N-1' 検証スクリプトカタログ

観測データ解析の目的と全スクリプトソースコード

2026年4月11日 坂口 忍（坂口製麺所）| 全10本 / 5,619行

目次

#	スクリプト	解析目的	結果
1	sparc_P2b_critical.py	P2b: 自己無撞着臨界変形条件	否定
2	sparc_P3_domainwall.py	P3: Z2 SSBドメインウォール仮説	否定
3	sparc_P5_deepMOND.py	P5: 深MONDリミットの摂動展開	成立
4	sparc_hR_decomposition.py	hR偏相関の完全分解	成立
5	sparc_hR_residual31.py	残り31%: 高次形状パラメータ	否定
6	sparc_gc_method_systematics.py	gc決定手法の系統効果	否定
7	sparc_cond14_residual.py	条件14（塑性領域）による残り31%の検証	否定
8	sparc_strain_gradient.py	歪みエネルギー勾配項	撤回/B-
9	sparc_tautology_check.py	トートロジーチェック	否定
10	sparc_grad_ratio_detail.py	E_grad/E_local（hR非含有勾配比）精密評価	撤回/B-

実行環境: uv run --with scipy --with matplotlib python [script]

データ: SPARC Rotmod_LTG/（175銀河回転曲線）+ sparc_gc.csv（gc/vflat/hR）

1. P2b: 自己無撞着臨界変形条件

ファイル: sparc_P2b_critical.py

解析目的

膜パラメータ c を「臨界変形に到達する条件」で決定する仮説（P2b）の検証。$d^2U/d\ epsilon^2 = 0$ の臨界点 $epsilon_c = 1 - sqrt(c)$ での自己無撞着方程式 $max[g_{eff}(r)/g_c] = 2(sqrt(c)-1)$ を SPARC 175銀河で数値的に解き、予測 gc と観測 gc を比較する。

結果

否定: R^2=-1.11, alpha=0.219, gc_pred/gc_obs=2.74。r_c/hR~0.93（記録）。

ソースコード

(478 行)

```
#!/usr/bin/env python3
"""
sparc_P2b_critical.py
=====================
P2b検証: 自己無撞着臨界変形条件からgcを予測する

物理的仮説:
  膜パラメータ c は「膜が臨界変形に丁度到達する」条件で決まる。

  臨界条件: ε_c = 1 - sqrtc で d^2U/dε^2 = 0
  このとき g_eff(r_c)/g_c = 2(sqrtc - 1)

  自己無撞着版: g_eff は MOND 補間で g_N と g_c = c*a_0 から決まる
  -&gt; max_r [g_eff(r)/g_c] = 2(sqrtc - 1) が c を決定

テスト:
  Test 1: 各銀河で c を数値的に解き、gc_pred = c*a_0 vs gc_obs を比較
  Test 2: gc_pred が幾何平均法則 gc ~ sqrt(a_0*vflat^2/hR) を再現するか
  Test 3: gc_pred vs gc_obs の残差構造 (hR, vflat, Σ_bar 依存性)
  Test 4: r_c/hR の分布 -&gt; 有限ディスク効果の大きさ

実行: uv run --with scipy --with matplotlib python sparc_P2b_critical.py
"""

import numpy as np
from scipy.optimize import brentq
from scipy.interpolate import interp1d
from pathlib import Path
import csv
import sys

# ============================================================
# 定数・パス
# ============================================================
a0 = 1.2e-10          # m/s^2 (MOND加速度スケール)
G_N = 4.30091e-3      # (km/s)^2 pc / M_sun
Yd_default = 0.5      # 恒星質量光度比 (ディスク)
Yb_default = 0.7      # 恒星質量光度比 (バルジ)
kpc_to_pc = 1000.0
kms2_to_ms2 = 1.0e6   # (km/s)^2 -&gt; (m/s)^2

# データパス (ローカルWindows環境)
BASE = Path(r"D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン")
ROTMOD_DIR = BASE / "Rotmod_LTG"
GC_CSV = BASE / "sparc_gc.csv"


# ============================================================
# ユーティリティ
# ============================================================
def load_gc_table(path):
    """sparc_gc.csv を読み込み {galaxy_name: dict} を返す"""
    data = {}
    with open(path, "r") as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = row["Galaxy"].strip()
            data[name] = {
                "gc_obs": float(row["gc"]),        # m/s^2 単位想定
                "vflat": float(row["Vflat"]),       # km/s
                "hR": float(row["hR"]),             # kpc
            }
```

```
    return data


def load_rotmod(galaxy_name, rotmod_dir):
    """Rotmod_LTG から回転曲線を読み込む
    戻り値: r_kpc, Vobs, Vgas, Vdisk, Vbul (全て km/s, rはkpc)
    """
    fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
    if not fpath.exists():
        return None

    r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
    with open(fpath, "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
            r.append(float(parts[0]))
            vobs.append(float(parts[1]))
            errv.append(float(parts[2]))
            vgas.append(float(parts[3]))
            vdisk.append(float(parts[4]))
            vbul.append(float(parts[5]))

    return (np.array(r), np.array(vobs), np.array(vgas),
            np.array(vdisk), np.array(vbul))


def compute_gN_profile(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
    """バリオン重力加速度プロファイル g_N(r) を m/s^2 で返す

    V の符号を保存: g = |V|*V*Y / r (符号付き)
    最終的にgasは符号なし(Vgas^2は常に正寄与)
    """
    r_m = r_kpc * kpc_to_pc * 3.0857e16   # kpc -&gt; m (1 pc = 3.0857e16 m)

    # V^2 -&gt; 加速度 (km/s)^2 / kpc -&gt; m/s^2
    # a = V^2 / r  where V in km/s, r in kpc
    # a [m/s^2] = V^2 [km^2/s^2] * 1e6 / (r [kpc] * 3.0857e19)

    conv = kms2_to_ms2 / (r_kpc * 3.0857e19)   # per-point conversion

    gN = np.abs(Vgas) * Vgas * conv ¥
       + Yd * np.abs(Vdisk) * Vdisk * conv ¥
       + Yb * np.abs(Vbul) * Vbul * conv

    return np.abs(gN)   # 大きさのみ


def g_eff_over_gc(x):
    """g_eff/g_c as function of x = g_N/g_c
    MOND simple interpolation: μ(y)=y/(1+y)
    -&gt; g_eff/g_c = [x + sqrt(x^2+4x)] / 2
    """
    return (x + np.sqrt(x**2 + 4*x)) / 2


def critical_RHS(c):
    """臨界条件の右辺: 2(sqrtc - 1)
    c &gt; 1 でのみ正値
    """
    return 2.0 * (np.sqrt(c) - 1.0)


# ============================================================
# メイン: 各銀河で c を解く
# ============================================================
def solve_c_for_galaxy(gN_profile, r_kpc, vflat_kms,
                       c_min=1.001, c_max=50.0):
    """
    自己無撞着臨界条件:
      max_r [g_eff(r)/g_c] = 2(sqrtc - 1)

    ここで g_c = c*a_0, x(r) = g_N(r)/g_c

    g_eff の最大値は g_N の最大値の位置で達成される（単調関数のため）。

    戻り値: c_sol, gc_pred, r_c_kpc, converged
    """
    # g_N の最大値とその位置
    idx_max = np.argmax(gN_profile)
    gN_max = gN_profile[idx_max]
    r_max = r_kpc[idx_max]
```

```
    if gN_max &lt;= 0:
        return np.nan, np.nan, np.nan, False

    def residual(c):
        gc = c * a0
        x_max = gN_max / gc
        lhs = g_eff_over_gc(x_max)
        rhs = critical_RHS(c)
        return lhs - rhs

    # 存在チェック
    try:
        r_lo = residual(c_min)
        r_hi = residual(c_max)
    except:
        return np.nan, np.nan, np.nan, False

    if r_lo * r_hi &gt; 0:
        # 符号が変わらない -&gt; 解なし or 範囲外
        # c_max を広げて再試行
        for c_try in [100, 500, 2000]:
            try:
                if residual(c_min) * residual(c_try) &lt; 0:
                    c_max = c_try
                    break
            except:
                continue
        else:
            return np.nan, np.nan, np.nan, False

    try:
        c_sol = brentq(residual, c_min, c_max, xtol=1e-8)
    except:
        return np.nan, np.nan, np.nan, False

    gc_pred = c_sol * a0

    # r_c の計算: g_eff(r_c)/g_c = 2(sqrtc-1) を満たす最大の r
    #  (g_N の減少域で閾値を横切る点)
    gc = gc_pred
    threshold = critical_RHS(c_sol)

    if threshold &lt;= 0:
        return c_sol, gc_pred, r_max, True

    x_arr = gN_profile / gc
    geff_gc_arr = g_eff_over_gc(x_arr)

    # 外側から探して閾値を超える最初の点
    r_c_kpc = r_max  # デフォルト
    for i in range(len(r_kpc)-1, 0, -1):
        if geff_gc_arr[i] &gt;= threshold:
            r_c_kpc = r_kpc[i]
            break

    return c_sol, gc_pred, r_c_kpc, True


def main():
    print("=" * 70)
    print("P2b: 自己無撞着臨界変形条件 -- gc予測検証")
    print("=" * 70)

    # データ読み込み
    if not GC_CSV.exists():
        print(f"ERROR: {GC_CSV} not found")
        sys.exit(1)

    gc_table = load_gc_table(GC_CSV)
    print(f"sparc_gc.csv: {len(gc_table)} galaxies loaded")

    # 各銀河を処理
    results = []
    skipped = 0

    for gal_name, gal_info in gc_table.items():
        rotmod = load_rotmod(gal_name, ROTMOD_DIR)
        if rotmod is None:
            skipped += 1
            continue

        r_kpc, Vobs, Vgas, Vdisk, Vbul = rotmod

        if len(r_kpc) &lt; 5:
            skipped += 1
            continue

```

```
        # g_N プロファイル
        gN = compute_gN_profile(r_kpc, Vgas, Vdisk, Vbul,
                                Yd=Yd_default, Yb=Yb_default)

        # c を解く
        c_sol, gc_pred, r_c, converged = solve_c_for_galaxy(
            gN, r_kpc, gal_info["vflat"])

        if not converged:
            skipped += 1
            continue

        results.append({
            "name": gal_name,
            "gc_obs": gal_info["gc_obs"],
            "gc_pred": gc_pred,
            "c_sol": c_sol,
            "r_c_kpc": r_c,
            "hR": gal_info["hR"],
            "vflat": gal_info["vflat"],
            "gN_max": np.max(gN),
            "r_c_over_hR": r_c / gal_info["hR"] if gal_info["hR"] &gt; 0 else np.nan,
        })

    N = len(results)
    print(f"Converged: {N} galaxies, Skipped: {skipped}")

    if N &lt; 10:
        print("Too few galaxies for analysis.")
        sys.exit(1)

    # NumPy配列化
    gc_obs = np.array([r["gc_obs"] for r in results])
    gc_pred = np.array([r["gc_pred"] for r in results])
    c_arr = np.array([r["c_sol"] for r in results])
    hR_arr = np.array([r["hR"] for r in results])
    vflat_arr = np.array([r["vflat"] for r in results])
    rc_hR = np.array([r["r_c_over_hR"] for r in results])
    gN_max_arr = np.array([r["gN_max"] for r in results])

    # ============================================================
    # Test 1: gc_pred vs gc_obs
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 1: gc_pred vs gc_obs")
    print("=" * 50)

    log_obs = np.log10(gc_obs)
    log_pred = np.log10(gc_pred)

    mask = np.isfinite(log_obs) &amp;amp; np.isfinite(log_pred)
    log_obs_f = log_obs[mask]
    log_pred_f = log_pred[mask]

    from scipy.stats import pearsonr, spearmanr
    rho_p, p_p = pearsonr(log_pred_f, log_obs_f)
    rho_s, p_s = spearmanr(log_pred_f, log_obs_f)

    ratio = gc_pred[mask] / gc_obs[mask]

    print(f"  N = {mask.sum()}")
    print(f"  Pearson  ρ = {rho_p:.4f}, p = {p_p:.2e}")
    print(f"  Spearman ρ = {rho_s:.4f}, p = {p_s:.2e}")
    print(f"  gc_pred/gc_obs: median = {np.median(ratio):.3f}, "
          f"mean = {np.mean(ratio):.3f}, std = {np.std(ratio):.3f}")
    print(f"  log10 offset: median = {np.median(log_pred_f - log_obs_f):.3f}")

    # R^2 (log-log)
    ss_res = np.sum((log_obs_f - log_pred_f)**2)
    ss_tot = np.sum((log_obs_f - np.mean(log_obs_f))**2)
    R2_direct = 1 - ss_res / ss_tot if ss_tot &gt; 0 else np.nan
    print(f"  R^2(direct) = {R2_direct:.4f}")

    # 線形回帰 log(gc_obs) vs log(gc_pred)
    from numpy.polynomial.polynomial import polyfit
    coef = np.polyfit(log_pred_f, log_obs_f, 1)
    print(f"  Linear fit: log(gc_obs) = {coef[0]:.3f} x log(gc_pred) + {coef[1]:.3f}")

    # ============================================================
    # Test 2: gc_pred vs 幾何平均法則
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 2: gc_pred vs geometric mean law")
    print("=" * 50)

    GS0_proxy = (vflat_arr**2) / (hR_arr * kpc_to_pc * 3.0857e16)  # m/s^2
    gc_geom = np.sqrt(a0 * GS0_proxy)   # 幾何平均法則 (η=1)
```

```
    log_geom = np.log10(gc_geom)
    log_pred_all = np.log10(gc_pred)

    mask2 = np.isfinite(log_geom) &amp;amp; np.isfinite(log_pred_all)
    rho2, p2 = pearsonr(log_geom[mask2], log_pred_all[mask2])
    print(f"  ρ(gc_pred, gc_geom) = {rho2:.4f}, p = {p2:.2e}")

    # gc_pred のスケーリング: log(gc_pred) vs log(GS0_proxy)
    log_GS0 = np.log10(GS0_proxy)
    mask3 = np.isfinite(log_GS0) &amp;amp; np.isfinite(log_pred_all)
    slope_fit = np.polyfit(log_GS0[mask3], log_pred_all[mask3], 1)
    print(f"  log(gc_pred) vs log(vflat^2/hR): slope = {slope_fit[0]:.4f} "
          f"(期待: 0.5)")

    # ============================================================
    # Test 3: 残差構造
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 3: 残差構造 Δlog(gc) = log(gc_pred) - log(gc_obs)")
    print("=" * 50)

    delta = log_pred_f - log_obs_f

    # vs hR
    log_hR = np.log10(hR_arr[mask])
    rho_hR, p_hR = pearsonr(log_hR, delta)
    print(f"  Δ vs log(hR):    ρ = {rho_hR:.4f}, p = {p_hR:.3e}")

    # vs vflat
    log_vf = np.log10(vflat_arr[mask])
    rho_vf, p_vf = pearsonr(log_vf, delta)
    print(f"  Δ vs log(vflat): ρ = {rho_vf:.4f}, p = {p_vf:.3e}")

    # vs gN_max
    log_gNm = np.log10(gN_max_arr[mask])
    rho_gN, p_gN = pearsonr(log_gNm, delta)
    print(f"  Δ vs log(gN_max): ρ = {rho_gN:.4f}, p = {p_gN:.3e}")

    # ============================================================
    # Test 4: r_c / hR 分布
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 4: r_c / hR 分布")
    print("=" * 50)

    rc_hR_f = rc_hR[np.isfinite(rc_hR)]
    print(f"  N = {len(rc_hR_f)}")
    print(f"  median(r_c/hR) = {np.median(rc_hR_f):.2f}")
    print(f"  mean(r_c/hR)   = {np.mean(rc_hR_f):.2f}")
    print(f"  std(r_c/hR)    = {np.std(rc_hR_f):.2f}")
    print(f"  min = {np.min(rc_hR_f):.2f}, max = {np.max(rc_hR_f):.2f}")
    print(f"  fraction r_c/hR &amp;lt; 3: {np.mean(rc_hR_f &amp;lt; 3):.2%}")
    print(f"  fraction r_c/hR &amp;lt; 5: {np.mean(rc_hR_f &amp;lt; 5):.2%}")

    # ============================================================
    # Test 5: c の分布
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 5: c の分布")
    print("=" * 50)

    print(f"  median(c) = {np.median(c_arr):.3f}")
    print(f"  mean(c)   = {np.mean(c_arr):.3f}")
    print(f"  std(c)    = {np.std(c_arr):.3f}")
    print(f"  min = {np.min(c_arr):.3f}, max = {np.max(c_arr):.3f}")
    print(f"  gc_pred range: [{np.min(gc_pred):.2e}, {np.max(gc_pred):.2e}] m/s^2")
    print(f"  gc_obs  range: [{np.min(gc_obs):.2e}, {np.max(gc_obs):.2e}] m/s^2")

    # ============================================================
    # プロット
    # ============================================================
    try:
        import matplotlib
        matplotlib.use("Agg")
        import matplotlib.pyplot as plt

        fig, axes = plt.subplots(2, 2, figsize=(12, 10))
        fig.suptitle("P2b: Self-consistent Critical Deformation", fontsize=14)

        # (a) gc_pred vs gc_obs
        ax = axes[0, 0]
        ax.scatter(log_obs_f, log_pred_f, s=8, alpha=0.5)
        lim = [min(log_obs_f.min(), log_pred_f.min()) - 0.1,
               max(log_obs_f.max(), log_pred_f.max()) + 0.1]
        ax.plot(lim, lim, "k--", lw=1, label="1:1")
        ax.set_xlabel("log_1_0(gc_obs)")
```

```
        ax.set_ylabel("log_1_0(gc_pred)")
        ax.set_title(f"(a) gc_pred vs gc_obs  R^2={R2_direct:.3f}")
        ax.legend()
        ax.set_aspect("equal")

        # (b) gc_pred vs geometric mean law
        ax = axes[0, 1]
        ax.scatter(log_geom[mask2], log_pred_all[mask2], s=8, alpha=0.5)
        ax.plot(lim, lim, "k--", lw=1)
        ax.set_xlabel("log_1_0(gc_geom = sqrt(a_0*vflat^2/hR))")
        ax.set_ylabel("log_1_0(gc_pred)")
        ax.set_title(f"(b) gc_pred vs geometric mean")

        # (c) 残差 vs hR
        ax = axes[1, 0]
        ax.scatter(log_hR, delta, s=8, alpha=0.5)
        ax.axhline(0, color="k", ls="--", lw=1)
        ax.set_xlabel("log_1_0(hR [kpc])")
        ax.set_ylabel("Δlog_1_0(gc)")
        ax.set_title(f"(c) Residual vs hR  ρ={rho_hR:.3f}")

        # (d) r_c/hR 分布
        ax = axes[1, 1]
        ax.hist(rc_hR_f, bins=30, edgecolor="black", alpha=0.7)
        ax.axvline(np.median(rc_hR_f), color="red", ls="--",
                   label=f"median={np.median(rc_hR_f):.1f}")
        ax.set_xlabel("r_c / hR")
        ax.set_ylabel("count")
        ax.set_title("(d) Critical radius distribution")
        ax.legend()

        plt.tight_layout()
        out_fig = BASE / "P2b_critical_results.png"
        plt.savefig(out_fig, dpi=150)
        print(f"¥nFigure saved: {out_fig}")
    except Exception as e:
        print(f"¥nPlot skipped: {e}")

    # ============================================================
    # サマリー
    # ============================================================
    print("¥n" + "=" * 70)
    print("SUMMARY")
    print("=" * 70)
    print(f"  P2b条件で解いた gc_pred と gc_obs の相関: ρ = {rho_p:.4f}")
    print(f"  R^2(direct 1:1) = {R2_direct:.4f}")
    print(f"  gc_pred のスケーリング指数: α = {slope_fit[0]:.4f} (期待 0.5)")
    print(f"  r_c/hR 中央値: {np.median(rc_hR_f):.2f}")

    if abs(slope_fit[0] - 0.5) &lt; 0.1:
        print("  -&gt; α ~= 0.5: 幾何平均法則と整合 [OK]")
    else:
        print(f"  -&gt; α = {slope_fit[0]:.3f}: 幾何平均法則から乖離")

    if R2_direct &gt; 0.3:
        print(f"  -&gt; R^2 = {R2_direct:.3f}: P2bは gc を有意に予測 [OK]")
    else:
        print(f"  -&gt; R^2 = {R2_direct:.3f}: P2bの予測力は弱い")

    if np.median(rc_hR_f) &lt; 5:
        print(f"  -&gt; r_c/hR ~ {np.median(rc_hR_f):.1f}: 有限ディスク効果が支配的 [OK]")
    else:
        print(f"  -&gt; r_c/hR ~ {np.median(rc_hR_f):.1f}: 漸近極限に近い")


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

2. P3: Z2 SSBドメインウォール仮説

ファイル: sparc_P3_domainwall.py

解析目的

epsilon(r) の平衡プロファイルがキンク（ドメインウォール）を形成し、キンク幅 delta が hR
と固定的関係にある条件で c を決定する仮説（P3）
の検証。各銀河のキンク特性量（幅、位置、エネルギー）を測定し、delta = beta x hR の拘束から gc を予測する。

結果

否定: 19/175銀河のみキンク形成（c>1限定）、delta/hR CV=1.154、E_kink R^2=-0.507。

ソースコード

(611 行)

```
#!/usr/bin/env python3
"""
sparc_P3_domainwall.py
======================
P3検証: Z2 SSBドメインウォール仮説

物理的仮説:
  ε(r)の平衡プロファイルがキンク（ドメインウォール）を形成し、
  その構造的拘束がcを決定する。

  ε(r) = [-x + sqrt((x+2)^2 - 4c)] / 2,  x(r) = g_N(r)/(c*a_0)
  存在条件: x &gt;= x_min = 2(sqrtc - 1)  (c &gt; 1)

テスト:
  Test 1: ε(r)プロファイルの計算とキンク幅δの測定
  Test 2: δ/hR の普遍性チェック（全銀河で一定か？）
  Test 3: キンクエネルギー E_kink = int(dε/dr)^2 r dr の銀河依存性
  Test 4: E_kink ~ 定数 の拘束からcを予測 -&gt; gc_pred vs gc_obs
  Test 5: 代替拘束 δ = β*hR からcを予測 -&gt; gc_pred vs gc_obs

実行: uv run --with scipy --with matplotlib python sparc_P3_domainwall.py
"""

import numpy as np
from scipy.optimize import brentq
from scipy.interpolate import interp1d
from scipy.stats import pearsonr, spearmanr
from pathlib import Path
import csv
import sys

# ============================================================
# 定数・パス
# ============================================================
a0 = 1.2e-10          # m/s^2
G_N = 4.30091e-3      # (km/s)^2 pc / M_sun
Yd_default = 0.5
Yb_default = 0.7
kpc_to_m = 3.0857e19  # 1 kpc in meters

BASE = Path(r"D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン")
ROTMOD_DIR = BASE / "Rotmod_LTG"
GC_CSV = BASE / "sparc_gc.csv"


# ============================================================
# データ読み込み
# ============================================================
def load_gc_table(path):
    data = {}
    with open(path, "r") as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = row["Galaxy"].strip()
            data[name] = {
                "gc_obs": float(row["gc"]),
                "vflat": float(row["Vflat"]),
                "hR": float(row["hR"]),
            }
    return data


```

```
def load_rotmod(galaxy_name, rotmod_dir):
    fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
    if not fpath.exists():
        return None
    r, vobs, vgas, vdisk, vbul = [], [], [], [], []
    with open(fpath, "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
            r.append(float(parts[0]))
            vobs.append(float(parts[1]))
            vgas.append(float(parts[3]))
            vdisk.append(float(parts[4]))
            vbul.append(float(parts[5]))
    return (np.array(r), np.array(vobs), np.array(vgas),
            np.array(vdisk), np.array(vbul))


def compute_gN_profile(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
    """g_N(r) in m/s^2"""
    r_m = r_kpc * kpc_to_m
    conv = 1.0e6 / r_m     # (km/s)^2 / r[m] -&gt; m/s^2
    gN = np.abs(Vgas) * Vgas * conv ¥
       + Yd * np.abs(Vdisk) * Vdisk * conv ¥
       + Yb * np.abs(Vbul) * Vbul * conv
    return np.abs(gN)


# ============================================================
# ε(r) プロファイル計算
# ============================================================
def epsilon_eq(x, c):
    """平衡εを x = g_N/gc で計算
     ε = [-x + sqrt((x+2)^2 - 4c)] / 2
    存在条件: (x+2)^2 &gt;= 4c -&gt; x &gt;= 2(sqrtc - 1)
    """
    disc = (x + 2.0)**2 - 4.0 * c
    if np.isscalar(x):
        if disc &lt; 0:
            return np.nan
        return (-x + np.sqrt(disc)) / 2.0
    else:
        result = np.full_like(x, np.nan, dtype=float)
        valid = disc &gt;= 0
        result[valid] = (-x[valid] + np.sqrt(disc[valid])) / 2.0
        return result


def compute_epsilon_profile(r_kpc, gN, c):
    """各半径でε(r)を計算
    戻り値: eps(r), valid_mask
    """
    gc = c * a0
    x = gN / gc
    eps = epsilon_eq(x, c)
    valid = np.isfinite(eps)
    return eps, valid


def measure_kink(r_kpc, eps, valid):
    """キンクの特性量を測定
    - δ: キンク幅 (εが最大値の90%-&gt;10%に落ちる幅)
    - r_wall: キンク中心 (dε/drが最大の位置)
    - E_kink: int(dε/dr)^2 dr (キンクエネルギー密度の代理量)
    """
    if valid.sum() &lt; 5:
        return np.nan, np.nan, np.nan, np.nan

    r_v = r_kpc[valid]
    e_v = eps[valid]

    # dε/dr (数値微分)
    if len(r_v) &lt; 3:
        return np.nan, np.nan, np.nan, np.nan

    dr = np.diff(r_v)
    de = np.diff(e_v)
    mask_dr = dr &gt; 0
    if mask_dr.sum() &lt; 2:
        return np.nan, np.nan, np.nan, np.nan

    dedr = de[mask_dr] / dr[mask_dr]
    r_mid = (r_v[:-1] + r_v[1:])[mask_dr] / 2.0
```

```
    # キンクエネルギー E = int(dε/dr)^2 dr
    E_kink = np.trapz(dedr**2, r_mid)

    # dε/dr が最大（最も急な変化）の位置 -&gt; ウォール中心
    idx_max_grad = np.argmax(np.abs(dedr))
    r_wall = r_mid[idx_max_grad]

    # キンク幅: εが [ε_max, ε_min] の 80%-&gt;20% に落ちる幅
    e_max = np.max(e_v)
    e_min = np.min(e_v)
    e_range = e_max - e_min
    if e_range &lt; 1e-10:
        return np.nan, r_wall, E_kink, np.nan

    e_hi = e_min + 0.8 * e_range
    e_lo = e_min + 0.2 * e_range

    # εは一般にrとともに減少
    r_hi = np.interp(e_hi, e_v[::-1], r_v[::-1]) if e_v[0] &gt; e_v[-1] else np.nan
    r_lo = np.interp(e_lo, e_v[::-1], r_v[::-1]) if e_v[0] &gt; e_v[-1] else np.nan

    if np.isfinite(r_hi) and np.isfinite(r_lo):
        delta = abs(r_lo - r_hi)
    else:
        # フォールバック: |Δε| / max|dε/dr|
        delta = e_range / (np.max(np.abs(dedr)) + 1e-30)

    return delta, r_wall, E_kink, np.max(np.abs(dedr))


# ============================================================
# P3拘束からcを予測
# ============================================================
def predict_c_from_delta_constraint(r_kpc, gN, hR, beta_target,
                                    c_min=1.001, c_max=100.0):
    """拘束 δ = β*hR からcを決定"""
    def residual(c):
        eps, valid = compute_epsilon_profile(r_kpc, gN, c)
        delta, _, _, _ = measure_kink(r_kpc, eps, valid)
        if not np.isfinite(delta) or delta &lt;= 0:
            return 1e10
        return delta / hR - beta_target

    try:
        # まず符号チェック
        r_lo = residual(c_min)
        r_hi = residual(c_max)
        if not (np.isfinite(r_lo) and np.isfinite(r_hi)):
            return np.nan
        if r_lo * r_hi &gt; 0:
            # 単調-&gt;ブラケットが見つからない
            return np.nan
        c_sol = brentq(residual, c_min, c_max, xtol=1e-6, maxiter=100)
        return c_sol
    except:
        return np.nan


def predict_c_from_energy_constraint(r_kpc, gN, E_target,
                                     c_min=1.001, c_max=100.0):
    """拘束 E_kink = E_target からcを決定"""
    def residual(c):
        eps, valid = compute_epsilon_profile(r_kpc, gN, c)
        _, _, E_kink, _ = measure_kink(r_kpc, eps, valid)
        if not np.isfinite(E_kink) or E_kink &lt;= 0:
            return 1e10
        return np.log10(E_kink) - np.log10(E_target)

    try:
        r_lo = residual(c_min)
        r_hi = residual(c_max)
        if not (np.isfinite(r_lo) and np.isfinite(r_hi)):
            return np.nan
        if r_lo * r_hi &gt; 0:
            return np.nan
        c_sol = brentq(residual, c_min, c_max, xtol=1e-6, maxiter=100)
        return c_sol
    except:
        return np.nan


# ============================================================
# メイン
# ============================================================
def main():
    print("=" * 70)
```

```
    print("P3: Z2 SSBドメインウォール仮説 -- gc予測検証")
    print("=" * 70)

    gc_table = load_gc_table(GC_CSV)
    print(f"sparc_gc.csv: {len(gc_table)} galaxies")

    # Phase 1: 観測gcでε(r)プロファイルを計算し、キンク特性を測定
    print("¥n--- Phase 1: ε(r) キンク特性の測定（観測gcを使用）---")

    records = []
    for gal_name, info in gc_table.items():
        rotmod = load_rotmod(gal_name, ROTMOD_DIR)
        if rotmod is None:
            continue
        r_kpc, Vobs, Vgas, Vdisk, Vbul = rotmod
        if len(r_kpc) &lt; 5:
            continue

        gN = compute_gN_profile(r_kpc, Vgas, Vdisk, Vbul)
        gc_obs = info["gc_obs"]
        hR = info["hR"]
        vflat = info["vflat"]

        # gc_obs の単位チェック: a_0単位なら m/s^2 に変換
        if gc_obs &lt; 1e-5:
            gc_val = gc_obs  # 既に m/s^2
        else:
            gc_val = gc_obs * a0  # a_0単位

        c_obs = gc_val / a0

        if c_obs &lt;= 1.0:
            # c &lt;= 1 では臨界条件なし、εは全域で定義可能
            # ただしドメインウォール構造は異なる
            pass

        eps, valid = compute_epsilon_profile(r_kpc, gN, c_obs)
        delta, r_wall, E_kink, max_grad = measure_kink(r_kpc, eps, valid)

        if np.isfinite(delta) and delta &gt; 0 and hR &gt; 0:
            records.append({
                "name": gal_name,
                "gc_obs": gc_val,
                "c_obs": c_obs,
                "hR": hR,
                "vflat": vflat,
                "delta": delta,
                "r_wall": r_wall,
                "E_kink": E_kink,
                "max_grad": max_grad,
                "delta_over_hR": delta / hR,
                "r_wall_over_hR": r_wall / hR if np.isfinite(r_wall) else np.nan,
                "n_valid": valid.sum(),
                "n_total": len(r_kpc),
                "gN_profile": gN,
                "r_profile": r_kpc,
            })

    N = len(records)
    print(f"  キンク測定成功: {N} 銀河")

    if N &lt; 20:
        print("  WARNING: 解析に十分な銀河数がありません")
        if N == 0:
            sys.exit(1)

    # ============================================================
    # Test 1: δ/hR の分布 -- 普遍性チェック
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 1: δ/hR 分布（普遍性チェック）")
    print("=" * 50)

    dh = np.array([r["delta_over_hR"] for r in records])
    dh_f = dh[np.isfinite(dh) &amp;amp; (dh &gt; 0) &amp;amp; (dh &lt; 100)]

    print(f"  N = {len(dh_f)}")
    print(f"  median(δ/hR) = {np.median(dh_f):.3f}")
    print(f"  mean(δ/hR)   = {np.mean(dh_f):.3f}")
    print(f"  std(δ/hR)    = {np.std(dh_f):.3f}")
    print(f"  CV (std/mean) = {np.std(dh_f)/np.mean(dh_f):.3f}")
    print(f"  min = {np.min(dh_f):.3f}, max = {np.max(dh_f):.3f}")

    if np.std(dh_f) / np.mean(dh_f) &lt; 0.3:
        print("  -&gt; CV &lt; 0.3: δ/hR はほぼ普遍的 [OK]")
        universal_beta = True
    else:
```

```
        print("  -&gt; CV &gt;= 0.3: δ/hR は銀河ごとに大きく変動")
        universal_beta = False

    # ============================================================
    # Test 2: δ/hR vs c, vflat, hR の相関
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 2: δ/hR の駆動因子")
    print("=" * 50)

    c_arr = np.array([r["c_obs"] for r in records])
    vf_arr = np.array([r["vflat"] for r in records])
    hR_arr = np.array([r["hR"] for r in records])
    E_arr = np.array([r["E_kink"] for r in records])

    mask = np.isfinite(dh) &amp; (dh &gt; 0) &amp; (dh &lt; 100)

    for label, arr in [("c_obs", c_arr), ("vflat", vf_arr),
                       ("hR", hR_arr), ("log E_kink", np.log10(E_arr + 1e-30))]:
        m = mask &amp; np.isfinite(arr)
        if m.sum() &gt; 10:
            rho, p = pearsonr(arr[m], dh[m])
            print(f"  δ/hR vs {label:12s}: ρ = {rho:+.4f}, p = {p:.3e}")

    # ============================================================
    # Test 3: r_wall / hR の分布
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 3: r_wall / hR 分布（ウォール位置）")
    print("=" * 50)

    rw_hR = np.array([r["r_wall_over_hR"] for r in records])
    rw_f = rw_hR[np.isfinite(rw_hR) &amp; (rw_hR &gt; 0)]

    print(f"  N = {len(rw_f)}")
    print(f"  median(r_wall/hR) = {np.median(rw_f):.2f}")
    print(f"  mean(r_wall/hR)   = {np.mean(rw_f):.2f}")
    print(f"  std               = {np.std(rw_f):.2f}")

    # ============================================================
    # Test 4: E_kink の普遍性 -&gt; cの予測
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 4: E_kink 拘束からgc予測")
    print("=" * 50)

    E_valid = E_arr[mask &amp; np.isfinite(E_arr) &amp; (E_arr &gt; 0)]
    E_median = np.median(E_valid)
    E_cv = np.std(E_valid) / np.mean(E_valid)

    print(f"  E_kink: median = {E_median:.4e}")
    print(f"  E_kink: CV = {E_cv:.3f}")

    if E_cv &lt; 0.5:
        print("  -&gt; E_kink のばらつきが小さい -&gt; 拘束として機能する可能性")
    else:
        print("  -&gt; E_kink のばらつきが大きい -&gt; 拘束としては弱い")

    # E_kink = E_median の拘束からcを予測
    gc_pred_E = []
    gc_obs_E = []
    count_E = 0
    for rec in records:
        c_pred = predict_c_from_energy_constraint(
            rec["r_profile"], rec["gN_profile"], E_median)
        if np.isfinite(c_pred) and c_pred &gt; 0:
            gc_pred_E.append(c_pred * a0)
            gc_obs_E.append(rec["gc_obs"])
            count_E += 1

    print(f"  E_kink拘束で収束: {count_E}/{N}")

    if count_E &gt; 10:
        gc_pred_E = np.array(gc_pred_E)
        gc_obs_E = np.array(gc_obs_E)
        lp = np.log10(gc_pred_E)
        lo = np.log10(gc_obs_E)
        m = np.isfinite(lp) &amp; np.isfinite(lo)
        if m.sum() &gt; 5:
            rho, p = pearsonr(lp[m], lo[m])
            ss_res = np.sum((lo[m] - lp[m])**2)
            ss_tot = np.sum((lo[m] - np.mean(lo[m]))**2)
            R2 = 1 - ss_res / ss_tot if ss_tot &gt; 0 else np.nan
            ratio = gc_pred_E[m] / gc_obs_E[m]
            print(f"  ρ(gc_pred, gc_obs) = {rho:.4f}, p = {p:.3e}")
            print(f"  R^2(1:1) = {R2:.4f}")
            print(f"  ratio median = {np.median(ratio):.3f}")
```

```
411|
412|     # ============================================================
413|     # Test 5: δ = β*hR 拘束からgc予測
414|     # ============================================================
415|     print("¥n" + "=" * 50)
416|     print("Test 5: δ = β*hR 拘束からgc予測")
417|     print("=" * 50)
418|
419|     beta_med = np.median(dh_f)
420|     print(f"  使用するβ = {beta_med:.3f}")
421|
422|     gc_pred_D = []
423|     gc_obs_D = []
424|     count_D = 0
425|     for rec in records:
426|         c_pred = predict_c_from_delta_constraint(
427|             rec["r_profile"], rec["gN_profile"], rec["hR"], beta_med)
428|         if np.isfinite(c_pred) and c_pred &gt; 0:
429|             gc_pred_D.append(c_pred * a0)
430|             gc_obs_D.append(rec["gc_obs"])
431|             count_D += 1
432|
433|     print(f"  δ拘束で収束: {count_D}/{N}")
434|
435|     if count_D &gt; 10:
436|         gc_pred_D = np.array(gc_pred_D)
437|         gc_obs_D = np.array(gc_obs_D)
438|         lp = np.log10(gc_pred_D)
439|         lo = np.log10(gc_obs_D)
440|         m = np.isfinite(lp) &amp;amp; np.isfinite(lo)
441|         if m.sum() &gt; 5:
442|             rho, p = pearsonr(lp[m], lo[m])
443|             ss_res = np.sum((lo[m] - lp[m])**2)
444|             ss_tot = np.sum((lo[m] - np.mean(lo[m]))**2)
445|             R2 = 1 - ss_res / ss_tot if ss_tot &gt; 0 else np.nan
446|             ratio = gc_pred_D[m] / gc_obs_D[m]
447|             print(f"  ρ(gc_pred, gc_obs) = {rho:.4f}, p = {p:.3e}")
448|             print(f"  R^2(1:1) = {R2:.4f}")
449|             print(f"  ratio median = {np.median(ratio):.3f}")
450|
451|             # スケーリング指数
452|             GS0 = (np.array([r["vflat"] for r in records[:count_D]])**2) ¥
453|                   / (np.array([r["hR"] for r in records[:count_D]]) * kpc_to_m)
454|             # 注意: records順と gc_pred_D順が一致しない可能性
455|             # -&gt; gc_pred_D のスケーリングを直接チェック
456|             log_GS0_all = np.log10(
457|                 np.array([r["vflat"]**2 / (r["hR"] * kpc_to_m)
458|                           for r in records]))
459|             # 対応するgc_pred_Dのスケーリングは別途
460|
461|     # ============================================================
462|     # Test 6: スケーリング指数チェック (観測gcベース)
463|     # ============================================================
464|     print("¥n" + "=" * 50)
465|     print("Test 6: キンク特性量 vs proxy スケーリング")
466|     print("=" * 50)
467|
468|     log_gc = np.log10(np.array([r["gc_obs"] for r in records]))
469|     log_GS0 = np.log10(np.array(
470|         [r["vflat"]**2 / (r["hR"] * kpc_to_m) for r in records]))
471|     log_delta = np.log10(np.array([r["delta"] for r in records]))
472|     log_E = np.log10(np.array([r["E_kink"] for r in records]) + 1e-30)
473|
474|     m = np.isfinite(log_gc) &amp;amp; np.isfinite(log_GS0)
475|
476|     # δ vs GS0
477|     m1 = np.isfinite(log_delta) &amp;amp; np.isfinite(log_GS0)
478|     if m1.sum() &gt; 10:
479|         sl = np.polyfit(log_GS0[m1], log_delta[m1], 1)
480|         rho, p = pearsonr(log_GS0[m1], log_delta[m1])
481|         print(f"  log(δ) vs log(vflat^2/hR): slope={sl[0]:.3f}, ρ={rho:.3f}")
482|
483|     # E_kink vs GS0
484|     m2 = np.isfinite(log_E) &amp;amp; np.isfinite(log_GS0) &amp;amp; (log_E &gt; -20)
485|     if m2.sum() &gt; 10:
486|         sl = np.polyfit(log_GS0[m2], log_E[m2], 1)
487|         rho, p = pearsonr(log_GS0[m2], log_E[m2])
488|         print(f"  log(E_kink) vs log(vflat^2/hR): slope={sl[0]:.3f}, ρ={rho:.3f}")
489|
490|     # E_kink vs gc
491|     m3 = np.isfinite(log_E) &amp;amp; np.isfinite(log_gc) &amp;amp; (log_E &gt; -20)
492|     if m3.sum() &gt; 10:
493|         sl = np.polyfit(log_gc[m3], log_E[m3], 1)
494|         rho, p = pearsonr(log_gc[m3], log_E[m3])
495|         print(f"  log(E_kink) vs log(gc): slope={sl[0]:.3f}, ρ={rho:.3f}")
496|
497|     # ============================================================
```

```
    # プロット
    # ============================================================
    try:
        import matplotlib
        matplotlib.use("Agg")
        import matplotlib.pyplot as plt

        fig, axes = plt.subplots(2, 3, figsize=(16, 10))
        fig.suptitle("P3: Z2 SSB Domain Wall Structure", fontsize=14)

        # (a) δ/hR 分布
        ax = axes[0, 0]
        ax.hist(dh_f, bins=30, edgecolor="black", alpha=0.7)
        ax.axvline(np.median(dh_f), color="red", ls="--",
                   label=f"median={np.median(dh_f):.2f}")
        ax.set_xlabel("δ / hR")
        ax.set_ylabel("count")
        ax.set_title("(a) Kink width / hR")
        ax.legend()

        # (b) r_wall/hR 分布
        ax = axes[0, 1]
        ax.hist(rw_f, bins=30, edgecolor="black", alpha=0.7)
        ax.axvline(np.median(rw_f), color="red", ls="--",
                   label=f"median={np.median(rw_f):.1f}")
        ax.set_xlabel("r_wall / hR")
        ax.set_title("(b) Wall position / hR")
        ax.legend()

        # (c) δ/hR vs c_obs
        ax = axes[0, 2]
        m = np.isfinite(dh) &amp; (dh &gt; 0) &amp; (dh &lt; 100)
        ax.scatter(c_arr[m], dh[m], s=8, alpha=0.5)
        ax.set_xlabel("c_obs = gc/a_0")
        ax.set_ylabel("δ / hR")
        ax.set_title("(c) δ/hR vs c")

        # (d) E_kink vs gc
        ax = axes[1, 0]
        m3 = np.isfinite(log_E) &amp; np.isfinite(log_gc) &amp; (log_E &gt; -20)
        if m3.sum() &gt; 0:
            ax.scatter(log_gc[m3], log_E[m3], s=8, alpha=0.5)
            ax.set_xlabel("log_1_0(gc_obs)")
            ax.set_ylabel("log_1_0(E_kink)")
            ax.set_title("(d) Kink energy vs gc")

        # (e) gc_pred (E拘束) vs gc_obs
        ax = axes[1, 1]
        if count_E &gt; 5:
            lp = np.log10(gc_pred_E)
            lo = np.log10(gc_obs_E)
            m = np.isfinite(lp) &amp; np.isfinite(lo)
            ax.scatter(lo[m], lp[m], s=8, alpha=0.5)
            lim = [min(lo[m].min(), lp[m].min())-0.1,
                   max(lo[m].max(), lp[m].max())+0.1]
            ax.plot(lim, lim, "k--")
            ax.set_xlabel("log_1_0(gc_obs)")
            ax.set_ylabel("log_1_0(gc_pred)")
            ax.set_title("(e) E_kink constraint")
            ax.set_aspect("equal")
        else:
            ax.text(0.5, 0.5, "Not enough data", transform=ax.transAxes,
                    ha="center")

        # (f) gc_pred (δ拘束) vs gc_obs
        ax = axes[1, 2]
        if count_D &gt; 5:
            lp = np.log10(gc_pred_D)
            lo = np.log10(gc_obs_D)
            m = np.isfinite(lp) &amp; np.isfinite(lo)
            ax.scatter(lo[m], lp[m], s=8, alpha=0.5)
            lim = [min(lo[m].min(), lp[m].min())-0.1,
                   max(lo[m].max(), lp[m].max())+0.1]
            ax.plot(lim, lim, "k--")
            ax.set_xlabel("log_1_0(gc_obs)")
            ax.set_ylabel("log_1_0(gc_pred)")
            ax.set_title("(f) δ=β*hR constraint")
            ax.set_aspect("equal")
        else:
            ax.text(0.5, 0.5, "Not enough data", transform=ax.transAxes,
                    ha="center")

        plt.tight_layout()
        out_fig = BASE / "P3_domainwall_results.png"
        plt.savefig(out_fig, dpi=150)
        print(f"¥nFigure saved: {out_fig}")
    except Exception as e:
```

```
585|         print(f"¥nPlot skipped: {e}")
586|
587|     # ============================================================
588|     # サマリー
589|     # ============================================================
590|     print("¥n" + "=" * 70)
591|     print("SUMMARY")
592|     print("=" * 70)
593|     print(f"  キンク幅 δ/hR: median={np.median(dh_f):.3f}, "
594|           f"CV={np.std(dh_f)/np.mean(dh_f):.3f}")
595|     print(f"  ウォール位置 r_wall/hR: median={np.median(rw_f):.2f}")
596|     print(f"  E_kink CV = {E_cv:.3f}")
597|     if count_E &gt; 10:
598|         print(f"  E_kink拘束: R^2={R2:.4f}" if 'R2' in dir() else "  E_kink拘束: 計算不能")
599|     if count_D &gt; 10:
600|         print(f"  δ拘束: 収束={count_D}/{N}")
601|
602|     print("¥n  P3評価:")
603|     if universal_beta:
604|         print("  -&gt; δ/hR が普遍的 -&gt; ドメインウォール仮説に整合")
605|     else:
606|         print("  -&gt; δ/hR が非普遍的 -&gt; 単純なドメインウォール仮説は否定的")
607|
608|
609| if __name__ == "__main__":
610|     main()
611|
```

3. P5: 深MONDリミットの摂動展開

ファイル: sparc_P5_deepMOND.py

解析目的

深MOND極限 g_{eff} = sqrt(g_N x g_c) の代数的帰結として alpha=0.5 が出るかを検証。深MONDフィット（gc_deep）と完全MONDフィット（gc_full）のスケーリング指数を比較し、有限ディスク効果を定量化する。BTFR + 幾何平均法則の自己無撞着条件 $vflat^2$ ~ hR も検証する。

結果

成立: gc_deep alpha=0.471+/-0.036, p(0.5)=0.43。x<0.1: alpha=0.478+/-0.083。

ソースコード

(589 行)

```
#!/usr/bin/env python3
"""
sparc_P5_deepMOND.py
====================
P5検証: 深MONDリミットの摂動展開からα=0.5が出るか

戦略:
  深MONDでは g_eff ~= sqrt(g_N*gc) -&gt; BTFR: vflat^4 = gc*GM_bar
  しかし BTFR は hR を含まない（大r極限で点質量と等価）。

  hR依存性は「有限r効果」= 深MONDからのずれから来る。
  -&gt; 完全MOND補間関数で回転曲線をフィットし、
     得られるgcのスケーリングを調べる。

テスト:
  Test 1: 深MONDフィット gc_DM（r^2*g_N -&gt; vflat^4からgcを決定）
  Test 2: 完全MONDフィット gc_full（全r範囲でV(r;gc)をフィット）
  Test 3: gc_DM, gc_full のスケーリング -&gt; α の値
  Test 4: gc_full - gc_DM の差（1次補正）のhR依存性
  Test 5: 「フラットになる半径」r_flat/hR と補正の関係
  Test 6: 自己無撞着条件の次元的帰結

実行: uv run --with scipy --with matplotlib python sparc_P5_deepMOND.py
"""

import numpy as np
from scipy.optimize import minimize_scalar, brentq
from scipy.stats import pearsonr, spearmanr, linregress
from pathlib import Path
import csv
import sys

# ============================================================
# 定数
# ============================================================
a0 = 1.2e-10          # m/s^2
kpc_to_m = 3.0857e19  # 1 kpc -&gt; m
kms2_to_ms2 = 1.0e6   # (km/s)^2 -&gt; (m/s)^2
Yd_default = 0.5
Yb_default = 0.7

BASE = Path(r"D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン")
ROTMOD_DIR = BASE / "Rotmod_LTG"
GC_CSV = BASE / "sparc_gc.csv"


# ============================================================
# データ読み込み
# ============================================================
def load_gc_table(path):
    data = {}
    with open(path, "r") as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = row["Galaxy"].strip()
            data[name] = {
                "gc_obs": float(row["gc"]),
                "vflat": float(row["Vflat"]),
                "hR": float(row["hR"]),
            }
    return data

```

```
63|
64| def load_rotmod(galaxy_name, rotmod_dir):
65|     fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
66|     if not fpath.exists():
67|         return None
68|     r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
69|     with open(fpath, "r") as f:
70|         for line in f:
71|             line = line.strip()
72|             if not line or line.startswith("#"):
73|                 continue
74|             parts = line.split()
75|             if len(parts) &lt; 6:
76|                 continue
77|             r.append(float(parts[0]))
78|             vobs.append(float(parts[1]))
79|             errv.append(float(parts[2]))
80|             vgas.append(float(parts[3]))
81|             vdisk.append(float(parts[4]))
82|             vbul.append(float(parts[5]))
83|     return (np.array(r), np.array(vobs), np.array(errv),
84|             np.array(vgas), np.array(vdisk), np.array(vbul))
85|
86|
87| def compute_gN_profile(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
88|     """g_N(r) in m/s^2, r in kpc"""
89|     r_m = r_kpc * kpc_to_m
90|     conv = kms2_to_ms2 / r_m  # V^2[m^2/s^2] / r[m] = a[m/s^2]
91|     gN = np.abs(Vgas) * Vgas * conv ¥
92|        + Yd * np.abs(Vdisk) * Vdisk * conv ¥
93|        + Yb * np.abs(Vbul) * Vbul * conv
94|     return np.abs(gN)
95|
96|
97| def compute_Vbar_sq(Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
98|     """V_bar^2(r) = Vgas^2 + Yd*Vdisk^2 + Yb*Vbul^2 in (km/s)^2
99|     符号を保存"""
100|     return (np.abs(Vgas) * Vgas
101|             + Yd * np.abs(Vdisk) * Vdisk
102|             + Yb * np.abs(Vbul) * Vbul)
103|
104|
105| # ============================================================
106| # MOND補間によるV予測
107| # ============================================================
108| def V_deepMOND(r_kpc, gN, gc):
109|     """深MONDリミット: g_eff = sqrt(g_N*gc)
110|     V^2 = r*g_eff -&gt; V = (r*sqrt(g_N*gc))^(1/2)  &lt;- これは間違い
111|     V^2[m^2/s^2] = r[m] * g_eff[m/s^2]
112|     V[km/s] = sqrt(r[m] * g_eff[m/s^2]) / 1e3 * 1e3 = sqrt(r*g_eff) in m/s -&gt; km/s
113|     """
114|     r_m = r_kpc * kpc_to_m
115|     g_eff = np.sqrt(gN * gc)
116|     Vsq = r_m * g_eff  # m^2/s^2
117|     return np.sqrt(np.abs(Vsq)) * 1e-3  # km/s
118|
119|
120| def V_fullMOND(r_kpc, gN, gc):
121|     """完全MOND補間: g_eff/gc = [x + sqrt(x^2+4x)]/2, x=g_N/gc
122|     V^2[m^2/s^2] = r[m] * g_eff[m/s^2]
123|     """
124|     r_m = r_kpc * kpc_to_m
125|     x = gN / gc
126|     g_eff = gc * (x + np.sqrt(x**2 + 4 * x)) / 2.0
127|     Vsq = r_m * g_eff
128|     return np.sqrt(np.abs(Vsq)) * 1e-3  # km/s
129|
130|
131| def V_fullMOND_with_correction(r_kpc, gN, gc):
132|     """完全MOND + 1次補正項を分離して返す"""
133|     r_m = r_kpc * kpc_to_m
134|     x = gN / gc
135|     # 完全
136|     g_full = gc * (x + np.sqrt(x**2 + 4*x)) / 2.0
137|     # 深MOND (0次)
138|     g_deep = np.sqrt(gN * gc)
139|     # 1次補正
140|     g_corr = g_full - g_deep
141|     return g_full, g_deep, g_corr
142|
143|
144| # ============================================================
145| # gcフィット
146| # ============================================================
147| def fit_gc(r_kpc, gN, Vobs, errV, mode="full",
148|            gc_min=1e-12, gc_max=1e-8):
149|     """回転曲線フィットでgcを決定
```

```
    mode: "deep" = 深MONDのみ, "full" = 完全MOND補間

    フラット領域（外側50%のデータ点）にフォーカス
    """
    N = len(r_kpc)
    # 外側50%（最低5点）
    n_flat = max(5, N // 2)
    idx_flat = slice(N - n_flat, N)

    r_f = r_kpc[idx_flat]
    V_f = Vobs[idx_flat]
    e_f = errV[idx_flat]
    gN_f = gN[idx_flat]

    # 誤差が0の点は除外
    valid = e_f &gt; 0
    if valid.sum() &lt; 3:
        valid = np.ones(len(e_f), dtype=bool)
        e_f = np.where(e_f &gt; 0, e_f, np.median(Vobs) * 0.1)

    def chi2(log_gc):
        gc = 10**log_gc
        if mode == "deep":
            V_pred = V_deepMOND(r_f[valid], gN_f[valid], gc)
        else:
            V_pred = V_fullMOND(r_f[valid], gN_f[valid], gc)
        resid = (V_f[valid] - V_pred) / e_f[valid]
        return np.sum(resid**2)

    result = minimize_scalar(chi2, bounds=(np.log10(gc_min), np.log10(gc_max)),
                             method="bounded")

    gc_best = 10**result.x
    chi2_min = result.fun
    ndof = valid.sum() - 1

    return gc_best, chi2_min, ndof


def fit_gc_BTFR(gN, r_kpc, vflat):
    """BTFR方式: vflat^4 = gc*r^2*g_N を大rで適用
    -&gt; gc = vflat^4 / (r^2 * g_N) の外側平均
    """
    N = len(r_kpc)
    n_out = max(3, N // 3)

    r_m = r_kpc[-n_out:] * kpc_to_m
    gN_out = gN[-n_out:]
    vflat_ms = vflat * 1e3  # km/s -&gt; m/s

    # gc = vflat^4 / (r^2 * g_N)... これは
    # V^4 = r^2 * g_N * gc (深MOND) より
    # gc = V^4 / (r^2 * g_N)

    valid = gN_out &gt; 0
    if valid.sum() &lt; 1:
        return np.nan

    gc_arr = vflat_ms**4 / (r_m[valid]**2 * gN_out[valid])
    return np.median(gc_arr)


# ============================================================
# メイン
# ============================================================
def main():
    print("=" * 70)
    print("P5: 深MONDリミットの摂動展開 -- α=0.5の導出")
    print("=" * 70)

    gc_table = load_gc_table(GC_CSV)
    print(f"sparc_gc.csv: {len(gc_table)} galaxies")

    records = []
    skipped = 0

    for gal_name, info in gc_table.items():
        rotmod = load_rotmod(gal_name, ROTMOD_DIR)
        if rotmod is None:
            skipped += 1
            continue

        r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
        if len(r_kpc) &lt; 8:
            skipped += 1
            continue

```

```
        gN = compute_gN_profile(r_kpc, Vgas, Vdisk, Vbul)

        # gc単位の確認・変換
        gc_obs = info["gc_obs"]
        if gc_obs &gt; 1e-5:  # a_0単位の可能性
            gc_obs_ms2 = gc_obs * a0
        else:
            gc_obs_ms2 = gc_obs

        hR = info["hR"]
        vflat = info["vflat"]

        # --- フィット ---
        # (a) 深MOND
        gc_deep, chi2_deep, ndof_deep = fit_gc(
            r_kpc, gN, Vobs, errV, mode="deep")

        # (b) 完全MOND
        gc_full, chi2_full, ndof_full = fit_gc(
            r_kpc, gN, Vobs, errV, mode="full")

        # (c) BTFR方式
        gc_btfr = fit_gc_BTFR(gN, r_kpc, vflat)

        if not (np.isfinite(gc_deep) and np.isfinite(gc_full)):
            skipped += 1
            continue

        # --- 1次補正の測定 ---
        g_full, g_deep_arr, g_corr = V_fullMOND_with_correction(
            r_kpc, gN, gc_full)

        # 補正の相対的大きさ（外側半分で評価）
        n_out = max(3, len(r_kpc) // 2)
        rel_corr = np.median(np.abs(g_corr[-n_out:]) / (g_deep_arr[-n_out:] + 1e-30))

        # x = g_N/gc の中央値（外側）
        x_outer = np.median(gN[-n_out:] / gc_full)

        # 「フラットになる半径」の推定: Vobs が vflat の 90% を超える最小r
        v90 = 0.9 * vflat
        r_flat_idx = np.where(Vobs &gt;= v90)[0]
        r_flat = r_kpc[r_flat_idx[0]] if len(r_flat_idx) &gt; 0 else r_kpc[-1]

        # proxy
        GS0_proxy = (vflat * 1e3)**2 / (hR * kpc_to_m)  # m/s^2
        gc_geom = np.sqrt(a0 * GS0_proxy)

        records.append({
            "name": gal_name,
            "gc_obs": gc_obs_ms2,
            "gc_deep": gc_deep,
            "gc_full": gc_full,
            "gc_btfr": gc_btfr,
            "gc_geom": gc_geom,
            "chi2_deep": chi2_deep,
            "chi2_full": chi2_full,
            "hR": hR,
            "vflat": vflat,
            "GS0_proxy": GS0_proxy,
            "rel_corr": rel_corr,
            "x_outer": x_outer,
            "r_flat_hR": r_flat / hR if hR &gt; 0 else np.nan,
            "delta_gc": np.log10(gc_full) - np.log10(gc_deep),
        })

    N = len(records)
    print(f"Processed: {N}, Skipped: {skipped}")

    # NumPy配列化
    gc_obs = np.array([r["gc_obs"] for r in records])
    gc_deep = np.array([r["gc_deep"] for r in records])
    gc_full = np.array([r["gc_full"] for r in records])
    gc_btfr = np.array([r["gc_btfr"] for r in records])
    gc_geom = np.array([r["gc_geom"] for r in records])
    hR_arr = np.array([r["hR"] for r in records])
    vflat_arr = np.array([r["vflat"] for r in records])
    GS0_arr = np.array([r["GS0_proxy"] for r in records])
    rel_corr = np.array([r["rel_corr"] for r in records])
    x_outer = np.array([r["x_outer"] for r in records])
    r_flat_hR = np.array([r["r_flat_hR"] for r in records])
    delta_gc = np.array([r["delta_gc"] for r in records])

    # ============================================================
    # Test 1: gc_deep（深MOND）のスケーリング
    # ============================================================
    print("¥n" + "=" * 50)
```

```
    print("Test 1: gc_deep (深MONDフィット) のスケーリング")
    print("=" * 50)

    log_gcd = np.log10(gc_deep)
    log_GS0 = np.log10(GS0_arr)
    log_vf = np.log10(vflat_arr)
    log_hR = np.log10(hR_arr)

    m = np.isfinite(log_gcd) &amp; np.isfinite(log_GS0)
    sl = linregress(log_GS0[m], log_gcd[m])
    print(f"  log(gc_deep) vs log(vflat^2/hR): slope = {sl.slope:.4f} +/- {sl.stderr:.4f}")
    print(f"  R^2 = {sl.rvalue**2:.4f}")
    print(f"  期待: slope = 0.5 -&gt; p(0.5) = {abs(sl.slope-0.5)/sl.stderr:.2f}σ")

    # 多変量: log(gc_deep) = a*log(vflat) + b*log(hR) + const
    from numpy.linalg import lstsq
    X = np.column_stack([log_vf[m], log_hR[m], np.ones(m.sum())])
    y = log_gcd[m]
    coef, res, _, _ = lstsq(X, y, rcond=None)
    y_pred = X @ coef
    ss_res = np.sum((y - y_pred)**2)
    ss_tot = np.sum((y - np.mean(y))**2)
    R2_mv = 1 - ss_res / ss_tot
    print(f"  多変量: gc_deep ~ vflat^{coef[0]:.3f} x hR^{coef[1]:.3f}, R^2={R2_mv:.4f}")
    print(f"  proxy^0.5期待: vflat^1.0, hR^(-0.5)")

    # ============================================================
    # Test 2: gc_full (完全MOND) のスケーリング
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 2: gc_full (完全MONDフィット) のスケーリング")
    print("=" * 50)

    log_gcf = np.log10(gc_full)
    m2 = np.isfinite(log_gcf) &amp; np.isfinite(log_GS0)
    sl2 = linregress(log_GS0[m2], log_gcf[m2])
    print(f"  log(gc_full) vs log(vflat^2/hR): slope = {sl2.slope:.4f} +/- {sl2.stderr:.4f}")
    print(f"  R^2 = {sl2.rvalue**2:.4f}")

    coef2, _, _, _ = lstsq(
        np.column_stack([log_vf[m2], log_hR[m2], np.ones(m2.sum())]),
        log_gcf[m2], rcond=None)
    y_pred2 = np.column_stack([log_vf[m2], log_hR[m2], np.ones(m2.sum())]) @ coef2
    R2_mv2 = 1 - np.sum((log_gcf[m2] - y_pred2)**2) / np.sum((log_gcf[m2] - np.mean(log_gcf[m2]))**2)
    print(f"  多変量: gc_full ~ vflat^{coef2[0]:.3f} x hR^{coef2[1]:.3f}, R^2={R2_mv2:.4f}")

    # ============================================================
    # Test 3: gc_obs vs gc_full, gc_deep の直接比較
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 3: gc予測 vs gc_obs 直接比較")
    print("=" * 50)

    log_gco = np.log10(gc_obs)

    for label, gc_p in [("gc_deep", gc_deep), ("gc_full", gc_full),
                        ("gc_btfr", gc_btfr), ("gc_geom", gc_geom)]:
        log_p = np.log10(gc_p)
        m = np.isfinite(log_gco) &amp; np.isfinite(log_p)
        if m.sum() &lt; 10:
            print(f"  {label}: insufficient data ({m.sum()})")
            continue
        rho, p = pearsonr(log_p[m], log_gco[m])
        ratio = gc_p[m] / gc_obs[m]
        ss_r = np.sum((log_gco[m] - log_p[m])**2)
        ss_t = np.sum((log_gco[m] - np.mean(log_gco[m]))**2)
        R2 = 1 - ss_r / ss_t if ss_t &gt; 0 else np.nan
        print(f"  {label:10s}: ρ={rho:.4f}, R^2(1:1)={R2:.4f}, "
              f"ratio median={np.median(ratio):.3f}")

    # ============================================================
    # Test 4: 1次補正 (gc_full - gc_deep) のhR依存性
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 4: 1次補正 Δlog(gc) = log(gc_full/gc_deep) の構造")
    print("=" * 50)

    m4 = np.isfinite(delta_gc) &amp; np.isfinite(log_hR)
    print(f"  Δlog(gc) 統計: median={np.median(delta_gc[m4]):.4f}, "
          f"std={np.std(delta_gc[m4]):.4f}")

    for label, arr in [("log(hR)", log_hR), ("log(vflat)", log_vf),
                       ("x_outer", x_outer), ("r_flat/hR", r_flat_hR)]:
        m = np.isfinite(delta_gc) &amp; np.isfinite(arr)
        if m.sum() &gt; 10:
            rho, p = pearsonr(arr[m], delta_gc[m])
            print(f"  Δ vs {label:12s}: ρ={rho:+.4f}, p={p:.3e}")
```

```
411|
412|    # ============================================================
413|    # Test 5: x_outer (深MOND度) の分布とスケーリングへの影響
414|    # ============================================================
415|    print("¥n" + "=" * 50)
416|    print("Test 5: 深MOND度 x_outer = g_N/gc の分布")
417|    print("=" * 50)
418|
419|    x_f = x_outer[np.isfinite(x_outer)]
420|    print(f"  median(x_outer) = {np.median(x_f):.3f}")
421|    print(f"  mean(x_outer)   = {np.mean(x_f):.3f}")
422|    print(f"  fraction x &lt; 0.1 (深MOND): {np.mean(x_f &lt; 0.1):.1%}")
423|    print(f"  fraction x &lt; 0.3:          {np.mean(x_f &lt; 0.3):.1%}")
424|    print(f"  fraction x &lt; 1.0:          {np.mean(x_f &lt; 1.0):.1%}")
425|    print(f"  fraction x &gt; 1.0 (高面密度): {np.mean(x_f &gt; 1.0):.1%}")
426|
427|    # x_outer が小さい銀河 (純粋な深MOND) だけでスケーリングを見る
428|    for x_cut in [0.1, 0.3, 0.5, 1.0]:
429|        mask_x = (x_outer &lt; x_cut) &amp; np.isfinite(log_gcf) &amp; np.isfinite(log_GS0)
430|        n_x = mask_x.sum()
431|        if n_x &gt; 10:
432|            sl_x = linregress(log_GS0[mask_x], log_gcf[mask_x])
433|            print(f"  x&lt;{x_cut}: N={n_x},  α={sl_x.slope:.4f}+/-{sl_x.stderr:.4f}, "
434|                  f"R^2={sl_x.rvalue**2:.4f}")
435|
436|    # ============================================================
437|    # Test 6: 自己無撞着条件の代数的帰結
438|    # ============================================================
439|    print("¥n" + "=" * 50)
440|    print("Test 6: BTFR + 幾何平均法則 の自己無撞着チェック")
441|    print("=" * 50)
442|
443|    # BTFR: vflat^4 = gc*GM_bar -&gt; gc_BTFR = vflat^4/(GM_bar)
444|    # 幾何平均: gc = η*sqrt(a_0*vflat^2/hR)
445|    # 連立: vflat^4 = η*sqrt(a_0*vflat^2/hR)*GM_bar
446|    #       vflat^3 = η*GM_bar*sqrt(a_0/hR)
447|    # つまり vflat^3 ~ M_bar / sqrthR (ηがΥdのみ依存なら)
448|
449|    # M_bar ~= Υd x L_disk, L_disk は光度
450|    # BTFR との比較: vflat^4 ~ M_bar vs vflat^3 ~ M_bar/sqrthR
451|
452|    # 後者を前者で割ると: 1/vflat ~ 1/sqrthR -&gt; vflat^2 ~ hR
453|    # これは観測的に検証可能
454|
455|    m6 = np.isfinite(log_vf) &amp; np.isfinite(log_hR)
456|    sl6 = linregress(log_hR[m6], log_vf[m6])
457|    print(f"  log(vflat) vs log(hR): slope={sl6.slope:.4f}+/-{sl6.stderr:.4f}, "
458|          f"R^2={sl6.rvalue**2:.4f}")
459|    print(f"  自己無撞着予測: slope=0.5 (vflat^2~hR)")
460|    print(f"  実測からの乖離: {abs(sl6.slope - 0.5)/sl6.stderr:.1f}σ")
461|
462|    # vflat^2 vs hR (直接)
463|    sl6b = linregress(log_hR[m6], 2*log_vf[m6])
464|    print(f"  log(vflat^2) vs log(hR): slope={sl6b.slope:.4f}")
465|
466|    # 残差のhR依存性 (BTFR残差)
467|    # BTFR: log(gc) ~= 4*log(vflat) + const
468|    # 幾何平均: log(gc) ~= log(vflat) - 0.5*log(hR) + const
469|    # 差: 3*log(vflat) ~= -0.5*log(hR) + const
470|    # -&gt; log(vflat) ~= -1/6*log(hR) &lt;- 反相関?
471|
472|    print(f"¥n  注: BTFR(指数4)と幾何平均(指数1,-0.5)の連立は")
473|    print(f"  vflat^3 ~ M_bar/sqrthR を予測。")
474|    print(f"  vflat vs hR の slope={sl6.slope:.3f} が 0.5 でないなら、")
475|    print(f"  BTFR指数は厳密に4ではない。")
476|
477|    # ============================================================
478|    # プロット
479|    # ============================================================
480|    try:
481|        import matplotlib
482|        matplotlib.use("Agg")
483|        import matplotlib.pyplot as plt
484|
485|        fig, axes = plt.subplots(2, 3, figsize=(16, 10))
486|        fig.suptitle("P5: Deep MOND Perturbation Analysis", fontsize=14)
487|
488|        # (a) gc_full vs gc_obs
489|        ax = axes[0, 0]
490|        m = np.isfinite(log_gcf) &amp; np.isfinite(log_gco)
491|        ax.scatter(log_gco[m], log_gcf[m], s=8, alpha=0.5)
492|        lim = [min(log_gco[m].min(), log_gcf[m].min())-0.1,
493|               max(log_gco[m].max(), log_gcf[m].max())+0.1]
494|        ax.plot(lim, lim, "k--", lw=1)
495|        ax.set_xlabel("log_1_0(gc_obs)")
496|        ax.set_ylabel("log_1_0(gc_full)")
497|        ax.set_title("(a) gc_full vs gc_obs")
```

```
        ax.set_aspect("equal")

        # (b) gc_full スケーリング
        ax = axes[0, 1]
        m = np.isfinite(log_gcf) &amp;amp; np.isfinite(log_GS0)
        ax.scatter(log_GS0[m], log_gcf[m], s=8, alpha=0.5)
        x_line = np.linspace(log_GS0[m].min(), log_GS0[m].max(), 50)
        ax.plot(x_line, sl2.slope * x_line + sl2.intercept, "r-",
                label=f"slope={sl2.slope:.3f}")
        ax.plot(x_line, 0.5 * x_line + np.median(log_gcf[m] - 0.5*log_GS0[m]),
                "g--", label="slope=0.5 (期待)")
        ax.set_xlabel("log_1_0(vflat^2/hR)")
        ax.set_ylabel("log_1_0(gc_full)")
        ax.set_title("(b) gc_full scaling")
        ax.legend(fontsize=9)

        # (c) gc_deep vs gc_full
        ax = axes[0, 2]
        m = np.isfinite(log_gcd) &amp;amp; np.isfinite(log_gcf)
        ax.scatter(log_gcf[m], log_gcd[m], s=8, alpha=0.5)
        lim2 = [min(log_gcf[m].min(), log_gcd[m].min())-0.1,
                max(log_gcf[m].max(), log_gcd[m].max())+0.1]
        ax.plot(lim2, lim2, "k--")
        ax.set_xlabel("log_1_0(gc_full)")
        ax.set_ylabel("log_1_0(gc_deep)")
        ax.set_title("(c) Deep vs Full MOND")
        ax.set_aspect("equal")

        # (d) 1次補正 vs x_outer
        ax = axes[1, 0]
        m = np.isfinite(delta_gc) &amp;amp; np.isfinite(x_outer)
        ax.scatter(x_outer[m], delta_gc[m], s=8, alpha=0.5)
        ax.axhline(0, color="k", ls="--")
        ax.set_xlabel("x_outer = g_N/gc")
        ax.set_ylabel("Δlog_1_0(gc_full/gc_deep)")
        ax.set_title("(d) 1st order correction vs x")

        # (e) x_outer 分布
        ax = axes[1, 1]
        ax.hist(x_f, bins=30, edgecolor="black", alpha=0.7)
        ax.axvline(np.median(x_f), color="r", ls="--",
                   label=f"median={np.median(x_f):.2f}")
        ax.axvline(1.0, color="blue", ls=":", label="x=1 (MOND)")
        ax.set_xlabel("x_outer = g_N/gc")
        ax.set_title("(e) Deep MOND degree")
        ax.legend(fontsize=9)

        # (f) vflat vs hR
        ax = axes[1, 2]
        m = np.isfinite(log_vf) &amp;amp; np.isfinite(log_hR)
        ax.scatter(log_hR[m], log_vf[m], s=8, alpha=0.5)
        x_line = np.linspace(log_hR[m].min(), log_hR[m].max(), 50)
        ax.plot(x_line, sl6.slope * x_line + sl6.intercept, "r-",
                label=f"slope={sl6.slope:.3f}")
        ax.plot(x_line, 0.5 * x_line + np.median(log_vf[m] - 0.5*log_hR[m]),
                "g--", label="slope=0.5 (予測)")
        ax.set_xlabel("log_1_0(hR [kpc])")
        ax.set_ylabel("log_1_0(vflat [km/s])")
        ax.set_title("(f) vflat vs hR")
        ax.legend(fontsize=9)

        plt.tight_layout()
        out_fig = BASE / "P5_deepMOND_results.png"
        plt.savefig(out_fig, dpi=150)
        print(f"¥nFigure saved: {out_fig}")
    except Exception as e:
        print(f"¥nPlot skipped: {e}")

    # ============================================================
    # SUMMARY
    # ============================================================
    print("¥n" + "=" * 70)
    print("SUMMARY")
    print("=" * 70)
    print(f"  gc_deep スケーリング: α = {sl.slope:.4f}+/-{sl.stderr:.4f}")
    print(f"  gc_full スケーリング: α = {sl2.slope:.4f}+/-{sl2.stderr:.4f}")
    print(f"  gc_full 多変量: vflat^{coef2[0]:.3f} x hR^{coef2[1]:.3f}")
    print(f"  深MOND度 median(x_outer) = {np.median(x_f):.3f}")
    print(f"  1次補正 median|Δlog(gc)| = {np.median(np.abs(delta_gc[np.isfinite(delta_gc)])):.4f}")
    print(f"  vflat vs hR: slope = {sl6.slope:.4f}")

    if abs(sl2.slope - 0.5) &lt; 2 * sl2.stderr:
        print("¥n  [*] gc_full のスケーリングが α~=0.5 と整合!")
        print("  -&gt; 完全MOND補間から幾何平均法則が再現される")
    else:
        print(f"¥n  gc_full α={sl2.slope:.3f} != 0.5")
        print("  -&gt; MOND補間のみでは幾何平均法則を説明不十分")
```

```

if __name__ == "__main__":
    main()

```

4. hR偏相関の完全分解

ファイル: sparc_hR_decomposition.py

解析目的

gc と hR の偏相関 rho=-0.472 を段階的に分解する。統制変数（vflat, x_outer, f_bul, f_gas, disk_conc, Yd）を逐次追加し、各寄与を定量化。eta vs hR の相関構造、gc残差の駆動因子、BTFR(大r) vs deep(全RC) のhR指数の差を分析する。

結果

x_outer が69%説明（rho: -0.472 -> -0.149）。raw ではhR-gc無相関（rho=-0.022）。

ソースコード

(612 行)

```
#!/usr/bin/env python3
"""
sparc_hR_decomposition.py
=========================
hR偏相関 rho=-0.312 の分解

問題:
  V-1b: gc ~ vflat^1.10 x hR^(-0.56)
  P5 deep MOND: gc_deep ~ vflat^0.94 x hR^(-0.44)
  差分: hR指数に -0.12 の未説明分

候補:
  (a) 有限x補正の間接的hR依存性
  (b) Yd-hR相関 -&gt; η(Yd)を通じたhR依存性
  (c) 指数ディスク近似の破れ
  (d) バルジ/ガス比の効果

テスト:
  Test 1: Yd vs hR の相関 -&gt; (b)の検証
  Test 2: η vs hR を Yd で制御した偏相関 -&gt; Yd経由を除いた純粋hR効果
  Test 3: gc_obs vs gc_deep の残差のhR依存性の分解
  Test 4: 「有効ディスク」補正 M_bar/(2πΣ_0hR^2) のhR依存性 -&gt; (c)の検証
  Test 5: バルジ割合 f_bul vs hR残差 -&gt; (d)の検証
  Test 6: 段階的偏相関（全候補を逐次統制して残差を追跡）

実行: uv run --with scipy --with matplotlib python sparc_hR_decomposition.py
"""

import numpy as np
from scipy.optimize import minimize_scalar
from scipy.stats import pearsonr, spearmanr, linregress
from numpy.linalg import lstsq
from pathlib import Path
import csv
import sys

# ============================================================
# 定数
# ============================================================
a0 = 1.2e-10
kpc_to_m = 3.0857e19
kms2_to_ms2 = 1.0e6
Yd_default = 0.5
Yb_default = 0.7

BASE = Path(r"D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン")
ROTMOD_DIR = BASE / "Rotmod_LTG"
GC_CSV = BASE / "sparc_gc.csv"


# ============================================================
# データ読み込み
# ============================================================
def load_gc_table(path):
    data = {}
    with open(path, "r") as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = row["Galaxy"].strip()
            data[name] = {
                "gc_obs": float(row["gc"]),
                "vflat": float(row["Vflat"]),
```

```
                "hR": float(row["hR"]),
            }
            # Yd がカラムにあれば読む
            if "Yd" in row:
                try:
                    data[name]["Yd"] = float(row["Yd"])
                except:
                    data[name]["Yd"] = Yd_default
            else:
                data[name]["Yd"] = Yd_default
    return data


def load_rotmod(galaxy_name, rotmod_dir):
    fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
    if not fpath.exists():
        return None
    r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
    with open(fpath, "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
            r.append(float(parts[0]))
            vobs.append(float(parts[1]))
            errv.append(float(parts[2]))
            vgas.append(float(parts[3]))
            vdisk.append(float(parts[4]))
            vbul.append(float(parts[5]))
    return (np.array(r), np.array(vobs), np.array(errv),
            np.array(vgas), np.array(vdisk), np.array(vbul))


def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
    r_m = r_kpc * kpc_to_m
    conv = kms2_to_ms2 / r_m
    gN = (np.abs(Vgas) * Vgas * conv
          + Yd * np.abs(Vdisk) * Vdisk * conv
          + Yb * np.abs(Vbul) * Vbul * conv)
    return np.abs(gN)


# ============================================================
# 各銀河の特性量計算
# ============================================================
def galaxy_properties(r_kpc, Vobs, errV, Vgas, Vdisk, Vbul, vflat, hR, Yd=0.5, Yb=0.7):
    """銀河の各種特性量を計算"""
    gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd, Yb)

    # バリオン速度成分
    Vbar_sq = (np.abs(Vgas)*Vgas + Yd*np.abs(Vdisk)*Vdisk + Yb*np.abs(Vbul)*Vbul)

    # バルジ割合: f_bul = Vbul^2/(Vbar^2) の中央値
    Vbar_sq_abs = np.abs(Vbar_sq)
    valid = Vbar_sq_abs &gt; 0
    if valid.sum() &gt; 0:
        f_bul = np.median(Yb * Vbul[valid]**2 / Vbar_sq_abs[valid])
    else:
        f_bul = 0.0

    # ガス割合: f_gas = Vgas^2/(Vbar^2)
    if valid.sum() &gt; 0:
        f_gas = np.median(Vgas[valid]**2 / Vbar_sq_abs[valid])
    else:
        f_gas = 0.0

    # M_bar の推定 (外側のg_Nから): GM_bar ~= r^2 x g_N(r_out)
    n_out = max(3, len(r_kpc) // 3)
    r_out_m = r_kpc[-n_out:] * kpc_to_m
    gN_out = gN[-n_out:]
    GM_bar = np.median(r_out_m**2 * gN_out)  # m^3/s^2

    # 有効面密度: Σ_eff = M_bar / (2π hR^2)
    hR_m = hR * kpc_to_m
    Sigma_eff = GM_bar / (2 * np.pi * hR_m**2)  # m/s^2 (加速度次元)
    # -&gt; 真の面密度は Σ = M/(2πhR^2) [kg/m^2] だが
    #   ここでは GxΣ 次元で統一 (m/s^2)

    # ディスク集中度: disk concentration = V_disk_peak / vflat
    Vdisk_peak = np.max(np.abs(Vdisk)) * np.sqrt(Yd)
    disk_conc = Vdisk_peak / vflat if vflat &gt; 0 else np.nan

    # 「指数ディスク理想比」: M_bar_ideal = (vflat^2/G) x hR x (2π)
    # 実際のM_bar との比 = M_bar / M_bar_ideal
```

```
    # これが1からずれていれば指数ディスク近似が破れている
    vflat_ms = vflat * 1e3
    GM_bar_ideal = vflat_ms**2 * hR_m * 2 * np.pi  # 次元注意: これは雑
    # より正確: BTFR gc ~ vflat^4/(GM_bar)
    # 指数ディスク: gc ~ vflat^4/(Gx2πΣ_0hR^2) = vflat^4/(2πGΣ_0hR^2)
    # proxy: GΣ_0 = vflat^2/hR -&gt; gc_ideal = vflat^4/(2πxvflat^2xhR) = vflat^2/(2πhR)
    # 実際: gc_btfr = vflat^4/GM_bar
    # 比: gc_btfr/gc_ideal = vflat^2x2πhR/GM_bar -&gt; 指数ディスクからの乖離

    # 深MONDフィット
    def fit_gc_deep(gc_trial):
        g_eff = np.sqrt(gN * gc_trial)
        V_pred = np.sqrt(r_kpc * kpc_to_m * g_eff) * 1e-3
        n_f = max(5, len(r_kpc) // 2)
        V_f = Vobs[-n_f:]
        V_p = V_pred[-n_f:]
        e_f = errV[-n_f:]
        e_f = np.where(e_f &gt; 0, e_f, np.median(Vobs) * 0.1)
        return np.sum(((V_f - V_p) / e_f)**2)

    res = minimize_scalar(fit_gc_deep, bounds=(1e-12, 1e-8), method='bounded')
    gc_deep = 10**res.x if res.x &gt; -30 else res.x
    # minimize_scalar returns the actual value, not log
    gc_deep = res.x  # 実はboundsで直接値を渡しているので...
    # 修正: minimize_scalarのboundsは直接gcの値
    res2 = minimize_scalar(lambda lgc: fit_gc_deep(10**lgc),
                           bounds=(-12, -8), method='bounded')
    gc_deep = 10**res2.x

    # x_outer (深MOND度)
    x_outer = np.median(gN[-n_out:] / gc_deep) if gc_deep &gt; 0 else np.nan

    # η の計算: gc_obs = η^2 x a_0 x proxy -&gt; η = gc_obs / sqrt(a_0 x proxy)
    proxy = vflat_ms**2 / hR_m
    # gc_obsは後で使う

    return {
        "gN": gN,
        "f_bul": f_bul,
        "f_gas": f_gas,
        "GM_bar": GM_bar,
        "Sigma_eff": Sigma_eff,
        "disk_conc": disk_conc,
        "gc_deep": gc_deep,
        "x_outer": x_outer,
        "proxy": proxy,
    }


# ============================================================
# 偏相関関数
# ============================================================
def partial_corr(x, y, controls):
    """x, y の偏相関 (controls をリストで制御) """
    if len(controls) == 0:
        return pearsonr(x, y)

    # 各変数からcontrolsを回帰で除去
    C = np.column_stack(controls + [np.ones(len(x))])

    coef_x, _, _, _ = lstsq(C, x, rcond=None)
    res_x = x - C @ coef_x

    coef_y, _, _, _ = lstsq(C, y, rcond=None)
    res_y = y - C @ coef_y

    return pearsonr(res_x, res_y)


# ============================================================
# メイン
# ============================================================
def main():
    print("=" * 70)
    print("hR偏相関の分解: rho=-0.312 の物理的起源")
    print("=" * 70)

    gc_table = load_gc_table(GC_CSV)
    print(f"sparc_gc.csv: {len(gc_table)} galaxies")

    records = []
    for gal_name, info in gc_table.items():
        rotmod = load_rotmod(gal_name, ROTMOD_DIR)
        if rotmod is None:
            continue
        r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
        if len(r_kpc) &lt; 8:
```

```
            continue

        gc_obs = info["gc_obs"]
        if gc_obs &gt; 1e-5:
            gc_obs_ms2 = gc_obs * a0
        else:
            gc_obs_ms2 = gc_obs

        hR = info["hR"]
        vflat = info["vflat"]
        Yd = info.get("Yd", Yd_default)

        props = galaxy_properties(r_kpc, Vobs, errV, Vgas, Vdisk, Vbul,
                                  vflat, hR, Yd=Yd)

        # η
        gc_geom = np.sqrt(a0 * props["proxy"])
        eta = gc_obs_ms2 / gc_geom if gc_geom &gt; 0 else np.nan

        records.append({
            "name": gal_name,
            "gc_obs": gc_obs_ms2,
            "gc_deep": props["gc_deep"],
            "hR": hR,
            "vflat": vflat,
            "Yd": Yd,
            "f_bul": props["f_bul"],
            "f_gas": props["f_gas"],
            "GM_bar": props["GM_bar"],
            "Sigma_eff": props["Sigma_eff"],
            "disk_conc": props["disk_conc"],
            "x_outer": props["x_outer"],
            "proxy": props["proxy"],
            "eta": eta,
        })

    N = len(records)
    print(f"Processed: {N}")

    # 配列化
    gc_obs = np.array([r["gc_obs"] for r in records])
    gc_deep = np.array([r["gc_deep"] for r in records])
    hR = np.array([r["hR"] for r in records])
    vflat = np.array([r["vflat"] for r in records])
    Yd = np.array([r["Yd"] for r in records])
    f_bul = np.array([r["f_bul"] for r in records])
    f_gas = np.array([r["f_gas"] for r in records])
    GM_bar = np.array([r["GM_bar"] for r in records])
    disk_conc = np.array([r["disk_conc"] for r in records])
    x_outer = np.array([r["x_outer"] for r in records])
    proxy = np.array([r["proxy"] for r in records])
    eta = np.array([r["eta"] for r in records])

    log_gc = np.log10(gc_obs)
    log_gcd = np.log10(gc_deep)
    log_hR = np.log10(hR)
    log_vf = np.log10(vflat)
    log_Yd = np.log10(Yd)
    log_eta = np.log10(np.abs(eta))
    log_proxy = np.log10(proxy)

    # 有効マスク
    m = (np.isfinite(log_gc) &amp; np.isfinite(log_hR) &amp; np.isfinite(log_vf)
         &amp; np.isfinite(log_gcd) &amp; np.isfinite(log_eta) &amp; np.isfinite(log_Yd)
         &amp; np.isfinite(x_outer) &amp; np.isfinite(f_bul) &amp; np.isfinite(f_gas)
         &amp; np.isfinite(disk_conc) &amp; (eta &gt; 0))

    print(f"Valid for analysis: {m.sum()}")

    # ============================================================
    # Test 0: ベースライン偏相関の再確認
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 0: ベースライン偏相関")
    print("=" * 50)

    rho_raw, p_raw = pearsonr(log_hR[m], log_gc[m])
    print(f"  raw: rho(hR, gc) = {rho_raw:+.4f}, p = {p_raw:.3e}")

    rho_pv, p_pv = partial_corr(log_hR[m], log_gc[m], [log_vf[m]])
    print(f"  |vflat: rho(hR, gc | vflat) = {rho_pv:+.4f}, p = {p_pv:.3e}")

    rho_pvy, p_pvy = partial_corr(log_hR[m], log_gc[m], [log_vf[m], log_Yd[m]])
    print(f"  |vflat,Yd: rho = {rho_pvy:+.4f}, p = {p_pvy:.3e}")

    # ============================================================
    # Test 1: Yd vs hR の相関
```

```
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 1: Yd vs hR の相関")
    print("=" * 50)

    rho_Yd_hR, p_Yd_hR = pearsonr(log_hR[m], log_Yd[m])
    print(f"  rho(log Yd, log hR) = {rho_Yd_hR:+.4f}, p = {p_Yd_hR:.3e}")

    # Yd が全て同じ値 (0.5) なら相関は出ない
    Yd_unique = np.unique(Yd[m])
    print(f"  Yd unique values: {len(Yd_unique)} (min={Yd[m].min():.3f}, max={Yd[m].max():.3f})")

    if len(Yd_unique) &lt;= 1:
        print("  WARNING: Yd が全銀河で同一値 -&gt; η(Yd)経由のhR効果は検証不能")
        print("  sparc_gc.csv に個別Yd情報がないため、SPS依存の検証は保留")
        Yd_varies = False
    else:
        Yd_varies = True
        # η vs hR をYd制御
        rho_eta_hR, p_eta_hR = partial_corr(log_hR[m], log_eta[m], [log_Yd[m]])
        print(f"  rho(eta, hR | Yd) = {rho_eta_hR:+.4f}, p = {p_eta_hR:.3e}")

    # ============================================================
    # Test 2: η vs hR (直接)
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 2: eta vs hR (直接＋制御) ")
    print("=" * 50)

    rho_eh, p_eh = pearsonr(log_hR[m], log_eta[m])
    print(f"  raw: rho(eta, hR) = {rho_eh:+.4f}, p = {p_eh:.3e}")

    # vflat 制御
    rho_ehv, p_ehv = partial_corr(log_hR[m], log_eta[m], [log_vf[m]])
    print(f"  |vflat: rho(eta, hR | vflat) = {rho_ehv:+.4f}, p = {p_ehv:.3e}")

    # proxy 制御
    rho_ehp, p_ehp = partial_corr(log_hR[m], log_eta[m], [log_proxy[m]])
    print(f"  |proxy: rho(eta, hR | proxy) = {rho_ehp:+.4f}, p = {p_ehp:.3e}")

    # vflat + x_outer 制御
    rho_ehvx, p_ehvx = partial_corr(log_hR[m], log_eta[m],
                                    [log_vf[m], np.log10(x_outer[m] + 1e-10)])
    print(f"  |vflat,x: rho(eta, hR | vflat,x) = {rho_ehvx:+.4f}, p = {p_ehvx:.3e}")

    # ============================================================
    # Test 3: gc残差 = log(gc_obs/gc_deep) のhR依存性
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 3: gc残差 = log(gc_obs/gc_deep) の分解")
    print("=" * 50)

    resid = log_gc[m] - log_gcd[m]
    print(f"  残差統計: median = {np.median(resid):.4f}, std = {np.std(resid):.4f}")

    rho_rh, p_rh = pearsonr(log_hR[m], resid)
    print(f"  rho(残差, hR) = {rho_rh:+.4f}, p = {p_rh:.3e}")

    rho_rv, p_rv = pearsonr(log_vf[m], resid)
    print(f"  rho(残差, vflat) = {rho_rv:+.4f}, p = {p_rv:.3e}")

    rho_rx, p_rx = pearsonr(np.log10(x_outer[m] + 1e-10), resid)
    print(f"  rho(残差, x_outer) = {rho_rx:+.4f}, p = {p_rx:.3e}")

    rho_rf, p_rf = pearsonr(f_bul[m], resid)
    print(f"  rho(残差, f_bul) = {rho_rf:+.4f}, p = {p_rf:.3e}")

    rho_rg, p_rg = pearsonr(f_gas[m], resid)
    print(f"  rho(残差, f_gas) = {rho_rg:+.4f}, p = {p_rg:.3e}")

    rho_rd, p_rd = pearsonr(disk_conc[m], resid)
    print(f"  rho(残差, disk_conc) = {rho_rd:+.4f}, p = {p_rd:.3e}")

    # 残差のhR偏相関 (vflat制御)
    rho_rhv, p_rhv = partial_corr(log_hR[m], resid, [log_vf[m]])
    print(f"  rho(残差, hR | vflat) = {rho_rhv:+.4f}, p = {p_rhv:.3e}")

    # ============================================================
    # Test 4: 指数ディスク近似の乖離度
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 4: 指数ディスク近似の乖離")
    print("=" * 50)

    # 指数ディスク理想: gc_ideal = vflat^4 / (2π G Σ_0 hR^2)
    # ここで GΣ_0 = proxy = vflat^2/hR を使うと gc_ideal = vflat^2/(2π hR)
    # 深MONDフィットの gc_deep との比: gc_deep / gc_ideal
```

```
    vflat_ms = vflat * 1e3
    hR_m = hR * kpc_to_m
    gc_ideal = vflat_ms[m]**2 / (2 * np.pi * hR_m[m])
    deviation = np.log10(gc_deep[m] / gc_ideal)

    print(f"  log(gc_deep / gc_ideal): median = {np.median(deviation):.4f}, "
          f"std = {np.std(deviation):.4f}")

    rho_dh, p_dh = pearsonr(log_hR[m], deviation)
    print(f"  rho(deviation, hR) = {rho_dh:+.4f}, p = {p_dh:.3e}")

    rho_dv, p_dv = pearsonr(log_vf[m], deviation)
    print(f"  rho(deviation, vflat) = {rho_dv:+.4f}, p = {p_dv:.3e}")

    # ============================================================
    # Test 5: バルジ・ガス割合の効果
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 5: バルジ・ガス割合の効果")
    print("=" * 50)

    # f_bul, f_gas vs hR
    rho_bh, p_bh = pearsonr(log_hR[m], f_bul[m])
    print(f"  rho(f_bul, hR) = {rho_bh:+.4f}, p = {p_bh:.3e}")

    rho_gh, p_gh = pearsonr(log_hR[m], f_gas[m])
    print(f"  rho(f_gas, hR) = {rho_gh:+.4f}, p = {p_gh:.3e}")

    # gc のhR偏相関に f_bul を追加制御
    rho_pvb, p_pvb = partial_corr(log_hR[m], log_gc[m],
                                  [log_vf[m], f_bul[m]])
    print(f"  rho(gc, hR | vflat, f_bul) = {rho_pvb:+.4f}, p = {p_pvb:.3e}")

    # gc のhR偏相関に f_gas を追加制御
    rho_pvg, p_pvg = partial_corr(log_hR[m], log_gc[m],
                                  [log_vf[m], f_gas[m]])
    print(f"  rho(gc, hR | vflat, f_gas) = {rho_pvg:+.4f}, p = {p_pvg:.3e}")

    # 全て制御
    rho_all, p_all = partial_corr(log_hR[m], log_gc[m],
                                  [log_vf[m], f_bul[m], f_gas[m],
                                   np.log10(x_outer[m] + 1e-10)])
    print(f"  rho(gc, hR | vflat, f_bul, f_gas, x) = {rho_all:+.4f}, p = {p_all:.3e}")

    # ============================================================
    # Test 6: 段階的偏相関（逐次統制）
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 6: 段階的偏相関（hR効果の逐次分解）")
    print("=" * 50)

    log_x = np.log10(x_outer[m] + 1e-10)

    steps = [
        ("raw", []),
        ("|vflat", [log_vf[m]]),
        ("|vflat,x_outer", [log_vf[m], log_x]),
        ("|vflat,x,f_bul", [log_vf[m], log_x, f_bul[m]]),
        ("|vflat,x,f_bul,f_gas", [log_vf[m], log_x, f_bul[m], f_gas[m]]),
        ("|vflat,x,f_bul,f_gas,disk_conc",
         [log_vf[m], log_x, f_bul[m], f_gas[m], disk_conc[m]]),
    ]

    print(f"  {'制御変数':&lt;40s} {'rho':&gt;8s} {'p':&gt;12s} {'|rho|変化':&gt;10s}")
    print("  " + "-" * 72)
    prev_rho = None
    for label, controls in steps:
        valid_controls = []
        for c in controls:
            if np.isfinite(c).all():
                valid_controls.append(c)
            else:
                mc = np.isfinite(c)
                if mc.sum() &gt; 0.9 * len(c):
                    # ほぼ有効 -&gt; NaNを中央値で埋める
                    c_filled = c.copy()
                    c_filled[~np.isfinite(c)] = np.nanmedian(c)
                    valid_controls.append(c_filled)

        rho, p = partial_corr(log_hR[m], log_gc[m], valid_controls)
        change = f"{abs(rho) - abs(prev_rho):+.4f}" if prev_rho is not None else "--"
        print(f"  {label:&lt;40s} {rho:+.4f}   p={p:.3e}   {change:&gt;10s}")
        prev_rho = rho

    # ============================================================
    # Test 7: 深MOND構造からの期待hR指数の導出
    # ============================================================
```

```
    print("¥n" + "=" * 50)
    print("Test 7: hR指数の理論値 vs 観測値")
    print("=" * 50)

    # 深MOND BTFR: gc = vflat^4 / (G M_bar)
    # M_bar = L x Υd, ディスク luminosity L ~ Σ_0_L x hR^2
    # Σ_0 = Υd x Σ_0_L -&gt; M_bar = Σ_0/Υd x hR^2 x 2π... &lt;- Υd が入る
    # gc = vflat^4 / (2π G Σ_0 hR^2)
    # proxy = GΣ_0 = vflat^2/hR (定義)
    # -&gt; gc = vflat^4 / (2π x vflat^2 x hR) = vflat^2/(2πhR) &lt;- hR^(-1)
    # しかし幾何平均法則は gc ~ proxy^0.5 = (vflat^2/hR)^0.5 &lt;- hR^(-0.5)
    #
    # 矛盾? いや:
    # gc = vflat^4/(GM_bar) は gc ~ vflat^4/M_bar
    # gc ~ (proxy)^0.5 は gc ~ vflat/hR^0.5
    # これらが同時成立 -&gt; vflat^4/M_bar ~ vflat/hR^0.5
    # -&gt; vflat^3 ~ M_bar/hR^0.5
    # -&gt; M_bar ~ vflat^3 x hR^0.5
    # 一方 M_bar ~ Σ_0 hR^2 で Σ_0 = vflat^2/(G hR) (proxy定義)
    # -&gt; M_bar ~ (vflat^2/hR) x hR^2 = vflat^2 x hR
    # -&gt; vflat^3 x hR^0.5 ~ vflat^2 x hR -&gt; vflat ~ hR^0.5
    # Test 6 で slope=0.545 -&gt; 自己無撞着

    # つまり:
    # BTFR単独: gc ~ vflat^4/M_bar -&gt; hR指数は M_bar の hR 依存性で決まる
    # M_bar ~ vflat^2 x hR (proxy定義から) -&gt; gc ~ vflat^2/hR -&gt; hR^(-1)
    # しかし vflat ~ hR^0.5 を代入すると gc ~ hR/hR = hR^0 ???

    # もう少し丁寧に:
    # gc ~ (vflat^2/hR)^alpha で、alpha の値を見る
    # gc = vflat^4/(GM_bar), M_bar = 2π Σ_0hR^2, GΣ_0 = vflat^2/hR
    # -&gt; gc = vflat^4/(2π x (vflat^2/hR) x hR^2) = vflat^4/(2π vflat^2 hR) = vflat^2/(2πhR)
    # -&gt; gc ~ vflat^2/hR = proxy^1
    # 深MONDのBTFRでは alpha=1 のはず!
    # しかし gc_deep フィットでは alpha=0.47 が出ている

    # なぜ? -&gt; gc_deep フィットは「回転曲線全体」をフィットしており、
    # 大rでの BTFR とは異なる。有限rでの g_N(r) の構造が入る。

    print("  理論的分析:")
    print("  BTFR厳密 (大r極限): gc = vflat^4/(GM_bar)")
    print("    M_bar = 2pi Sigma0 hR^2, G Sigma0 = vflat^2/hR")
    print("    -&gt; gc = vflat^2/(2pi hR) -&gt; proxy^1.0")
    print("")
    print("  gc_deep フィット (回転曲線全体): alpha = 0.47")
    print("    -&gt; 有限rでの g_N(r) プロファイル構造が alpha を下げている")
    print("")

    # 直接検証: gc_btfr (大r極限) vs gc_deep (全体フィット)
    n_out = 3
    gc_btfr_arr = []
    for rec in records:
        rm = load_rotmod(rec["name"], ROTMOD_DIR)
        if rm is None:
            gc_btfr_arr.append(np.nan)
            continue
        r_kpc_i, _, _, Vgas_i, Vdisk_i, Vbul_i = rm
        gN_i = compute_gN(r_kpc_i, Vgas_i, Vdisk_i, Vbul_i)
        vf_ms = rec["vflat"] * 1e3
        n_o = max(3, len(r_kpc_i) // 3)
        r_out = r_kpc_i[-n_o:] * kpc_to_m
        gN_out = gN_i[-n_o:]
        valid_o = gN_out &gt; 0
        if valid_o.sum() &gt; 0:
            gc_btfr_i = np.median(vf_ms**4 / (r_out[valid_o]**2 * gN_out[valid_o]))
            gc_btfr_arr.append(gc_btfr_i)
        else:
            gc_btfr_arr.append(np.nan)

    gc_btfr = np.array(gc_btfr_arr)
    log_gcb = np.log10(gc_btfr)
    m_btfr = m &amp;amp; np.isfinite(log_gcb)

    if m_btfr.sum() &gt; 10:
        sl_btfr = linregress(log_proxy[m_btfr], log_gcb[m_btfr])
        sl_deep = linregress(log_proxy[m], log_gcd[m])
        print(f"  gc_btfr (大r) vs proxy: alpha = {sl_btfr.slope:.4f} +/- {sl_btfr.stderr:.4f}")
        print(f"  gc_deep (全体) vs proxy: alpha = {sl_deep.slope:.4f} +/- {sl_deep.stderr:.4f}")
        print(f"  差: {sl_btfr.slope - sl_deep.slope:.4f}")
        print(f"  -&gt; 有限rの g_N 構造が alpha を {sl_btfr.slope:.3f} -&gt; {sl_deep.slope:.3f} に変える")

    # ============================================================
    # SUMMARY
    # ============================================================
    print("¥n" + "=" * 70)
    print("SUMMARY: hR偏相関の分解")
    print("=" * 70)
```

```
585|
586|     print(f"¥n  ベースライン: rho(gc, hR | vflat) = {rho_pv:+.4f}")
587|     print(f"  全変数統制後: rho(gc, hR | vflat,x,f_bul,f_gas,disk_conc) = {rho:+.4f}")
588|     print(f"  低減量: {abs(rho_pv) - abs(rho):.4f}")
589|     print(f"¥n  gc残差(obs-deep) の hR 相関: {rho_rh:+.4f}")
590|     print(f"  eta の hR 相関: {rho_eh:+.4f}")
591|     print(f"  eta|vflat の hR 相関: {rho_ehv:+.4f}")
592|
593|     print(f"¥n  hR偏相関の構成:")
594|     print(f"    (i)  深MOND構造 (gc_deep hR^-0.44): 大部分")
595|     print(f"    (ii) gc残差(obs-deep)中の hR: rho={rho_rh:+.4f}")
596|     print(f"    (iii) eta中のhR: rho={rho_eh:+.4f}")
597|
598|     if abs(rho_rh) &lt; 0.1 and abs(rho_eh) &lt; 0.1:
599|         print("¥n  -&gt; 残差にもetaにもhR効果はほぼなし")
600|         print("  -&gt; hR偏相関は 深MOND構造 (hR^-0.44) でほぼ完全に説明される")
601|     elif abs(rho_rh) &gt; 0.1:
602|         print(f"¥n  -&gt; gc残差にhR効果あり (rho={rho_rh:+.3f})")
603|         print("  -&gt; 有限x補正 or ディスク形状効果が寄与")
604|
605|     if abs(rho_eh) &gt; 0.15:
606|         print(f"¥n  -&gt; etaにhR依存性あり (rho={rho_eh:+.3f})")
607|         print("  -&gt; Υd-hR相関 or 追加の物理的効果")
608|
609|
610| if __name__ == "__main__":
611|     main()
612|
```

5. 残り31%: 高次形状パラメータ

ファイル: sparc_hR_residual31.py

解析目的

x_outer で69%説明後の残り31%（rho=-0.145）を、9つの新しい形状パラメータ（gN曲率、勾配比、遷移幅、Vdiskピーク比、resid_asymm、gN_curvature、Vbar/Vobs、flatness、rmax/hR）で説明できるか検証する。

結果

無効: 全パラメータ delta|rho|<0.01。rmax/hR>8で rho=-0.507 と急増（物理的効果の示唆）。

ソースコード

(580 行)

```
#!/usr/bin/env python3
"""
sparc_hR_residual31.py
======================
hR偏相関の残り31%（rho=-0.21）の追求

既に否定: f_bul, f_gas, disk_conc, Yd(定数) -&gt; 効果なし
x_outer が 69% を説明済み

新しい候補:
  (i)   g_N プロファイルの高次形状パラメータ（曲率、勾配比）
  (ii)  rmax/hR（観測範囲の影響）
  (iii) g_N 遷移幅（g_N ~ a_0 付近の急峻さ）
  (iv)  Vdisk プロファイル形状（指数ディスクからの乖離度）
  (v)   深MONDフィットの残差構造（内側vs外側の系統差）

テスト:
  Test 1: 新しい形状パラメータの定義と計算
  Test 2: 各パラメータ vs hR の相関
  Test 3: 段階的偏相関（x_outer + 新パラメータ）
  Test 4: gc残差(obs-deep)の新パラメータ依存性
  Test 5: 「完全モデル」-- 何%まで説明可能か

実行: uv run --with scipy --with matplotlib python sparc_hR_residual31.py
"""

import numpy as np
from scipy.optimize import minimize_scalar
from scipy.stats import pearsonr, linregress
from numpy.linalg import lstsq
from pathlib import Path
import csv
import sys

# ============================================================
a0 = 1.2e-10
kpc_to_m = 3.0857e19
kms2_to_ms2 = 1.0e6
Yd_default = 0.5
Yb_default = 0.7

BASE = Path(r"D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン")
ROTMOD_DIR = BASE / "Rotmod_LTG"
GC_CSV = BASE / "sparc_gc.csv"


def load_gc_table(path):
    data = {}
    with open(path, "r") as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = row["Galaxy"].strip()
            data[name] = {
                "gc_obs": float(row["gc"]),
                "vflat": float(row["Vflat"]),
                "hR": float(row["hR"]),
            }
    return data


def load_rotmod(galaxy_name, rotmod_dir):
    fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
    if not fpath.exists():
```

```
        return None
    r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
    with open(fpath, "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
            r.append(float(parts[0]))
            vobs.append(float(parts[1]))
            errv.append(float(parts[2]))
            vgas.append(float(parts[3]))
            vdisk.append(float(parts[4]))
            vbul.append(float(parts[5]))
    return (np.array(r), np.array(vobs), np.array(errv),
            np.array(vgas), np.array(vdisk), np.array(vbul))


def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
    r_m = r_kpc * kpc_to_m
    conv = kms2_to_ms2 / r_m
    gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
          + Yb*np.abs(Vbul)*Vbul*conv)
    return np.abs(gN)


def partial_corr(x, y, controls):
    if len(controls) == 0:
        return pearsonr(x, y)
    C = np.column_stack(controls + [np.ones(len(x))])
    coef_x, _, _, _ = lstsq(C, x, rcond=None)
    res_x = x - C @ coef_x
    coef_y, _, _, _ = lstsq(C, y, rcond=None)
    res_y = y - C @ coef_y
    return pearsonr(res_x, res_y)


# ============================================================
# 形状パラメータの計算
# ============================================================
def compute_shape_params(r_kpc, Vobs, errV, Vgas, Vdisk, Vbul,
                         vflat, hR, gc_obs_ms2):
    """g_Nプロファイルの高次形状パラメータ"""
    gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)
    N = len(r_kpc)
    params = {}

    # --- (1) rmax/hR: 観測範囲 ---
    params["rmax_hR"] = r_kpc[-1] / hR if hR &gt; 0 else np.nan

    # --- (2) g_N プロファイルの対数勾配 ---
    # d log(g_N) / d log(r) の中央値と変動
    log_r = np.log10(r_kpc)
    log_gN = np.log10(gN + 1e-30)
    valid = np.isfinite(log_gN) &amp; np.isfinite(log_r) &amp; (gN &gt; 0)

    if valid.sum() &gt;= 5:
        dr = np.diff(log_r[valid])
        dg = np.diff(log_gN[valid])
        mask_dr = np.abs(dr) &gt; 1e-10
        if mask_dr.sum() &gt;= 3:
            grad = dg[mask_dr] / dr[mask_dr]
            r_mid = (log_r[valid][:-1] + log_r[valid][1:])[mask_dr] / 2.0

            # 外側半分の勾配
            n_out = max(1, len(grad) // 2)
            params["grad_outer"] = np.median(grad[-n_out:])

            # 内側半分の勾配
            params["grad_inner"] = np.median(grad[:n_out])

            # 勾配比 (inner/outer) -- ディスク形状の指標
            if abs(params["grad_outer"]) &gt; 0.01:
                params["grad_ratio"] = params["grad_inner"] / params["grad_outer"]
            else:
                params["grad_ratio"] = np.nan

            # 勾配の変動 (std) -- プロファイルの非単調性
            params["grad_std"] = np.std(grad)
        else:
            params["grad_outer"] = np.nan
            params["grad_inner"] = np.nan
            params["grad_ratio"] = np.nan
            params["grad_std"] = np.nan
    else:
```

```
        params["grad_outer"] = np.nan
        params["grad_inner"] = np.nan
        params["grad_ratio"] = np.nan
        params["grad_std"] = np.nan

    # --- (3) 遷移幅: g_N が gc 付近を横切る半径範囲 ---
    # g_N/gc が 0.3 -&gt; 3.0 を横切る r の幅 (hR単位)
    gc = gc_obs_ms2
    if gc &gt; 0:
        x_arr = gN / gc
        r_lo = np.nan  # x = 3.0 の位置 (内側)
        r_hi = np.nan  # x = 0.3 の位置 (外側)

        for i in range(N-1):
            if x_arr[i] &gt;= 3.0 and x_arr[i+1] &lt; 3.0:
                # 線形補間
                frac = (3.0 - x_arr[i+1]) / (x_arr[i] - x_arr[i+1])
                r_lo = r_kpc[i+1] + frac * (r_kpc[i] - r_kpc[i+1])
                break

        for i in range(N-1):
            if x_arr[i] &gt;= 0.3 and x_arr[i+1] &lt; 0.3:
                frac = (0.3 - x_arr[i+1]) / (x_arr[i] - x_arr[i+1])
                r_hi = r_kpc[i+1] + frac * (r_kpc[i] - r_kpc[i+1])
                break

        if np.isfinite(r_lo) and np.isfinite(r_hi) and r_hi &gt; r_lo:
            params["transition_width"] = (r_hi - r_lo) / hR
            params["transition_center"] = (r_hi + r_lo) / 2.0 / hR
        else:
            params["transition_width"] = np.nan
            params["transition_center"] = np.nan
    else:
        params["transition_width"] = np.nan
        params["transition_center"] = np.nan

    # --- (4) Vdisk プロファイル形状: 指数ディスクからの乖離 ---
    # 指数ディスク理論: Vdisk(r) ~ r x [I_0K_0 - I_1K_1]^(1/2)
    # 乖離度 = Vdisk実測のピーク位置 / (2.2 x hR)
    Vdisk_abs = np.abs(Vdisk)
    if np.max(Vdisk_abs) &gt; 0:
        idx_peak = np.argmax(Vdisk_abs)
        r_peak = r_kpc[idx_peak]
        params["peak_ratio"] = r_peak / (2.2 * hR)  # 理想 = 1.0

        # Vdisk の「形状集中度」: Vdisk(hR) / Vdisk_peak
        # hR での Vdisk を補間
        if r_kpc[-1] &gt; hR and r_kpc[0] &lt; hR:
            V_at_hR = np.interp(hR, r_kpc, Vdisk_abs)
            params["shape_conc"] = V_at_hR / np.max(Vdisk_abs)
        else:
            params["shape_conc"] = np.nan
    else:
        params["peak_ratio"] = np.nan
        params["shape_conc"] = np.nan

    # --- (5) 深MONDフィット残差の内外比 ---
    # gc_deep でフィットした V_pred の残差を内側 vs 外側で比較
    def fit_gc_deep_full(r, gN, Vobs, errV):
        def chi2(lgc):
            gc = 10**lgc
            g_eff = np.sqrt(gN * gc)
            V_pred = np.sqrt(r * kpc_to_m * g_eff) * 1e-3
            e = np.where(errV &gt; 0, errV, np.median(Vobs) * 0.1)
            return np.sum(((Vobs - V_pred) / e)**2)
        res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
        return 10**res.x

    gc_deep = fit_gc_deep_full(r_kpc, gN, Vobs, errV)
    params["gc_deep"] = gc_deep

    g_eff_deep = np.sqrt(gN * gc_deep)
    V_deep = np.sqrt(r_kpc * kpc_to_m * g_eff_deep) * 1e-3
    e = np.where(errV &gt; 0, errV, np.median(Vobs) * 0.1)
    frac_resid = (Vobs - V_deep) / e

    n_half = N // 2
    if n_half &gt;= 3:
        params["resid_inner"] = np.median(frac_resid[:n_half])
        params["resid_outer"] = np.median(frac_resid[n_half:])
        params["resid_asymm"] = params["resid_inner"] - params["resid_outer"]
    else:
        params["resid_inner"] = np.nan
        params["resid_outer"] = np.nan
        params["resid_asymm"] = np.nan

    # --- (6) x_outer (再計算) ---
```

```
    n_out = max(3, N // 3)
    params["x_outer"] = np.median(gN[-n_out:] / gc_deep) if gc_deep &gt; 0 else np.nan

    # --- (7) g_N の曲率 (2次微分的) ---
    if valid.sum() &gt;= 7:
        # log(gN) vs log(r) の2次フィット → 曲率
        try:
            coeffs = np.polyfit(log_r[valid], log_gN[valid], 2)
            params["gN_curvature"] = coeffs[0]  # 2次係数
        except:
            params["gN_curvature"] = np.nan
    else:
        params["gN_curvature"] = np.nan

    # --- (8) Vbar/Vobs のフラット到達度 ---
    # 外側での Vbar/Vobs = 「暗黒物質」支配度
    Vbar = np.sqrt(np.abs(
        np.abs(Vgas)*Vgas + Yd_default*np.abs(Vdisk)*Vdisk
        + Yb_default*np.abs(Vbul)*Vbul)) * np.sign(
            np.abs(Vgas)*Vgas + Yd_default*np.abs(Vdisk)*Vdisk
            + Yb_default*np.abs(Vbul)*Vbul + 1e-30)

    Vbar_abs = np.abs(Vbar)
    ratio_out = np.median(Vbar_abs[-n_out:] / (Vobs[-n_out:] + 1e-10))
    params["Vbar_Vobs_outer"] = ratio_out

    # --- (9) フラット到達度: Vobs の外側変動 ---
    V_out = Vobs[-n_out:]
    params["flatness"] = np.std(V_out) / np.mean(V_out) if np.mean(V_out) &gt; 0 else np.nan

    return params


# ============================================================
def main():
    print("=" * 70)
    print("hR偏相関の残り31%: 高次形状パラメータによる分解")
    print("=" * 70)

    gc_table = load_gc_table(GC_CSV)

    records = []
    for gal_name, info in gc_table.items():
        rotmod = load_rotmod(gal_name, ROTMOD_DIR)
        if rotmod is None:
            continue
        r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
        if len(r_kpc) &lt; 8:
            continue

        gc_obs = info["gc_obs"]
        gc_obs_ms2 = gc_obs * a0 if gc_obs &gt; 1e-5 else gc_obs
        hR = info["hR"]
        vflat = info["vflat"]

        params = compute_shape_params(r_kpc, Vobs, errV, Vgas, Vdisk, Vbul,
                                      vflat, hR, gc_obs_ms2)

        vflat_ms = vflat * 1e3
        hR_m = hR * kpc_to_m
        proxy = vflat_ms**2 / hR_m
        gc_geom = np.sqrt(a0 * proxy)
        eta = gc_obs_ms2 / gc_geom if gc_geom &gt; 0 else np.nan

        records.append({
            "name": gal_name,
            "gc_obs": gc_obs_ms2,
            "gc_deep": params["gc_deep"],
            "hR": hR,
            "vflat": vflat,
            "eta": eta,
            **params,
        })

    N = len(records)
    print(f"Processed: {N}")

    # 配列化
    def arr(key):
        return np.array([r[key] for r in records], dtype=float)

    gc_obs = arr("gc_obs")
    gc_deep = arr("gc_deep")
    hR_a = arr("hR")
    vflat_a = arr("vflat")
    eta_a = arr("eta")

```

```
    log_gc = np.log10(gc_obs)
    log_gcd = np.log10(gc_deep)
    log_hR = np.log10(hR_a)
    log_vf = np.log10(vflat_a)
    log_eta = np.log10(np.abs(eta_a))

    x_outer = arr("x_outer")
    rmax_hR = arr("rmax_hR")
    grad_outer = arr("grad_outer")
    grad_inner = arr("grad_inner")
    grad_ratio = arr("grad_ratio")
    grad_std = arr("grad_std")
    trans_width = arr("transition_width")
    trans_center = arr("transition_center")
    peak_ratio = arr("peak_ratio")
    shape_conc = arr("shape_conc")
    resid_asymm = arr("resid_asymm")
    gN_curvature = arr("gN_curvature")
    Vbar_Vobs = arr("Vbar_Vobs_outer")
    flatness = arr("flatness")
    f_bul = arr("f_bul") if "f_bul" in records[0] else np.zeros(N)

    # f_bul を再計算
    f_bul_arr = []
    f_gas_arr = []
    dc_arr = []
    for rec in records:
        rm = load_rotmod(rec["name"], ROTMOD_DIR)
        if rm is None:
            f_bul_arr.append(np.nan); f_gas_arr.append(np.nan); dc_arr.append(np.nan)
            continue
        _, Vo, _, Vg, Vd, Vb = rm
        Vbar_sq = np.abs(Vg)*Vg + Yd_default*np.abs(Vd)*Vd + Yb_default*np.abs(Vb)*Vb
        va = np.abs(Vbar_sq)
        v = va &gt; 0
        f_bul_arr.append(np.median(Yb_default*Vb[v]**2 / va[v]) if v.sum()&gt;0 else 0)
        f_gas_arr.append(np.median(Vg[v]**2 / va[v]) if v.sum()&gt;0 else 0)
        dc_arr.append(np.max(np.abs(Vd))*np.sqrt(Yd_default)/rec["vflat"] if rec["vflat"]&gt;0 else np.nan)

    f_bul = np.array(f_bul_arr)
    f_gas = np.array(f_gas_arr)
    disk_conc = np.array(dc_arr)

    # 有効マスク
    log_x = np.log10(x_outer + 1e-10)
    m = (np.isfinite(log_gc) &amp; np.isfinite(log_hR) &amp; np.isfinite(log_vf)
         &amp; np.isfinite(log_x) &amp; np.isfinite(log_gcd) &amp; (eta_a &gt; 0)
         &amp; np.isfinite(log_eta))

    print(f"Valid: {m.sum()}")

    # ============================================================
    # Test 1: 新パラメータの基本統計
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 1: 新パラメータの基本統計")
    print("=" * 50)

    new_params = {
        "rmax/hR": rmax_hR,
        "grad_outer": grad_outer,
        "grad_inner": grad_inner,
        "grad_ratio": grad_ratio,
        "grad_std": grad_std,
        "trans_width": trans_width,
        "trans_center": trans_center,
        "peak_ratio": peak_ratio,
        "shape_conc": shape_conc,
        "resid_asymm": resid_asymm,
        "gN_curvature": gN_curvature,
        "Vbar/Vobs_out": Vbar_Vobs,
        "flatness": flatness,
    }

    for name, arr_p in new_params.items():
        valid = np.isfinite(arr_p) &amp; m
        if valid.sum() &gt; 10:
            print(f"  {name:20s}: N={valid.sum():3d}, "
                  f"median={np.median(arr_p[valid]):+.3f}, "
                  f"std={np.std(arr_p[valid]):.3f}")

    # ============================================================
    # Test 2: 各パラメータ vs hR, gc の相関
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 2: パラメータ vs hR, gc の相関")
    print("=" * 50)
```

```
412|
413|     print(f"  {'パラメータ':&lt;20s} {'rho(hR)':&gt;10s} {'p(hR)':&gt;12s} "
414|           f"{'rho(gc)':&gt;10s} {'p(gc)':&gt;12s}")
415|     print("  " + "-" * 66)
416|
417|     for name, arr_p in new_params.items():
418|         valid = np.isfinite(arr_p) &amp; m
419|         if valid.sum() &gt; 15:
420|             rh, ph = pearsonr(log_hR[valid], arr_p[valid])
421|             rg, pg = pearsonr(log_gc[valid], arr_p[valid])
422|             flag_h = " *" if ph &lt; 0.01 else ""
423|             flag_g = " *" if pg &lt; 0.01 else ""
424|             print(f"  {name:&lt;20s} {rh:+.4f}{flag_h:3s}  p={ph:.3e}  "
425|                   f"{rg:+.4f}{flag_g:3s}  p={pg:.3e}")
426|
427|     # ============================================================
428|     # Test 3: 段階的偏相関 (x_outer + 新パラメータ)
429|     # ============================================================
430|     print("¥n" + "=" * 50)
431|     print("Test 3: 段階的偏相関 -- x_outer に新パラメータを追加")
432|     print("=" * 50)
433|
434|     base_controls = [log_vf[m], log_x[m]]
435|     rho_base, p_base = partial_corr(log_hR[m], log_gc[m], base_controls)
436|     print(f"  ベースライン |vflat, x_outer: rho = {rho_base:+.4f}, p = {p_base:.3e}")
437|     print()
438|
439|     # 各パラメータを1つずつ追加して効果を見る
440|     print(f"  {'追加パラメータ':&lt;20s} {'rho':&gt;8s} {'p':&gt;12s} {'Δ|rho|':&gt;10s} {'N':&gt;5s}")
441|     print("  " + "-" * 58)
442|
443|     results = []
444|     for name, arr_p in new_params.items():
445|         valid = np.isfinite(arr_p) &amp; m
446|         if valid.sum() &lt; 30:
447|             continue
448|
449|         # NaN を中央値で埋める
450|         arr_filled = arr_p.copy()
451|         nan_mask = ~np.isfinite(arr_filled)
452|         if nan_mask.sum() &gt; 0:
453|             arr_filled[nan_mask] = np.nanmedian(arr_p)
454|
455|         try:
456|             controls = base_controls + [arr_filled[m]]
457|             rho_new, p_new = partial_corr(log_hR[m], log_gc[m], controls)
458|             delta = abs(rho_new) - abs(rho_base)
459|             print(f"  {name:&lt;20s} {rho_new:+.4f}   p={p_new:.3e}   {delta:+.4f}   {valid.sum():&gt;5d}")
460|             results.append((name, rho_new, delta, arr_filled))
461|         except Exception as e:
462|             print(f"  {name:&lt;20s} ERROR: {e}")
463|
464|     # ============================================================
465|     # Test 4: 最も効果的なパラメータの組み合わせ
466|     # ============================================================
467|     print("¥n" + "=" * 50)
468|     print("Test 4: 最も効果的な組み合わせ")
469|     print("=" * 50)
470|
471|     # 効果が大きい順 (|rho|を最も減らす) にソート
472|     results.sort(key=lambda x: x[2])  # delta が最も負 = 最も効果的
473|
474|     if len(results) &gt;= 3:
475|         top3 = results[:3]
476|         print(f"  Top 3 (|rho|を最も減らす):")
477|         for name, rho, delta, _ in top3:
478|             print(f"    {name}: delta = {delta:+.4f}, rho = {rho:+.4f}")
479|
480|         # Top 3 を全て同時投入
481|         controls_top3 = base_controls + [r[3][m] for r in top3]
482|         rho_top3, p_top3 = partial_corr(log_hR[m], log_gc[m], controls_top3)
483|         print(f"¥n  Top 3 同時投入: rho = {rho_top3:+.4f}, p = {p_top3:.3e}")
484|         print(f"  ベースからの削減: {abs(rho_base) - abs(rho_top3):.4f}")
485|         print(f"  削減率: {(abs(rho_base) - abs(rho_top3))/abs(rho_base)*100:.1f}%")
486|
487|     # ============================================================
488|     # Test 5: 全パラメータ同時投入
489|     # ============================================================
490|     print("¥n" + "=" * 50)
491|     print("Test 5: 全パラメータ同時投入")
492|     print("=" * 50)
493|
494|     all_controls = base_controls.copy()
495|     for name, _, _, arr_filled in results:
496|         all_controls.append(arr_filled[m])
497|
498|     try:
```

```
        rho_all, p_all = partial_corr(log_hR[m], log_gc[m], all_controls)
        print(f"  rho = {rho_all:+.4f}, p = {p_all:.3e}")
        print(f"  ベースからの削減: {abs(rho_base) - abs(rho_all):.4f}")
        print(f"  vflat のみからの削減率: "
              f"{(0.472 - abs(rho_all))/0.472*100:.1f}% "
              f"(x_outer: 69%, 新パラメータ: {(abs(rho_base) - abs(rho_all))/0.472*100:.1f}%)")
    except Exception as e:
        print(f"  ERROR: {e}")

    # ============================================================
    # Test 6: gc残差のrmax/hR依存性（系統効果チェック）
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 6: 観測的系統効果の検証")
    print("=" * 50)

    resid = log_gc[m] - log_gcd[m]

    # rmax/hR vs 残差
    rmax_valid = np.isfinite(rmax_hR[m])
    if rmax_valid.sum() &gt; 10:
        rho_rm, p_rm = pearsonr(rmax_hR[m][rmax_valid], resid[rmax_valid])
        print(f"  rho(gc残差, rmax/hR) = {rho_rm:+.4f}, p = {p_rm:.3e}")

        # rmax/hR が大きい銀河 vs 小さい銀河で gc_deep の精度が変わるか
        rmax_med = np.median(rmax_hR[m][rmax_valid])
        hi = rmax_hR[m][rmax_valid] &gt; rmax_med
        lo = ~hi
        print(f"  rmax/hR &gt; {rmax_med:.1f}: 残差 median = {np.median(resid[rmax_valid][hi]):.4f}")
        print(f"  rmax/hR &lt; {rmax_med:.1f}: 残差 median = {np.median(resid[rmax_valid][lo]):.4f}")

    # flatness vs 残差
    flat_valid = np.isfinite(flatness[m])
    if flat_valid.sum() &gt; 10:
        rho_fl, p_fl = pearsonr(flatness[m][flat_valid], resid[flat_valid])
        print(f"  rho(gc残差, flatness) = {rho_fl:+.4f}, p = {p_fl:.3e}")

    # ============================================================
    # Test 7: rmax/hR 制限下での偏相関
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 7: rmax/hR 制限下での偏相関")
    print("=" * 50)

    for rmax_cut in [5, 8, 10, 15]:
        m_cut = m &amp;amp; (rmax_hR &gt; rmax_cut)
        if m_cut.sum() &gt; 20:
            rho_c, p_c = partial_corr(log_hR[m_cut], log_gc[m_cut],
                                      [log_vf[m_cut], log_x[m_cut]])
            print(f"  rmax/hR &gt; {rmax_cut:2d}: N={m_cut.sum():3d}, "
                  f"rho(hR|vflat,x) = {rho_c:+.4f}, p = {p_c:.3e}")

    # ============================================================
    # SUMMARY
    # ============================================================
    print("¥n" + "=" * 70)
    print("SUMMARY")
    print("=" * 70)

    print(f"¥n  |vflat:           rho = -0.472")
    print(f"  |vflat, x_outer:  rho = {rho_base:+.4f}  (x_outer が 69% 説明)")

    if len(results) &gt;= 3:
        print(f"  |vflat, x, top3:  rho = {rho_top3:+.4f}  "
              f"(追加 {(abs(rho_base) - abs(rho_top3))/0.472*100:.1f}% 説明)")

    try:
        print(f"  |全パラメータ:    rho = {rho_all:+.4f}  "
              f"(追加 {(abs(rho_base) - abs(rho_all))/0.472*100:.1f}% 説明)")
    except:
        pass

    print(f"¥n  残り31%の内訳:")
    if len(results) &gt;= 1:
        for name, rho, delta, _ in results[:5]:
            pct = -delta / 0.472 * 100
            print(f"    {name:&lt;20s}: {pct:+.1f}%")


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

6. gc決定手法の系統効果

ファイル: sparc_gc_method_systematics.py

解析目的

残り31%が TA3 tanh フィットの系統効果かを検証。7つのgc決定手法（gc_obs, gc_deep(all/out/in), gc_full(all/out), gc_tanh）でhR偏相関を比較。内側/外側フィットの差のhR依存性、最適重み付きgc、rmax/hR
サブセットでの手法間比較を実施する。

結果

逆効果: TA3 gc_obs の rho=-0.145 が全手法中最小。gc_deep -0.230、gc_full -0.351。

ソースコード

(562 行)

```
#!/usr/bin/env python3
"""
sparc_gc_method_systematics.py
==============================
gc決定手法の系統効果: TA3 tanh gc_obs vs 深MOND gc_deep

問題:
  hR偏相関の残り31%（rho=-0.145 after |vflat,x）が高次形状パラメータで消えない。
  rmax/hR &gt; 8 で rho=-0.507 と強化 -&gt; 手法依存性の可能性。

仮説:
  gc_obs（TA3 tanh fit）と gc_deep（深MONDフィット）は異なる手法で gc を決定。
  両者の差 Δ = log(gc_obs/gc_deep) が hR と相関し、
  これが残り31%の原因である可能性。

テスト:
  Test 1: Δ = log(gc_obs/gc_deep) の hR 依存性
  Test 2: gc_obs, gc_deep それぞれの hR 偏相関の比較
  Test 3: rmax/hR サブセットでの系統的比較
  Test 4: gc_deep で幾何平均法則を再フィット -&gt; alpha の値
  Test 5: gc_deep ベースでの hR 偏相関 -&gt; 31%が消えるか
  Test 6: 完全MOND gc_full でも同じ検証
  Test 7: 「外側フィット」vs「全域フィット」の gc の差

実行: uv run --with scipy --with matplotlib python sparc_gc_method_systematics.py
"""

import numpy as np
from scipy.optimize import minimize_scalar, minimize
from scipy.stats import pearsonr, linregress
from numpy.linalg import lstsq
from pathlib import Path
import csv
import sys

# ============================================================
a0 = 1.2e-10
kpc_to_m = 3.0857e19
kms2_to_ms2 = 1.0e6
Yd_default = 0.5
Yb_default = 0.7

BASE = Path(r"D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン")
ROTMOD_DIR = BASE / "Rotmod_LTG"
GC_CSV = BASE / "sparc_gc.csv"


def load_gc_table(path):
    data = {}
    with open(path, "r") as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = row["Galaxy"].strip()
            data[name] = {
                "gc_obs": float(row["gc"]),
                "vflat": float(row["Vflat"]),
                "hR": float(row["hR"]),
            }
    return data


def load_rotmod(galaxy_name, rotmod_dir):
```

```
    fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
    if not fpath.exists():
        return None
    r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
    with open(fpath, "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
            r.append(float(parts[0]))
            vobs.append(float(parts[1]))
            errv.append(float(parts[2]))
            vgas.append(float(parts[3]))
            vdisk.append(float(parts[4]))
            vbul.append(float(parts[5]))
    return (np.array(r), np.array(vobs), np.array(errv),
            np.array(vgas), np.array(vdisk), np.array(vbul))


def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
    r_m = r_kpc * kpc_to_m
    conv = kms2_to_ms2 / r_m
    gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
          + Yb*np.abs(Vbul)*Vbul*conv)
    return np.abs(gN)


def partial_corr(x, y, controls):
    if len(controls) == 0:
        return pearsonr(x, y)
    C = np.column_stack(controls + [np.ones(len(x))])
    cx, _, _, _ = lstsq(C, x, rcond=None)
    cy, _, _, _ = lstsq(C, y, rcond=None)
    return pearsonr(x - C @ cx, y - C @ cy)


# ============================================================
# 複数の gc フィット手法
# ============================================================
def fit_gc_deep(r_kpc, gN, Vobs, errV, r_range="all"):
    """深MOND: g_eff = sqrt(g_N * gc)
    r_range: "all", "outer50", "outer30", "inner50"
    """
    N = len(r_kpc)
    if r_range == "outer50":
        sl = slice(N - max(5, N//2), N)
    elif r_range == "outer30":
        sl = slice(N - max(5, N//3), N)
    elif r_range == "inner50":
        sl = slice(0, max(5, N//2))
    else:
        sl = slice(0, N)

    r_f = r_kpc[sl]
    V_f = Vobs[sl]
    e_f = errV[sl]
    gN_f = gN[sl]
    e_f = np.where(e_f &gt; 0, e_f, np.median(Vobs) * 0.1)

    def chi2(lgc):
        gc = 10**lgc
        g_eff = np.sqrt(gN_f * gc)
        V_pred = np.sqrt(r_f * kpc_to_m * g_eff) * 1e-3
        return np.sum(((V_f - V_pred) / e_f)**2)

    res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
    return 10**res.x, res.fun


def fit_gc_full(r_kpc, gN, Vobs, errV, r_range="all"):
    """完全MOND: g_eff/gc = [x + sqrt(x^2+4x)]/2"""
    N = len(r_kpc)
    if r_range == "outer50":
        sl = slice(N - max(5, N//2), N)
    elif r_range == "outer30":
        sl = slice(N - max(5, N//3), N)
    elif r_range == "inner50":
        sl = slice(0, max(5, N//2))
    else:
        sl = slice(0, N)

    r_f = r_kpc[sl]
    V_f = Vobs[sl]
    e_f = errV[sl]
```

```
    gN_f = gN[sl]
    e_f = np.where(e_f &gt; 0, e_f, np.median(Vobs) * 0.1)

    def chi2(lgc):
        gc = 10**lgc
        x = gN_f / gc
        g_eff = gc * (x + np.sqrt(x**2 + 4*x)) / 2.0
        V_pred = np.sqrt(r_f * kpc_to_m * g_eff) * 1e-3
        return np.sum(((V_f - V_pred) / e_f)**2)

    res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
    return 10**res.x, res.fun


def fit_gc_tanh(r_kpc, Vobs, errV, vflat):
    """TA3風 tanhフィット: V(r) = vflat x tanh(r/r_t)
    gc = vflat^2/(r_t x kpc_to_m) で近似
    """
    V_norm = Vobs / vflat
    e_norm = errV / vflat
    e_norm = np.where(e_norm &gt; 0, e_norm, 0.1)

    def chi2(log_rt):
        rt = 10**log_rt
        V_pred = np.tanh(r_kpc / rt)
        return np.sum(((V_norm - V_pred) / e_norm)**2)

    res = minimize_scalar(chi2, bounds=(-2, 2), method='bounded')
    rt = 10**res.x  # kpc
    gc_tanh = (vflat * 1e3)**2 / (rt * kpc_to_m)
    return gc_tanh, rt, res.fun


# ============================================================
def main():
    print("=" * 70)
    print("gc決定手法の系統効果: TA3 vs 深MOND vs 完全MOND")
    print("=" * 70)

    gc_table = load_gc_table(GC_CSV)

    records = []
    for gal_name, info in gc_table.items():
        rotmod = load_rotmod(gal_name, ROTMOD_DIR)
        if rotmod is None:
            continue
        r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
        if len(r_kpc) &lt; 8:
            continue

        gc_obs = info["gc_obs"]
        gc_obs_ms2 = gc_obs * a0 if gc_obs &gt; 1e-5 else gc_obs
        hR = info["hR"]
        vflat = info["vflat"]

        gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)

        # 各種 gc フィット
        gc_deep_all, chi2_da = fit_gc_deep(r_kpc, gN, Vobs, errV, "all")
        gc_deep_out, chi2_do = fit_gc_deep(r_kpc, gN, Vobs, errV, "outer50")
        gc_deep_in, chi2_di  = fit_gc_deep(r_kpc, gN, Vobs, errV, "inner50")
        gc_deep_o30, _       = fit_gc_deep(r_kpc, gN, Vobs, errV, "outer30")

        gc_full_all, chi2_fa = fit_gc_full(r_kpc, gN, Vobs, errV, "all")
        gc_full_out, chi2_fo = fit_gc_full(r_kpc, gN, Vobs, errV, "outer50")

        gc_tanh, rt_tanh, chi2_t = fit_gc_tanh(r_kpc, Vobs, errV, vflat)

        # x_outer
        n_out = max(3, len(r_kpc) // 3)
        x_outer = np.median(gN[-n_out:] / gc_deep_all) if gc_deep_all &gt; 0 else np.nan

        # rmax/hR
        rmax_hR = r_kpc[-1] / hR if hR &gt; 0 else np.nan

        # proxy
        vflat_ms = vflat * 1e3
        hR_m = hR * kpc_to_m
        proxy = vflat_ms**2 / hR_m

        # 内外 gc 比
        gc_ratio_io = gc_deep_in / gc_deep_out if gc_deep_out &gt; 0 else np.nan

        records.append({
            "name": gal_name,
            "gc_obs": gc_obs_ms2,
            "gc_deep_all": gc_deep_all,
```

```
            "gc_deep_out": gc_deep_out,
            "gc_deep_in": gc_deep_in,
            "gc_deep_o30": gc_deep_o30,
            "gc_full_all": gc_full_all,
            "gc_full_out": gc_full_out,
            "gc_tanh": gc_tanh,
            "rt_tanh": rt_tanh,
            "gc_ratio_io": gc_ratio_io,
            "hR": hR,
            "vflat": vflat,
            "proxy": proxy,
            "x_outer": x_outer,
            "rmax_hR": rmax_hR,
        })

    N = len(records)
    print(f"Processed: {N}")

    # 配列化
    def arr(k): return np.array([r[k] for r in records], dtype=float)

    gc_obs     = arr("gc_obs")
    gc_deep    = arr("gc_deep_all")
    gc_deep_o  = arr("gc_deep_out")
    gc_deep_i  = arr("gc_deep_in")
    gc_deep_o3 = arr("gc_deep_o30")
    gc_full    = arr("gc_full_all")
    gc_full_o  = arr("gc_full_out")
    gc_tanh    = arr("gc_tanh")
    gc_ratio   = arr("gc_ratio_io")
    hR_a       = arr("hR")
    vflat_a    = arr("vflat")
    proxy_a    = arr("proxy")
    x_outer    = arr("x_outer")
    rmax_hR    = arr("rmax_hR")

    log_gco  = np.log10(gc_obs)
    log_gcd  = np.log10(gc_deep)
    log_gcdo = np.log10(gc_deep_o)
    log_gcdi = np.log10(gc_deep_i)
    log_gcf  = np.log10(gc_full)
    log_gcfo = np.log10(gc_full_o)
    log_gct  = np.log10(gc_tanh)
    log_hR   = np.log10(hR_a)
    log_vf   = np.log10(vflat_a)
    log_pr   = np.log10(proxy_a)
    log_x    = np.log10(x_outer + 1e-10)

    m = (np.isfinite(log_gco) &amp;amp; np.isfinite(log_gcd) &amp;amp; np.isfinite(log_hR)
         &amp;amp; np.isfinite(log_vf) &amp;amp; np.isfinite(log_x) &amp;amp; np.isfinite(log_gcdo)
         &amp;amp; np.isfinite(log_gcdi) &amp;amp; np.isfinite(log_gcf) &amp;amp; np.isfinite(log_gct)
         &amp;amp; np.isfinite(log_gcfo) &amp;amp; np.isfinite(rmax_hR))

    print(f"Valid: {m.sum()}")

    # ============================================================
    # Test 1: Δ = log(gc_obs/gc_X) の hR 依存性
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 1: 手法間差 Δ の hR 依存性")
    print("=" * 50)

    gc_methods = {
        "gc_deep(all)":   log_gcd,
        "gc_deep(out50)": log_gcdo,
        "gc_deep(out30)": np.log10(gc_deep_o3),
        "gc_deep(in50)":  log_gcdi,
        "gc_full(all)":   log_gcf,
        "gc_full(out50)": log_gcfo,
        "gc_tanh":        log_gct,
    }

    print(f"  {'手法':&amp;lt;20s} {'Δ median':&amp;gt;10s} {'ρ(Δ,hR)':&amp;gt;10s} {'p':&amp;gt;12s} "
          f"{'ρ(Δ,hR|vf)':&amp;gt;12s} {'p':&amp;gt;12s}")
    print("  " + "-" * 78)

    for name, log_gc_x in gc_methods.items():
        mx = m &amp;amp; np.isfinite(log_gc_x)
        if mx.sum() &amp;lt; 20:
            continue
        delta = log_gco[mx] - log_gc_x[mx]
        rho_h, p_h = pearsonr(log_hR[mx], delta)
        rho_hv, p_hv = partial_corr(log_hR[mx], delta, [log_vf[mx]])
        print(f"  {name:&amp;lt;20s} {np.median(delta):+.4f}   {rho_h:+.4f}  p={p_h:.3e}  "
              f"{rho_hv:+.4f}  p={p_hv:.3e}")

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

```
    # Test 2: 各手法の gc での hR 偏相関
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 2: gc(各手法) の hR 偏相関 |vflat, x_outer")
    print("=" * 50)

    all_gc_methods = {
        "gc_obs (TA3)":     log_gco,
        "gc_deep(all)":     log_gcd,
        "gc_deep(out50)":   log_gcdo,
        "gc_deep(in50)":    log_gcdi,
        "gc_full(all)":     log_gcf,
        "gc_full(out50)":   log_gcfo,
        "gc_tanh":          log_gct,
    }

    print(f"  {'手法':&lt;20s} {'ρ|vf':&gt;8s} {'ρ|vf,x':&gt;8s} {'α(proxy)':&gt;10s} {'R^2':&gt;8s}")
    print("  " + "-" * 58)

    for name, log_gc_x in all_gc_methods.items():
        mx = m &amp;amp; np.isfinite(log_gc_x)
        if mx.sum() &lt; 20:
            continue
        rho_vf, _ = partial_corr(log_hR[mx], log_gc_x[mx], [log_vf[mx]])
        rho_vfx, _ = partial_corr(log_hR[mx], log_gc_x[mx], [log_vf[mx], log_x[mx]])
        sl = linregress(log_pr[mx], log_gc_x[mx])
        print(f"  {name:&lt;20s} {rho_vf:+.4f}   {rho_vfx:+.4f}   {sl.slope:+.4f}   {sl.rvalue**2:.4f}")

    # ============================================================
    # Test 3: rmax/hR サブセット比較
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 3: rmax/hR サブセットでの比較")
    print("=" * 50)

    for rcut in [5, 8, 10, 15]:
        mc = m &amp;amp; (rmax_hR &gt; rcut)
        if mc.sum() &lt; 15:
            continue
        print(f"¥n  rmax/hR &gt; {rcut} (N={mc.sum()}):")

        for name, log_gc_x in [("gc_obs", log_gco), ("gc_deep(all)", log_gcd),
                               ("gc_full(all)", log_gcf), ("gc_tanh", log_gct)]:
            mx = mc &amp;amp; np.isfinite(log_gc_x)
            if mx.sum() &lt; 10:
                continue
            rho_vf, p_vf = partial_corr(log_hR[mx], log_gc_x[mx], [log_vf[mx]])
            rho_vfx, p_vfx = partial_corr(log_hR[mx], log_gc_x[mx],
                                          [log_vf[mx], log_x[mx]])
            print(f"    {name:&lt;18s}: |vf: {rho_vf:+.4f} (p={p_vf:.3e})  "
                  f"|vf,x: {rho_vfx:+.4f} (p={p_vfx:.3e})")

    # ============================================================
    # Test 4: gc_deep ベースで幾何平均法則を再フィット
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 4: 各手法の幾何平均法則フィット")
    print("=" * 50)

    for name, log_gc_x in all_gc_methods.items():
        mx = m &amp;amp; np.isfinite(log_gc_x)
        if mx.sum() &lt; 20:
            continue
        # 単変量: log(gc) = α x log(proxy) + const
        sl = linregress(log_pr[mx], log_gc_x[mx])
        # 多変量: log(gc) = a x log(vflat) + b x log(hR) + const
        X = np.column_stack([log_vf[mx], log_hR[mx], np.ones(mx.sum())])
        coef, _, _, _ = lstsq(X, log_gc_x[mx], rcond=None)
        y_pred = X @ coef
        ss_r = np.sum((log_gc_x[mx] - y_pred)**2)
        ss_t = np.sum((log_gc_x[mx] - np.mean(log_gc_x[mx]))**2)
        R2 = 1 - ss_r / ss_t

        print(f"  {name:&lt;20s}: α={sl.slope:.4f}+/-{sl.stderr:.4f}  "
              f"vf^{coef[0]:.3f} x hR^{coef[1]:.3f}  R^2={R2:.4f}")

    # ============================================================
    # Test 5: 内側 vs 外側フィットの hR 依存性
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 5: 内側 vs 外側フィットの差")
    print("=" * 50)

    delta_io = log_gcdi[m] - log_gcdo[m]
    rho_io_h, p_io_h = pearsonr(log_hR[m], delta_io)
    rho_io_hv, p_io_hv = partial_corr(log_hR[m], delta_io, [log_vf[m]])
    rho_io_hvx, p_io_hvx = partial_corr(log_hR[m], delta_io,
```

```
                                        [log_vf[m], log_x[m]])

    print(f"  Δ(inner-outer) の hR 相関:")
    print(f"    raw:      ρ = {rho_io_h:+.4f}, p = {p_io_h:.3e}")
    print(f"    |vflat:   ρ = {rho_io_hv:+.4f}, p = {p_io_hv:.3e}")
    print(f"    |vflat,x: ρ = {rho_io_hvx:+.4f}, p = {p_io_hvx:.3e}")

    print(f"¥n  Δ(inner-outer) 統計:")
    print(f"    median = {np.median(delta_io):.4f}")
    print(f"    std    = {np.std(delta_io):.4f}")
    print(f"    |Δ|&gt;0.1: {np.mean(np.abs(delta_io) &gt; 0.1):.1%}")
    print(f"    |Δ|&gt;0.3: {np.mean(np.abs(delta_io) &gt; 0.3):.1%}")

    # gc_obs は inner, outer どちらに近い?
    d_obs_in  = np.abs(log_gco[m] - log_gcdi[m])
    d_obs_out = np.abs(log_gco[m] - log_gcdo[m])
    d_obs_all = np.abs(log_gco[m] - log_gcd[m])
    print(f"¥n  gc_obs との距離:")
    print(f"    |gc_obs - gc_deep(all)|  median = {np.median(d_obs_all):.4f}")
    print(f"    |gc_obs - gc_deep(out)|  median = {np.median(d_obs_out):.4f}")
    print(f"    |gc_obs - gc_deep(in)|   median = {np.median(d_obs_in):.4f}")

    # ============================================================
    # Test 6: gc_obs をどの手法で置き換えると hR 偏相関が消えるか
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 6: hR偏相関が最小になる gc の定義")
    print("=" * 50)

    # 線形結合: log(gc_best) = w x log(gc_deep_out) + (1-w) x log(gc_deep_in)
    # -&gt; wを最適化して hR偏相関を最小化

    def hR_partial_corr_for_weight(w):
        log_gc_w = w * log_gcdo[m] + (1-w) * log_gcdi[m]
        try:
            rho, _ = partial_corr(log_hR[m], log_gc_w, [log_vf[m], log_x[m]])
            return abs(rho)
        except:
            return 1.0

    from scipy.optimize import minimize_scalar as ms
    res_w = ms(hR_partial_corr_for_weight, bounds=(-1, 2), method='bounded')
    w_opt = res_w.x
    rho_opt = res_w.fun

    log_gc_opt = w_opt * log_gcdo[m] + (1-w_opt) * log_gcdi[m]
    rho_actual, p_actual = partial_corr(log_hR[m], log_gc_opt, [log_vf[m], log_x[m]])

    print(f"  最適重み: w_outer = {w_opt:.4f}")
    print(f"  最小 |ρ|: {rho_opt:.4f} (sign: {rho_actual:+.4f})")
    print(f"  (gc_obs: ρ = -0.145, gc_deep: ρ = ?? )")

    # gc_obs の暗黙的重みを推定
    # gc_obs ~= w_obs x gc_deep_out + (1-w_obs) x gc_deep_in ?
    # 最小二乗で推定
    X_w = np.column_stack([log_gcdo[m], log_gcdi[m], np.ones(m.sum())])
    coef_w, _, _, _ = lstsq(X_w, log_gco[m], rcond=None)
    print(f"¥n  gc_obs の分解:")
    print(f"    gc_obs ~= gc_deep_out^{coef_w[0]:.3f} x gc_deep_in^{coef_w[1]:.3f}")
    print(f"    暗黙的外側重み: {coef_w[0]:.3f}")

    # ============================================================
    # Test 7: TA3 tanh fit の rt/hR 比とhR偏相関
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 7: tanh fit パラメータの構造")
    print("=" * 50)

    rt = arr("rt_tanh")
    log_rt = np.log10(rt)
    rt_hR = rt / hR_a

    mr = m &amp; np.isfinite(log_rt) &amp; np.isfinite(rt_hR)

    if mr.sum() &gt; 20:
        print(f"  rt/hR: median = {np.median(rt_hR[mr]):.3f}, "
              f"std = {np.std(rt_hR[mr]):.3f}")

        rho_rth, p_rth = pearsonr(log_hR[mr], np.log10(rt_hR[mr]))
        print(f"  ρ(log(rt/hR), log(hR)) = {rho_rth:+.4f}, p = {p_rth:.3e}")

        # rt/hR を統制して hR 偏相関を見る
        rho_t, p_t = partial_corr(log_hR[mr], log_gco[mr],
                                  [log_vf[mr], log_x[mr], np.log10(rt_hR[mr])])
        print(f"  ρ(gc_obs, hR | vflat, x, rt/hR) = {rho_t:+.4f}, p = {p_t:.3e}")

        rho_base_r, _ = partial_corr(log_hR[mr], log_gco[mr],
```

```
                                    [log_vf[mr], log_x[mr]])
        print(f"  (ベースライン |vflat, x: ρ = {rho_base_r:+.4f})")

    # ============================================================
    # Test 8: gc_deep(out50) ベースでの偏相関構造
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 8: gc_deep(out50) ベースの偏相関構造")
    print("=" * 50)

    for label, lgc in [("gc_obs", log_gco), ("gc_deep(all)", log_gcd),
                       ("gc_deep(out50)", log_gcdo), ("gc_deep(in50)", log_gcdi),
                       ("gc_full(all)", log_gcf), ("gc_full(out50)", log_gcfo)]:
        mx = m &amp;amp; np.isfinite(lgc)
        if mx.sum() &lt; 20:
            continue
        rho_raw, _ = pearsonr(log_hR[mx], lgc[mx])
        rho_vf, _ = partial_corr(log_hR[mx], lgc[mx], [log_vf[mx]])
        rho_vfx, _ = partial_corr(log_hR[mx], lgc[mx], [log_vf[mx], log_x[mx]])
        print(f"  {label:&lt;20s}: raw={rho_raw:+.4f}  |vf={rho_vf:+.4f}  |vf,x={rho_vfx:+.4f}")

    # ============================================================
    # SUMMARY
    # ============================================================
    print("¥n" + "=" * 70)
    print("SUMMARY")
    print("=" * 70)

    print(f"¥n  gc決定手法によるhR偏相関の変化 (|vflat, x_outer):")
    for label, lgc in all_gc_methods.items():
        mx = m &amp;amp; np.isfinite(lgc)
        if mx.sum() &gt; 20:
            rho, p = partial_corr(log_hR[mx], lgc[mx], [log_vf[mx], log_x[mx]])
            print(f"    {label:&lt;20s}: ρ = {rho:+.4f} (p={p:.3e})")

    print(f"¥n  hR偏相関を最小にする gc:")
    print(f"    最適: gc_out^{w_opt:.3f} x gc_in^{1-w_opt:.3f} -&gt; |ρ| = {rho_opt:.4f}")
    print(f"    gc_obs の暗黙的外側重み: {coef_w[0]:.3f}")

    print(f"¥n  核心的問い: 31%は手法の系統効果か物理的効果か?")
    print(f"  -&gt; gc_deep ベースでの hR 偏相関を確認:")

    rho_deep_base, p_deep = partial_corr(log_hR[m], log_gcd[m],
                                         [log_vf[m], log_x[m]])
    print(f"     gc_deep(all) |vflat,x: ρ = {rho_deep_base:+.4f} (p={p_deep:.3e})")

    rho_obs_base, p_obs = partial_corr(log_hR[m], log_gco[m],
                                       [log_vf[m], log_x[m]])
    print(f"     gc_obs (TA3) |vflat,x: ρ = {rho_obs_base:+.4f} (p={p_obs:.3e})")

    if abs(rho_deep_base) &lt; abs(rho_obs_base) * 0.5:
        print(f"¥n  [*] gc_deep では hR 偏相関が大幅に減少")
        print(f"  -&gt; 31%の大部分は TA3 gc 決定の系統効果")
    elif abs(rho_deep_base) &gt; abs(rho_obs_base) * 0.8:
        print(f"¥n  [*] gc_deep でも hR 偏相関がほぼ同じ")
        print(f"  -&gt; 31%は手法非依存の物理的効果")
    else:
        print(f"¥n  [*] 部分的に手法依存、部分的に物理的")
        pct = (abs(rho_obs_base) - abs(rho_deep_base)) / abs(rho_obs_base) * 100
        print(f"  -&gt; 手法効果: {pct:.0f}%, 物理的: {100-pct:.0f}%")


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

7. 条件14(塑性領域)による残り31%の検証

ファイル: sparc_cond14_residual.py

解析目的

条件14の塑性領域が残り31%を「微小補正」として説明するか検証。8つの塑性パラメータ（s_c, r_plastic/hR, f_p(gc), E_strain, excess_strain, central_conc, strain_gradient, r_peak/hR）を計算し、段階的偏相関と塑性補正モデルで評価する。

結果

否定: s_c vs x_outer rho=+0.527（共線）。統制でrho悪化（-0.145 -> -0.273）。

ソースコード

(563 行)

```
#!/usr/bin/env python3
"""
sparc_cond14_residual.py
========================
条件14（塑性領域）が残り31%のhR偏相関を説明するか

前セッションで否定されたもの:
  - f_p (g_N &lt; a_0 質量分率) -&gt; gc: f_p ~= 1 で判別力なし
  - gc ~ Σ_bar^(1/3): R^2 = 0.016

今回の仮説（弱い版）:
  塑性領域は「gc を決める主因」ではなく、
  深MOND gc に「微小な補正」を加える (R^2 ~ 0.04, 5%効果)。

  メカニズム:
  (1) ディスク中心で g_N が高い -&gt; 歪み大 -&gt; 塑性領域が集中
  (2) 塑性領域の rho_eff = u_strain/c^2 が有効質量として作用
  (3) -&gt; gc_in &gt; gc_out の系統差を生む
  (4) hR が小さい銀河ほどバリオン集中 -&gt; 中心 g_N/a_0 が高い -&gt; 塑性補正大

  定量的予測:
  - 「中心歪みパラメータ」s_c = g_N_peak / gc_obs ~ 塑性度
  - Δgc = gc_obs - gc_deep ~ s_c（塑性補正）
  - s_c を統制すると hR 偏相関の残り31%が減少

テスト:
  Test 1: 中心歪みパラメータ s_c の定義と分布
  Test 2: s_c vs gc_in - gc_out の相関（塑性メカニズムの直接検証）
  Test 3: s_c vs hR の相関（hR依存性の経路確認）
  Test 4: s_c を統制した hR 偏相関（31%が減るか）
  Test 5: 塑性補正モデル gc_corrected = gc_deep x (1 + β x s_c^γ) のフィット
  Test 6: rmax/hR &gt; 8 サブセットでの検証
  Test 7: 代替パラメータ（歪みエネルギー密度プロキシ）

実行: uv run --with scipy --with matplotlib python sparc_cond14_residual.py
"""

import numpy as np
from scipy.optimize import minimize_scalar, minimize
from scipy.stats import pearsonr, spearmanr, linregress
from numpy.linalg import lstsq
from pathlib import Path
import csv
import sys

# ============================================================
a0 = 1.2e-10
kpc_to_m = 3.0857e19
kms2_to_ms2 = 1.0e6
Yd_default = 0.5
Yb_default = 0.7

BASE = Path(r"D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン")
ROTMOD_DIR = BASE / "Rotmod_LTG"
GC_CSV = BASE / "sparc_gc.csv"


def load_gc_table(path):
    data = {}
    with open(path, "r") as f:
        reader = csv.DictReader(f)
        for row in reader:
```

```
            name = row["Galaxy"].strip()
            data[name] = {
                "gc_obs": float(row["gc"]),
                "vflat": float(row["Vflat"]),
                "hR": float(row["hR"]),
            }
    return data


def load_rotmod(galaxy_name, rotmod_dir):
    fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
    if not fpath.exists():
        return None
    r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
    with open(fpath, "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
            r.append(float(parts[0]))
            vobs.append(float(parts[1]))
            errv.append(float(parts[2]))
            vgas.append(float(parts[3]))
            vdisk.append(float(parts[4]))
            vbul.append(float(parts[5]))
    return (np.array(r), np.array(vobs), np.array(errv),
            np.array(vgas), np.array(vdisk), np.array(vbul))


def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
    r_m = r_kpc * kpc_to_m
    conv = kms2_to_ms2 / r_m
    gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
          + Yb*np.abs(Vbul)*Vbul*conv)
    return np.abs(gN)


def partial_corr(x, y, controls):
    if len(controls) == 0:
        return pearsonr(x, y)
    C = np.column_stack(controls + [np.ones(len(x))])
    cx, _, _, _ = lstsq(C, x, rcond=None)
    cy, _, _, _ = lstsq(C, y, rcond=None)
    return pearsonr(x - C @ cx, y - C @ cy)


def fit_gc_deep(r_kpc, gN, Vobs, errV, r_range="all"):
    N = len(r_kpc)
    if r_range == "outer50":
        sl = slice(N - max(5, N//2), N)
    elif r_range == "inner50":
        sl = slice(0, max(5, N//2))
    else:
        sl = slice(0, N)
    r_f, V_f, e_f, gN_f = r_kpc[sl], Vobs[sl], errV[sl], gN[sl]
    e_f = np.where(e_f &gt; 0, e_f, np.median(Vobs) * 0.1)

    def chi2(lgc):
        gc = 10**lgc
        V_pred = np.sqrt(r_f * kpc_to_m * np.sqrt(gN_f * gc)) * 1e-3
        return np.sum(((V_f - V_pred) / e_f)**2)

    res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
    return 10**res.x


# ============================================================
# 塑性パラメータの定義
# ============================================================
def compute_plasticity_params(r_kpc, gN, gc_obs, hR):
    """条件14に基づく塑性パラメータ群"""
    params = {}

    # --- (1) 中心歪みパラメータ s_c = g_N_peak / gc ---
    gN_peak = np.max(gN)
    params["s_c"] = gN_peak / gc_obs if gc_obs &gt; 0 else np.nan

    # --- (2) ピーク位置での歪み ---
    idx_peak = np.argmax(gN)
    params["r_peak_hR"] = r_kpc[idx_peak] / hR if hR &gt; 0 else np.nan

    # --- (3) 歪みエネルギー密度プロキシ ---
    # u_strain ~ ε^2 ~ (g_N/gc)^2 の積分
    # E_strain = int (g_N(r)/gc)^2 x r dr
```

```
    x_arr = gN / gc_obs
    if len(r_kpc) &gt; 2:
        E_strain = np.trapz(x_arr**2 * r_kpc, r_kpc)
        params["E_strain"] = E_strain
    else:
        params["E_strain"] = np.nan

    # --- (4) 「塑性半径」: g_N &gt; gc の領域の広さ ---
    # r_plastic = g_N &gt; gc を満たす最大 r (hR単位)
    above = r_kpc[gN &gt; gc_obs]
    params["r_plastic_hR"] = above[-1] / hR if len(above) &gt; 0 and hR &gt; 0 else 0.0

    # --- (5) 塑性質量分率 (修正版) ---
    # f_p = int_{g_N &gt; gc} g_N r dr / int g_N r dr
    # g_N &gt; gc の領域の質量寄与の割合
    total_int = np.trapz(gN * r_kpc, r_kpc)
    if total_int &gt; 0:
        mask_plastic = gN &gt; gc_obs
        if mask_plastic.any():
            plastic_int = np.trapz(gN[mask_plastic] * r_kpc[mask_plastic],
                                   r_kpc[mask_plastic])
            params["f_p_gc"] = plastic_int / total_int
        else:
            params["f_p_gc"] = 0.0
    else:
        params["f_p_gc"] = np.nan

    # --- (6) 中心集中度: g_N(r &lt; hR) / g_N(全体) ---
    inner = r_kpc &lt; hR
    if inner.sum() &gt; 0 and len(r_kpc) &gt; 2:
        gN_inner = np.trapz(gN[inner] * r_kpc[inner], r_kpc[inner])
        gN_total = np.trapz(gN * r_kpc, r_kpc)
        params["central_conc"] = gN_inner / gN_total if gN_total &gt; 0 else np.nan
    else:
        params["central_conc"] = np.nan

    # --- (7) 歪み勾配: d(g_N/gc)/d(r/hR) at r = hR ---
    x_arr_norm = gN / gc_obs
    r_norm = r_kpc / hR
    # r/hR = 1 付近での勾配
    mask_near = np.abs(r_norm - 1.0) &lt; 0.5
    if mask_near.sum() &gt;= 3:
        sl = linregress(r_norm[mask_near], x_arr_norm[mask_near])
        params["strain_gradient"] = sl.slope
    else:
        params["strain_gradient"] = np.nan

    # --- (8) 「超過歪み」: max(0, g_N - gc) の積分 / (gc x hR^2) ---
    excess = np.maximum(0, gN - gc_obs)
    if len(r_kpc) &gt; 2 and gc_obs &gt; 0 and hR &gt; 0:
        E_excess = np.trapz(excess * r_kpc, r_kpc)
        params["excess_strain"] = E_excess / (gc_obs * (hR)**2)
    else:
        params["excess_strain"] = np.nan

    return params


# ============================================================
def main():
    print("=" * 70)
    print("条件14 (塑性領域) による残り31%の説明可能性")
    print("=" * 70)

    gc_table = load_gc_table(GC_CSV)

    records = []
    for gal_name, info in gc_table.items():
        rotmod = load_rotmod(gal_name, ROTMOD_DIR)
        if rotmod is None:
            continue
        r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
        if len(r_kpc) &lt; 8:
            continue

        gc_obs = info["gc_obs"]
        gc_obs_ms2 = gc_obs * a0 if gc_obs &gt; 1e-5 else gc_obs
        hR = info["hR"]
        vflat = info["vflat"]

        gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)

        # gc フィット
        gc_deep = fit_gc_deep(r_kpc, gN, Vobs, errV, "all")
        gc_deep_in = fit_gc_deep(r_kpc, gN, Vobs, errV, "inner50")
        gc_deep_out = fit_gc_deep(r_kpc, gN, Vobs, errV, "outer50")

```

```
        # 塑性パラメータ
        plast = compute_plasticity_params(r_kpc, gN, gc_obs_ms2, hR)

        # x_outer
        n_out = max(3, len(r_kpc) // 3)
        x_outer = np.median(gN[-n_out:] / gc_deep) if gc_deep &gt; 0 else np.nan

        # rmax/hR
        rmax_hR = r_kpc[-1] / hR if hR &gt; 0 else np.nan

        # 残差
        delta_gc = np.log10(gc_obs_ms2) - np.log10(gc_deep) if gc_deep &gt; 0 else np.nan
        delta_io = np.log10(gc_deep_in) - np.log10(gc_deep_out) if gc_deep_out &gt; 0 else np.nan

        records.append({
            "name": gal_name,
            "gc_obs": gc_obs_ms2,
            "gc_deep": gc_deep,
            "gc_deep_in": gc_deep_in,
            "gc_deep_out": gc_deep_out,
            "delta_gc": delta_gc,
            "delta_io": delta_io,
            "hR": hR,
            "vflat": vflat,
            "x_outer": x_outer,
            "rmax_hR": rmax_hR,
            **plast,
        })

    N = len(records)
    print(f"Processed: {N}")

    def arr(k): return np.array([r[k] for r in records], dtype=float)

    gc_obs = arr("gc_obs"); gc_deep = arr("gc_deep")
    hR_a = arr("hR"); vflat_a = arr("vflat")
    x_outer = arr("x_outer"); rmax_hR = arr("rmax_hR")
    delta_gc = arr("delta_gc"); delta_io = arr("delta_io")

    s_c = arr("s_c")
    r_peak_hR = arr("r_peak_hR")
    E_strain = arr("E_strain")
    r_plastic_hR = arr("r_plastic_hR")
    f_p_gc = arr("f_p_gc")
    central_conc = arr("central_conc")
    strain_grad = arr("strain_gradient")
    excess_strain = arr("excess_strain")

    log_gc = np.log10(gc_obs)
    log_gcd = np.log10(gc_deep)
    log_hR = np.log10(hR_a)
    log_vf = np.log10(vflat_a)
    log_x = np.log10(x_outer + 1e-10)
    log_sc = np.log10(s_c + 1e-10)

    m = (np.isfinite(log_gc) &amp;amp; np.isfinite(log_hR) &amp;amp; np.isfinite(log_vf)
         &amp;amp; np.isfinite(log_x) &amp;amp; np.isfinite(log_gcd) &amp;amp; np.isfinite(log_sc)
         &amp;amp; np.isfinite(delta_gc))

    print(f"Valid: {m.sum()}")

    # ============================================================
    # Test 1: 塑性パラメータの分布
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 1: 塑性パラメータの分布")
    print("=" * 50)

    plast_params = {
        "s_c (gN_peak/gc)": s_c,
        "r_peak/hR": r_peak_hR,
        "E_strain": E_strain,
        "r_plastic/hR": r_plastic_hR,
        "f_p(gc)": f_p_gc,
        "central_conc": central_conc,
        "strain_gradient": strain_grad,
        "excess_strain": excess_strain,
    }

    for name, arr_p in plast_params.items():
        v = np.isfinite(arr_p) &amp;amp; m
        if v.sum() &gt; 10:
            print(f"  {name:20s}: N={v.sum():3d}  "
                  f"median={np.median(arr_p[v]):.4f}  "
                  f"std={np.std(arr_p[v]):.4f}  "
                  f"range=[{np.min(arr_p[v]):.3f}, {np.max(arr_p[v]):.3f}]")

```

```
    # ============================================================
    # Test 2: s_c vs gc_in - gc_out (塑性メカニズムの直接検証)
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 2: 中心歪み s_c vs gc内外差")
    print("=" * 50)

    m_io = m &amp;amp; np.isfinite(delta_io)
    if m_io.sum() &gt; 10:
        rho_sio, p_sio = pearsonr(log_sc[m_io], delta_io[m_io])
        print(f"  rho(s_c, gc_in - gc_out) = {rho_sio:+.4f}, p = {p_sio:.3e}")
        print(f"  予測: 正の相関 (s_c大 -&gt; gc_inが相対的に高い) ")

        # s_c の四分位で delta_io を比較
        q25, q75 = np.percentile(s_c[m_io], [25, 75])
        lo = s_c[m_io] &lt; q25
        hi = s_c[m_io] &gt; q75
        print(f"  s_c &lt; Q25 ({q25:.2f}): delta_io median = {np.median(delta_io[m_io][lo]):.4f}")
        print(f"  s_c &gt; Q75 ({q75:.2f}): delta_io median = {np.median(delta_io[m_io][hi]):.4f}")

    # 全塑性パラメータ vs delta_io
    print(f"¥n  全塑性パラメータ vs delta_io:")
    for name, arr_p in plast_params.items():
        v = m_io &amp;amp; np.isfinite(arr_p)
        if v.sum() &gt; 15:
            # 対数化が適切なものは対数で
            if np.min(arr_p[v]) &gt; 0:
                rho, p = pearsonr(np.log10(arr_p[v] + 1e-10), delta_io[v])
                print(f"    {name:20s}: rho = {rho:+.4f}, p = {p:.3e}")
            else:
                rho, p = pearsonr(arr_p[v], delta_io[v])
                print(f"    {name:20s}: rho = {rho:+.4f}, p = {p:.3e}")

    # ============================================================
    # Test 3: 塑性パラメータ vs hR の相関
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 3: 塑性パラメータ vs hR, gc")
    print("=" * 50)

    print(f"  {'パラメータ':&lt;20s} {'rho(hR)':&gt;10s} {'p':&gt;12s} "
          f"{'rho(gc)':&gt;10s} {'p':&gt;12s}")
    print("  " + "-" * 66)
    for name, arr_p in plast_params.items():
        v = m &amp;amp; np.isfinite(arr_p)
        if v.sum() &gt; 15:
            if np.min(arr_p[v]) &gt; 0:
                ap = np.log10(arr_p[v] + 1e-10)
            else:
                ap = arr_p[v]
            rh, ph = pearsonr(log_hR[v], ap)
            rg, pg = pearsonr(log_gc[v], ap)
            print(f"  {name:&lt;20s} {rh:+.4f}   p={ph:.3e}   {rg:+.4f}   p={pg:.3e}")

    # ============================================================
    # Test 4: 塑性パラメータを統制したhR偏相関 [*]核心[*]
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 4: 塑性パラメータ統制後のhR偏相関")
    print("=" * 50)

    base_controls = [log_vf[m], log_x[m]]
    rho_base, p_base = partial_corr(log_hR[m], log_gc[m], base_controls)
    print(f"  ベースライン |vflat, x: rho = {rho_base:+.4f}")

    print(f"¥n  {'追加パラメータ':&lt;20s} {'rho':&gt;8s} {'Δ|rho|':&gt;10s}")
    print("  " + "-" * 40)

    best_delta = 0
    best_name = ""
    for name, arr_p in plast_params.items():
        v = m &amp;amp; np.isfinite(arr_p)
        if v.sum() &lt; m.sum() * 0.7:
            continue
        arr_filled = arr_p.copy()
        arr_filled[~np.isfinite(arr_filled)] = np.nanmedian(arr_p)
        if np.min(arr_filled[m]) &gt; 0:
            ctrl = np.log10(arr_filled[m] + 1e-10)
        else:
            ctrl = arr_filled[m]
        try:
            rho_new, p_new = partial_corr(log_hR[m], log_gc[m],
                                          base_controls + [ctrl])
            delta = abs(rho_new) - abs(rho_base)
            print(f"  {name:&lt;20s} {rho_new:+.4f}   {delta:+.4f}")
            if delta &lt; best_delta:
                best_delta = delta
```

```
                best_name = name
        except:
            pass

    if best_delta &lt; -0.01:
        print(f"¥n  最も効果的: {best_name} (Δ = {best_delta:+.4f})")
    else:
        print(f"¥n  どの塑性パラメータも hR 偏相関を有意に減らさない")

    # ============================================================
    # Test 5: 塑性補正モデル
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 5: 塑性補正モデル gc_corr = gc_deep x (1 + beta x s_c^gamma)")
    print("=" * 50)

    # gc_obs / gc_deep = 1 + β x s_c^γ をフィット
    ratio = gc_obs[m] / gc_deep[m]
    log_ratio = np.log10(ratio)

    # まず線形: log(gc_obs/gc_deep) = a x log(s_c) + b
    sl_sc = linregress(log_sc[m], log_ratio)
    print(f"  log(gc_obs/gc_deep) vs log(s_c):")
    print(f"    slope = {sl_sc.slope:.4f} +/- {sl_sc.stderr:.4f}")
    print(f"    R&lt;super&gt;2&lt;/super&gt; = {sl_sc.rvalue**2:.4f}")
    print(f"    p = {sl_sc.pvalue:.3e}")

    # s_c 補正後の gc
    gc_corrected = gc_deep[m] * (s_c[m] ** sl_sc.slope) * 10**sl_sc.intercept

    log_gcc = np.log10(gc_corrected)
    rho_cc, p_cc = partial_corr(log_hR[m], log_gcc,
                                [log_vf[m], log_x[m]])
    print(f"¥n  補正後 gc の hR 偏相関 |vflat, x:")
    print(f"    rho = {rho_cc:+.4f} (ベースライン: {rho_base:+.4f})")
    print(f"    Δ|rho| = {abs(rho_cc) - abs(rho_base):+.4f}")

    # 他の塑性パラメータでも同様の補正
    print(f"¥n  各塑性パラメータによる補正効果:")
    for name, arr_p in plast_params.items():
        v = m &amp; np.isfinite(arr_p) &amp; (arr_p &gt; 0)
        if v.sum() &lt; m.sum() * 0.5:
            continue
        log_p = np.log10(arr_p[v] + 1e-10)
        sl = linregress(log_p, log_ratio[np.where(m)[0][v[m]]] if v.sum() == m.sum()
                        else np.nan)
        # 簡易版: m内でのフィット
        try:
            arr_m = arr_p[m]
            valid_m = arr_m &gt; 0
            if valid_m.sum() &gt; 20:
                log_pm = np.log10(arr_m[valid_m] + 1e-10)
                sl2 = linregress(log_pm, log_ratio[valid_m])
                gc_c2 = gc_deep[m].copy()
                gc_c2[valid_m] *= (arr_m[valid_m] ** sl2.slope) * 10**sl2.intercept
                log_gc2 = np.log10(gc_c2)
                if np.isfinite(log_gc2).sum() &gt; 20:
                    m2 = np.isfinite(log_gc2)
                    rho2, _ = partial_corr(log_hR[m][m2], log_gc2[m2],
                                           [log_vf[m][m2], log_x[m][m2]])
                    delta2 = abs(rho2) - abs(rho_base)
                    print(f"    {name:20s}: R2={sl2.rvalue**2:.4f}, "
                          f"rho_corr={rho2:+.4f}, delta={delta2:+.4f}")
        except:
            pass

    # ============================================================
    # Test 6: rmax/hR &gt; 8 サブセット
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 6: rmax/hR &gt; 8 サブセットでの塑性効果")
    print("=" * 50)

    m8 = m &amp; (rmax_hR &gt; 8)
    if m8.sum() &gt; 15:
        rho_b8, _ = partial_corr(log_hR[m8], log_gc[m8],
                                 [log_vf[m8], log_x[m8]])
        print(f"  ベースライン (rmax/hR&gt;8): rho = {rho_b8:+.4f} (N={m8.sum()})")

        for name, arr_p in [("s_c", s_c), ("excess_strain", excess_strain),
                            ("f_p(gc)", f_p_gc), ("central_conc", central_conc)]:
            v8 = m8 &amp; np.isfinite(arr_p)
            if v8.sum() &lt; 10:
                continue
            arr_f = arr_p.copy()
            arr_f[~np.isfinite(arr_f)] = np.nanmedian(arr_p)
            if np.min(arr_f[m8]) &gt; 0:
```

```
                ctrl = np.log10(arr_f[m8] + 1e-10)
            else:
                ctrl = arr_f[m8]
            try:
                rho8, p8 = partial_corr(log_hR[m8], log_gc[m8],
                                        [log_vf[m8], log_x[m8], ctrl])
                delta8 = abs(rho8) - abs(rho_b8)
                print(f"  +{name:20s}: rho={rho8:+.4f}, delta={delta8:+.4f}")
            except:
                pass

    # ============================================================
    # Test 7: x_outer と s_c の独立性チェック
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 7: x_outer と s_c の関係")
    print("=" * 50)

    rho_xs, p_xs = pearsonr(log_x[m], log_sc[m])
    print(f"  rho(x_outer, s_c) = {rho_xs:+.4f}, p = {p_xs:.3e}")

    # s_c|x_outer の残差が hR 偏相関を説明するか
    # つまり x_outer では捉えきれない塑性情報が s_c にあるか
    rho_sc_hR, p_sc_hR = partial_corr(log_hR[m], log_gc[m],
                                      [log_vf[m], log_x[m], log_sc[m]])
    print(f"  rho(gc, hR | vflat, x, s_c) = {rho_sc_hR:+.4f}, p = {p_sc_hR:.3e}")
    print(f"  (ベースライン |vflat, x: {rho_base:+.4f})")
    delta_sc = abs(rho_sc_hR) - abs(rho_base)
    print(f"  delta = {delta_sc:+.4f}")

    if delta_sc &lt; -0.02:
        pct = -delta_sc / 0.472 * 100
        print(f"  -&gt; s_c が hR 偏相関の追加 {pct:.1f}% を説明")
    else:
        print(f"  -&gt; s_c は x_outer 以上の説明力を持たない")

    # ============================================================
    # SUMMARY
    # ============================================================
    print("¥n" + "=" * 70)
    print("SUMMARY: 条件14（塑性領域）の残り31%への寄与")
    print("=" * 70)

    print(f"¥n  hR偏相関:")
    print(f"    |vflat:        rho = -0.472")
    print(f"    |vflat, x:     rho = {rho_base:+.4f} (x が 69% 説明)")
    print(f"    |vflat, x, sc: rho = {rho_sc_hR:+.4f} (s_c 追加効果: {delta_sc:+.4f})")

    print(f"¥n  s_c vs gc内外差: rho = {rho_sio:+.4f}" if m_io.sum() &gt; 10 else "")
    print(f"  s_c vs x_outer:  rho = {rho_xs:+.4f}")

    print(f"¥n  条件14の評価:")
    if delta_sc &lt; -0.03:
        print(f"  -&gt; 塑性パラメータが hR 偏相関の追加分を部分的に説明 [*]")
        print(f"  -&gt; 条件14は「微小補正」として機能する可能性あり")
    elif abs(delta_sc) &lt; 0.03:
        print(f"  -&gt; 塑性パラメータは x_outer と冗長（独立な説明力なし）")
        print(f"  -&gt; 条件14は残り31%を説明しない")
    else:
        print(f"  -&gt; 塑性パラメータの追加で hR 偏相関がむしろ増加")
        print(f"  -&gt; 条件14は残り31%と無関係")


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

8. 歪みエネルギー勾配項

ファイル: sparc_strain_gradient.py

解析目的

膜ラグランジアン L=U(epsilon;c)+kappa(d epsilon/dr)^2 の勾配項の効果を検証。局所歪み epsilon^2（x_outer）と勾配(d epsilon/dr)^2 を分離し、E_grad, E_grad_norm, E_grad/E_local
等の勾配パラメータがhR偏相関の残り31%を説明するか評価する。

結果

E_grad_norm で rho=+0.023（95%消失）→ 後にトートロジーと判明し撤回。

ソースコード

(574 行)

```
#!/usr/bin/env python3
"""
sparc_strain_gradient.py
========================
歪みエネルギー勾配項による残り31%の説明

新しい着眼:
  膜の歪みエネルギーには2成分:
    u_local ~ ε^2 -&gt; x_outer と同等（既にテスト済み、69%説明）
    u_grad  ~ (dε/dr)^2 -&gt; hR に独立に依存（未テスト）

  指数ディスクでは:
    dε/dr ~ (dg_N/dr)/gc ~ g_N/(gc x hR)
    u_grad ~ (g_N/(gc x hR))^2
    -&gt; 勾配エネルギー ~ hR^(-2) の依存性

  仮説: 勾配エネルギーが gc に微小補正を加える
    gc_eff = gc_deep x (1 + λ x E_grad / E_0)
    E_grad = int (dε/dr)^2 r dr
    E_0 は正規化定数

テスト:
  Test 1: ε(r) プロファイルの勾配計算
  Test 2: E_grad vs hR, x_outer の相関（独立性チェック）
  Test 3: E_grad を統制した hR 偏相関
  Test 4: 勾配補正モデルの検証
  Test 5: E_grad vs E_local の独立性
  Test 6: rmax/hR &gt; 8 サブセット
  Test 7: 偏相関の段階的分解（x_outer -&gt; E_grad）

実行: uv run --with scipy --with matplotlib python sparc_strain_gradient.py
"""

import numpy as np
from scipy.optimize import minimize_scalar
from scipy.stats import pearsonr, linregress
from numpy.linalg import lstsq
from pathlib import Path
import csv
import sys

# ============================================================
a0 = 1.2e-10
kpc_to_m = 3.0857e19
kms2_to_ms2 = 1.0e6
Yd_default = 0.5
Yb_default = 0.7

BASE = Path(r"D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン")
ROTMOD_DIR = BASE / "Rotmod_LTG"
GC_CSV = BASE / "sparc_gc.csv"


def load_gc_table(path):
    data = {}
    with open(path, "r") as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = row["Galaxy"].strip()
            data[name] = {
                "gc_obs": float(row["gc"]),
                "vflat": float(row["Vflat"]),
```

```
                "hR": float(row["hR"]),
            }
    return data

def load_rotmod(galaxy_name, rotmod_dir):
    fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
    if not fpath.exists():
        return None
    r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
    with open(fpath, "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
            r.append(float(parts[0]))
            vobs.append(float(parts[1]))
            errv.append(float(parts[2]))
            vgas.append(float(parts[3]))
            vdisk.append(float(parts[4]))
            vbul.append(float(parts[5]))
    return (np.array(r), np.array(vobs), np.array(errv),
            np.array(vgas), np.array(vdisk), np.array(vbul))


def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
    r_m = r_kpc * kpc_to_m
    conv = kms2_to_ms2 / r_m
    gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
          + Yb*np.abs(Vbul)*Vbul*conv)
    return np.abs(gN)


def partial_corr(x, y, controls):
    if len(controls) == 0:
        return pearsonr(x, y)
    C = np.column_stack(controls + [np.ones(len(x))])
    cx, _, _, _ = lstsq(C, x, rcond=None)
    cy, _, _, _ = lstsq(C, y, rcond=None)
    return pearsonr(x - C @ cx, y - C @ cy)


def fit_gc_deep(r_kpc, gN, Vobs, errV):
    e = np.where(errV &gt; 0, errV, np.median(Vobs) * 0.1)
    def chi2(lgc):
        gc = 10**lgc
        V_pred = np.sqrt(r_kpc * kpc_to_m * np.sqrt(gN * gc)) * 1e-3
        return np.sum(((Vobs - V_pred) / e)**2)
    res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
    return 10**res.x


# ============================================================
# ε(r) と勾配エネルギーの計算
# ============================================================
def compute_epsilon_profile(r_kpc, gN, gc):
    """平衡εプロファイル
    平衡条件: (c - 1 + ε^2)/(1-ε) = g_N/gc = x
    -&gt; ε = [-x + sqrt((x+2)^2 - 4c)] / 2  where c = gc/a_0

    深MOND近似 (x &lt;&lt; 1): ε ~= sqrt(c) - 1 + x/(2sqrtc) + ...
    -&gt; εはほぼ定数 + xに比例する小さな変動

    しかし c &lt; 1 の銀河では上の解は複素数になる可能性。
    代替: 深MOND で g_eff = sqrt(gN x gc) から
    ε を 「g_eff/gc - 1 の関数」として定義:
    y = g_eff/gc とすると μ(y) = g_N/g_eff -&gt; ε は μ の逆関数的量

    実用的定義: ε(r) = sqrt(g_N(r)/gc) (深MOND限定の正規化歪み)
    """
    x = gN / gc
    epsilon = np.sqrt(x)  # 深MOND近似での歪みパラメータ
    return epsilon


def compute_strain_energies(r_kpc, gN, gc, hR):
    """歪みエネルギーの局所項と勾配項を計算"""
    eps = compute_epsilon_profile(r_kpc, gN, gc)
    r_norm = r_kpc / hR  # hR 正規化

    N = len(r_kpc)
    result = {}

    # --- 局所エネルギー: E_local = int ε^2 x r dr ---
```

```
    if N &gt; 2:
        result["E_local"] = np.trapz(eps**2 * r_kpc, r_kpc)
    else:
        result["E_local"] = np.nan

    # --- 勾配エネルギー: E_grad = int (dε/dr)^2 x r dr ---
    if N &gt; 3:
        dr = np.diff(r_kpc)
        deps = np.diff(eps)
        mask = dr &gt; 0
        if mask.sum() &gt; 2:
            dedr = deps[mask] / dr[mask]  # dε/dr [1/kpc]
            r_mid = (r_kpc[:-1] + r_kpc[1:])[mask] / 2.0
            result["E_grad"] = np.trapz(dedr**2 * r_mid, r_mid)

            # hR正規化版: (dε/d(r/hR))^2 = (hR x dε/dr)^2
            dedr_norm = dedr * hR
            result["E_grad_norm"] = np.trapz(dedr_norm**2 * r_mid, r_mid)

            # 勾配のピーク値
            result["grad_peak"] = np.max(np.abs(dedr))
            result["grad_peak_norm"] = np.max(np.abs(dedr_norm))

            # 勾配ピーク位置 / hR
            idx_gp = np.argmax(np.abs(dedr))
            result["r_grad_peak_hR"] = r_mid[idx_gp] / hR

            # 勾配の外側 vs 内側の比
            n_half = len(dedr) // 2
            if n_half &gt;= 2:
                result["grad_outer_inner"] = (
                    np.median(np.abs(dedr[n_half:])) /
                    (np.median(np.abs(dedr[:n_half])) + 1e-30))
            else:
                result["grad_outer_inner"] = np.nan
        else:
            for k in ["E_grad", "E_grad_norm", "grad_peak", "grad_peak_norm",
                      "r_grad_peak_hR", "grad_outer_inner"]:
                result[k] = np.nan
    else:
        for k in ["E_grad", "E_grad_norm", "grad_peak", "grad_peak_norm",
                  "r_grad_peak_hR", "grad_outer_inner"]:
            result[k] = np.nan

    # --- 勾配/局所比: E_grad/E_local ---
    if np.isfinite(result.get("E_grad", np.nan)) and result.get("E_local", 0) &gt; 0:
        result["grad_local_ratio"] = result["E_grad"] / result["E_local"]
    else:
        result["grad_local_ratio"] = np.nan

    # --- 理論的予測: E_grad_theory ~ (gN_peak/(gcxhR))^2 x hR^3 ---
    gN_peak = np.max(gN)
    result["E_grad_theory"] = (gN_peak / (gc * hR * kpc_to_m))**2 * (hR * kpc_to_m)**3

    # --- 正規化勾配エネルギー密度: e_grad = E_grad / (hR^2 x gc^2) ---
    # これが「hR に依存しない普遍量」なら、gc への寄与が推定できる
    if np.isfinite(result.get("E_grad", np.nan)):
        result["e_grad_density"] = result["E_grad"] / ((hR)**2 * gc**2 + 1e-30)
    else:
        result["e_grad_density"] = np.nan

    return result


# ============================================================
def main():
    print("=" * 70)
    print("歪みエネルギー勾配項: (d epsilon/dr)^2 による hR 偏相関の説明")
    print("=" * 70)

    gc_table = load_gc_table(GC_CSV)

    records = []
    for gal_name, info in gc_table.items():
        rotmod = load_rotmod(gal_name, ROTMOD_DIR)
        if rotmod is None:
            continue
        r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
        if len(r_kpc) &lt; 8:
            continue

        gc_obs = info["gc_obs"]
        gc_obs_ms2 = gc_obs * a0 if gc_obs &gt; 1e-5 else gc_obs
        hR = info["hR"]
        vflat = info["vflat"]

        gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)
```

```
        gc_deep = fit_gc_deep(r_kpc, gN, Vobs, errV)

        # 歪みエネルギー (gc_obs ベースと gc_deep ベースの両方)
        strain_obs = compute_strain_energies(r_kpc, gN, gc_obs_ms2, hR)
        strain_deep = compute_strain_energies(r_kpc, gN, gc_deep, hR)

        # x_outer
        n_out = max(3, len(r_kpc) // 3)
        x_outer = np.median(gN[-n_out:] / gc_deep) if gc_deep &gt; 0 else np.nan

        rmax_hR = r_kpc[-1] / hR if hR &gt; 0 else np.nan

        records.append({
            "name": gal_name,
            "gc_obs": gc_obs_ms2,
            "gc_deep": gc_deep,
            "hR": hR, "vflat": vflat,
            "x_outer": x_outer, "rmax_hR": rmax_hR,
            # gc_deep ベースの勾配量 (循環を避ける)
            "E_grad_d": strain_deep.get("E_grad", np.nan),
            "E_grad_norm_d": strain_deep.get("E_grad_norm", np.nan),
            "E_local_d": strain_deep.get("E_local", np.nan),
            "grad_peak_d": strain_deep.get("grad_peak", np.nan),
            "grad_peak_norm_d": strain_deep.get("grad_peak_norm", np.nan),
            "grad_local_ratio_d": strain_deep.get("grad_local_ratio", np.nan),
            "r_grad_peak_hR_d": strain_deep.get("r_grad_peak_hR", np.nan),
            "grad_oi_d": strain_deep.get("grad_outer_inner", np.nan),
            "e_grad_density_d": strain_deep.get("e_grad_density", np.nan),
            # gc_obs ベース (循環の可能性あり、参考用)
            "E_grad_o": strain_obs.get("E_grad", np.nan),
            "grad_local_ratio_o": strain_obs.get("grad_local_ratio", np.nan),
        })

    N = len(records)
    print(f"Processed: {N}")

    def arr(k): return np.array([r[k] for r in records], dtype=float)

    gc_obs = arr("gc_obs"); gc_deep = arr("gc_deep")
    hR_a = arr("hR"); vflat_a = arr("vflat")
    x_outer = arr("x_outer"); rmax_hR = arr("rmax_hR")

    log_gc = np.log10(gc_obs); log_gcd = np.log10(gc_deep)
    log_hR = np.log10(hR_a); log_vf = np.log10(vflat_a)
    log_x = np.log10(x_outer + 1e-10)

    # 勾配量 (gc_deep ベース)
    E_grad = arr("E_grad_d")
    E_grad_norm = arr("E_grad_norm_d")
    E_local = arr("E_local_d")
    grad_peak = arr("grad_peak_d")
    grad_peak_norm = arr("grad_peak_norm_d")
    grad_local = arr("grad_local_ratio_d")
    r_grad_hR = arr("r_grad_peak_hR_d")
    grad_oi = arr("grad_oi_d")
    e_grad_dens = arr("e_grad_density_d")

    m = (np.isfinite(log_gc) &amp; np.isfinite(log_hR) &amp; np.isfinite(log_vf)
         &amp; np.isfinite(log_x) &amp; np.isfinite(log_gcd)
         &amp; np.isfinite(E_grad) &amp; (E_grad &gt; 0)
         &amp; np.isfinite(E_local) &amp; (E_local &gt; 0))

    print(f"Valid: {m.sum()}")

    log_Eg = np.log10(E_grad + 1e-30)
    log_Egn = np.log10(E_grad_norm + 1e-30)
    log_El = np.log10(E_local + 1e-30)
    log_glr = np.log10(grad_local + 1e-30)
    log_gp = np.log10(grad_peak + 1e-30)
    log_gpn = np.log10(grad_peak_norm + 1e-30)

    # ============================================================
    # Test 1: 勾配パラメータの基本統計
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 1: 勾配パラメータの基本統計")
    print("=" * 50)

    grad_params = {
        "E_grad": E_grad,
        "E_grad_norm": E_grad_norm,
        "E_local": E_local,
        "grad_peak": grad_peak,
        "grad_peak_norm": grad_peak_norm,
        "E_grad/E_local": grad_local,
        "r_grad_peak/hR": r_grad_hR,
        "grad_outer/inner": grad_oi,
```

```
        "e_grad_density": e_grad_dens,
    }

    for name, ap in grad_params.items():
        v = np.isfinite(ap) &amp;amp; m &amp;amp; (ap &amp;gt; 0)
        if v.sum() &amp;gt; 10:
            print(f"  {name:20s}: N={v.sum():3d}  "
                  f"median={np.median(ap[v]):.4e}  "
                  f"std/mean={np.std(ap[v])/np.mean(ap[v]):.3f}")

    # ============================================================
    # Test 2: 勾配量 vs hR, x_outer の相関 [*]核心[*]
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 2: 勾配量 vs hR, x_outer, gc の相関")
    print("=" * 50)

    print(f"  {'パラメータ':&amp;lt;20s} {'rho(hR)':&amp;gt;10s} {'rho(x)':&amp;gt;10s} "
          f"{'rho(gc)':&amp;gt;10s} {'rho(hR|x)':&amp;gt;10s}")
    print("  " + "-" * 64)

    key_params = {}
    for name, ap in grad_params.items():
        v = np.isfinite(ap) &amp;amp; m &amp;amp; (ap &amp;gt; 0)
        if v.sum() &amp;lt; 20:
            continue
        log_ap = np.log10(ap[v] + 1e-30)

        rh, _ = pearsonr(log_hR[v], log_ap)
        rx, _ = pearsonr(log_x[v], log_ap)
        rg, _ = pearsonr(log_gc[v], log_ap)
        # hR|x_outer 偏相関: 勾配量が x を超える hR 情報を持つか
        rhx, phx = partial_corr(log_hR[v], log_ap, [log_x[v]])

        print(f"  {name:&amp;lt;20s} {rh:+.4f}     {rx:+.4f}     {rg:+.4f}     {rhx:+.4f}")
        key_params[name] = (ap, log_ap, v, rhx)

    # ============================================================
    # Test 3: 勾配量を統制した hR 偏相関 [*][*]核心[*][*]
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 3: 勾配量統制後の hR 偏相関")
    print("=" * 50)

    base_controls = [log_vf[m], log_x[m]]
    rho_base, p_base = partial_corr(log_hR[m], log_gc[m], base_controls)
    print(f"  ベースライン |vflat, x: rho = {rho_base:+.4f}")

    print(f"¥n  {'追加パラメータ':&amp;lt;20s} {'rho':&amp;gt;8s} {'Δ|rho|':&amp;gt;10s} {'p':&amp;gt;12s}")
    print("  " + "-" * 52)

    best_delta = 0; best_name = ""; best_ctrl = None
    for name, ap in grad_params.items():
        v = np.isfinite(ap) &amp;amp; m &amp;amp; (ap &amp;gt; 0)
        if v.sum() &amp;lt; m.sum() * 0.7:
            continue
        ap_filled = ap.copy()
        ap_filled[~(np.isfinite(ap) &amp;amp; (ap &amp;gt; 0))] = np.nanmedian(ap[ap &amp;gt; 0])
        ctrl = np.log10(ap_filled[m] + 1e-30)

        try:
            rho_new, p_new = partial_corr(log_hR[m], log_gc[m],
                                          base_controls + [ctrl])
            delta = abs(rho_new) - abs(rho_base)
            print(f"  {name:&amp;lt;20s} {rho_new:+.4f}   {delta:+.4f}   p={p_new:.3e}")
            if delta &amp;lt; best_delta:
                best_delta = delta; best_name = name; best_ctrl = ctrl
        except:
            pass

    if best_delta &amp;lt; -0.01:
        print(f"¥n  [*] 最も効果的: {best_name} (Δ = {best_delta:+.4f})")
        pct = -best_delta / 0.472 * 100
        print(f"  -&amp;gt; hR偏相関の追加 {pct:.1f}% を説明")
    else:
        print(f"¥n  勾配パラメータも hR 偏相関を有意に減らさない")

    # ============================================================
    # Test 4: E_grad/E_local 比の独立性
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 4: E_grad/E_local 比の独立性")
    print("=" * 50)

    v_glr = np.isfinite(grad_local) &amp;amp; m &amp;amp; (grad_local &amp;gt; 0)
    if v_glr.sum() &amp;gt; 20:
        log_glr_v = np.log10(grad_local[v_glr])
```

```
411|
412|         # x_outer との相関
413|         rho_glx, p_glx = pearsonr(log_x[v_glr], log_glr_v)
414|         print(f"  rho(E_grad/E_local, x_outer) = {rho_glx:+.4f}, p = {p_glx:.3e}")
415|
416|         # hR との偏相関 (x 統制)
417|         rho_glh, p_glh = partial_corr(log_hR[v_glr], log_glr_v, [log_x[v_glr]])
418|         print(f"  rho(E_grad/E_local, hR | x) = {rho_glh:+.4f}, p = {p_glh:.3e}")
419|
420|         # 理論的予測: E_grad/E_local ~ 1/hR^2 (勾配項が hR に依存)
421|         sl = linregress(log_hR[v_glr], log_glr_v)
422|         print(f"  log(E_grad/E_local) vs log(hR): slope = {sl.slope:.4f} +/- {sl.stderr:.4f}")
423|         print(f"  理論予測: slope = -2.0")
424|
425|     # ============================================================
426|     # Test 5: gc残差 vs 勾配量
427|     # ============================================================
428|     print("¥n" + "=" * 50)
429|     print("Test 5: gc残差(obs-deep) vs 勾配量")
430|     print("=" * 50)
431|
432|     resid = log_gc[m] - log_gcd[m]
433|
434|     for name, ap in grad_params.items():
435|         v = np.isfinite(ap) &amp;amp; m &amp;amp; (ap &amp;gt; 0)
436|         if v.sum() &amp;lt; 20:
437|             continue
438|         log_ap = np.log10(ap[v] + 1e-30)
439|         # 残差をm内のインデックスに合わせる
440|         resid_v = (log_gc[v] - log_gcd[v])
441|         rho_r, p_r = pearsonr(log_ap, resid_v)
442|         # x_outer 統制後
443|         rho_rx, p_rx = partial_corr(log_ap, resid_v, [log_x[v]])
444|         print(f"  {name:20s}: rho(resid)={rho_r:+.4f}  rho(resid|x)={rho_rx:+.4f}")
445|
446|     # ============================================================
447|     # Test 6: rmax/hR &amp;gt; 8 サブセット
448|     # ============================================================
449|     print("¥n" + "=" * 50)
450|     print("Test 6: rmax/hR &amp;gt; 8 サブセット")
451|     print("=" * 50)
452|
453|     m8 = m &amp;amp; (rmax_hR &amp;gt; 8)
454|     if m8.sum() &amp;gt; 15:
455|         rho_b8, _ = partial_corr(log_hR[m8], log_gc[m8],
456|                                  [log_vf[m8], log_x[m8]])
457|         print(f"  ベースライン (rmax/hR&amp;gt;8, N={m8.sum()}): rho = {rho_b8:+.4f}")
458|
459|         for name, ap in [("E_grad", E_grad), ("E_grad/E_local", grad_local),
460|                          ("grad_peak_norm", grad_peak_norm),
461|                          ("E_grad_norm", E_grad_norm)]:
462|             v8 = m8 &amp;amp; np.isfinite(ap) &amp;amp; (ap &amp;gt; 0)
463|             if v8.sum() &amp;lt; 10:
464|                 continue
465|             ap_f = ap.copy()
466|             ap_f[~(np.isfinite(ap) &amp;amp; (ap &amp;gt; 0))] = np.nanmedian(ap[ap &amp;gt; 0])
467|             ctrl = np.log10(ap_f[m8] + 1e-30)
468|             try:
469|                 rho8, p8 = partial_corr(log_hR[m8], log_gc[m8],
470|                                         [log_vf[m8], log_x[m8], ctrl])
471|                 delta8 = abs(rho8) - abs(rho_b8)
472|                 print(f"  +{name:20s}: rho={rho8:+.4f}, delta={delta8:+.4f}")
473|             except:
474|                 pass
475|
476|     # ============================================================
477|     # Test 7: 段階的偏相関 (vflat -&amp;gt; x -&amp;gt; E_grad/E_local)
478|     # ============================================================
479|     print("¥n" + "=" * 50)
480|     print("Test 7: 段階的偏相関 (勾配比を追加)")
481|     print("=" * 50)
482|
483|     # grad_local を使用 (最も理論的に動機づけられた量)
484|     glr_filled = grad_local.copy()
485|     glr_filled[~(np.isfinite(glr_filled) &amp;amp; (glr_filled &amp;gt; 0))] = np.nanmedian(
486|         grad_local[np.isfinite(grad_local) &amp;amp; (grad_local &amp;gt; 0)])
487|     log_glr_all = np.log10(glr_filled[m] + 1e-30)
488|
489|     steps = [
490|         ("raw", []),
491|         ("|vflat", [log_vf[m]]),
492|         ("|vflat, x_outer", [log_vf[m], log_x[m]]),
493|         ("|vflat, x, E_grad/E_local", [log_vf[m], log_x[m], log_glr_all]),
494|     ]
495|
496|     # best_ctrl があれば追加
497|     if best_ctrl is not None and best_name != "E_grad/E_local":
```

```
        steps.append(
            (f"|vflat, x, {best_name}", [log_vf[m], log_x[m], best_ctrl])
        )
        # 両方同時
        steps.append(
            (f"|vflat, x, ratio, {best_name}",
             [log_vf[m], log_x[m], log_glr_all, best_ctrl])
        )

    print(f"  {'統制変数':&lt;40s} {'rho':&gt;8s} {'p':&gt;12s}")
    print("  " + "-" * 62)
    for label, ctrls in steps:
        try:
            rho_s, p_s = partial_corr(log_hR[m], log_gc[m], ctrls)
            print(f"  {label:&lt;40s} {rho_s:+.4f}   p={p_s:.3e}")
        except Exception as e:
            print(f"  {label:&lt;40s} ERROR: {e}")

    # ============================================================
    # Test 8: 理論的スケーリングの検証
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 8: 理論的スケーリング")
    print("=" * 50)

    # E_grad ~ hR^a の a を測定
    v_eg = np.isfinite(E_grad) &amp;amp; m &amp;amp; (E_grad &gt; 0)
    if v_eg.sum() &gt; 20:
        sl_eg = linregress(log_hR[v_eg], np.log10(E_grad[v_eg]))
        print(f"  E_grad vs hR: slope = {sl_eg.slope:.4f} +/- {sl_eg.stderr:.4f}")
        print(f"  理論予測: E_grad ~ hR^(-2+3) = hR^1 (次元から)")

    # E_local ~ hR^a
    v_el = np.isfinite(E_local) &amp;amp; m &amp;amp; (E_local &gt; 0)
    if v_el.sum() &gt; 20:
        sl_el = linregress(log_hR[v_el], np.log10(E_local[v_el]))
        print(f"  E_local vs hR: slope = {sl_el.slope:.4f} +/- {sl_el.stderr:.4f}")

    # E_grad/E_local ~ hR^a -&gt; aが負なら hR 小でgradient支配
    if v_glr.sum() &gt; 20:
        print(f"  E_grad/E_local vs hR: slope = {sl.slope:.4f} (再掲)")
        if sl.slope &lt; -0.5:
            print(f"  -&gt; hR が小さい銀河ほど勾配項が支配的 [*]")

    # ============================================================
    # SUMMARY
    # ============================================================
    print("¥n" + "=" * 70)
    print("SUMMARY")
    print("=" * 70)

    print(f"¥n  hR偏相関の段階的分解:")
    print(f"    |vflat:           rho = -0.472")
    print(f"    |vflat, x:        rho = {rho_base:+.4f} (x_outer 69%)")
    if best_delta &lt; -0.01:
        rho_best, _ = partial_corr(log_hR[m], log_gc[m],
                                   base_controls + [best_ctrl])
        print(f"    |vflat, x, {best_name}: rho = {rho_best:+.4f} "
              f"(追加 {-best_delta/0.472*100:.1f}%)")

    print(f"¥n  勾配エネルギー仮説の評価:")
    if best_delta &lt; -0.02:
        print(f"  -&gt; 勾配項が hR 偏相関の追加分を部分的に説明 [*]")
        print(f"  -&gt; 膜の歪み勾配エネルギーは物理的に妥当な候補")
    elif best_delta &lt; -0.01:
        print(f"  -&gt; わずかな効果あり ({-best_delta/0.472*100:.1f}%) だが決定的ではない")
    else:
        print(f"  -&gt; 勾配項も残り31%を説明しない")

    if v_glr.sum() &gt; 20 and sl.slope &lt; -1.0:
        print(f"¥n  ただし E_grad/E_local ~ hR^{sl.slope:.1f} は物理的に興味深い:")
        print(f"  小さい hR の銀河で勾配項が相対的に重要 -&gt; hR 依存性の定性的起源")


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

9. トートロジーチェック

ファイル: sparc_tautology_check.py

解析目的

E_grad_norm = hR^2 x E_grad の定義にhR^2が含まれるため、hR偏相関の消失がトートロジーかを検証。8テスト: E_grad単独、プラシーボ(E_local x hR^2)、ランダム x hR^2、分解、hRシャッフル(1000回)、E_grad/E_local精密、最適alpha探索、hR直接統制。

結果

トートロジー確定: hR^2単独でrho=+0.011。最適alpha=1.75。E_grad_norm 26%を撤回。

ソースコード

(493 行)

```
#!/usr/bin/env python3
"""
sparc_tautology_check.py
========================
トートロジーチェック: E_grad_norm の hR 含有が偏相関消失の原因か

問題:
  E_grad_norm = int(hR x dε/dr)^2 r dr -&gt; 定義に hR が含まれる
  hR 偏相関を統制すると ρ=-0.145 -&gt; +0.023 に消失
  -&gt; hR^2 を掛けただけのトートロジーか？

テスト:
  Test 1: E_grad (非正規化、hR非含有) 単独での偏相関
  Test 2: プラシーボ -- E_local x hR^2 で同じ効果が出るか
  Test 3: プラシーボ -- ランダム量 x hR^2 で同じ効果が出るか
  Test 4: E_grad_norm = hR^2 x G(r) と分解、G(r) の独立寄与
  Test 5: hR シャッフルテスト (1000回)
  Test 6: E_grad/E_local (hR 非含有) の精密評価
  Test 7: 「任意の X x hR^α」で最適α を探索 -&gt; α=2 が特別か

実行: uv run --with scipy --with matplotlib python sparc_tautology_check.py
"""

import numpy as np
from scipy.optimize import minimize_scalar
from scipy.stats import pearsonr
from numpy.linalg import lstsq
from pathlib import Path
import csv
import sys

a0 = 1.2e-10
kpc_to_m = 3.0857e19
kms2_to_ms2 = 1.0e6
Yd_default = 0.5
Yb_default = 0.7

BASE = Path(r"D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン")
ROTMOD_DIR = BASE / "Rotmod_LTG"
GC_CSV = BASE / "sparc_gc.csv"


def load_gc_table(path):
    data = {}
    with open(path, "r") as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = row["Galaxy"].strip()
            data[name] = {
                "gc_obs": float(row["gc"]),
                "vflat": float(row["Vflat"]),
                "hR": float(row["hR"]),
            }
    return data


def load_rotmod(galaxy_name, rotmod_dir):
    fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
    if not fpath.exists():
        return None
    r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
    with open(fpath, "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
            r.append(float(parts[0]))
            vobs.append(float(parts[1]))
            errv.append(float(parts[2]))
            vgas.append(float(parts[3]))
            vdisk.append(float(parts[4]))
            vbul.append(float(parts[5]))
    return (np.array(r), np.array(vobs), np.array(errv),
            np.array(vgas), np.array(vdisk), np.array(vbul))


def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
    r_m = r_kpc * kpc_to_m
    conv = kms2_to_ms2 / r_m
    gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
          + Yb*np.abs(Vbul)*Vbul*conv)
    return np.abs(gN)


def partial_corr(x, y, controls):
    if len(controls) == 0:
        return pearsonr(x, y)
    C = np.column_stack(controls + [np.ones(len(x))])
    cx, _, _, _ = lstsq(C, x, rcond=None)
    cy, _, _, _ = lstsq(C, y, rcond=None)
    return pearsonr(x - C @ cx, y - C @ cy)


def fit_gc_deep(r_kpc, gN, Vobs, errV):
    e = np.where(errV &gt; 0, errV, np.median(Vobs) * 0.1)
    def chi2(lgc):
        gc = 10**lgc
        V_pred = np.sqrt(r_kpc * kpc_to_m * np.sqrt(gN * gc)) * 1e-3
        return np.sum(((Vobs - V_pred) / e)**2)
    res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
    return 10**res.x


def compute_strain_energies(r_kpc, gN, gc, hR):
    eps = np.sqrt(gN / gc)
    N = len(r_kpc)
    result = {"E_local": np.nan, "E_grad": np.nan, "E_grad_norm": np.nan}

    if N &lt; 4:
        return result

    result["E_local"] = np.trapz(eps**2 * r_kpc, r_kpc)

    dr = np.diff(r_kpc)
    deps = np.diff(eps)
    mask = dr &gt; 0
    if mask.sum() &lt; 2:
        return result

    dedr = deps[mask] / dr[mask]
    r_mid = (r_kpc[:-1] + r_kpc[1:])[mask] / 2.0

    result["E_grad"] = np.trapz(dedr**2 * r_mid, r_mid)
    dedr_norm = dedr * hR
    result["E_grad_norm"] = np.trapz(dedr_norm**2 * r_mid, r_mid)

    return result


# ============================================================
def main():
    print("=" * 70)
    print("トートロジーチェック: E_grad_norm の hR 含有効果")
    print("=" * 70)

    gc_table = load_gc_table(GC_CSV)

    records = []
    for gal_name, info in gc_table.items():
        rotmod = load_rotmod(gal_name, ROTMOD_DIR)
        if rotmod is None:
            continue
        r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
        if len(r_kpc) &lt; 8:
            continue

```

```
        gc_obs = info["gc_obs"]
        gc_obs_ms2 = gc_obs * a0 if gc_obs &gt; 1e-5 else gc_obs
        hR = info["hR"]
        vflat = info["vflat"]

        gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)
        gc_deep = fit_gc_deep(r_kpc, gN, Vobs, errV)

        strain = compute_strain_energies(r_kpc, gN, gc_deep, hR)

        n_out = max(3, len(r_kpc) // 3)
        x_outer = np.median(gN[-n_out:] / gc_deep) if gc_deep &gt; 0 else np.nan

        records.append({
            "gc_obs": gc_obs_ms2, "gc_deep": gc_deep,
            "hR": hR, "vflat": vflat, "x_outer": x_outer,
            "E_local": strain["E_local"],
            "E_grad": strain["E_grad"],
            "E_grad_norm": strain["E_grad_norm"],
        })

    N = len(records)

    def arr(k): return np.array([r[k] for r in records], dtype=float)

    gc_obs = arr("gc_obs"); hR_a = arr("hR"); vflat_a = arr("vflat")
    x_outer = arr("x_outer")
    E_local = arr("E_local"); E_grad = arr("E_grad"); E_grad_norm = arr("E_grad_norm")

    log_gc = np.log10(gc_obs)
    log_hR = np.log10(hR_a)
    log_vf = np.log10(vflat_a)
    log_x = np.log10(x_outer + 1e-10)

    m = (np.isfinite(log_gc) &amp; np.isfinite(log_hR) &amp; np.isfinite(log_vf)
         &amp; np.isfinite(log_x) &amp; np.isfinite(E_grad) &amp; (E_grad &gt; 0)
         &amp; np.isfinite(E_local) &amp; (E_local &gt; 0)
         &amp; np.isfinite(E_grad_norm) &amp; (E_grad_norm &gt; 0))

    print(f"Valid: {m.sum()}")

    base = [log_vf[m], log_x[m]]
    rho_base, p_base = partial_corr(log_hR[m], log_gc[m], base)
    print(f"¥nベースライン |vflat, x: rho = {rho_base:+.4f}, p = {p_base:.3e}")

    # ============================================================
    # Test 1: E_grad (非正規化、hR非含有)
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 1: E_grad (非正規化、hR非含有) ")
    print("=" * 50)

    log_Eg = np.log10(E_grad[m])
    rho_Eg, p_Eg = partial_corr(log_hR[m], log_gc[m], base + [log_Eg])
    print(f"  |vflat, x, E_grad: rho = {rho_Eg:+.4f}, p = {p_Eg:.3e}")
    print(f"  Δ|rho| = {abs(rho_Eg) - abs(rho_base):+.4f}")

    # E_grad_norm (比較)
    log_Egn = np.log10(E_grad_norm[m])
    rho_Egn, p_Egn = partial_corr(log_hR[m], log_gc[m], base + [log_Egn])
    print(f"  |vflat, x, E_grad_norm: rho = {rho_Egn:+.4f}, p = {p_Egn:.3e}")
    print(f"  Δ|rho| = {abs(rho_Egn) - abs(rho_base):+.4f}")

    diff = abs(rho_Eg) - abs(rho_Egn)
    print(f"¥n  E_grad vs E_grad_norm の差: {diff:+.4f}")
    if abs(rho_Eg) &gt; abs(rho_base) * 0.9:
        print("  -&gt; E_grad(非正規化)は hR偏相関をほとんど減らさない")
        print("  -&gt; E_grad_normの効果は hR^2正規化に依存")
    else:
        print(f"  -&gt; E_grad(非正規化)も偏相関を {(abs(rho_base)-abs(rho_Eg))/abs(rho_base)*100:.1f}% 削減")

    # ============================================================
    # Test 2: プラシーボ -- E_local x hR^2
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 2: プラシーボ -- E_local x hR^2")
    print("=" * 50)

    placebo1 = E_local[m] * hR_a[m]**2
    log_p1 = np.log10(placebo1 + 1e-30)
    rho_p1, p_p1 = partial_corr(log_hR[m], log_gc[m], base + [log_p1])
    print(f"  |vflat, x, E_local*hR^2: rho = {rho_p1:+.4f}, p = {p_p1:.3e}")
    print(f"  Δ|rho| = {abs(rho_p1) - abs(rho_base):+.4f}")

    # 純粋 hR^2 だけ
    log_hR2 = 2 * log_hR[m]
    rho_hR2, p_hR2 = partial_corr(log_hR[m], log_gc[m], base + [log_hR2])
```

```
    print(f"¥n  |vflat, x, hR^2: rho = {rho_hR2:+.4f}, p = {p_hR2:.3e}")
    print(f"  (hR^2 は hR の単調変換なので、hR統制と同義のはず)")

    # E_local x hR^n for various n
    print(f"¥n  E_local x hR^n のスキャン:")
    for n in [0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0]:
        pn = E_local[m] * hR_a[m]**n
        log_pn = np.log10(pn + 1e-30)
        rho_pn, _ = partial_corr(log_hR[m], log_gc[m], base + [log_pn])
        tag = " &lt;-- E_grad_norm相当" if abs(n - 2.0) &lt; 0.01 else ""
        print(f"    n={n:.1f}: rho = {rho_pn:+.4f}{tag}")

    # ============================================================
    # Test 3: ランダム量 x hR^2
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 3: ランダム量 x hR^2 のプラシーボ")
    print("=" * 50)

    np.random.seed(42)
    n_trials = 5
    rho_randoms = []
    for i in range(n_trials):
        rand_qty = np.random.lognormal(0, 1, m.sum())
        placebo_rand = rand_qty * hR_a[m]**2
        log_pr = np.log10(placebo_rand + 1e-30)
        rho_r, _ = partial_corr(log_hR[m], log_gc[m], base + [log_pr])
        rho_randoms.append(rho_r)
        print(f"  Trial {i+1}: rho = {rho_r:+.4f}")

    print(f"  Mean: {np.mean(rho_randoms):+.4f}, std: {np.std(rho_randoms):.4f}")
    print(f"  E_grad_norm: {rho_Egn:+.4f}")

    if abs(np.mean(rho_randoms)) &lt; abs(rho_base) * 0.3:
        print("  -&gt; ランダム x hR^2 でも偏相関がほぼ消える")
        print("  -&gt; [*] hR^2 を掛けるだけで偏相関が消える（トートロジー）")
        tautology_random = True
    else:
        print("  -&gt; ランダム x hR^2 では偏相関が残る")
        print("  -&gt; E_grad_norm の物理的内容が寄与している")
        tautology_random = False

    # ============================================================
    # Test 4: E_grad_norm = hR^2 x G(r) の分解
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 4: E_grad_norm = hR^2 x G(r) の分解")
    print("=" * 50)

    # G(r) = E_grad_norm / hR^2 = E_grad（定義から）
    # つまり G(r) = E_grad そのもの
    print("  E_grad_norm = hR^2 x int(dε/dr)^2 r dr = hR^2 x E_grad")
    print("  つまり G(r) = E_grad (hR非含有成分)")
    print(f"  E_grad のみでの偏相関: rho = {rho_Eg:+.4f} (Test 1 再掲)")
    print(f"  E_grad_norm での偏相関: rho = {rho_Egn:+.4f}")
    print(f"  差: {abs(rho_Eg) - abs(rho_Egn):+.4f}")
    print(f"¥n  -&gt; E_grad_norm の効果 = E_grad の効果 + hR^2 の効果")

    # E_grad と hR^2 の両方を同時統制
    rho_both, p_both = partial_corr(log_hR[m], log_gc[m],
                                    base + [log_Eg, log_hR2])
    print(f"  |vflat, x, E_grad, hR^2: rho = {rho_both:+.4f}")
    print(f"  (E_grad + hR^2 同時 = E_grad_norm と同等のはず)")

    # ============================================================
    # Test 5: hR シャッフルテスト (1000回)
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 5: hR シャッフルテスト (1000回)")
    print("=" * 50)

    # E_grad は固定、hR だけシャッフルして E_grad_norm_shuffled = E_grad x hR_shuffled^2
    np.random.seed(123)
    n_shuffle = 1000
    rho_shuffled = []
    for _ in range(n_shuffle):
        hR_shuf = hR_a[m].copy()
        np.random.shuffle(hR_shuf)
        Egn_shuf = E_grad[m] * hR_shuf**2
        log_shuf = np.log10(Egn_shuf + 1e-30)
        rho_s, _ = partial_corr(log_hR[m], log_gc[m], base + [log_shuf])
        rho_shuffled.append(rho_s)

    rho_shuffled = np.array(rho_shuffled)
    print(f"  Shuffled: mean = {np.mean(rho_shuffled):+.4f}, "
          f"std = {np.std(rho_shuffled):.4f}")
    print(f"  Shuffled: median = {np.median(rho_shuffled):+.4f}")
```

```
    print(f"  Shuffled: 5th-95th = [{np.percentile(rho_shuffled,5):+.4f}, "
          f"{np.percentile(rho_shuffled,95):+.4f}]")
    print(f"  Real E_grad_norm: rho = {rho_Egn:+.4f}")

    # E_grad_norm の rho が shuffled 分布の何パーセンタイルか
    pct = np.mean(rho_shuffled &lt;= rho_Egn) * 100
    print(f"  Real は shuffled の {pct:.1f} パーセンタイル")

    if pct &gt; 5 and pct &lt; 95:
        print("  -&gt; Real は shuffled と区別できない")
        print("  -&gt; [*] E_grad_norm の偏相関消失は hR^2 x (何か) の構造的効果")
        tautology_shuffle = True
    else:
        print(f"  -&gt; Real ({rho_Egn:+.4f}) は shuffled 分布の外")
        print("  -&gt; E_grad_norm の物理的内容 (E_grad の gN 構造) が寄与")
        tautology_shuffle = False

    # ============================================================
    # Test 6: E_grad/E_local (hR 非含有) の精密評価
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 6: E_grad/E_local (hR 非含有) の精密評価")
    print("=" * 50)

    ratio = E_grad[m] / E_local[m]
    log_ratio = np.log10(ratio + 1e-30)

    rho_ratio, p_ratio = partial_corr(log_hR[m], log_gc[m], base + [log_ratio])
    print(f"  |vflat, x, E_grad/E_local: rho = {rho_ratio:+.4f}, p = {p_ratio:.3e}")
    print(f"  Δ|rho| = {abs(rho_ratio) - abs(rho_base):+.4f}")
    pct_explained = (abs(rho_base) - abs(rho_ratio)) / 0.472 * 100
    print(f"  hR偏相関全体に対する説明率: {pct_explained:.1f}%")

    # E_grad/E_local が hR と独立に gc と相関するか
    rho_ratio_gc, p_rg = partial_corr(log_ratio, log_gc[m], [log_hR[m], log_vf[m]])
    print(f"  rho(ratio, gc | hR, vflat) = {rho_ratio_gc:+.4f}, p = {p_rg:.3e}")

    # E_grad/E_local と hR の相関
    rho_ratio_hR, p_rh = pearsonr(log_hR[m], log_ratio)
    print(f"  rho(ratio, hR) = {rho_ratio_hR:+.4f}")

    # E_grad/E_local と x_outer の相関
    rho_ratio_x, p_rx = pearsonr(log_x[m], log_ratio)
    print(f"  rho(ratio, x_outer) = {rho_ratio_x:+.4f}")

    # ============================================================
    # Test 7: 「X x hR^α」で最適αを探索
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 7: E_grad x hR^alpha で最適alpha")
    print("=" * 50)

    alphas = np.arange(-2.0, 4.1, 0.25)
    rho_alpha = []
    for a in alphas:
        qty = E_grad[m] * hR_a[m]**a
        log_qty = np.log10(qty + 1e-30)
        rho_a, _ = partial_corr(log_hR[m], log_gc[m], base + [log_qty])
        rho_alpha.append(rho_a)
        if abs(a - 0) &lt; 0.01 or abs(a - 2) &lt; 0.01 or abs(rho_a) &lt; 0.05:
            print(f"  alpha={a:+.2f}: rho = {rho_a:+.4f}")

    rho_alpha = np.array(rho_alpha)
    best_idx = np.argmin(np.abs(rho_alpha))
    best_alpha = alphas[best_idx]
    best_rho = rho_alpha[best_idx]
    print(f"¥n  最適 alpha = {best_alpha:.2f}: rho = {best_rho:+.4f}")
    print(f"  alpha=0 (E_grad のみ): rho = {rho_alpha[alphas==0][0] if 0 in alphas else 'N/A'}")
    print(f"  alpha=2 (E_grad_norm): rho = {rho_alpha[np.argmin(np.abs(alphas-2))]:+.4f}")

    # alphaが0付近で最適なら物理的、2付近なら hR^2 効果
    if abs(best_alpha) &lt; 0.5:
        print("  -&gt; 最適αが0付近 -&gt; E_grad自体に物理的説明力あり")
    elif abs(best_alpha - 2) &lt; 0.5:
        print("  -&gt; 最適αが2付近 -&gt; hR^2正規化が本質 (トートロジー寄り) ")
    else:
        print(f"  -&gt; 最適αが{best_alpha:.1f} -&gt; 中間的")

    # ============================================================
    # Test 8: log(hR) を直接統制変数に追加 (自明なテスト)
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 8: hR 自体を統制 (参照テスト) ")
    print("=" * 50)

    # hR を統制変数に追加したら偏相関は定義上ゼロになるはず
    rho_hR_ctrl, p_hR_ctrl = partial_corr(log_hR[m], log_gc[m],
```

```
                                        base + [log_hR[m]])
    print(f"  |vflat, x, hR: rho = {rho_hR_ctrl:+.4f}")
    print(f"  (hR を直接統制 -&gt; 偏相関は定義上ゼロ)")
    print(f"  E_grad_norm で rho={rho_Egn:+.4f} はこれにどれだけ近いか:")
    print(f"  -&gt; {abs(rho_Egn)/abs(rho_hR_ctrl)*100:.0f}% of hR直接統制"
          if abs(rho_hR_ctrl) &gt; 0.001
          else f"  -&gt; hR直接統制もゼロ、E_grad_normもゼロ")

    # ============================================================
    # SUMMARY
    # ============================================================
    print("¥n" + "=" * 70)
    print("SUMMARY: トートロジー判定")
    print("=" * 70)

    print(f"¥n  偏相関の比較:")
    print(f"    |vflat, x:                  rho = {rho_base:+.4f} (ベースライン)")
    print(f"    |vflat, x, E_grad:          rho = {rho_Eg:+.4f} (hR非含有)")
    print(f"    |vflat, x, E_grad/E_local:  rho = {rho_ratio:+.4f} (hR非含有)")
    print(f"    |vflat, x, E_local*hR^2:    rho = {rho_p1:+.4f} (プラシーボ)")
    print(f"    |vflat, x, E_grad_norm:     rho = {rho_Egn:+.4f} (hR含有)")
    print(f"    |vflat, x, hR:              rho = {rho_hR_ctrl:+.4f} (hR直接)")

    print(f"¥n  シャッフルテスト:")
    print(f"    shuffled mean = {np.mean(rho_shuffled):+.4f}")
    print(f"    real = {rho_Egn:+.4f}")
    print(f"    percentile = {pct:.1f}%")

    print(f"¥n  最適 alpha = {best_alpha:.2f}")

    print(f"¥n  ■ 判定:")

    # 判定ロジック
    Eg_effective = abs(rho_Eg) &lt; abs(rho_base) * 0.7
    ratio_effective = abs(rho_ratio) &lt; abs(rho_base) * 0.7
    placebo_works = abs(rho_p1) &lt; abs(rho_base) * 0.3
    shuffle_indistinguishable = (pct &gt; 5 and pct &lt; 95)

    if placebo_works and shuffle_indistinguishable:
        print("  [*] トートロジー確定")
        print("    E_localxhR^2 (プラシーボ) で同じ効果が出る")
        print("    シャッフルでも区別できない")
        print("    -&gt; E_grad_normの26%はhR^2正規化のアーティファクト")
        if ratio_effective:
            print(f"    ただし E_grad/E_local (hR非含有) で rho={rho_ratio:+.4f}")
            pct_real = (abs(rho_base) - abs(rho_ratio)) / 0.472 * 100
            print(f"    -&gt; 真の物理的効果: {pct_real:.1f}% (勾配/局所比)")
        else:
            print("    E_grad/E_local も効果なし -&gt; 勾配効果全体がアーティファクト")
    elif not placebo_works and not shuffle_indistinguishable:
        print("  [*] 物理的効果確定")
        print("    プラシーボでは効果なし、シャッフルでも区別できる")
        print("    -&gt; E_grad_normの26%は真の物理的効果")
    else:
        print("  [*] 部分的トートロジー")
        print(f"    プラシーボ効果: {'あり' if placebo_works else 'なし'}")
        print(f"    シャッフル区別: {'不能' if shuffle_indistinguishable else '可能'}")
        if Eg_effective:
            pct_Eg = (abs(rho_base) - abs(rho_Eg)) / 0.472 * 100
            print(f"    E_grad(hR非含有)の物理的寄与: {pct_Eg:.1f}%")
        if ratio_effective:
            pct_real = (abs(rho_base) - abs(rho_ratio)) / 0.472 * 100
            print(f"    E_grad/E_local(hR非含有)の物理的寄与: {pct_real:.1f}%")

    # 最終的な hR 偏相関分解
    print(f"¥n  ■ hR偏相関の最終分解:")
    print(f"    x_outer (MOND遷移): 69%")
    if ratio_effective:
        pct_real = (abs(rho_base) - abs(rho_ratio)) / 0.472 * 100
        print(f"    E_grad/E_local (勾配比, hR非含有): {pct_real:.1f}%")
        print(f"    残差: {100 - 69 - pct_real:.1f}%")
    elif Eg_effective:
        pct_Eg = (abs(rho_base) - abs(rho_Eg)) / 0.472 * 100
        print(f"    E_grad (勾配, hR非含有): {pct_Eg:.1f}%")
        print(f"    残差: {100 - 69 - pct_Eg:.1f}%")
    else:
        print(f"    勾配項 (hR非含有): 効果なし")
        print(f"    残差: 31% (未説明)")


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

10. E_grad/E_local(hR非含有勾配比)精密評価

ファイル: sparc_grad_ratio_detail.py

解析目的

トートロジー訂正後、E_grad/E_local（hR非含有）の物理的寄与を精密評価。8テスト: 偏相関詳細(Pearson/Spearman/効果量)、rmax/hRサブセット、ブートストラップCI(5000回)、独立性(ratio分散分解)、gc残差相関、勾配形状パラメータ、permutation test(2000回)。

結果

B-級: perm p=0.0000, gc残差rho=-0.373(p=8e-6)。CI含0(N=136統計力不足)。18%説明。

ソースコード

(567 行)

```
#!/usr/bin/env python3
"""
sparc_grad_ratio_detail.py
==========================
E_grad/E_local の精密評価

トートロジーチェックで確定:
  - E_grad_norm の26%はhR^2トートロジー（撤回）
  - E_grad/E_local（hR非含有）で18%は示唆的だがp=0.49

本スクリプトの目的:
  E_grad/E_local の物理的寄与をあらゆる角度から精査し、
  「弱いが本物」か「ノイズ」かを判定する。

テスト:
  Test 1: E_grad/E_local の偏相関の詳細（p値、CI、効果量）
  Test 2: rmax/hR サブセットでの有意性
  Test 3: ブートストラップ信頼区間（10000回）
  Test 4: E_grad/E_local のhR・x_outerとの独立性の精密評価
  Test 5: E_grad/E_local の物理的内容の分解（何がratioを駆動するか）
  Test 6: gc残差(obs-deep)との相関の精密評価
  Test 7: E_grad/E_local をさらに分解: 勾配の形状 vs 振幅
  Test 8: permutation test（E_grad/E_localの偏相関のp値を正確に計算）

実行: uv run --with scipy --with matplotlib python sparc_grad_ratio_detail.py
"""

import numpy as np
from scipy.optimize import minimize_scalar
from scipy.stats import pearsonr, spearmanr, linregress
from numpy.linalg import lstsq
from pathlib import Path
import csv
import sys

a0 = 1.2e-10
kpc_to_m = 3.0857e19
kms2_to_ms2 = 1.0e6
Yd_default = 0.5
Yb_default = 0.7

BASE = Path(r"D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン")
ROTMOD_DIR = BASE / "Rotmod_LTG"
GC_CSV = BASE / "sparc_gc.csv"


def load_gc_table(path):
    data = {}
    with open(path, "r") as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = row["Galaxy"].strip()
            data[name] = {
                "gc_obs": float(row["gc"]),
                "vflat": float(row["Vflat"]),
                "hR": float(row["hR"]),
            }
    return data


def load_rotmod(galaxy_name, rotmod_dir):
    fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
```

```
    if not fpath.exists():
        return None
    r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
    with open(fpath, "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
            r.append(float(parts[0]))
            vobs.append(float(parts[1]))
            errv.append(float(parts[2]))
            vgas.append(float(parts[3]))
            vdisk.append(float(parts[4]))
            vbul.append(float(parts[5]))
    return (np.array(r), np.array(vobs), np.array(errv),
            np.array(vgas), np.array(vdisk), np.array(vbul))


def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
    r_m = r_kpc * kpc_to_m
    conv = kms2_to_ms2 / r_m
    gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
          + Yb*np.abs(Vbul)*Vbul*conv)
    return np.abs(gN)


def partial_corr(x, y, controls):
    if len(controls) == 0:
        return pearsonr(x, y)
    C = np.column_stack(controls + [np.ones(len(x))])
    cx, _, _, _ = lstsq(C, x, rcond=None)
    cy, _, _, _ = lstsq(C, y, rcond=None)
    rx = x - C @ cx
    ry = y - C @ cy
    return pearsonr(rx, ry)


def partial_corr_residuals(x, y, controls):
    """偏相関の残差を返す（ブートストラップ用）"""
    if len(controls) == 0:
        return x, y
    C = np.column_stack(controls + [np.ones(len(x))])
    cx, _, _, _ = lstsq(C, x, rcond=None)
    cy, _, _, _ = lstsq(C, y, rcond=None)
    return x - C @ cx, y - C @ cy


def fit_gc_deep(r_kpc, gN, Vobs, errV):
    e = np.where(errV &gt; 0, errV, np.median(Vobs) * 0.1)
    def chi2(lgc):
        gc = 10**lgc
        V_pred = np.sqrt(r_kpc * kpc_to_m * np.sqrt(gN * gc)) * 1e-3
        return np.sum(((Vobs - V_pred) / e)**2)
    res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
    return 10**res.x


def compute_strain(r_kpc, gN, gc, hR):
    eps = np.sqrt(gN / gc)
    N = len(r_kpc)
    if N &lt; 4:
        return np.nan, np.nan, np.nan, None, None

    E_local = np.trapz(eps**2 * r_kpc, r_kpc)

    dr = np.diff(r_kpc)
    deps = np.diff(eps)
    mask = dr &gt; 0
    if mask.sum() &lt; 2:
        return E_local, np.nan, np.nan, None, None

    dedr = deps[mask] / dr[mask]
    r_mid = (r_kpc[:-1] + r_kpc[1:])[mask] / 2.0
    E_grad = np.trapz(dedr**2 * r_mid, r_mid)

    ratio = E_grad / E_local if E_local &gt; 0 else np.nan

    return E_local, E_grad, ratio, dedr, r_mid


# ============================================================
def main():
    print("=" * 70)
    print("E_grad/E_local の精密評価")
```

```
    print("=" * 70)

    gc_table = load_gc_table(GC_CSV)

    records = []
    for gal_name, info in gc_table.items():
        rotmod = load_rotmod(gal_name, ROTMOD_DIR)
        if rotmod is None:
            continue
        r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
        if len(r_kpc) &lt; 8:
            continue

        gc_obs = info["gc_obs"]
        gc_obs_ms2 = gc_obs * a0 if gc_obs &gt; 1e-5 else gc_obs
        hR = info["hR"]
        vflat = info["vflat"]

        gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)
        gc_deep = fit_gc_deep(r_kpc, gN, Vobs, errV)

        E_local, E_grad, ratio, dedr, r_mid = compute_strain(r_kpc, gN, gc_deep, hR)

        n_out = max(3, len(r_kpc) // 3)
        x_outer = np.median(gN[-n_out:] / gc_deep) if gc_deep &gt; 0 else np.nan
        rmax_hR = r_kpc[-1] / hR if hR &gt; 0 else np.nan

        # 勾配プロファイルの形状パラメータ (ratio以外の独立な情報)
        grad_shape = {}
        if dedr is not None and len(dedr) &gt; 4:
            # 勾配の集中度: ピーク付近 vs 全体
            abs_dedr = np.abs(dedr)
            peak_idx = np.argmax(abs_dedr)
            grad_shape["grad_peak_frac"] = abs_dedr[peak_idx] / (np.mean(abs_dedr) + 1e-30)

            # 勾配の非対称性: 内側 vs 外側
            n_half = len(dedr) // 2
            if n_half &gt;= 2:
                grad_shape["grad_asymm"] = (
                    np.mean(abs_dedr[:n_half]) / (np.mean(abs_dedr[n_half:]) + 1e-30))

            # 勾配ピーク位置/hR
            grad_shape["grad_peak_r_hR"] = r_mid[peak_idx] / hR if hR &gt; 0 else np.nan

            # 勾配の2次モーメント (幅)
            total = np.sum(abs_dedr)
            if total &gt; 0:
                r_mean = np.sum(abs_dedr * r_mid) / total
                r_var = np.sum(abs_dedr * (r_mid - r_mean)**2) / total
                grad_shape["grad_width_hR"] = np.sqrt(r_var) / hR if hR &gt; 0 else np.nan
            else:
                grad_shape["grad_width_hR"] = np.nan
        else:
            grad_shape = {"grad_peak_frac": np.nan, "grad_asymm": np.nan,
                          "grad_peak_r_hR": np.nan, "grad_width_hR": np.nan}

        # バルジ/ガス割合
        Vbar_sq = (np.abs(Vgas)*Vgas + Yd_default*np.abs(Vdisk)*Vdisk
                   + Yb_default*np.abs(Vbul)*Vbul)
        va = np.abs(Vbar_sq)
        v = va &gt; 0
        f_bul = np.median(Yb_default*Vbul[v]**2 / va[v]) if v.sum() &gt; 0 else 0
        f_gas = np.median(Vgas[v]**2 / va[v]) if v.sum() &gt; 0 else 0

        records.append({
            "gc_obs": gc_obs_ms2, "gc_deep": gc_deep,
            "hR": hR, "vflat": vflat,
            "x_outer": x_outer, "rmax_hR": rmax_hR,
            "E_local": E_local, "E_grad": E_grad, "ratio": ratio,
            "f_bul": f_bul, "f_gas": f_gas,
            **grad_shape,
        })

    N = len(records)
    def arr(k): return np.array([r[k] for r in records], dtype=float)

    gc_obs = arr("gc_obs"); gc_deep = arr("gc_deep")
    hR_a = arr("hR"); vflat_a = arr("vflat")
    x_outer = arr("x_outer"); rmax_hR = arr("rmax_hR")
    E_local = arr("E_local"); E_grad = arr("E_grad"); ratio = arr("ratio")
    f_bul = arr("f_bul"); f_gas = arr("f_gas")
    grad_pf = arr("grad_peak_frac"); grad_as = arr("grad_asymm")
    grad_pr = arr("grad_peak_r_hR"); grad_wd = arr("grad_width_hR")

    log_gc = np.log10(gc_obs); log_gcd = np.log10(gc_deep)
    log_hR = np.log10(hR_a); log_vf = np.log10(vflat_a)
    log_x = np.log10(x_outer + 1e-10)
```

```
    log_ratio = np.log10(ratio + 1e-30)

    m = (np.isfinite(log_gc) &amp;amp; np.isfinite(log_hR) &amp;amp; np.isfinite(log_vf)
         &amp;amp; np.isfinite(log_x) &amp;amp; np.isfinite(log_ratio)
         &amp;amp; np.isfinite(E_grad) &amp;amp; (E_grad &amp;gt; 0)
         &amp;amp; np.isfinite(E_local) &amp;amp; (E_local &amp;gt; 0))

    print(f"Valid: {m.sum()}")

    base = [log_vf[m], log_x[m]]
    rho_base, p_base = partial_corr(log_hR[m], log_gc[m], base)
    print(f"ベースライン |vflat, x: rho = {rho_base:+.4f}, p = {p_base:.3e}")

    # ============================================================
    # Test 1: E_grad/E_local 偏相関の詳細
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 1: E_grad/E_local 偏相関の詳細")
    print("=" * 50)

    rho_r, p_r = partial_corr(log_hR[m], log_gc[m], base + [log_ratio[m]])
    print(f"  |vflat, x, ratio: rho = {rho_r:+.4f}, p = {p_r:.3e}")
    delta_pct = (abs(rho_base) - abs(rho_r)) / 0.472 * 100
    print(f"  hR偏相関全体に対する削減: {delta_pct:.1f}%")

    # Spearman
    # 残差を取ってからSpearman
    rx_hR, rx_gc = partial_corr_residuals(log_hR[m], log_gc[m], base)
    rho_sp_base, p_sp_base = spearmanr(rx_hR, rx_gc)
    rx_hR2, rx_gc2 = partial_corr_residuals(log_hR[m], log_gc[m], base + [log_ratio[m]])
    rho_sp_r, p_sp_r = spearmanr(rx_hR2, rx_gc2)
    print(f"¥n  Spearman (残差):")
    print(f"    base: rho_s = {rho_sp_base:+.4f}, p = {p_sp_base:.3e}")
    print(f"    +ratio: rho_s = {rho_sp_r:+.4f}, p = {p_sp_r:.3e}")

    # 効果量 (Cohen's f^2)
    R2_base = rho_base**2
    R2_full = rho_r**2
    # partial R^2 の変化
    delta_R2 = R2_base - R2_full
    print(f"¥n  効果量:")
    print(f"    partial R^2 base: {R2_base:.4f}")
    print(f"    partial R^2 +ratio: {R2_full:.4f}")
    print(f"    ΔR^2: {delta_R2:.4f}")

    # ============================================================
    # Test 2: rmax/hR サブセット
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 2: rmax/hR サブセットでの ratio 効果")
    print("=" * 50)

    for rcut in [3, 5, 8, 10]:
        mc = m &amp;amp; (rmax_hR &amp;gt; rcut)
        if mc.sum() &amp;lt; 15:
            continue
        rb, pb = partial_corr(log_hR[mc], log_gc[mc], [log_vf[mc], log_x[mc]])
        rr, pr = partial_corr(log_hR[mc], log_gc[mc],
                              [log_vf[mc], log_x[mc], log_ratio[mc]])
        delta = abs(rb) - abs(rr)
        print(f"  rmax/hR &amp;gt; {rcut:2d} (N={mc.sum():3d}): "
              f"base={rb:+.4f}, +ratio={rr:+.4f}, Δ|ρ|={delta:+.4f}, p={pr:.3e}")

    # ============================================================
    # Test 3: ブートストラップ信頼区間
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 3: ブートストラップ信頼区間 (5000回)")
    print("=" * 50)

    np.random.seed(42)
    n_boot = 5000
    n_samp = m.sum()
    idx_all = np.where(m)[0]

    rho_boot_base = []
    rho_boot_ratio = []
    delta_boot = []

    for _ in range(n_boot):
        idx = np.random.choice(n_samp, n_samp, replace=True)
        try:
            rb, _ = partial_corr(log_hR[m][idx], log_gc[m][idx],
                                 [log_vf[m][idx], log_x[m][idx]])
            rr, _ = partial_corr(log_hR[m][idx], log_gc[m][idx],
                                 [log_vf[m][idx], log_x[m][idx], log_ratio[m][idx]])
            rho_boot_base.append(rb)
```

```
            rho_boot_ratio.append(rr)
            delta_boot.append(abs(rb) - abs(rr))
        except:
            pass

    rho_boot_base = np.array(rho_boot_base)
    rho_boot_ratio = np.array(rho_boot_ratio)
    delta_boot = np.array(delta_boot)

    print(f"  成功: {len(delta_boot)}/{n_boot}")
    print(f"¥n  rho(base) の 95% CI: [{np.percentile(rho_boot_base, 2.5):.4f}, "
          f"{np.percentile(rho_boot_base, 97.5):.4f}]")
    print(f"  rho(+ratio) の 95% CI: [{np.percentile(rho_boot_ratio, 2.5):.4f}, "
          f"{np.percentile(rho_boot_ratio, 97.5):.4f}]")
    print(f"¥n  Δ|rho| の分布:")
    print(f"    mean = {np.mean(delta_boot):.4f}")
    print(f"    median = {np.median(delta_boot):.4f}")
    print(f"    95% CI = [{np.percentile(delta_boot, 2.5):.4f}, "
          f"{np.percentile(delta_boot, 97.5):.4f}]")
    print(f"    P(Δ &gt; 0) = {np.mean(delta_boot &gt; 0):.3f}")

    if np.percentile(delta_boot, 2.5) &gt; 0:
        print("  -&gt; 95% CI が正の領域にある -&gt; ratio の改善は有意")
    elif np.mean(delta_boot &gt; 0) &gt; 0.8:
        print("  -&gt; P(Δ&gt;0)&gt;80% -&gt; ratio の改善は probable だが決定的でない")
    else:
        print("  -&gt; 改善はノイズの範囲")

    # ============================================================
    # Test 4: ratio とhR, x_outer の独立性
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 4: ratio の独立性")
    print("=" * 50)

    rho_rh, p_rh = pearsonr(log_hR[m], log_ratio[m])
    rho_rx, p_rx = pearsonr(log_x[m], log_ratio[m])
    rho_rv, p_rv = pearsonr(log_vf[m], log_ratio[m])
    rho_rhx, p_rhx = partial_corr(log_hR[m], log_ratio[m], [log_x[m]])
    rho_rxh, p_rxh = partial_corr(log_x[m], log_ratio[m], [log_hR[m]])

    print(f"  ratio vs hR:     rho = {rho_rh:+.4f}, p = {p_rh:.3e}")
    print(f"  ratio vs x:      rho = {rho_rx:+.4f}, p = {p_rx:.3e}")
    print(f"  ratio vs vflat:  rho = {rho_rv:+.4f}, p = {p_rv:.3e}")
    print(f"  ratio vs hR|x:   rho = {rho_rhx:+.4f}, p = {p_rhx:.3e}")
    print(f"  ratio vs x|hR:   rho = {rho_rxh:+.4f}, p = {p_rxh:.3e}")

    # ratio が gc と独立に相関するか (hR, vflat 統制後)
    rho_rgc, p_rgc = partial_corr(log_ratio[m], log_gc[m],
                                  [log_hR[m], log_vf[m]])
    print(f"¥n  ratio vs gc | hR, vflat: rho = {rho_rgc:+.4f}, p = {p_rgc:.3e}")

    rho_rgcx, p_rgcx = partial_corr(log_ratio[m], log_gc[m],
                                    [log_hR[m], log_vf[m], log_x[m]])
    print(f"  ratio vs gc | hR, vflat, x: rho = {rho_rgcx:+.4f}, p = {p_rgcx:.3e}")

    # ============================================================
    # Test 5: ratio を駆動する物理量
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 5: ratio を駆動する物理量")
    print("=" * 50)

    # E_grad/E_local = int(dε/dr)^2rdr / intε^2rdr
    # 次元的に: ratio ~ 1/L^2 (Lは特徴的長さスケール)
    # 何が ratio を駆動するか

    # ratio vs hR のスケーリング
    sl = linregress(log_hR[m], log_ratio[m])
    print(f"  ratio vs hR: slope = {sl.slope:.4f} +/- {sl.stderr:.4f}")
    print(f"  (理論予測: slope ~ -1 to -2)")

    # ratio vs gN_peak/gc_deep
    gN_peak = []
    for rec in records:
        rm = load_rotmod(rec.get("name", ""), ROTMOD_DIR) if "name" in rec else None
        if rm is None:
            # name を使えないので代替計算
            gN_peak.append(np.nan)
        else:
            gN_i = compute_gN(rm[0], rm[3], rm[4], rm[5])
            gN_peak.append(np.max(gN_i))
    # name がないので records から再計算
    # 簡略化: E_local/hR^2 ~ &lt;ε^2&gt; ~ &lt;gN/gc&gt; ~ x_average
    # ratio ~ 1/hR^2 x (勾配形状) -&gt; x と hR の組み合わせ

    print(f"¥n  ratio の分散分解:")
```

```
    # ratio = f(hR, x, 形状パラメータ)
    X = np.column_stack([log_hR[m], log_x[m], np.ones(m.sum())])
    coef, _, _, _ = lstsq(X, log_ratio[m], rcond=None)
    y_pred = X @ coef
    R2_hx = 1 - np.sum((log_ratio[m] - y_pred)**2) / np.sum((log_ratio[m] - np.mean(log_ratio[m]))**2)
    print(f"  R^2(ratio ~ hR + x): {R2_hx:.4f}")
    print(f"  ratio ~ hR^{coef[0]:.3f} x x^{coef[1]:.3f}")
    print(f"  -&gt; ratio の {R2_hx*100:.1f}% は hR と x で説明される")
    print(f"  -&gt; 残り {(1-R2_hx)*100:.1f}% が「独立な物理情報」")

    # ============================================================
    # Test 6: gc残差 vs ratio の精密評価
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 6: gc残差(obs-deep) vs ratio")
    print("=" * 50)

    resid = log_gc[m] - log_gcd[m]

    rho_res_r, p_res_r = pearsonr(log_ratio[m], resid)
    print(f"  raw: rho(resid, ratio) = {rho_res_r:+.4f}, p = {p_res_r:.3e}")

    rho_res_rx, p_res_rx = partial_corr(log_ratio[m], resid, [log_x[m]])
    print(f"  |x: rho(resid, ratio | x) = {rho_res_rx:+.4f}, p = {p_res_rx:.3e}")

    rho_res_rxh, p_res_rxh = partial_corr(log_ratio[m], resid,
                                          [log_x[m], log_hR[m]])
    print(f"  |x,hR: rho(resid, ratio | x,hR) = {rho_res_rxh:+.4f}, p = {p_res_rxh:.3e}")

    rho_res_full, p_res_full = partial_corr(log_ratio[m], resid,
                                            [log_x[m], log_hR[m], log_vf[m]])
    print(f"  |x,hR,vf: rho(resid, ratio | x,hR,vf) = {rho_res_full:+.4f}, p = {p_res_full:.3e}")

    # ============================================================
    # Test 7: 勾配形状パラメータの独立寄与
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 7: 勾配形状パラメータの独立寄与")
    print("=" * 50)

    shape_params = {
        "grad_peak_frac": grad_pf,
        "grad_asymm": grad_as,
        "grad_peak_r/hR": grad_pr,
        "grad_width/hR": grad_wd,
    }

    for name, sp_arr in shape_params.items():
        v = m &amp; np.isfinite(sp_arr) &amp; (sp_arr &gt; 0)
        if v.sum() &lt; 20:
            continue
        log_sp = np.log10(sp_arr[v] + 1e-10)

        # hR偏相関への寄与
        rho_s, p_s = partial_corr(log_hR[v], log_gc[v],
                                  [log_vf[v], log_x[v], log_sp])
        delta = abs(rho_s) - abs(rho_base)
        # gc残差との相関
        resid_v = log_gc[v] - log_gcd[v]
        rho_sr, p_sr = partial_corr(log_sp, resid_v,
                                    [log_x[v], log_hR[v]])
        print(f"  {name:20s}: hR偏相関 rho={rho_s:+.4f} (Δ={delta:+.4f}), "
              f"resid|x,hR: rho={rho_sr:+.4f}")

    # ============================================================
    # Test 8: Permutation test (正確なp値)
    # ============================================================
    print("¥n" + "=" * 50)
    print("Test 8: Permutation test (2000回)")
    print("=" * 50)

    # ratio をシャッフルして偏相関の分布を作り、
    # 実測の偏相関改善が偶然で起きる確率を計算
    np.random.seed(777)
    n_perm = 2000

    # 実測の改善量
    real_delta = abs(rho_base) - abs(rho_r)

    perm_deltas = []
    for _ in range(n_perm):
        ratio_shuf = log_ratio[m].copy()
        np.random.shuffle(ratio_shuf)
        try:
            rho_perm, _ = partial_corr(log_hR[m], log_gc[m],
                                       base + [ratio_shuf])
            perm_delta = abs(rho_base) - abs(rho_perm)
```

```
            perm_deltas.append(perm_delta)
        except:
            pass

    perm_deltas = np.array(perm_deltas)
    p_perm = np.mean(perm_deltas &gt;= real_delta)

    print(f"  実測 Δ|rho| = {real_delta:.4f}")
    print(f"  Permutation: mean = {np.mean(perm_deltas):.4f}, "
          f"std = {np.std(perm_deltas):.4f}")
    print(f"  Permutation p-value = {p_perm:.4f}")
    print(f"  95th percentile = {np.percentile(perm_deltas, 95):.4f}")

    if p_perm &lt; 0.05:
        print("  -&gt; p &lt; 0.05: ratio の改善は有意 [*]")
    elif p_perm &lt; 0.10:
        print("  -&gt; 0.05 &lt; p &lt; 0.10: marginally significant")
    else:
        print("  -&gt; p &gt;= 0.10: ratio の改善はノイズ")

    # ============================================================
    # SUMMARY
    # ============================================================
    print("¥n" + "=" * 70)
    print("SUMMARY: E_grad/E_local の最終評価")
    print("=" * 70)

    print(f"¥n  偏相関:")
    print(f"    base |vflat,x:       rho = {rho_base:+.4f}")
    print(f"    +ratio:              rho = {rho_r:+.4f}")
    print(f"    Δ|rho| = {real_delta:+.4f} ({delta_pct:.1f}% of total)")
    print(f"    Pearson p = {p_r:.3e}")
    print(f"    Permutation p = {p_perm:.4f}")

    print(f"¥n  ブートストラップ:")
    print(f"    P(Δ&gt;0) = {np.mean(delta_boot &gt; 0):.3f}")
    print(f"    95% CI of Δ = [{np.percentile(delta_boot, 2.5):.4f}, "
          f"{np.percentile(delta_boot, 97.5):.4f}]")

    print(f"¥n  独立性:")
    print(f"    ratio の {R2_hx*100:.1f}% は hR+x で説明")
    print(f"    ratio vs gc | hR,vflat,x: rho = {rho_rgcx:+.4f}")

    print(f"¥n  gc残差との相関:")
    print(f"    ratio vs resid | x,hR,vf: rho = {rho_res_full:+.4f}")

    print(f"¥n  ■ 最終判定:")
    if p_perm &lt; 0.05 and np.mean(delta_boot &gt; 0) &gt; 0.9:
        print("  [*] E_grad/E_local は統計的に有意な物理的効果")
        print(f"  -&gt; hR偏相関の追加 {delta_pct:.0f}% を説明")
    elif p_perm &lt; 0.10 or np.mean(delta_boot &gt; 0) &gt; 0.7:
        print("  △ E_grad/E_local は示唆的だが決定的でない")
        print(f"  -&gt; 方向は正しいが統計力不足 (N={m.sum()})")
    else:
        print("  x E_grad/E_local の効果はノイズ")
        print("  -&gt; 勾配エネルギー仮説は支持されない")

    print(f"¥n  ■ hR偏相関の最終分解:")
    print(f"    x_outer (MOND遷移): 69%")
    if p_perm &lt; 0.10:
        print(f"    E_grad/E_local (勾配比): {delta_pct:.0f}% (p_perm={p_perm:.3f})")
        print(f"    残差: {100-69-delta_pct:.0f}%")
    else:
        print(f"    E_grad/E_local: 効果不明 ({delta_pct:.0f}%, p={p_perm:.3f})")
        print(f"    残差: 31%")


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

付録: 共通ユーティリティ関数

全スクリプトで共通して使用される関数:

関数	目的	使用スクリプト
load_gc_table()	sparc_gc.csv の読み込み	全10本
load_rotmod()	Rotmod_LTG/*.dat の読み込み	全10本
compute_gN()	g_N(r) プロファイル計算 [m/s^2]	全10本
partial_corr()	偏相関（統制変数付きPearson）	#4-10
fit_gc_deep()	深MONDフィット gc 推定	#3,6,7,8,9,10
fit_gc_full()	完全MONDフィット gc 推定	#3,6
compute_strain()	epsilon(r) と歪みエネルギー計算	#8,9,10

データフロー:

sparc_gc.csv (gc, vflat, hR) + Rotmod_LTG/*.dat (r, Vobs, Vgas, Vdisk, Vbul)
-> compute_gN() -> fit_gc_deep/full() -> partial_corr() -> 段階的分解

単位系:

- gc: m/s^2 (a0単位の場合はスクリプト内で自動変換)
- r: kpc -> m (1 kpc = 3.0857 x 10^{19} m)
- V: km/s -> m/s (x 10^3)
- a0 = 1.2 x 10^{-10} m/s^2
- Yd = 0.5 (デフォルト)、Yb = 0.7 (デフォルト)