed exponent) 。(4) f_gas_deep の決定因子。(5) Yd 微分の直接テスト (Yd +/-10% の数値微分) 。(6) 導出精度評価。

結果

部分的成功: d ln(gc)/d ln(Yd) = -(1-f_gas) は銀河内で成立 (r=0.616, p=7e-18) 。しかし銀河間回帰 beta=-0.41 の予測には失敗。f_gas_deep median=0.284 (予測 0.59 の半分) 。原因: f_gas と Yd の負相関 (rho=-0.46) による Simpson's paradox 的効果。A-1 (eta導出) は A 級維持だが「運動学的に制約された」ことは記録。

実行方法

```
uv run --with scipy --with matplotlib python sparc_eta_derivation.py
```

入力データ: sparc_gc.csv + Rotmod_LTG/*.dat
出力: eta_derivation_results.png + 標準出力

ソースコード全文

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
sparc_eta_derivation.py
=======================
eta □□□□□□□: □MOND□□□□□□□□□□

■ □□□□□:

□MOND□□: gc_deep(r) = g_eff(r)^2 / gN(r)

gN(r) = Yd × v_disk(r)^2/r + v_gas(r)^2/r + v_bul(r)^2/r

Yd □□□□□□□□□□ gc_deep □□□:

  d ln(gc_deep) / d ln(Yd) = -Yd × v_disk^2 / (Yd × v_disk^2 + v_gas^2 + v_bul^2)
                           = -(1 - f_gas_deep)

□□□ f_gas_deep = (v_gas^2 + v_bul^2) / gN □□MOND□□□□Yd□□□□□□□□□

η = gc / sqrt(a0 × G Sigma0) □ G Sigma0 = vflat^2/hR □ Yd □□□□□□:

  d ln(η) / d ln(Yd) = d ln(gc) / d ln(Yd) = -(1 - f_gas_deep)

□□: η □ Yd^(-0.41) → d ln(η)/d ln(Yd) = -0.41

□□: f_gas_deep = 1 - 0.41 = 0.59

□□□: □MOND□□□□□□(+bulge)□□□□59%□□□
η □ Yd^(-0.41) □□□□□□□

■ □□□:
  (1) SPARC □□□□□MOND□□□ f_gas □□□□□
  (2) median f_gas_deep ≈ 0.59 □□□□□□
  (3) □□□□□ f_gas_deep vs Yd □□□□□
  (4) eta_predicted = C × Yd^(-(1-f_gas_deep)) □□□□□□□

□□: uv run --with scipy --with matplotlib python sparc_eta_derivation.py
"""

import os, sys
import numpy as np
from scipy.optimize import minimize
from scipy.stats import spearmanr, pearsonr
import warnings
warnings.filterwarnings('ignore')

BASE = r"D:\□□□□□□\□□□□□□\□□□□□\□□□□□□□\□□□□"
ROTMOD = os.path.join(BASE, "Rotmod_LTG")
GC_CSV = os.path.join(BASE, "sparc_gc.csv")
a0 = 1.2e-10

# ============================================================
# □□□□□□□□□□
# ============================================================
```

```
def load_gc_csv():
    import csv
    data = {}
    with open(GC_CSV, 'r') as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = None
            for k in row:
                if 'gal' in k.lower() or 'name' in k.lower():
                    name = row[k].strip(); break
            if not name: continue
            gc_val = None
            for k in ['gc','gc_obs','g_c','gc_a0']:
                if k in row and row[k]:
                    try: gc_val = float(row[k]); break
                    except: pass
            if gc_val is None: continue
            vflat, hR, Yd = None, None, None
            for k in ['vflat','Vflat','v_flat']:
                if k in row and row[k]:
                    try: vflat = float(row[k]); break
                    except: pass
            for k in ['hR','Reff','h_R','hr']:
                if k in row and row[k]:
                    try: hR = float(row[k]); break
                    except: pass
            for k in ['Yd','yd','Y_d','SPS']:
                if k in row and row[k]:
                    try: Yd = float(row[k]); break
                    except: pass
            data[name] = {'gc': gc_val, 'vflat': vflat, 'hR': hR, 'Yd': Yd}
    gc_vals = [d['gc'] for d in data.values()]
    if np.median(gc_vals) &gt; 1e-5:
        for n in data: data[n]['gc'] *= a0
    return data

def load_rotcurve(gname):
    fname = os.path.join(ROTMOD, f"{gname}_rotmod.dat")
    if not os.path.exists(fname): return None
    cols = {k: [] for k in ['r','vobs','vgas','vdisk','vbul']}
    with open(fname, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith('#'): continue
            p = line.split()
            if len(p) &lt; 6: continue
            try:
                cols['r'].append(float(p[0]))
                cols['vobs'].append(float(p[1]))
                cols['vgas'].append(float(p[3]))
                cols['vdisk'].append(float(p[4]))
                cols['vbul'].append(float(p[5]))
            except: continue
    if len(cols['r']) &lt; 5: return None
    return {k: np.array(v) for k, v in cols.items()}

def compute_gc_deep(rc, Yd=0.5):
    r_m = rc['r'] * 3.086e19
    v_bar2 = (Yd * rc['vdisk']**2 + rc['vgas']**2 + rc['vbul']**2) * 1e6
    v_obs2 = (rc['vobs'] * 1e3)**2
    gN = v_bar2 / r_m
    gobs = v_obs2 / r_m
    mask = gN &gt; 0
    if np.sum(mask) &lt; 3: return None
    gc_pts = gobs[mask]**2 / gN[mask]
    gc_med = np.median(gc_pts)
    if gc_med &lt;= 0: return None
    x = gN[mask] / gc_med
    deep = x &lt; 1.0
    return np.median(gc_pts[deep]) if np.sum(deep) &gt;= 3 else gc_med

# ============================================================
# □□□
# ============================================================
def main():
    print("=" * 70)
    print("eta □□□□□□□: □MOND □□□□□□")
    print("=" * 70)

    print("""
■ □□□□□:
  eta ~ Yd^beta  where  beta = -(1 - f_gas_deep)
  □□: beta = -0.41 → f_gas_deep = 0.59

  f_gas_deep = &lt;v_gas^2 + v_bul^2&gt; / &lt;Yd*v_disk^2 + v_gas^2 + v_bul^2&gt;
  □□□ &lt;...&gt; □□MOND□□□x = gN/gc &lt; 1□□□□□□□
""")

    gc_data = load_gc_csv()

    results = []
    for gname in sorted(gc_data.keys()):
```

```
        gd = gc_data[gname]
        gc_obs, vflat, hR = gd['gc'], gd.get('vflat'), gd.get('hR')
        Yd = gd.get('Yd') or 0.5
        if gc_obs &lt;= 0 or not vflat or not hR or vflat &lt;= 0 or hR &lt;= 0:
            continue
        rc = load_rotcurve(gname)
        if rc is None: continue
        gc_deep = compute_gc_deep(rc, Yd)
        if gc_deep is None or gc_deep &lt;= 0: continue

        # □□□□□□ gN □□
        r_m = rc['r'] * 3.086e19
        v_disk2 = rc['vdisk']**2 * 1e6  # (m/s)^2
        v_gas2 = rc['vgas']**2 * 1e6
        v_bul2 = rc['vbul']**2 * 1e6

        gN_disk = Yd * v_disk2 / r_m  # Yd□□□□
        gN_gas = v_gas2 / r_m          # Yd□□□□□
        gN_bul = v_bul2 / r_m          # Yd□□□□□
        gN_total = gN_disk + gN_gas + gN_bul

        # □MOND□□□□□ (x = gN/gc &lt; 1)
        x = gN_total / gc_deep
        deep_mask = (x &lt; 1.0) &amp; (gN_total &gt; 0)

        if np.sum(deep_mask) &lt; 3:
            continue

        # f_gas_deep = (□Yd□□) / (□□□)  at deep MOND points
        gN_nonYd = gN_gas[deep_mask] + gN_bul[deep_mask]
        gN_tot_deep = gN_total[deep_mask]

        # □□□□□gN □□□□
        f_gas_deep = np.sum(gN_nonYd) / np.sum(gN_tot_deep)

        # gN^2 □□□□□gc_deep □□□□□□□□
        weights = gN_tot_deep**2  # gc_deep = gobs^2/gN □□□ gN^2 □□□□□
        f_gas_deep_w = np.sum(gN_nonYd * weights) / np.sum(gN_tot_deep * weights)

        # □□□□ f_gas □□
        f_gas_pts = gN_nonYd / gN_tot_deep

        # eta_obs
        GSigma0 = (vflat * 1e3)**2 / (hR * 3.086e19)  # m/s^2
        eta_obs = gc_obs / np.sqrt(a0 * GSigma0)

        # beta_predicted = -(1 - f_gas_deep)
        beta_pred = -(1 - f_gas_deep)

        results.append({
            'name': gname,
            'gc_obs': gc_obs,
            'gc_deep': gc_deep,
            'vflat': vflat,
            'hR': hR,
            'Yd': Yd,
            'f_gas_deep': f_gas_deep,
            'f_gas_deep_w': f_gas_deep_w,
            'f_gas_median': np.median(f_gas_pts),
            'beta_pred': beta_pred,
            'eta_obs': eta_obs,
            'n_deep': int(np.sum(deep_mask)),
            'disk_frac_deep': 1 - f_gas_deep,
        })

    N = len(results)
    print(f"□□□: {N}\n")

    f_gas = np.array([r['f_gas_deep'] for r in results])
    f_gas_w = np.array([r['f_gas_deep_w'] for r in results])
    f_gas_med = np.array([r['f_gas_median'] for r in results])
    beta_pred = np.array([r['beta_pred'] for r in results])
    eta_obs = np.array([r['eta_obs'] for r in results])
    Yd = np.array([r['Yd'] for r in results])
    vflat = np.array([r['vflat'] for r in results])
    hR = np.array([r['hR'] for r in results])

    # ============================================================
    # (1) f_gas_deep □□□
    # ============================================================
    print("=" * 70)
    print("(1) f_gas_deep □□□")
    print("=" * 70)
    print(f"  mean      = {np.mean(f_gas):.4f}")
    print(f"  median    = {np.median(f_gas):.4f}")
    print(f"  std       = {np.std(f_gas):.4f}")
    print(f"  IQR       = [{np.percentile(f_gas,25):.4f}, {np.percentile(f_gas,75):.4f}]")
    print(f"  □□□□□: median = {np.median(f_gas_w):.4f}")
    print(f"  □□□    = 0.59")
    print(f"  □        = {abs(np.median(f_gas) - 0.59):.4f} ({abs(np.median(f_gas)-0.59)/0.59*100:.1f}%)")

    # ============================================================
```

```
    # (2) □□□□□ beta_predicted vs eta_obs
    # ============================================================
    print("\n" + "=" * 70)
    print("(2) □□□□□ beta □□□□□")
    print("=" * 70)

    # □□□ beta
    log_eta = np.log10(eta_obs)
    log_Yd = np.log10(Yd)

    # □□□□□ beta (OLS)
    from numpy.linalg import lstsq
    X = np.column_stack([log_Yd, np.ones(N)])
    sol, _, _, _ = lstsq(X, log_eta, rcond=None)
    beta_obs_global = sol[0]

    print(f"  beta_obs (global OLS)     = {beta_obs_global:.4f}")
    print(f"  beta_obs (expected)       = -0.41 (from eta~Yd^-0.41)")
    print(f"  beta_pred (median)        = {np.median(beta_pred):.4f}")
    print(f"  beta_pred = -(1-f_gas)    = {-(1-np.median(f_gas)):.4f}")

    # ============================================================
    # (3) eta_predicted vs eta_obs
    # ============================================================
    print("\n" + "=" * 70)
    print("(3) eta □□□□□")
    print("=" * 70)

    # Model A: eta = C × Yd^(-0.41) [empirical, fixed exponent]
    log_eta_pred_A = -0.41 * log_Yd
    C_A = 10**(np.median(log_eta - log_eta_pred_A))
    pred_A = np.log10(C_A) + log_eta_pred_A
    R2_A = 1 - np.sum((log_eta - pred_A)**2) / np.sum((log_eta - np.mean(log_eta))**2)

    # Model B: eta = C × Yd^(-(1-f_gas_i)) [per-galaxy gas fraction]
    log_eta_pred_B = beta_pred * log_Yd
    C_B = 10**(np.median(log_eta - log_eta_pred_B))
    pred_B = np.log10(C_B) + log_eta_pred_B
    R2_B = 1 - np.sum((log_eta - pred_B)**2) / np.sum((log_eta - np.mean(log_eta))**2)

    # Model C: eta = C × Yd^beta_global_obs [global fit]
    pred_C = sol[1] + sol[0] * log_Yd
    R2_C = 1 - np.sum((log_eta - pred_C)**2) / np.sum((log_eta - np.mean(log_eta))**2)

    # Model D: eta = C (constant, no Yd dependence)
    R2_D = 0.0

    print(f"  Model A (Yd^-0.41 fixed):     R2 = {R2_A:.4f}")
    print(f"  Model B (Yd^-(1-f_gas_i)):     R2 = {R2_B:.4f}")
    print(f"  Model C (Yd^beta_fit):         R2 = {R2_C:.4f}, beta={sol[0]:.4f}")
    print(f"  Model D (eta=const):           R2 = {R2_D:.4f}")

    if R2_B &gt; R2_A:
        print(f"\n  □ Per-galaxy f_gas □□□ (B) □ fixed exponent (A) □□□□!")
        print(f"     → f_gas_deep □ eta □ Yd □□□□□□□□□□")
    else:
        print(f"\n  Per-galaxy f_gas (B) □ fixed (A) □□□□□□")
        print(f"  → f_gas_deep □□□□□□□□□□□□□□□□□□□□")

    # ============================================================
    # (4) f_gas_deep □□□□□□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("(4) f_gas_deep □□□□□")
    print("=" * 70)

    params_test = {
        'log(Yd)': log_Yd,
        'log(vflat)': np.log10(vflat),
        'log(hR)': np.log10(hR),
        'log(vflat^2/hR)': np.log10(vflat**2 / hR),
    }

    for pname, pvals in params_test.items():
        rho, p = spearmanr(f_gas, pvals)
        print(f"  rho(f_gas_deep, {pname:20s}) = {rho:+.3f} (p={p:.2e})")

    # ============================================================
    # (5) Yd □□□□□□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("(5) Yd □□□□□□□□")
    print("=" * 70)
    print("□□□□ Yd □ +/-10% □□□□□d ln(gc_deep)/d ln(Yd) □□□□□")

    beta_numerical = []
    for r in results:
        rc = load_rotcurve(r['name'])
        if rc is None: continue
        Yd_base = r['Yd']

        gc_lo = compute_gc_deep(rc, Yd_base * 0.9)
```

```
        gc_hi = compute_gc_deep(rc, Yd_base * 1.1)

        if gc_lo is None or gc_hi is None or gc_lo &lt;= 0 or gc_hi &lt;= 0:
            continue

        dlngc_dlnYd = (np.log(gc_hi) - np.log(gc_lo)) / (np.log(1.1) - np.log(0.9))
        beta_numerical.append(dlngc_dlnYd)

    beta_num = np.array(beta_numerical)
    print(f"  □□□□ d ln(gc)/d ln(Yd):")
    print(f"    mean   = {np.mean(beta_num):.4f}")
    print(f"    median = {np.median(beta_num):.4f}")
    print(f"    std    = {np.std(beta_num):.4f}")
    print(f"  □□□□ -(1-f_gas_deep): median = {np.median(beta_pred):.4f}")
    print(f"  □□ eta exponent:               = -0.41")

    # □□□□ vs □□□□□□□
    if len(beta_num) == len(beta_pred):
        rho_bn, p_bn = pearsonr(beta_num, beta_pred)
        print(f"\n  r(numerical, analytical) = {rho_bn:.4f} (p={p_bn:.2e})")

    # ============================================================
    # (6) □□□□□□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("(6) □□□□□□□")
    print("=" * 70)

    # eta_theory = eta0 × Yd^(-(1-&lt;f_gas&gt;))
    # where &lt;f_gas&gt; = 0.59 (if prediction holds)
    # vs
    # eta_theory = eta0 × Yd^(-(1-f_gas_i)) (per-galaxy)

    # Residual after theory prediction
    resid_B = log_eta - pred_B
    resid_A = log_eta - pred_A

    print(f"  Model A (Yd^-0.41):        resid std = {np.std(resid_A):.4f} dex")
    print(f"  Model B (per-galaxy f_gas): resid std = {np.std(resid_B):.4f} dex")
    print(f"  eta total scatter:         std = {np.std(log_eta):.4f} dex")
    print(f"  Yd explained fraction: A={R2_A*100:.1f}%, B={R2_B*100:.1f}%")

    # ============================================================
    # (7) □□□□
    # ============================================================
    print("\n" + "=" * 70)
    print("□□□□")
    print("=" * 70)

    print(f"""
■ □□□□□:
  eta ~ Yd^beta, beta = -(1 - f_gas_deep)
  f_gas_deep = □MOND□□□□□□(+bulge)□□□

■ □□□□□:
  f_gas_deep median = {np.median(f_gas):.4f} (□□: 0.59)
  beta_predicted    = {np.median(beta_pred):.4f} (□□: -0.41)
  □□□□ beta     = {np.median(beta_num):.4f}

■ □□:
  f_gas_deep ≈ 0.59 □□ → eta □ Yd^(-0.41) □□□□□□□□□□□□
  □□□□eta □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

■ □□□□□:
  □□□□□□□:
    (a) f_gas_deep median □ [0.50, 0.68] (0.59 □ +/-15%)
    (b) beta_numerical ≈ beta_analytical (r &gt; 0.5)
    (c) Model B □ R2 &gt;= Model A □ R2

■ □□□□□□□□□□:
  A-1 (eta□□) □ A□□□□□ B□□□□□□
  □eta □□□□□□□□□→□eta □□□□□□□□□□□□
  □□□□□□□□□□□□□□ 1 □□□
""")

    # ============================================================
    # □
    # ============================================================
    try:
        import matplotlib
        matplotlib.use('Agg')
        import matplotlib.pyplot as plt

        fig, axes = plt.subplots(2, 2, figsize=(12, 10))
        fig.suptitle('eta first-principles derivation: gas fraction hypothesis', fontsize=14)

        # (a) f_gas_deep distribution
        ax = axes[0, 0]
        ax.hist(f_gas, bins=30, alpha=0.7, color='steelblue', edgecolor='white')
        ax.axvline(0.59, color='red', lw=2, ls='--', label='predicted (0.59)')
        ax.axvline(np.median(f_gas), color='orange', lw=2, label=f'median ({np.median(f_gas):.3f})')
        ax.set_xlabel('f_gas_deep')
```

```
        ax.set_ylabel('count')
        ax.set_title('Gas fraction at deep-MOND radii')
        ax.legend()

        # (b) eta_obs vs Yd with theory curves
        ax = axes[0, 1]
        ax.scatter(log_Yd, log_eta, s=10, alpha=0.5, c='steelblue')
        Yd_range = np.linspace(log_Yd.min(), log_Yd.max(), 50)
        ax.plot(Yd_range, np.log10(C_A) - 0.41*Yd_range, 'r-', lw=2, label='Yd^(-0.41) empirical')
        ax.plot(Yd_range, np.log10(C_B) + np.median(beta_pred)*Yd_range, 'g--', lw=2,
                label=f'Yd^({np.median(beta_pred):.2f}) predicted')
        ax.set_xlabel('log10(Yd)')
        ax.set_ylabel('log10(eta)')
        ax.set_title('eta vs Yd')
        ax.legend()

        # (c) beta_numerical vs beta_analytical
        ax = axes[1, 0]
        if len(beta_num) == len(beta_pred):
            ax.scatter(beta_pred, beta_num, s=10, alpha=0.5, c='steelblue')
            lim = [min(beta_pred.min(), beta_num.min())-0.1,
                   max(beta_pred.max(), beta_num.max())+0.1]
            ax.plot(lim, lim, 'k--', lw=1)
            ax.set_xlabel('beta_analytical = -(1-f_gas)')
            ax.set_ylabel('beta_numerical (Yd +/-10%)')
            ax.set_title(f'Analytical vs numerical beta (r={rho_bn:.3f})')

        # (d) f_gas vs Yd
        ax = axes[1, 1]
        ax.scatter(log_Yd, f_gas, s=10, alpha=0.5, c='steelblue')
        ax.axhline(0.59, color='red', ls='--', lw=1, label='prediction 0.59')
        ax.set_xlabel('log10(Yd)')
        ax.set_ylabel('f_gas_deep')
        ax.set_title('Gas fraction vs Yd')
        ax.legend()

        plt.tight_layout()
        figpath = os.path.join(BASE, 'eta_derivation_results.png')
        plt.savefig(figpath, dpi=150)
        print(f"\nFigure saved: {figpath}")
    except Exception as e:
        print(f"\nFigure error: {e}")

    print("\n▫▫▫")

if __name__ == '__main__':
    main()
```