

# 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  $\max[g_{\text{eff}}(r)/g_c] = 2(\sqrt{c}-1)$  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:  $R^2 = -1.11$ ,  $\alpha = 0.219$ ,  $gc_{\text{pred}}/gc_{\text{obs}} = 2.74$ ,  $r_c/hR = 0.93$  (記録)。

## ソースコード

(478 行)

```
1 | #!/usr/bin/env python3
2 | """
3 | sparc_P2b_critical.py
4 | =====
5 | P2b検証: 自己無撞着臨界変形条件からgcを予測する
6 |
7 | 物理的仮説:
8 |   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。
9 |
10 |  境界条件:  $\epsilon_c = 1 - \sqrt{c}$  で  $d^2U/d\epsilon^2 = 0$ 
11 |   このとき  $g_{\text{eff}}(r_c)/g_c = 2(\sqrt{c} - 1)$ 
12 |
13 |  自己無撞着版:  $g_{\text{eff}}$  は MOND 補間で  $g_N$  と  $g_c = c \cdot a_0$  から決まる
14 |   ->  $\max_r [g_{\text{eff}}(r)/g_c] = 2(\sqrt{c} - 1)$  が  $c$  を決定
15 |
16 | テスト:
17 |   Test 1: 各銀河で  $c$  を数値的に解き、 $gc_{\text{pred}} = c \cdot a_0$  vs  $gc_{\text{obs}}$  を比較
18 |   Test 2:  $gc_{\text{pred}}$  が幾何平均法則  $gc \sim \sqrt{a_0 \cdot v_{\text{flat}}^2 / hR}$  を再現するか
19 |   Test 3:  $gc_{\text{pred}}$  vs  $gc_{\text{obs}}$  の残差構造 ( $hR$ ,  $v_{\text{flat}}$ ,  $\Sigma_{\text{bar}}$  依存性)
20 |   Test 4:  $r_c/hR$  の分布 -> 有限ディスク効果の大きさ
21 |
22 | 実行: uv run --with scipy --with matplotlib python sparc_P2b_critical.py
23 | """
24 |
25 | import numpy as np
26 | from scipy.optimize import brentq
27 | from scipy.interpolate import interp1d
28 | from pathlib import Path
29 | import csv
30 | import sys
31 |
32 | # =====
33 | # 定数・パス
34 | # =====
35 | a0 = 1.2e-10          # m/s^2 (MOND加速度スケール)
36 | GN = 4.30091e-3      # (km/s)^2 pc / Msun
37 | Yd_default = 0.5     # 恒星質量光度比 (ディスク)
38 | Yb_default = 0.7     # 恒星質量光度比 (バルジ)
39 | kpc_to_pc = 1000.0
40 | kms2_to_ms2 = 1.0e6 # (km/s)^2 -> (m/s)^2
41 |
42 | # データパス (ローカルWindows環境)
43 | BASE = Path(r"D:\ドキュメント\アントロピー\新膜宇宙論\これまでの軌跡\パイソン")
44 | ROTMOD_DIR = BASE / "Rotmod_LTG"
45 | GC_CSV = BASE / "sparc_gc.csv"
46 |
47 |
48 | # =====
49 | # ユーティリティ
50 | # =====
51 | def load_gc_table(path):
52 |     """sparc_gc.csv を読み込み {galaxy_name: dict} を返す"""
53 |     data = {}
54 |     with open(path, "r") as f:
55 |         reader = csv.DictReader(f)
56 |         for row in reader:
57 |             name = row["Galaxy"].strip()
58 |             data[name] = {
59 |                 "gc_obs": float(row["gc"]),          # m/s^2 単位想定
60 |                 "vflat": float(row["Vflat"]),       # km/s
61 |                 "hR": float(row["hR"]),             # kpc
62 |             }
```

```

63|     return data
64|
65|
66| def load_rotmod(galaxy_name, rotmod_dir):
67|     """Rotmod_LTG から回転曲線を読み込む
68|     戻り値: r_kpc, Vobs, Vgas, Vdisk, Vbul (全て km/s, rはkpc)
69|     """
70|     fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
71|     if not fpath.exists():
72|         return None
73|
74|     r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
75|     with open(fpath, "r") as f:
76|         for line in f:
77|             line = line.strip()
78|             if not line or line.startswith("#"):
79|                 continue
80|             parts = line.split()
81|             if len(parts) < 6:
82|                 continue
83|             r.append(float(parts[0]))
84|             vobs.append(float(parts[1]))
85|             errv.append(float(parts[2]))
86|             vgas.append(float(parts[3]))
87|             vdisk.append(float(parts[4]))
88|             vbul.append(float(parts[5]))
89|
90|     return (np.array(r), np.array(vobs), np.array(vgas),
91|            np.array(vdisk), np.array(vbul))
92|
93|
94| def compute_gN_profile(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
95|     """バリエーション重力加速度プロファイル g_N(r) を m/s^2 で返す
96|
97|     V の符号を保存: g = |V|*V*Y / r (符号付き)
98|     最終的にgasは符号なし(Vgas^2は常に正寄与)
99|     """
100|    r_m = r_kpc * kpc_to_pc * 3.0857e16 # kpc -> m (1 pc = 3.0857e16 m)
101|
102|    # V^2 -> 加速度 (km/s)^2 / kpc -> m/s^2
103|    # a = V^2 / r where V in km/s, r in kpc
104|    # a [m/s^2] = V^2 [km^2/s^2] * 1e6 / (r [kpc] * 3.0857e19)
105|
106|    conv = kms2_to_ms2 / (r_kpc * 3.0857e19) # per-point conversion
107|
108|    gN = np.abs(Vgas) * Vgas * conv *
109|         + Yd * np.abs(Vdisk) * Vdisk * conv *
110|         + Yb * np.abs(Vbul) * Vbul * conv
111|
112|    return np.abs(gN) # 大きさのみ
113|
114|
115| def g_eff_over_gc(x):
116|     """g_eff/g_c as function of x = g_N/g_c
117|     MOND simple interpolation: mu(y)=y/(1+y)
118|     -> g_eff/g_c = [x + sqrt(x^2+4x)] / 2
119|     """
120|    return (x + np.sqrt(x**2 + 4*x)) / 2
121|
122|
123| def critical_RHS(c):
124|     """臨界条件の右辺: 2(sqrt(c) - 1)
125|     c > 1 でのみ正値
126|     """
127|    return 2.0 * (np.sqrt(c) - 1.0)
128|
129|
130| # =====
131| # メイン: 各銀河で c を解く
132| # =====
133| def solve_c_for_galaxy(gN_profile, r_kpc, vflat_kms,
134|                        c_min=1.001, c_max=50.0):
135|     """
136|     自己無撞着臨界条件:
137|     max_r [g_eff(r)/g_c] = 2(sqrt(c) - 1)
138|
139|     ここで g_c = c*a_0, x(r) = g_N(r)/g_c
140|
141|     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)。
142|
143|     戻り値: c_sol, gc_pred, r_c_kpc, converged
144|     """
145|    # g_N の最大値とその位置
146|    idx_max = np.argmax(gN_profile)
147|    gN_max = gN_profile[idx_max]
148|    r_max = r_kpc[idx_max]
149|

```

```

150|     if gN_max &lt;= 0:
151|         return np.nan, np.nan, np.nan, False
152|
153|     def residual(c):
154|         gc = c * a0
155|         x_max = gN_max / gc
156|         lhs = g_eff_over_gc(x_max)
157|         rhs = critical_RHS(c)
158|         return lhs - rhs
159|
160|     # 存在チェック
161|     try:
162|         r_lo = residual(c_min)
163|         r_hi = residual(c_max)
164|     except:
165|         return np.nan, np.nan, np.nan, False
166|
167|     if r_lo * r_hi &gt; 0:
168|         # 符号が変わらない -&gt; 解なし or 範囲外
169|         # c_max を広げて再試行
170|         for c_try in [100, 500, 2000]:
171|             try:
172|                 if residual(c_min) * residual(c_try) &lt; 0:
173|                     c_max = c_try
174|                     break
175|             except:
176|                 continue
177|         else:
178|             return np.nan, np.nan, np.nan, False
179|
180|     try:
181|         c_sol = brentq(residual, c_min, c_max, xtol=1e-8)
182|     except:
183|         return np.nan, np.nan, np.nan, False
184|
185|     gc_pred = c_sol * a0
186|
187|     # r_c の計算: g_eff(r_c)/g_c = 2(sqrtc-1) を満たす最大の r
188|     # (g_N の減少域で閾値を横切る点)
189|     gc = gc_pred
190|     threshold = critical_RHS(c_sol)
191|
192|     if threshold &lt;= 0:
193|         return c_sol, gc_pred, r_max, True
194|
195|     x_arr = gN_profile / gc
196|     geff_gc_arr = g_eff_over_gc(x_arr)
197|
198|     # 外側から探して閾値を超える最初の点
199|     r_c_kpc = r_max # デフォルト
200|     for i in range(len(r_kpc)-1, 0, -1):
201|         if geff_gc_arr[i] &gt;= threshold:
202|             r_c_kpc = r_kpc[i]
203|             break
204|
205|     return c_sol, gc_pred, r_c_kpc, True
206|
207|
208| def main():
209|     print("=" * 70)
210|     print("P2b: 自己無撞着臨界変形条件 -- gc予測検証")
211|     print("=" * 70)
212|
213|     # データ読み込み
214|     if not GC_CSV.exists():
215|         print(f"ERROR: {GC_CSV} not found")
216|         sys.exit(1)
217|
218|     gc_table = load_gc_table(GC_CSV)
219|     print(f"sparc_gc.csv: {len(gc_table)} galaxies loaded")
220|
221|     # 各銀河を処理
222|     results = []
223|     skipped = 0
224|
225|     for gal_name, gal_info in gc_table.items():
226|         rotmod = load_rotmod(gal_name, ROTMOD_DIR)
227|         if rotmod is None:
228|             skipped += 1
229|             continue
230|
231|         r_kpc, Vobs, Vgas, Vdisk, Vbul = rotmod
232|
233|         if len(r_kpc) &lt; 5:
234|             skipped += 1
235|             continue
236|

```

```

237| # g_N プロファイル
238| gN = compute_gN_profile(r_kpc, Vgas, Vdisk, Vbul,
239|                       Yd=Yd_default, Yb=Yb_default)
240|
241| # c を解く
242| c_sol, gc_pred, r_c, converged = solve_c_for_galaxy(
243|     gN, r_kpc, gal_info["vflat"])
244|
245| if not converged:
246|     skipped += 1
247|     continue
248|
249| results.append({
250|     "name": gal_name,
251|     "gc_obs": gal_info["gc_obs"],
252|     "gc_pred": gc_pred,
253|     "c_sol": c_sol,
254|     "r_c_kpc": r_c,
255|     "hR": gal_info["hR"],
256|     "vflat": gal_info["vflat"],
257|     "gN_max": np.max(gN),
258|     "r_c_over_hR": r_c / gal_info["hR"] if gal_info["hR"] > 0 else np.nan,
259| })
260|
261| N = len(results)
262| print(f"Converged: {N} galaxies, Skipped: {skipped}")
263|
264| if N < 10:
265|     print("Too few galaxies for analysis.")
266|     sys.exit(1)
267|
268| # NumPy配列化
269| gc_obs = np.array([r["gc_obs"] for r in results])
270| gc_pred = np.array([r["gc_pred"] for r in results])
271| c_arr = np.array([r["c_sol"] for r in results])
272| hR_arr = np.array([r["hR"] for r in results])
273| vflat_arr = np.array([r["vflat"] for r in results])
274| rc_hR = np.array([r["r_c_over_hR"] for r in results])
275| gN_max_arr = np.array([r["gN_max"] for r in results])
276|
277| # =====
278| # Test 1: gc_pred vs gc_obs
279| # =====
280| print("#n" + "=" * 50)
281| print("Test 1: gc_pred vs gc_obs")
282| print("=" * 50)
283|
284| log_obs = np.log10(gc_obs)
285| log_pred = np.log10(gc_pred)
286|
287| mask = np.isfinite(log_obs) & np.isfinite(log_pred)
288| log_obs_f = log_obs[mask]
289| log_pred_f = log_pred[mask]
290|
291| from scipy.stats import pearsonr, spearmanr
292| rho_p, p_p = pearsonr(log_pred_f, log_obs_f)
293| rho_s, p_s = spearmanr(log_pred_f, log_obs_f)
294|
295| ratio = gc_pred[mask] / gc_obs[mask]
296|
297| print(f" N = {mask.sum()}")
298| print(f" Pearson  $\rho$  = {rho_p:.4f}, p = {p_p:.2e}")
299| print(f" Spearman  $\rho$  = {rho_s:.4f}, p = {p_s:.2e}")
300| print(f" gc_pred/gc_obs: median = {np.median(ratio):.3f}, "
301|       f"mean = {np.mean(ratio):.3f}, std = {np.std(ratio):.3f}")
302| print(f" log10 offset: median = {np.median(log_pred_f - log_obs_f):.3f}")
303|
304| # R^2 (log-log)
305| ss_res = np.sum((log_obs_f - log_pred_f)**2)
306| ss_tot = np.sum((log_obs_f - np.mean(log_obs_f))**2)
307| R2_direct = 1 - ss_res / ss_tot if ss_tot > 0 else np.nan
308| print(f" R^2(direct) = {R2_direct:.4f}")
309|
310| # 線形回帰 log(gc_obs) vs log(gc_pred)
311| from numpy.polynomial.polynomial import polyfit
312| coef = np.polyfit(log_pred_f, log_obs_f, 1)
313| print(f" Linear fit: log(gc_obs) = {coef[0]:.3f} x log(gc_pred) + {coef[1]:.3f}")
314|
315| # =====
316| # Test 2: gc_pred vs 幾何平均法則
317| # =====
318| print("#n" + "=" * 50)
319| print("Test 2: gc_pred vs geometric mean law")
320| print("=" * 50)
321|
322| GS0_proxy = (vflat_arr**2) / (hR_arr * kpc_to_pc * 3.0857e16) # m/s^2
323| gc_geom = np.sqrt(a0 * GS0_proxy) # 幾何平均法則 ( $\eta=1$ )

```

```

324|
325| log_geom = np.log10(gc_geom)
326| log_pred_all = np.log10(gc_pred)
327|
328| mask2 = np.isfinite(log_geom) & np.isfinite(log_pred_all)
329| rho2, p2 = pearsonr(log_geom[mask2], log_pred_all[mask2])
330| print(f" ρ (gc_pred, gc_geom) = {rho2:.4f}, p = {p2:.2e}")
331|
332| # gc_pred のスケールリング: log(gc_pred) vs log(GS0_proxy)
333| log_GS0 = np.log10(GS0_proxy)
334| mask3 = np.isfinite(log_GS0) & np.isfinite(log_pred_all)
335| slope_fit = np.polyfit(log_GS0[mask3], log_pred_all[mask3], 1)
336| print(f" log(gc_pred) vs log(vflat^2/hR): slope = {slope_fit[0]:.4f} "
337|       f"(期待: 0.5)")
338|
339| # =====
340| # Test 3: 残差構造
341| # =====
342| print("\n" + "=" * 50)
343| print("Test 3: 残差構造 Δ log(gc) = log(gc_pred) - log(gc_obs)")
344| print("=" * 50)
345|
346| delta = log_pred_f - log_obs_f
347|
348| # vs hR
349| log_hR = np.log10(hR_arr[mask])
350| rho_hR, p_hR = pearsonr(log_hR, delta)
351| print(f" Δ vs log(hR): ρ = {rho_hR:.4f}, p = {p_hR:.3e}")
352|
353| # vs vflat
354| log_vf = np.log10(vflat_arr[mask])
355| rho_vf, p_vf = pearsonr(log_vf, delta)
356| print(f" Δ vs log(vflat): ρ = {rho_vf:.4f}, p = {p_vf:.3e}")
357|
358| # vs gN_max
359| log_gNm = np.log10(gN_max_arr[mask])
360| rho_gN, p_gN = pearsonr(log_gNm, delta)
361| print(f" Δ vs log(gN_max): ρ = {rho_gN:.4f}, p = {p_gN:.3e}")
362|
363| # =====
364| # Test 4: r_c / hR 分布
365| # =====
366| print("\n" + "=" * 50)
367| print("Test 4: r_c / hR 分布")
368| print("=" * 50)
369|
370| rc_hR_f = rc_hR[np.isfinite(rc_hR)]
371| print(f" N = {len(rc_hR_f)}")
372| print(f" median(r_c/hR) = {np.median(rc_hR_f):.2f}")
373| print(f" mean(r_c/hR) = {np.mean(rc_hR_f):.2f}")
374| print(f" std(r_c/hR) = {np.std(rc_hR_f):.2f}")
375| print(f" min = {np.min(rc_hR_f):.2f}, max = {np.max(rc_hR_f):.2f}")
376| print(f" fraction r_c/hR &lt; 3: {np.mean(rc_hR_f &lt; 3):.2%}")
377| print(f" fraction r_c/hR &lt; 5: {np.mean(rc_hR_f &lt; 5):.2%}")
378|
379| # =====
380| # Test 5: c の分布
381| # =====
382| print("\n" + "=" * 50)
383| print("Test 5: c の分布")
384| print("=" * 50)
385|
386| print(f" median(c) = {np.median(c_arr):.3f}")
387| print(f" mean(c) = {np.mean(c_arr):.3f}")
388| print(f" std(c) = {np.std(c_arr):.3f}")
389| print(f" min = {np.min(c_arr):.3f}, max = {np.max(c_arr):.3f}")
390| print(f" gc_pred range: [{np.min(gc_pred):.2e}, {np.max(gc_pred):.2e}] m/s^2")
391| print(f" gc_obs range: [{np.min(gc_obs):.2e}, {np.max(gc_obs):.2e}] m/s^2")
392|
393| # =====
394| # プロット
395| # =====
396| try:
397|     import matplotlib
398|     matplotlib.use("Agg")
399|     import matplotlib.pyplot as plt
400|
401|     fig, axes = plt.subplots(2, 2, figsize=(12, 10))
402|     fig.suptitle("P2b: Self-consistent Critical Deformation", fontsize=14)
403|
404|     # (a) gc_pred vs gc_obs
405|     ax = axes[0, 0]
406|     ax.scatter(log_obs_f, log_pred_f, s=8, alpha=0.5)
407|     lim = [min(log_obs_f.min(), log_pred_f.min()) - 0.1,
408|           max(log_obs_f.max(), log_pred_f.max()) + 0.1]
409|     ax.plot(lim, lim, "k--", lw=1, label="1:1")
410|     ax.set_xlabel("log_1_0(gc_obs)")

```

```

411|     ax.set_ylabel("log_1_0(gc_pred)")
412|     ax.set_title(f"(a) gc_pred vs gc_obs R^2={R2_direct:.3f}")
413|     ax.legend()
414|     ax.set_aspect("equal")
415|
416|     # (b) gc_pred vs geometric mean law
417|     ax = axes[0, 1]
418|     ax.scatter(log_geom[mask2], log_pred_all[mask2], s=8, alpha=0.5)
419|     ax.plot(lim, lim, "k--", lw=1)
420|     ax.set_xlabel("log_1_0(gc_geom = sqrt(a_0*vflat^2/hR))")
421|     ax.set_ylabel("log_1_0(gc_pred)")
422|     ax.set_title(f"(b) gc_pred vs geometric mean")
423|
424|     # (c) 残差 vs hR
425|     ax = axes[1, 0]
426|     ax.scatter(log_hR, delta, s=8, alpha=0.5)
427|     ax.axhline(0, color="k", ls="--", lw=1)
428|     ax.set_xlabel("log_1_0(hR [kpc])")
429|     ax.set_ylabel("Δ log_1_0(gc)")
430|     ax.set_title(f"(c) Residual vs hR ρ={rho_hR:.3f}")
431|
432|     # (d) r_c/hR 分布
433|     ax = axes[1, 1]
434|     ax.hist(rc_hR_f, bins=30, edgecolor="black", alpha=0.7)
435|     ax.axvline(np.median(rc_hR_f), color="red", ls="--",
436|               label=f"median={np.median(rc_hR_f):.1f}")
437|     ax.set_xlabel("r_c / hR")
438|     ax.set_ylabel("count")
439|     ax.set_title("(d) Critical radius distribution")
440|     ax.legend()
441|
442|     plt.tight_layout()
443|     out_fig = BASE / "P2b_critical_results.png"
444|     plt.savefig(out_fig, dpi=150)
445|     print(f"Figure saved: {out_fig}")
446| except Exception as e:
447|     print(f"Plot skipped: {e}")
448|
449| # =====
450| # サマリー
451| # =====
452| print("\n" + "=" * 70)
453| print("SUMMARY")
454| print("=" * 70)
455| print(f" P2b条件で解いた gc_pred と gc_obs の相関: ρ = {rho_p:.4f}")
456| print(f" R^2(direct 1:1) = {R2_direct:.4f}")
457| print(f" gc_pred のスケール指数: α = {slope_fit[0]:.4f} (期待 0.5)")
458| print(f" r_c/hR 中央値: {np.median(rc_hR_f):.2f}")
459|
460| if abs(slope_fit[0] - 0.5) < 0.1:
461|     print(f" α ≈ 0.5: 幾何平均法則と整合 [OK]")
462| else:
463|     print(f" α = {slope_fit[0]:.3f}: 幾何平均法則から乖離")
464|
465| if R2_direct > 0.3:
466|     print(f" R^2 = {R2_direct:.3f}: P2bは gc を有意に予測 [OK]")
467| else:
468|     print(f" R^2 = {R2_direct:.3f}: P2bの予測力は弱い")
469|
470| if np.median(rc_hR_f) < 5:
471|     print(f" r_c/hR ≈ {np.median(rc_hR_f):.1f}: 有限ディスク効果が支配的 [OK]")
472| else:
473|     print(f" r_c/hR ≈ {np.median(rc_hR_f):.1f}: 漸近極限に近い")
474|
475|
476| if __name__ == "__main__":
477|     main()
478|

```

## 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、 $\delta = \beta \times hR$  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、 $E_{\text{kink}} R^2 = -0.507$ 。

### ソースコード

(611 行)

```
1| #!/usr/bin/env python3
2| """
3| sparc_P3_domainwall.py
4| =====
5| P3検証: Z2 SSBドメインウォール仮説
6|
7| 物理的仮説:
8|  $\epsilon(r)$ 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、
9| その構造的拘束が $c$ を決定する。
10|
11|  $\epsilon(r) = [-x + \sqrt{(x+2)^2 - 4c}] / 2$ ,  $x(r) = g_N(r)/(c \cdot a_0)$ 
12| 存在条件:  $x \geq x_{\min} = 2(\sqrt{c} - 1)$  ( $c \geq 1$ )
13|
14| テスト:
15| Test 1:  $\epsilon(r)$ プロファイルの計算とキンク幅 $\delta$ の測定
16| Test 2:  $\delta/hR$ の普遍性チェック（全銀河で一定か?）
17| Test 3: キンクエネルギー  $E_{\text{kink}} = \int (d\epsilon/dr)^2 r dr$ の銀河依存性
18| Test 4:  $E_{\text{kink}}$  定数の拘束から $c$ を予測 -&gt;  $gc_{\text{pred}}$  vs  $gc_{\text{obs}}$ 
19| Test 5: 代替拘束  $\delta = \beta \cdot hR$  から $c$ を予測 -&gt;  $gc_{\text{pred}}$  vs  $gc_{\text{obs}}$ 
20|
21| 実行: uv run --with scipy --with matplotlib python sparc_P3_domainwall.py
22| """
23|
24| import numpy as np
25| from scipy.optimize import brentq
26| from scipy.interpolate import interp1d
27| from scipy.stats import pearsonr, spearmanr
28| from pathlib import Path
29| import csv
30| import sys
31|
32| # =====
33| # 定数・パス
34| # =====
35| a0 = 1.2e-10 # m/s^2
36| G_N = 4.30091e-3 # (km/s)^2 pc / M_sun
37| Yd_default = 0.5
38| Yb_default = 0.7
39| kpc_to_m = 3.0857e19 # 1 kpc in meters
40|
41| BASE = Path(r"D:\ドキュメント\アントロピー\新膜宇宙論\これまでの軌跡\パイソン")
42| ROTMOD_DIR = BASE / "Rotmod_LTG"
43| GC_CSV = BASE / "sparc_gc.csv"
44|
45|
46| # =====
47| # データ読み込み
48| # =====
49| def load_gc_table(path):
50|     data = {}
51|     with open(path, "r") as f:
52|         reader = csv.DictReader(f)
53|         for row in reader:
54|             name = row["Galaxy"].strip()
55|             data[name] = {
56|                 "gc_obs": float(row["gc"]),
57|                 "vflat": float(row["Vflat"]),
58|                 "hR": float(row["hR"]),
59|             }
60|     return data
61|
62|
```

```

63| def load_rotmod(galaxy_name, rotmod_dir):
64|     fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
65|     if not fpath.exists():
66|         return None
67|     r, vobs, vgas, vdisk, vbul = [], [], [], [], []
68|     with open(fpath, "r") as f:
69|         for line in f:
70|             line = line.strip()
71|             if not line or line.startswith("#"):
72|                 continue
73|             parts = line.split()
74|             if len(parts) < 6:
75|                 continue
76|             r.append(float(parts[0]))
77|             vobs.append(float(parts[1]))
78|             vgas.append(float(parts[3]))
79|             vdisk.append(float(parts[4]))
80|             vbul.append(float(parts[5]))
81|     return (np.array(r), np.array(vobs), np.array(vgas),
82|           np.array(vdisk), np.array(vbul))
83|
84|
85| def compute_gN_profile(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
86|     """g_N(r) in m/s^2"""
87|     r_m = r_kpc * kpc_to_m
88|     conv = 1.0e6 / r_m # (km/s)^2 / r[m] -> m/s^2
89|     gN = np.abs(Vgas) * Vgas * conv *
90|         + Yd * np.abs(Vdisk) * Vdisk * conv *
91|         + Yb * np.abs(Vbul) * Vbul * conv
92|     return np.abs(gN)
93|
94|
95| # =====
96| # ε(r) プロファイル計算
97| # =====
98| def epsilon_eq(x, c):
99|     """平衡εを x = g_N/gc で計算
100|     ε = [-x + sqrt((x+2)^2 - 4c)] / 2
101|     存在条件: (x+2)^2 >= 4c -> x >= 2(sqrt(c) - 1)
102|     """
103|     disc = (x + 2.0)**2 - 4.0 * c
104|     if np.isscalar(x):
105|         if disc < 0:
106|             return np.nan
107|         return (-x + np.sqrt(disc)) / 2.0
108|     else:
109|         result = np.full_like(x, np.nan, dtype=float)
110|         valid = disc >= 0
111|         result[valid] = (-x[valid] + np.sqrt(disc[valid])) / 2.0
112|         return result
113|
114|
115| def compute_epsilon_profile(r_kpc, gN, c):
116|     """各半径で ε(r) を計算
117|     戻り値: eps(r), valid_mask
118|     """
119|     gc = c * a0
120|     x = gN / gc
121|     eps = epsilon_eq(x, c)
122|     valid = np.isfinite(eps)
123|     return eps, valid
124|
125|
126| def measure_kink(r_kpc, eps, valid):
127|     """キングの特性量を測定
128|     - δ: キング幅 (ε が最大値の90%~>10%に落ちる幅)
129|     - r_wall: キング中心 (dε/drが最大の位置)
130|     - E_kink: int(dε/dr)^2 dr (キングエネルギー密度の代理量)
131|     """
132|     if valid.sum() < 5:
133|         return np.nan, np.nan, np.nan, np.nan
134|
135|     r_v = r_kpc[valid]
136|     e_v = eps[valid]
137|
138|     # dε/dr (数値微分)
139|     if len(r_v) < 3:
140|         return np.nan, np.nan, np.nan, np.nan
141|
142|     dr = np.diff(r_v)
143|     de = np.diff(e_v)
144|     mask_dr = dr > 0
145|     if mask_dr.sum() < 2:
146|         return np.nan, np.nan, np.nan, np.nan
147|
148|     dedr = de[mask_dr] / dr[mask_dr]
149|     r_mid = (r_v[:-1] + r_v[1:])[mask_dr] / 2.0

```

```

150|
151| # キンクエネルギー E = int(dε/dr)^2 dr
152| E_kink = np.trapz(dedr**2, r_mid)
153|
154| # dε/dr が最大 (最も急な変化) の位置 -&gt; ウォール中心
155| idx_max_grad = np.argmax(np.abs(dedr))
156| r_wall = r_mid[idx_max_grad]
157|
158| # キンク幅: ε が [ε_max, ε_min] の 80%-&gt;20% に落ちる幅
159| e_max = np.max(e_v)
160| e_min = np.min(e_v)
161| e_range = e_max - e_min
162| if e_range &lt; 1e-10:
163|     return np.nan, r_wall, E_kink, np.nan
164|
165| e_hi = e_min + 0.8 * e_range
166| e_lo = e_min + 0.2 * e_range
167|
168| # ε は一般に r とともに減少
169| r_hi = np.interp(e_hi, e_v[::-1], r_v[::-1]) if e_v[0] &gt; e_v[-1] else np.nan
170| r_lo = np.interp(e_lo, e_v[::-1], r_v[::-1]) if e_v[0] &gt; e_v[-1] else np.nan
171|
172| if np.isfinite(r_hi) and np.isfinite(r_lo):
173|     delta = abs(r_lo - r_hi)
174| else:
175|     # フォールバック: |Δε| / max|dε/dr|
176|     delta = e_range / (np.max(np.abs(dedr)) + 1e-30)
177|
178| return delta, r_wall, E_kink, np.max(np.abs(dedr))
179|
180|
181| # =====
182| # P3拘束からcを予測
183| # =====
184| def predict_c_from_delta_constraint(r_kpc, gN, hR, beta_target,
185|                                   c_min=1.001, c_max=100.0):
186|     """拘束 δ = β*hR からcを決定"""
187|     def residual(c):
188|         eps, valid = compute_epsilon_profile(r_kpc, gN, c)
189|         delta, _, _ = measure_kink(r_kpc, eps, valid)
190|         if not np.isfinite(delta) or delta &lt;= 0:
191|             return 1e10
192|         return delta / hR - beta_target
193|
194|     try:
195|         # まず符号チェック
196|         r_lo = residual(c_min)
197|         r_hi = residual(c_max)
198|         if not (np.isfinite(r_lo) and np.isfinite(r_hi)):
199|             return np.nan
200|         if r_lo * r_hi &gt; 0:
201|             # 単調-&gt;ブラケットが見つからない
202|             return np.nan
203|         c_sol = brentq(residual, c_min, c_max, xtol=1e-6, maxiter=100)
204|         return c_sol
205|     except:
206|         return np.nan
207|
208|
209| def predict_c_from_energy_constraint(r_kpc, gN, E_target,
210|                                     c_min=1.001, c_max=100.0):
211|     """拘束 E_kink = E_target からcを決定"""
212|     def residual(c):
213|         eps, valid = compute_epsilon_profile(r_kpc, gN, c)
214|         _, _, E_kink, _ = measure_kink(r_kpc, eps, valid)
215|         if not np.isfinite(E_kink) or E_kink &lt;= 0:
216|             return 1e10
217|         return np.log10(E_kink) - np.log10(E_target)
218|
219|     try:
220|         r_lo = residual(c_min)
221|         r_hi = residual(c_max)
222|         if not (np.isfinite(r_lo) and np.isfinite(r_hi)):
223|             return np.nan
224|         if r_lo * r_hi &gt; 0:
225|             return np.nan
226|         c_sol = brentq(residual, c_min, c_max, xtol=1e-6, maxiter=100)
227|         return c_sol
228|     except:
229|         return np.nan
230|
231|
232| # =====
233| # メイン
234| # =====
235| def main():
236|     print(" = " * 70)

```

```

237| print("P3: Z2 SSBドメインウォール仮説 -- gc予測検証")
238| print("=" * 70)
239|
240| gc_table = load_gc_table(GC_CSV)
241| print(f"sparc_gc.csv: {len(gc_table)} galaxies")
242|
243| # Phase 1: 観測gcで  $\epsilon(r)$  プロファイルを計算し、キンク特性を測定
244| print(f"Phase 1:  $\epsilon(r)$  キンク特性の測定 (観測gcを使用) ---")
245|
246| records = []
247| for gal_name, info in gc_table.items():
248|     rotmod = load_rotmod(gal_name, ROTMOD_DIR)
249|     if rotmod is None:
250|         continue
251|     r_kpc, Vobs, Vgas, Vdisk, Vbul = rotmod
252|     if len(r_kpc) < 5:
253|         continue
254|
255|     gN = compute_gN_profile(r_kpc, Vgas, Vdisk, Vbul)
256|     gc_obs = info["gc_obs"]
257|     hR = info["hR"]
258|     vflat = info["vflat"]
259|
260|     # gc_obs の単位チェック: a_0単位なら m/s^2 に変換
261|     if gc_obs < 1e-5:
262|         gc_val = gc_obs # 既に m/s^2
263|     else:
264|         gc_val = gc_obs * a0 # a_0単位
265|
266|     c_obs = gc_val / a0
267|
268|     if c_obs <= 1.0:
269|         # c <= 1 では臨界条件なし、 $\epsilon$  は全域で定義可能
270|         # ただしドメインウォール構造は異なる
271|         pass
272|
273|     eps, valid = compute_epsilon_profile(r_kpc, gN, c_obs)
274|     delta, r_wall, E_kink, max_grad = measure_kink(r_kpc, eps, valid)
275|
276|     if np.isfinite(delta) and delta > 0 and hR > 0:
277|         records.append({
278|             "name": gal_name,
279|             "gc_obs": gc_val,
280|             "c_obs": c_obs,
281|             "hR": hR,
282|             "vflat": vflat,
283|             "delta": delta,
284|             "r_wall": r_wall,
285|             "E_kink": E_kink,
286|             "max_grad": max_grad,
287|             "delta_over_hR": delta / hR,
288|             "r_wall_over_hR": r_wall / hR if np.isfinite(r_wall) else np.nan,
289|             "n_valid": valid.sum(),
290|             "n_total": len(r_kpc),
291|             "gN_profile": gN,
292|             "r_profile": r_kpc,
293|         })
294|
295| N = len(records)
296| print(f"キンク測定成功: {N} 銀河")
297|
298| if N < 20:
299|     print("WARNING: 解析に十分な銀河数がありません")
300|     if N == 0:
301|         sys.exit(1)
302|
303| # =====
304| # Test 1:  $\delta/hR$  の分布 -- 普遍性チェック
305| # =====
306| print(f"Test 1:  $\delta/hR$  分布 (普遍性チェック)")
307| print(f"Test 1:  $\delta/hR$  分布 (普遍性チェック)")
308| print(f"Test 1:  $\delta/hR$  分布 (普遍性チェック)")
309|
310| dh = np.array([r["delta_over_hR"] for r in records])
311| dh_f = dh[np.isfinite(dh) & (dh > 0) & (dh < 100)]
312|
313| print(f"N = {len(dh_f)}")
314| print(f"median( $\delta/hR$ ) = {np.median(dh_f):.3f}")
315| print(f"mean( $\delta/hR$ ) = {np.mean(dh_f):.3f}")
316| print(f"std( $\delta/hR$ ) = {np.std(dh_f):.3f}")
317| print(f"CV (std/mean) = {np.std(dh_f)/np.mean(dh_f):.3f}")
318| print(f"min = {np.min(dh_f):.3f}, max = {np.max(dh_f):.3f}")
319|
320| if np.std(dh_f) / np.mean(dh_f) < 0.3:
321|     print(f"CV < 0.3:  $\delta/hR$  はほぼ普遍的 [OK]")
322|     universal_beta = True
323| else:

```

```

324|         print(" -&gt; CV &gt;= 0.3:  $\delta/hR$  は銀河ごとに大きく変動")
325|         universal_beta = False
326|
327|         # =====
328|         # Test 2:  $\delta/hR$  vs c, vflat, hR の相関
329|         # =====
330|         print("\n" + "=" * 50)
331|         print("Test 2:  $\delta/hR$  の駆動因子")
332|         print("=" * 50)
333|
334|         c_arr = np.array([r["c_obs"] for r in records])
335|         vf_arr = np.array([r["vflat"] for r in records])
336|         hR_arr = np.array([r["hR"] for r in records])
337|         E_arr = np.array([r["E_kink"] for r in records])
338|
339|         mask = np.isfinite(dh) & (dh > 0) & (dh < 100)
340|
341|         for label, arr in [("c_obs", c_arr), ("vflat", vf_arr),
342|                             ("hR", hR_arr), ("log E_kink", np.log10(E_arr + 1e-30))]:
343|             m = mask & np.isfinite(arr)
344|             if m.sum() > 10:
345|                 rho, p = pearsonr(arr[m], dh[m])
346|                 print(f"  $\delta/hR$  vs {label:12s}:  $\rho = \{rho:.4f\}$ ,  $p = \{p:.3e\}$ ")
347|
348|         # =====
349|         # Test 3: r_wall / hR の分布
350|         # =====
351|         print("\n" + "=" * 50)
352|         print("Test 3: r_wall / hR 分布 (ウォール位置)")
353|         print("=" * 50)
354|
355|         rw_hR = np.array([r["r_wall_over_hR"] for r in records])
356|         rw_f = rw_hR[np.isfinite(rw_hR) & (rw_hR > 0)]
357|
358|         print(f" N = {len(rw_f)}")
359|         print(f" median(r_wall/hR) = {np.median(rw_f):.2f}")
360|         print(f" mean(r_wall/hR) = {np.mean(rw_f):.2f}")
361|         print(f" std = {np.std(rw_f):.2f}")
362|
363|         # =====
364|         # Test 4: E_kink の普遍性 -&gt; cの予測
365|         # =====
366|         print("\n" + "=" * 50)
367|         print("Test 4: E_kink 拘束からgc予測")
368|         print("=" * 50)
369|
370|         E_valid = E_arr[mask & np.isfinite(E_arr) & (E_arr > 0)]
371|         E_median = np.median(E_valid)
372|         E_cv = np.std(E_valid) / np.mean(E_valid)
373|
374|         print(f" E_kink: median = {E_median:.4e}")
375|         print(f" E_kink: CV = {E_cv:.3f}")
376|
377|         if E_cv < 0.5:
378|             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")
379|         else:
380|             print(" -&gt; E_kink のばらつきが大きい -&gt; 拘束としては弱い")
381|
382|         # E_kink = E_median の拘束からcを予測
383|         gc_pred_E = []
384|         gc_obs_E = []
385|         count_E = 0
386|         for rec in records:
387|             c_pred = predict_c_from_energy_constraint(
388|                 rec["r_profile"], rec["gN_profile"], E_median)
389|             if np.isfinite(c_pred) and c_pred > 0:
390|                 gc_pred_E.append(c_pred * a0)
391|                 gc_obs_E.append(rec["gc_obs"])
392|                 count_E += 1
393|
394|         print(f" E_kink拘束で収束: {count_E}/{N}")
395|
396|         if count_E > 10:
397|             gc_pred_E = np.array(gc_pred_E)
398|             gc_obs_E = np.array(gc_obs_E)
399|             lp = np.log10(gc_pred_E)
400|             lo = np.log10(gc_obs_E)
401|             m = np.isfinite(lp) & np.isfinite(lo)
402|             if m.sum() > 5:
403|                 rho, p = pearsonr(lp[m], lo[m])
404|                 ss_res = np.sum((lo[m] - lp[m])**2)
405|                 ss_tot = np.sum((lo[m] - np.mean(lo[m]))**2)
406|                 R2 = 1 - ss_res / ss_tot if ss_tot > 0 else np.nan
407|                 ratio = gc_pred_E[m] / gc_obs_E[m]
408|                 print(f"  $\rho$  (gc_pred, gc_obs) = {rho:.4f},  $p = \{p:.3e\}$ ")
409|                 print(f"  $R^2(1:1) = \{R2:.4f\}$ ")
410|                 print(f" ratio median = {np.median(ratio):.3f}")

```

```

411|
412| # =====
413| # Test 5:  $\delta = \beta \cdot hR$  拘束からgc予測
414| # =====
415| print("\n" + "=" * 50)
416| print("Test 5:  $\delta = \beta \cdot hR$  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  $\beta = \{beta\_med:.3f\}$ ")
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 > 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"  $\delta$  拘束で収束: {count_D}/{N}")
434|
435| if count_D > 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) & np.isfinite(lo)
441|     if m.sum() > 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 > 0 else np.nan
446|         ratio = gc_pred_D[m] / gc_obs_D[m]
447|         print(f"  $\rho$  (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|                   # -> 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) & np.isfinite(log_GS0)
475|
476| #  $\delta$  vs GS0
477| m1 = np.isfinite(log_delta) & np.isfinite(log_GS0)
478| if m1.sum() > 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"  $\log(\delta)$  vs  $\log(vflat^2/hR)$ : slope={sl[0]:.3f},  $\rho = \{rho:.3f\}$ ")
482|
483| # E_kink vs GS0
484| m2 = np.isfinite(log_E) & np.isfinite(log_GS0) & (log_E > -20)
485| if m2.sum() > 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},  $\rho = \{rho:.3f\}$ ")
489|
490| # E_kink vs gc
491| m3 = np.isfinite(log_E) & np.isfinite(log_gc) & (log_E > -20)
492| if m3.sum() > 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},  $\rho = \{rho:.3f\}$ ")
496|
497| # =====

```

```

498| # プロット
499| # =====
500| try:
501|     import matplotlib
502|     matplotlib.use("Agg")
503|     import matplotlib.pyplot as plt
504|
505|     fig, axes = plt.subplots(2, 3, figsize=(16, 10))
506|     fig.suptitle("P3: Z2 SSB Domain Wall Structure", fontsize=14)
507|
508|     # (a)  $\delta$ /hR 分布
509|     ax = axes[0, 0]
510|     ax.hist(dh_f, bins=30, edgecolor="black", alpha=0.7)
511|     ax.axvline(np.median(dh_f), color="red", ls="--",
512|               label=f"median={np.median(dh_f):.2f}")
513|     ax.set_xlabel(" $\delta$  / hR")
514|     ax.set_ylabel("count")
515|     ax.set_title("(a) Kink width / hR")
516|     ax.legend()
517|
518|     # (b) r_wall/hR 分布
519|     ax = axes[0, 1]
520|     ax.hist(rw_f, bins=30, edgecolor="black", alpha=0.7)
521|     ax.axvline(np.median(rw_f), color="red", ls="--",
522|               label=f"median={np.median(rw_f):.1f}")
523|     ax.set_xlabel("r_wall / hR")
524|     ax.set_title("(b) Wall position / hR")
525|     ax.legend()
526|
527|     # (c)  $\delta$ /hR vs c_obs
528|     ax = axes[0, 2]
529|     m = np.isfinite(dh) & (dh > 0) & (dh < 100)
530|     ax.scatter(c_arr[m], dh[m], s=8, alpha=0.5)
531|     ax.set_xlabel("c_obs = gc/a_0")
532|     ax.set_ylabel(" $\delta$  / hR")
533|     ax.set_title("(c)  $\delta$ /hR vs c")
534|
535|     # (d) E_kink vs gc
536|     ax = axes[1, 0]
537|     m3 = np.isfinite(log_E) & np.isfinite(log_gc) & (log_E > -20)
538|     if m3.sum() > 0:
539|         ax.scatter(log_gc[m3], log_E[m3], s=8, alpha=0.5)
540|         ax.set_xlabel("log_1_0(gc_obs)")
541|         ax.set_ylabel("log_1_0(E_kink)")
542|         ax.set_title("(d) Kink energy vs gc")
543|
544|     # (e) gc_pred (E拘束) vs gc_obs
545|     ax = axes[1, 1]
546|     if count_E > 5:
547|         lp = np.log10(gc_pred_E)
548|         lo = np.log10(gc_obs_E)
549|         m = np.isfinite(lp) & np.isfinite(lo)
550|         ax.scatter(lo[m], lp[m], s=8, alpha=0.5)
551|         lim = [min(lo[m].min(), lp[m].min())-0.1,
552|               max(lo[m].max(), lp[m].max())+0.1]
553|         ax.plot(lim, lim, "k--")
554|         ax.set_xlabel("log_1_0(gc_obs)")
555|         ax.set_ylabel("log_1_0(gc_pred)")
556|         ax.set_title("(e) E_kink constraint")
557|         ax.set_aspect("equal")
558|     else:
559|         ax.text(0.5, 0.5, "Not enough data", transform=ax.transAxes,
560|                ha="center")
561|
562|     # (f) gc_pred ( $\delta$ 拘束) vs gc_obs
563|     ax = axes[1, 2]
564|     if count_D > 5:
565|         lp = np.log10(gc_pred_D)
566|         lo = np.log10(gc_obs_D)
567|         m = np.isfinite(lp) & np.isfinite(lo)
568|         ax.scatter(lo[m], lp[m], s=8, alpha=0.5)
569|         lim = [min(lo[m].min(), lp[m].min())-0.1,
570|               max(lo[m].max(), lp[m].max())+0.1]
571|         ax.plot(lim, lim, "k--")
572|         ax.set_xlabel("log_1_0(gc_obs)")
573|         ax.set_ylabel("log_1_0(gc_pred)")
574|         ax.set_title("(f)  $\delta = \beta \cdot hR$  constraint")
575|         ax.set_aspect("equal")
576|     else:
577|         ax.text(0.5, 0.5, "Not enough data", transform=ax.transAxes,
578|                ha="center")
579|
580|     plt.tight_layout()
581|     out_fig = BASE / "P3_domainwall_results.png"
582|     plt.savefig(out_fig, dpi=150)
583|     print(f"Figure saved: {out_fig}")
584| 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  $\delta/hR$ : 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 > 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 > 10:
600|     print(f"  $\delta$  拘束: 収束={count_D}/{N}")
601|
602| print("%n P3評価:")
603| if universal_beta:
604|     print(" ->  $\delta/hR$  が普遍的 -> ドメインウォール仮説に整合")
605| else:
606|     print(" ->  $\delta/hR$  が非普遍的 -> 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  $g_{\text{eff}} = \sqrt{g_N \times g_c}$  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 ( $gc_{\text{deep}}$ ) と完全MONDフィット ( $gc_{\text{full}}$ ) 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  $v_{\text{flat}}^2 \sim hR$  も検証する。

#### 結果

成立:  $gc_{\text{deep}} \alpha=0.471+/-0.036$ ,  $p(0.5)=0.43$ .  $x<0.1$ :  $\alpha=0.478+/-0.083$ .

#### ソースコード

(589 行)

```
1 | #!/usr/bin/env python3
2 | """
3 | sparc_P5_deepMOND.py
4 | =====
5 | P5検証: 深MONDリミットの摂動展開から  $\alpha=0.5$ が出るか
6 |
7 | 戦略:
8 | 深MONDでは  $g_{\text{eff}} = \sqrt{g_N \times g_c}$  -> BTFR:  $v_{\text{flat}}^4 = gc \times GM_{\text{bar}}$ 
9 | しかし BTFR は  $hR$  を含まない (大 $r$ 極限で点質量と等価)。
10 |
11 |  $hR$ 依存性は「有限 $r$ 効果」= 深MONDからのずれから来る。
12 | -> 完全MOND補間関数で回転曲線をフィットし、
13 | 得られる $gc$ のスケーリングを調べる。
14 |
15 | テスト:
16 | Test 1: 深MONDフィット  $gc_{\text{DM}}$  ( $r^2 \times g_N$  ->  $v_{\text{flat}}^4$ から $gc$ を決定)
17 | Test 2: 完全MONDフィット  $gc_{\text{full}}$  (全 $r$ 範囲で $V(r;gc)$ をフィット)
18 | Test 3:  $gc_{\text{DM}}$ ,  $gc_{\text{full}}$  のスケーリング ->  $\alpha$  の値
19 | Test 4:  $gc_{\text{full}} - gc_{\text{DM}}$  の差 (1次補正) の $hR$ 依存性
20 | Test 5: 「フラットになる半径」 $r_{\text{flat}}/hR$  と補正の関係
21 | Test 6: 自己無撞着条件の次元帰結
22 |
23 | 実行: uv run --with scipy --with matplotlib python sparc_P5_deepMOND.py
24 | """
25 |
26 | import numpy as np
27 | from scipy.optimize import minimize_scalar, brentq
28 | from scipy.stats import pearsonr, spearmanr, linregress
29 | from pathlib import Path
30 | import csv
31 | import sys
32 |
33 | # =====
34 | # 定数
35 | # =====
36 | a0 = 1.2e-10 # m/s^2
37 | kpc_to_m = 3.0857e19 # 1 kpc -> m
38 | kms2_to_ms2 = 1.0e6 # (km/s)^2 -> (m/s)^2
39 | Yd_default = 0.5
40 | Yb_default = 0.7
41 |
42 | 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")
43 | ROTMOD_DIR = BASE / "Rotmod_LTG"
44 | GC_CSV = BASE / "sparc_gc.csv"
45 |
46 |
47 | # =====
48 | # データ読み込み
49 | # =====
50 | def load_gc_table(path):
51 |     data = {}
52 |     with open(path, "r") as f:
53 |         reader = csv.DictReader(f)
54 |         for row in reader:
55 |             name = row["Galaxy"].strip()
56 |             data[name] = {
57 |                 "gc_obs": float(row["gc"]),
58 |                 "vflat": float(row["Vflat"]),
59 |                 "hR": float(row["hR"]),
60 |             }
61 |     return data
62 |
```

```

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) < 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 -> 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 -> 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

```

```

150| mode: "deep" = 深MONDのみ, "full" = 完全MOND補間
151|
152| フラット領域（外側50%のデータ点）にフォーカス
153| """
154| N = len(r_kpc)
155| # 外側50%（最低5点）
156| n_flat = max(5, N // 2)
157| idx_flat = slice(N - n_flat, N)
158|
159| r_f = r_kpc[idx_flat]
160| V_f = Vobs[idx_flat]
161| e_f = errV[idx_flat]
162| gN_f = gN[idx_flat]
163|
164| # 誤差が0の点は除外
165| valid = e_f > 0
166| if valid.sum() < 3:
167|     valid = np.ones(len(e_f), dtype=bool)
168|     e_f = np.where(e_f > 0, e_f, np.median(Vobs) * 0.1)
169|
170| def chi2(log_gc):
171|     gc = 10**log_gc
172|     if mode == "deep":
173|         V_pred = V_deepMOND(r_f[valid], gN_f[valid], gc)
174|     else:
175|         V_pred = V_fullMOND(r_f[valid], gN_f[valid], gc)
176|     resid = (V_f[valid] - V_pred) / e_f[valid]
177|     return np.sum(resid**2)
178|
179| result = minimize_scalar(chi2, bounds=(np.log10(gc_min), np.log10(gc_max)),
180|                          method="bounded")
181|
182| gc_best = 10**result.x
183| chi2_min = result.fun
184| ndof = valid.sum() - 1
185|
186| return gc_best, chi2_min, ndof
187|
188|
189| def fit_gc_BTFR(gN, r_kpc, vflat):
190|     """BTFR方式: vflat^4 = gc*r^2*g_N を大rで適用
191|     -> gc = vflat^4 / (r^2 * g_N) の外側平均
192|     """
193|     N = len(r_kpc)
194|     n_out = max(3, N // 3)
195|
196|     r_m = r_kpc[-n_out:] * kpc_to_m
197|     gN_out = gN[-n_out:]
198|     vflat_ms = vflat * 1e3 # km/s -> m/s
199|
200|     # gc = vflat^4 / (r^2 * g_N)... これは
201|     # V^4 = r^2 * g_N * gc (深MOND) より
202|     # gc = V^4 / (r^2 * g_N)
203|
204|     valid = gN_out > 0
205|     if valid.sum() < 1:
206|         return np.nan
207|
208|     gc_arr = vflat_ms**4 / (r_m[valid]**2 * gN_out[valid])
209|     return np.median(gc_arr)
210|
211|
212| # =====
213| # メイン
214| # =====
215| def main():
216|     print("=" * 70)
217|     print("P5: 深MONDリミットの摂動展開 --  $\alpha=0.5$ の導出")
218|     print("=" * 70)
219|
220|     gc_table = load_gc_table(GC_CSV)
221|     print(f"sparc_gc.csv: {len(gc_table)} galaxies")
222|
223|     records = []
224|     skipped = 0
225|
226|     for gal_name, info in gc_table.items():
227|         rotmod = load_rotmod(gal_name, ROTMOD_DIR)
228|         if rotmod is None:
229|             skipped += 1
230|             continue
231|
232|         r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
233|         if len(r_kpc) < 8:
234|             skipped += 1
235|             continue
236|

```

```

237|         gN = compute_gN_profile(r_kpc, Vgas, Vdisk, Vbul)
238|
239|         # gc単位の確認・変換
240|         gc_obs = info["gc_obs"]
241|         if gc_obs > 1e-5: # a_0単位の可能性
242|             gc_obs_ms2 = gc_obs * a0
243|         else:
244|             gc_obs_ms2 = gc_obs
245|
246|         hR = info["hR"]
247|         vflat = info["vflat"]
248|
249|         # --- フィット ---
250|         # (a) 深MOND
251|         gc_deep, chi2_deep, ndof_deep = fit_gc(
252|             r_kpc, gN, Vobs, errV, mode="deep")
253|
254|         # (b) 完全MOND
255|         gc_full, chi2_full, ndof_full = fit_gc(
256|             r_kpc, gN, Vobs, errV, mode="full")
257|
258|         # (c) BTFR方式
259|         gc_btfr = fit_gc_BTFR(gN, r_kpc, vflat)
260|
261|         if not (np.isfinite(gc_deep) and np.isfinite(gc_full)):
262|             skipped += 1
263|             continue
264|
265|         # --- 1次補正の測定 ---
266|         g_full, g_deep_arr, g_corr = V_fullMOND_with_correction(
267|             r_kpc, gN, gc_full)
268|
269|         # 補正の相対的大きさ (外側半分で評価)
270|         n_out = max(3, len(r_kpc) // 2)
271|         rel_corr = np.median(np.abs(g_corr[-n_out:])) / (g_deep_arr[-n_out:] + 1e-30)
272|
273|         # x = g_N/gc の中央値 (外側)
274|         x_outer = np.median(gN[-n_out:] / gc_full)
275|
276|         # 「フラットになる半径」の推定: Vobs が vflat の 90% を超える最小r
277|         v90 = 0.9 * vflat
278|         r_flat_idx = np.where(Vobs >= v90)[0]
279|         r_flat = r_kpc[r_flat_idx[0]] if len(r_flat_idx) > 0 else r_kpc[-1]
280|
281|         # proxy
282|         GS0_proxy = (vflat * 1e3)**2 / (hR * kpc_to_m) # m/s^2
283|         gc_geom = np.sqrt(a0 * GS0_proxy)
284|
285|         records.append({
286|             "name": gal_name,
287|             "gc_obs": gc_obs_ms2,
288|             "gc_deep": gc_deep,
289|             "gc_full": gc_full,
290|             "gc_btfr": gc_btfr,
291|             "gc_geom": gc_geom,
292|             "chi2_deep": chi2_deep,
293|             "chi2_full": chi2_full,
294|             "hR": hR,
295|             "vflat": vflat,
296|             "GS0_proxy": GS0_proxy,
297|             "rel_corr": rel_corr,
298|             "x_outer": x_outer,
299|             "r_flat_hR": r_flat / hR if hR > 0 else np.nan,
300|             "delta_gc": np.log10(gc_full) - np.log10(gc_deep),
301|         })
302|
303|     N = len(records)
304|     print(f"Processed: {N}, Skipped: {skipped}")
305|
306|     # NumPy配列化
307|     gc_obs = np.array([r["gc_obs"] for r in records])
308|     gc_deep = np.array([r["gc_deep"] for r in records])
309|     gc_full = np.array([r["gc_full"] for r in records])
310|     gc_btfr = np.array([r["gc_btfr"] for r in records])
311|     gc_geom = np.array([r["gc_geom"] for r in records])
312|     hR_arr = np.array([r["hR"] for r in records])
313|     vflat_arr = np.array([r["vflat"] for r in records])
314|     GS0_arr = np.array([r["GS0_proxy"] for r in records])
315|     rel_corr = np.array([r["rel_corr"] for r in records])
316|     x_outer = np.array([r["x_outer"] for r in records])
317|     r_flat_hR = np.array([r["r_flat_hR"] for r in records])
318|     delta_gc = np.array([r["delta_gc"] for r in records])
319|
320|     # =====
321|     # Test 1: gc_deep (深MOND) のスケーリング
322|     # =====
323|     print("¥n" + "=" * 50)

```

```

324| print("Test 1: gc_deep (深MONDフィット) のスケーリング")
325| print("=" * 50)
326|
327| log_gcd = np.log10(gc_deep)
328| log_GS0 = np.log10(GS0_arr)
329| log_vf = np.log10(vflat_arr)
330| log_hR = np.log10(hR_arr)
331|
332| m = np.isfinite(log_gcd) & np.isfinite(log_GS0)
333| sl = linregress(log_GS0[m], log_gcd[m])
334| print(f" log(gc_deep) vs log(vflat^2/hR): slope = {sl.slope:.4f} +/- {sl.stderr:.4f}")
335| print(f" R^2 = {sl.rvalue**2:.4f}")
336| print(f" 期待: slope = 0.5 -> p(0.5) = {abs(sl.slope-0.5)/sl.stderr:.2f} σ")
337|
338| # 多変量: log(gc_deep) = a*log(vflat) + b*log(hR) + const
339| from numpy.linalg import lstsq
340| X = np.column_stack([log_vf[m], log_hR[m], np.ones(m.sum())])
341| y = log_gcd[m]
342| coef, res, _, _ = lstsq(X, y, rcond=None)
343| y_pred = X @ coef
344| ss_res = np.sum((y - y_pred)**2)
345| ss_tot = np.sum((y - np.mean(y))**2)
346| R2_mv = 1 - ss_res / ss_tot
347| print(f" 多変量: gc_deep ~ vflat^{coef[0]:.3f} x hR^{coef[1]:.3f}, R^2={R2_mv:.4f}")
348| print(f" proxy^0.5期待: vflat^1.0, hR^{-(0.5)}")
349|
350| # =====
351| # Test 2: gc_full (完全MOND) のスケーリング
352| # =====
353| print("\n" + "=" * 50)
354| print("Test 2: gc_full (完全MONDフィット) のスケーリング")
355| print("=" * 50)
356|
357| log_gcf = np.log10(gc_full)
358| m2 = np.isfinite(log_gcf) & np.isfinite(log_GS0)
359| sl2 = linregress(log_GS0[m2], log_gcf[m2])
360| print(f" log(gc_full) vs log(vflat^2/hR): slope = {sl2.slope:.4f} +/- {sl2.stderr:.4f}")
361| print(f" R^2 = {sl2.rvalue**2:.4f}")
362|
363| coef2, _, _, _ = lstsq(
364|     np.column_stack([log_vf[m2], log_hR[m2], np.ones(m2.sum())]),
365|     log_gcf[m2], rcond=None)
366| y_pred2 = np.column_stack([log_vf[m2], log_hR[m2], np.ones(m2.sum())]) @ coef2
367| R2_mv2 = 1 - np.sum((log_gcf[m2] - y_pred2)**2) / np.sum((log_gcf[m2] - np.mean(log_gcf[m2]))**2)
368| print(f" 多変量: gc_full ~ vflat^{coef2[0]:.3f} x hR^{coef2[1]:.3f}, R^2={R2_mv2:.4f}")
369|
370| # =====
371| # Test 3: gc_obs vs gc_full, gc_deep の直接比較
372| # =====
373| print("\n" + "=" * 50)
374| print("Test 3: gc予測 vs gc_obs 直接比較")
375| print("=" * 50)
376|
377| log_gco = np.log10(gc_obs)
378|
379| for label, gc_p in [("gc_deep", gc_deep), ("gc_full", gc_full),
380|                  ("gc_btfr", gc_btfr), ("gc_geom", gc_geom)]:
381|     log_p = np.log10(gc_p)
382|     m = np.isfinite(log_gco) & np.isfinite(log_p)
383|     if m.sum() < 10:
384|         print(f" {label}: insufficient data ({m.sum()})")
385|         continue
386|     rho, p = pearsonr(log_p[m], log_gco[m])
387|     ratio = gc_p[m] / gc_obs[m]
388|     ss_r = np.sum((log_gco[m] - log_p[m])**2)
389|     ss_t = np.sum((log_gco[m] - np.mean(log_gco[m]))**2)
390|     R2 = 1 - ss_r / ss_t if ss_t > 0 else np.nan
391|     print(f" {label:10s}: ρ={rho:.4f}, R^2(1:1)={R2:.4f}, "
392|           f"ratio median={np.median(ratio):.3f}")
393|
394| # =====
395| # Test 4: 1次補正 (gc_full - gc_deep) のhR依存性
396| # =====
397| print("\n" + "=" * 50)
398| print("Test 4: 1次補正 Δ log(gc) = log(gc_full/gc_deep) の構造")
399| print("=" * 50)
400|
401| m4 = np.isfinite(delta_gc) & np.isfinite(log_hR)
402| print(f" Δ log(gc) 統計: median={np.median(delta_gc[m4]):.4f}, "
403|       f"std={np.std(delta_gc[m4]):.4f}")
404|
405| for label, arr in [("log(hR)", log_hR), ("log(vflat)", log_vf),
406|                  ("x_outer", x_outer), ("r_flat/hR", r_flat_hR)]:
407|     m = np.isfinite(delta_gc) & np.isfinite(arr)
408|     if m.sum() > 10:
409|         rho, p = pearsonr(arr[m], delta_gc[m])
410|         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},  $\alpha$ ={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:  $v_{\text{flat}}^4 = gc \cdot GM_{\text{bar}} - \text{gc}$ ;  $gc_{\text{BTFR}} = v_{\text{flat}}^4 / (GM_{\text{bar}})$ 
444| # 幾何平均:  $gc = \eta \cdot \sqrt{a_0 \cdot v_{\text{flat}}^2 / hR}$ 
445| # 連立:  $v_{\text{flat}}^4 = \eta \cdot \sqrt{a_0 \cdot v_{\text{flat}}^2 / hR} \cdot GM_{\text{bar}}$ 
446| #  $v_{\text{flat}}^3 = \eta \cdot GM_{\text{bar}} \cdot \sqrt{a_0 / hR}$ 
447| # つまり  $v_{\text{flat}}^3 \sim M_{\text{bar}} / \sqrt{hR}$  ( $\eta$  が Yd のみ依存なら)
448|
449| #  $M_{\text{bar}} \sim Yd \cdot L_{\text{disk}}$ ,  $L_{\text{disk}}$  は光度
450| # BTFR との比較:  $v_{\text{flat}}^4 \sim M_{\text{bar}}$  vs  $v_{\text{flat}}^3 \sim M_{\text{bar}} / \sqrt{hR}$ 
451|
452| # 後者を前者で割ると:  $1/v_{\text{flat}} \sim 1/\sqrt{hR} - \text{gc}$ ;  $v_{\text{flat}}^2 \sim hR$ 
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 ( $v_{\text{flat}}^2 \sim hR$ )")
460| print(f" 実測からの乖離: {abs(sl6.slope - 0.5)/sl6.stderr:.1f}  $\sigma$ ")
461|
462| #  $v_{\text{flat}}^2$  vs hR (直接)
463| sl6b = linregress(log_hR[m6], 2*log_vf[m6])
464| print(f" log( $v_{\text{flat}}^2$ ) vs log(hR): slope={sl6b.slope:.4f}")
465|
466| # 残差のhR依存性 (BTFR残差)
467| # BTFR:  $\log(gc) \sim 4 \cdot \log(v_{\text{flat}}) + \text{const}$ 
468| # 幾何平均:  $\log(gc) \sim \log(v_{\text{flat}}) - 0.5 \cdot \log(hR) + \text{const}$ 
469| # 差:  $3 \cdot \log(v_{\text{flat}}) \sim -0.5 \cdot \log(hR) + \text{const}$ 
470| #  $-\text{gc}$ ;  $\log(v_{\text{flat}}) \sim -1/6 \cdot \log(hR)$  &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"  $v_{\text{flat}}^3 \sim M_{\text{bar}} / \sqrt{hR}$  を予測。")
474| print(f"  $v_{\text{flat}}$  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")

```

```

498|         ax.set_aspect("equal")
499|
500|         # (b) gc_full スケーリング
501|         ax = axes[0, 1]
502|         m = np.isfinite(log_gcf) & np.isfinite(log_GS0)
503|         ax.scatter(log_GS0[m], log_gcf[m], s=8, alpha=0.5)
504|         x_line = np.linspace(log_GS0[m].min(), log_GS0[m].max(), 50)
505|         ax.plot(x_line, sl2.slope * x_line + sl2.intercept, "r-",
506|                 label=f"slope={sl2.slope:.3f}")
507|         ax.plot(x_line, 0.5 * x_line + np.median(log_gcf[m] - 0.5*log_GS0[m]),
508|                 "g--", label="slope=0.5 (期待)")
509|         ax.set_xlabel("log_1_0(vflat^2/hR)")
510|         ax.set_ylabel("log_1_0(gc_full)")
511|         ax.set_title("(b) gc_full scaling")
512|         ax.legend(fontsize=9)
513|
514|         # (c) gc_deep vs gc_full
515|         ax = axes[0, 2]
516|         m = np.isfinite(log_gcd) & np.isfinite(log_gcf)
517|         ax.scatter(log_gcf[m], log_gcd[m], s=8, alpha=0.5)
518|         lim2 = [min(log_gcf[m].min(), log_gcd[m].min())-0.1,
519|                 max(log_gcf[m].max(), log_gcd[m].max())+0.1]
520|         ax.plot(lim2, lim2, "k--")
521|         ax.set_xlabel("log_1_0(gc_full)")
522|         ax.set_ylabel("log_1_0(gc_deep)")
523|         ax.set_title("(c) Deep vs Full MOND")
524|         ax.set_aspect("equal")
525|
526|         # (d) 1次補正 vs x_outer
527|         ax = axes[1, 0]
528|         m = np.isfinite(delta_gc) & np.isfinite(x_outer)
529|         ax.scatter(x_outer[m], delta_gc[m], s=8, alpha=0.5)
530|         ax.axhline(0, color="k", ls="--")
531|         ax.set_xlabel("x_outer = g_N/gc")
532|         ax.set_ylabel("Δ log_1_0(gc_full/gc_deep)")
533|         ax.set_title("(d) 1st order correction vs x")
534|
535|         # (e) x_outer 分布
536|         ax = axes[1, 1]
537|         ax.hist(x_f, bins=30, edgecolor="black", alpha=0.7)
538|         ax.axvline(np.median(x_f), color="r", ls="--",
539|                   label=f"median={np.median(x_f):.2f}")
540|         ax.axvline(1.0, color="blue", ls=":", label="x=1 (MOND)")
541|         ax.set_xlabel("x_outer = g_N/gc")
542|         ax.set_title("(e) Deep MOND degree")
543|         ax.legend(fontsize=9)
544|
545|         # (f) vflat vs hR
546|         ax = axes[1, 2]
547|         m = np.isfinite(log_vf) & np.isfinite(log_hR)
548|         ax.scatter(log_hR[m], log_vf[m], s=8, alpha=0.5)
549|         x_line = np.linspace(log_hR[m].min(), log_hR[m].max(), 50)
550|         ax.plot(x_line, sl6.slope * x_line + sl6.intercept, "r-",
551|                 label=f"slope={sl6.slope:.3f}")
552|         ax.plot(x_line, 0.5 * x_line + np.median(log_vf[m] - 0.5*log_hR[m]),
553|                 "g--", label="slope=0.5 (予測)")
554|         ax.set_xlabel("log_1_0(hR [kpc])")
555|         ax.set_ylabel("log_1_0(vflat [km/s])")
556|         ax.set_title("(f) vflat vs hR")
557|         ax.legend(fontsize=9)
558|
559|         plt.tight_layout()
560|         out_fig = BASE / "P5_deepMOND_results.png"
561|         plt.savefig(out_fig, dpi=150)
562|         print(f"Figure saved: {out_fig}")
563|     except Exception as e:
564|         print(f"Plot skipped: {e}")
565|
566|     # =====
567|     # SUMMARY
568|     # =====
569|     print("\n" + "=" * 70)
570|     print("SUMMARY")
571|     print("=" * 70)
572|     print(f"gc_deep スケーリング: α = {sl.slope:.4f} +/- {sl.stdev:.4f}")
573|     print(f"gc_full スケーリング: α = {sl2.slope:.4f} +/- {sl2.stdev:.4f}")
574|     print(f"gc_full 多変量: vflat {coef2[0]:.3f} x hR {coef2[1]:.3f}")
575|     print(f"深MOND度 median(x_outer) = {np.median(x_f):.3f}")
576|     print(f"1次補正 median|Δ log(gc)| = {np.median(np.abs(delta_gc[np.isfinite(delta_gc)]):.4f}")
577|     print(f"vflat vs hR: slope = {sl6.slope:.4f}")
578|
579|     if abs(sl2.slope - 0.5) < 2 * sl2.stdev:
580|         print(f"[*] gc_full のスケールリングが α=0.5 と整合!")
581|         print(f"-&gt; 完全MOND補間から幾何平均法則が再現される")
582|     else:
583|         print(f"n gc_full α={sl2.slope:.3f} != 0.5")
584|         print(f"-&gt; MOND補間のみでは幾何平均法則を説明不十分")

```

```
585|  
586|  
587| if __name__ == "__main__":  
588|     main()  
589|
```

## 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 ( $\rho: -0.472 \rightarrow -0.149$ )。raw ではhR-gc無相関 ( $\rho = -0.022$ )。

### ソースコード

(612 行)

```
1 | #!/usr/bin/env python3
2 | """
3 | sparc_hR_decomposition.py
4 | =====
5 | hR偏相関 rho=-0.312 の分解
6 |
7 | 問題:
8 | V-1b: gc ~ vflat1.10 x hR(-0.56)
9 | P5 deep MOND: gc_deep ~ vflat0.94 x hR(-0.44)
10 | 差分: hR指数に -0.12 の未説明分
11 |
12 | 候補:
13 | (a) 有限x補正の間接的hR依存性
14 | (b) Yd-hR相関 -&gt; η (Yd)を通じたhR依存性
15 | (c) 指数ディスク近似の破れ
16 | (d) バルジ/ガス比の効果
17 |
18 | テスト:
19 | Test 1: Yd vs hR の相関 -&gt; (b)の検証
20 | 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
21 | Test 3: gc_obs vs gc_deep の残差のhR依存性の分解
22 | Test 4: 「有効ディスク」補正  $M_{\text{bar}} / (2\pi \Sigma_{\text{0hR}}^2)$  のhR依存性 -&gt; (c)の検証
23 | Test 5: バルジ割合 f_bul vs hR残差 -&gt; (d)の検証
24 | Test 6: 段階的偏相関 (全候補を逐次統制して残差を追跡)
25 |
26 | 実行: uv run --with scipy --with matplotlib python sparc_hR_decomposition.py
27 | """
28 |
29 | import numpy as np
30 | from scipy.optimize import minimize_scalar
31 | from scipy.stats import pearsonr, spearmanr, linregress
32 | from numpy.linalg import lstsq
33 | from pathlib import Path
34 | import csv
35 | import sys
36 |
37 | # =====
38 | # 定数
39 | # =====
40 | a0 = 1.2e-10
41 | kpc_to_m = 3.0857e19
42 | kms2_to_ms2 = 1.0e6
43 | Yd_default = 0.5
44 | Yb_default = 0.7
45 |
46 | BASE = Path(r"D:\ドキュメント\アントロピー\新膜宇宙論\これまでの軌跡\パイソン")
47 | ROTMOD_DIR = BASE / "Rotmod_LTG"
48 | GC_CSV = BASE / "sparc_gc.csv"
49 |
50 |
51 | # =====
52 | # データ読み込み
53 | # =====
54 | def load_gc_table(path):
55 |     data = {}
56 |     with open(path, "r") as f:
57 |         reader = csv.DictReader(f)
58 |         for row in reader:
59 |             name = row["Galaxy"].strip()
60 |             data[name] = {
61 |                 "gc_obs": float(row["gc"]),
62 |                 "vflat": float(row["Vflat"]),
```

```

63|         "hR": float(row["hR"]),
64|     }
65|     # Yd がカラムにあれば読む
66|     if "Yd" in row:
67|         try:
68|             data[name]["Yd"] = float(row["Yd"])
69|         except:
70|             data[name]["Yd"] = Yd_default
71|     else:
72|         data[name]["Yd"] = Yd_default
73| return data
74|
75|
76| def load_rotmod(galaxy_name, rotmod_dir):
77|     fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
78|     if not fpath.exists():
79|         return None
80|     r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
81|     with open(fpath, "r") as f:
82|         for line in f:
83|             line = line.strip()
84|             if not line or line.startswith("#"):
85|                 continue
86|             parts = line.split()
87|             if len(parts) < 6:
88|                 continue
89|             r.append(float(parts[0]))
90|             vobs.append(float(parts[1]))
91|             errv.append(float(parts[2]))
92|             vgas.append(float(parts[3]))
93|             vdisk.append(float(parts[4]))
94|             vbul.append(float(parts[5]))
95|     return (np.array(r), np.array(vobs), np.array(errv),
96|           np.array(vgas), np.array(vdisk), np.array(vbul))
97|
98|
99| def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
100|     r_m = r_kpc * kpc_to_m
101|     conv = kms2_to_ms2 / r_m
102|     gN = (np.abs(Vgas) * Vgas * conv
103|          + Yd * np.abs(Vdisk) * Vdisk * conv
104|          + Yb * np.abs(Vbul) * Vbul * conv)
105|     return np.abs(gN)
106|
107|
108| # =====
109| # 各銀河の特性量計算
110| # =====
111| def galaxy_properties(r_kpc, Vobs, errV, Vgas, Vdisk, Vbul, vflat, hR, Yd=0.5, Yb=0.7):
112|     """銀河の各種特性量を計算"""
113|     gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd, Yb)
114|
115|     # バリオン速度成分
116|     Vbar_sq = (np.abs(Vgas)*Vgas + Yd*np.abs(Vdisk)*Vdisk + Yb*np.abs(Vbul)*Vbul)
117|
118|     # バルジ割合: f_bul = Vbul^2/(Vbar^2) の中央値
119|     Vbar_sq_abs = np.abs(Vbar_sq)
120|     valid = Vbar_sq_abs > 0
121|     if valid.sum() > 0:
122|         f_bul = np.median(Yb * Vbul[valid]**2 / Vbar_sq_abs[valid])
123|     else:
124|         f_bul = 0.0
125|
126|     # ガス割合: f_gas = Vgas^2/(Vbar^2)
127|     if valid.sum() > 0:
128|         f_gas = np.median(Vgas[valid]**2 / Vbar_sq_abs[valid])
129|     else:
130|         f_gas = 0.0
131|
132|     # M_bar の推定 (外側のg_Nから): GM_bar ~ r^2 x g_N(r_out)
133|     n_out = max(3, len(r_kpc) // 3)
134|     r_out_m = r_kpc[-n_out:] * kpc_to_m
135|     gN_out = gN[-n_out:]
136|     GM_bar = np.median(r_out_m**2 * gN_out) # m^3/s^2
137|
138|     # 有効面密度: Σ_eff = M_bar / (2π hR^2)
139|     hR_m = hR * kpc_to_m
140|     Sigma_eff = GM_bar / (2 * np.pi * hR_m**2) # m/s^2 (加速度次元)
141|     # -> 真の面密度は Σ = M/(2πhR^2) [kg/m^2] だが
142|     # ここでは GxΣ 次元で統一 (m/s^2)
143|
144|     # ディスク集中度: disk concentration = V_disk_peak / vflat
145|     Vdisk_peak = np.max(np.abs(Vdisk)) * np.sqrt(Yd)
146|     disk_conc = Vdisk_peak / vflat if vflat > 0 else np.nan
147|
148|     # 「指数ディスク理想比」: M_bar_ideal = (vflat^2/G) x hR x (2π)
149|     # 実際のM_bar との比 = M_bar / M_bar_ideal

```

```

150| # 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
151| vflat_ms = vflat * 1e3
152| GM_bar_ideal = vflat_ms**2 * hR_m * 2 * np.pi # 次元注意: これは雑
153| # より正確: BTFR gc ~ vflat^4/(GM_bar)
154| # 指数ディスク: gc ~ vflat^4/(Gx2πΣ_0hR^2) = vflat^4/(2πGΣ_0hR^2)
155| # proxy: GΣ_0 = vflat^2/hR -> gc_ideal = vflat^4/(2πxvflat^2xhR) = vflat^2/(2πhR)
156| # 実際: gc_btfr = vflat^4/GM_bar
157| # 比: gc_btfr/gc_ideal = vflat^2x2πhR/GM_bar -> 指数ディスクからの乖離
158|
159| # 深MONDフィット
160| def fit_gc_deep(gc_trial):
161|     g_eff = np.sqrt(gN * gc_trial)
162|     V_pred = np.sqrt(r_kpc * kpc_to_m * g_eff) * 1e-3
163|     n_f = max(5, len(r_kpc) // 2)
164|     V_f = Vobs[-n_f:]
165|     V_p = V_pred[-n_f:]
166|     e_f = errV[-n_f:]
167|     e_f = np.where(e_f > 0, e_f, np.median(Vobs) * 0.1)
168|     return np.sum(((V_f - V_p) / e_f)**2)
169|
170| res = minimize_scalar(fit_gc_deep, bounds=(1e-12, 1e-8), method='bounded')
171| gc_deep = 10**res.x if res.x > -30 else res.x
172| # minimize_scalar returns the actual value, not log
173| gc_deep = res.x # 実はboundsで直接値を渡しているの...
174| # 修正: minimize_scalarのboundsは直接gcの値
175| res2 = minimize_scalar(lambda lgc: fit_gc_deep(10**lgc),
176|                        bounds=(-12, -8), method='bounded')
177| gc_deep = 10**res2.x
178|
179| # x_outer (深MOND度)
180| x_outer = np.median(gN[-n_out:] / gc_deep) if gc_deep > 0 else np.nan
181|
182| # η の計算: gc_obs = η^2 x a_0 x proxy -> η = gc_obs / sqrt(a_0 x proxy)
183| proxy = vflat_ms**2 / hR_m
184| # gc_obsは後で使う
185|
186| return {
187|     "gN": gN,
188|     "f_bul": f_bul,
189|     "f_gas": f_gas,
190|     "GM_bar": GM_bar,
191|     "Sigma_eff": Sigma_eff,
192|     "disk_conc": disk_conc,
193|     "gc_deep": gc_deep,
194|     "x_outer": x_outer,
195|     "proxy": proxy,
196| }
197|
198|
199| # =====
200| # 偏相関関数
201| # =====
202| def partial_corr(x, y, controls):
203|     """x, y の偏相関 (controls をリストで制御) """
204|     if len(controls) == 0:
205|         return pearsonr(x, y)
206|
207|     # 各変数からcontrolsを回帰で除去
208|     C = np.column_stack(controls + [np.ones(len(x))])
209|
210|     coef_x, _, _ = lstsq(C, x, rcond=None)
211|     res_x = x - C @ coef_x
212|
213|     coef_y, _, _ = lstsq(C, y, rcond=None)
214|     res_y = y - C @ coef_y
215|
216|     return pearsonr(res_x, res_y)
217|
218|
219| # =====
220| # メイン
221| # =====
222| def main():
223|     print("=" * 70)
224|     print("hR偏相関の分解: rho=-0.312 の物理的起源")
225|     print("=" * 70)
226|
227|     gc_table = load_gc_table(GC_CSV)
228|     print(f"sparc_gc.csv: {len(gc_table)} galaxies")
229|
230|     records = []
231|     for gal_name, info in gc_table.items():
232|         rotmod = load_rotmod(gal_name, ROTMOD_DIR)
233|         if rotmod is None:
234|             continue
235|         r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
236|         if len(r_kpc) < 8:

```

```

237|         continue
238|
239|         gc_obs = info["gc_obs"]
240|         if gc_obs > 1e-5:
241|             gc_obs_ms2 = gc_obs * a0
242|         else:
243|             gc_obs_ms2 = gc_obs
244|
245|         hR = info["hR"]
246|         vflat = info["vflat"]
247|         Yd = info.get("Yd", Yd_default)
248|
249|         props = galaxy_properties(r_kpc, Vobs, errV, Vgas, Vdisk, Vbul,
250|                                 vflat, hR, Yd=Yd)
251|
252|         #  $\eta$ 
253|         gc_geom = np.sqrt(a0 * props["proxy"])
254|         eta = gc_obs_ms2 / gc_geom if gc_geom > 0 else np.nan
255|
256|         records.append({
257|             "name": gal_name,
258|             "gc_obs": gc_obs_ms2,
259|             "gc_deep": props["gc_deep"],
260|             "hR": hR,
261|             "vflat": vflat,
262|             "Yd": Yd,
263|             "f_bul": props["f_bul"],
264|             "f_gas": props["f_gas"],
265|             "GM_bar": props["GM_bar"],
266|             "Sigma_eff": props["Sigma_eff"],
267|             "disk_conc": props["disk_conc"],
268|             "x_outer": props["x_outer"],
269|             "proxy": props["proxy"],
270|             "eta": eta,
271|         })
272|
273|         N = len(records)
274|         print(f"Processed: {N}")
275|
276|         # 配列化
277|         gc_obs = np.array([r["gc_obs"] for r in records])
278|         gc_deep = np.array([r["gc_deep"] for r in records])
279|         hR = np.array([r["hR"] for r in records])
280|         vflat = np.array([r["vflat"] for r in records])
281|         Yd = np.array([r["Yd"] for r in records])
282|         f_bul = np.array([r["f_bul"] for r in records])
283|         f_gas = np.array([r["f_gas"] for r in records])
284|         GM_bar = np.array([r["GM_bar"] for r in records])
285|         disk_conc = np.array([r["disk_conc"] for r in records])
286|         x_outer = np.array([r["x_outer"] for r in records])
287|         proxy = np.array([r["proxy"] for r in records])
288|         eta = np.array([r["eta"] for r in records])
289|
290|         log_gc = np.log10(gc_obs)
291|         log_gcd = np.log10(gc_deep)
292|         log_hR = np.log10(hR)
293|         log_vf = np.log10(vflat)
294|         log_Yd = np.log10(Yd)
295|         log_eta = np.log10(np.abs(eta))
296|         log_proxy = np.log10(proxy)
297|
298|         # 有効マスク
299|         m = (np.isfinite(log_gc) & np.isfinite(log_hR) & np.isfinite(log_vf)
300|             & np.isfinite(log_gcd) & np.isfinite(log_eta) & np.isfinite(log_Yd)
301|             & np.isfinite(x_outer) & np.isfinite(f_bul) & np.isfinite(f_gas)
302|             & np.isfinite(disk_conc) & (eta > 0))
303|
304|         print(f"Valid for analysis: {m.sum()}")
305|
306|         # =====
307|         # Test 0: ベースライン偏相関の再確認
308|         # =====
309|         print("\n" + "=" * 50)
310|         print("Test 0: ベースライン偏相関")
311|         print("=" * 50)
312|
313|         rho_raw, p_raw = pearsonr(log_hR[m], log_gc[m])
314|         print(f" raw: rho(hR, gc) = {rho_raw:.4f}, p = {p_raw:.3e}")
315|
316|         rho_pv, p_pv = partial_corr(log_hR[m], log_gc[m], [log_vf[m]])
317|         print(f" |vflat: rho(hR, gc | vflat) = {rho_pv:.4f}, p = {p_pv:.3e}")
318|
319|         rho_pvy, p_pvy = partial_corr(log_hR[m], log_gc[m], [log_vf[m], log_Yd[m]])
320|         print(f" |vflat,Yd: rho = {rho_pvy:.4f}, p = {p_pvy:.3e}")
321|
322|         # =====
323|         # Test 1: Yd vs hR の相関

```

```

324| # =====
325| print("≠n" + "=" * 50)
326| print("Test 1: Yd vs hR の相関")
327| print("=" * 50)
328|
329| rho_Yd_hR, p_Yd_hR = pearsonr(log_hR[m], log_Yd[m])
330| print(f" rho(log Yd, log hR) = {rho_Yd_hR:+.4f}, p = {p_Yd_hR:.3e}")
331|
332| # Yd が全て同じ値 (0.5) なら相関は出ない
333| Yd_unique = np.unique(Yd[m])
334| print(f" Yd unique values: {len(Yd_unique)} (min={Yd[m].min():.3f}, max={Yd[m].max():.3f}")
335|
336| if len(Yd_unique) &lt;= 1:
337|     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")
338|     print(" sparc_gc.csv に個別Yd情報がないため、SPS依存の検証は保留")
339|     Yd_varies = False
340| else:
341|     Yd_varies = True
342|     # η vs hR をYd制御
343|     rho_eta_hR, p_eta_hR = partial_corr(log_hR[m], log_eta[m], [log_Yd[m]])
344|     print(f" rho(eta, hR | Yd) = {rho_eta_hR:+.4f}, p = {p_eta_hR:.3e}")
345|
346| # =====
347| # Test 2: η vs hR (直接)
348| # =====
349| print("≠n" + "=" * 50)
350| print("Test 2: eta vs hR (直接+制御) ")
351| print("=" * 50)
352|
353| rho_eh, p_eh = pearsonr(log_hR[m], log_eta[m])
354| print(f" raw: rho(eta, hR) = {rho_eh:+.4f}, p = {p_eh:.3e}")
355|
356| # vflat 制御
357| rho_ehv, p_ehv = partial_corr(log_hR[m], log_eta[m], [log_vf[m]])
358| print(f" |vflat: rho(eta, hR | vflat) = {rho_ehv:+.4f}, p = {p_ehv:.3e}")
359|
360| # proxy 制御
361| rho_ehp, p_ehp = partial_corr(log_hR[m], log_eta[m], [log_proxy[m]])
362| print(f" |proxy: rho(eta, hR | proxy) = {rho_ehp:+.4f}, p = {p_ehp:.3e}")
363|
364| # vflat + x_outer 制御
365| rho_ehvx, p_ehvx = partial_corr(log_hR[m], log_eta[m],
366|                                 [log_vf[m], np.log10(x_outer[m] + 1e-10)])
367| print(f" |vflat,x: rho(eta, hR | vflat,x) = {rho_ehvx:+.4f}, p = {p_ehvx:.3e}")
368|
369| # =====
370| # Test 3: gc残差 = log(gc_obs/gc_deep) のhR依存性
371| # =====
372| print("≠n" + "=" * 50)
373| print("Test 3: gc残差 = log(gc_obs/gc_deep) の分解")
374| print("=" * 50)
375|
376| resid = log_gc[m] - log_gcd[m]
377| print(f" 残差統計: median = {np.median(resid):.4f}, std = {np.std(resid):.4f}")
378|
379| rho_rh, p_rh = pearsonr(log_hR[m], resid)
380| print(f" rho(残差, hR) = {rho_rh:+.4f}, p = {p_rh:.3e}")
381|
382| rho_rv, p_rv = pearsonr(log_vf[m], resid)
383| print(f" rho(残差, vflat) = {rho_rv:+.4f}, p = {p_rv:.3e}")
384|
385| rho_rx, p_rx = pearsonr(np.log10(x_outer[m] + 1e-10), resid)
386| print(f" rho(残差, x_outer) = {rho_rx:+.4f}, p = {p_rx:.3e}")
387|
388| rho_rf, p_rf = pearsonr(f_bul[m], resid)
389| print(f" rho(残差, f_bul) = {rho_rf:+.4f}, p = {p_rf:.3e}")
390|
391| rho_rg, p_rg = pearsonr(f_gas[m], resid)
392| print(f" rho(残差, f_gas) = {rho_rg:+.4f}, p = {p_rg:.3e}")
393|
394| rho_rd, p_rd = pearsonr(disk_conc[m], resid)
395| print(f" rho(残差, disk_conc) = {rho_rd:+.4f}, p = {p_rd:.3e}")
396|
397| # 残差のhR偏相関 (vflat制御)
398| rho_rhv, p_rhv = partial_corr(log_hR[m], resid, [log_vf[m]])
399| print(f" rho(残差, hR | vflat) = {rho_rhv:+.4f}, p = {p_rhv:.3e}")
400|
401| # =====
402| # Test 4: 指数ディスク近似の乖離度
403| # =====
404| print("≠n" + "=" * 50)
405| print("Test 4: 指数ディスク近似の乖離")
406| print("=" * 50)
407|
408| # 指数ディスク理想: gc_ideal = vflat^4 / (2π G Σ_0 hR^2)
409| # ここで GΣ_0 = proxy = vflat^2/hR を使うと gc_ideal = vflat^2/(2π hR)
410| # 深MONDフィットの gc_deep との比: gc_deep / gc_ideal

```

```

411| vflat_ms = vflat * 1e3
412| hR_m = hR * kpc_to_m
413| gc_ideal = vflat_ms[m]**2 / (2 * np.pi * hR_m[m])
414| deviation = np.log10(gc_deep[m] / gc_ideal)
415|
416| print(f" log(gc_deep / gc_ideal): median = {np.median(deviation):.4f}, "
417|       f"std = {np.std(deviation):.4f}")
418|
419| rho_dh, p_dh = pearsonr(log_hR[m], deviation)
420| print(f" rho(deviation, hR) = {rho_dh:.4f}, p = {p_dh:.3e}")
421|
422| rho_dv, p_dv = pearsonr(log_vf[m], deviation)
423| print(f" rho(deviation, vflat) = {rho_dv:.4f}, p = {p_dv:.3e}")
424|
425| # =====
426| # Test 5: バルジ・ガス割合の効果
427| # =====
428| print("#n" + "=" * 50)
429| print("Test 5: バルジ・ガス割合の効果")
430| print("=" * 50)
431|
432| # f_bul, f_gas vs hR
433| rho_bh, p_bh = pearsonr(log_hR[m], f_bul[m])
434| print(f" rho(f_bul, hR) = {rho_bh:.4f}, p = {p_bh:.3e}")
435|
436| rho_gh, p_gh = pearsonr(log_hR[m], f_gas[m])
437| print(f" rho(f_gas, hR) = {rho_gh:.4f}, p = {p_gh:.3e}")
438|
439| # gc のhR偏相関に f_bul を追加制御
440| rho_pvb, p_pvb = partial_corr(log_hR[m], log_gc[m],
441|                               [log_vf[m], f_bul[m]])
442| print(f" rho(gc, hR | vflat, f_bul) = {rho_pvb:.4f}, p = {p_pvb:.3e}")
443|
444| # gc のhR偏相関に f_gas を追加制御
445| rho_pvg, p_pvg = partial_corr(log_hR[m], log_gc[m],
446|                               [log_vf[m], f_gas[m]])
447| print(f" rho(gc, hR | vflat, f_gas) = {rho_pvg:.4f}, p = {p_pvg:.3e}")
448|
449| # 全て制御
450| rho_all, p_all = partial_corr(log_hR[m], log_gc[m],
451|                               [log_vf[m], f_bul[m], f_gas[m],
452|                                np.log10(x_outer[m] + 1e-10)])
453| print(f" rho(gc, hR | vflat, f_bul, f_gas, x) = {rho_all:.4f}, p = {p_all:.3e}")
454|
455| # =====
456| # Test 6: 段階的偏相関 (逐次統制)
457| # =====
458| print("#n" + "=" * 50)
459| print("Test 6: 段階的偏相関 (hR効果の逐次分解)")
460| print("=" * 50)
461|
462| log_x = np.log10(x_outer[m] + 1e-10)
463|
464| steps = [
465|     ("raw", []),
466|     ("vflat", [log_vf[m]]),
467|     ("vflat,x_outer", [log_vf[m], log_x]),
468|     ("vflat,x,f_bul", [log_vf[m], log_x, f_bul[m]]),
469|     ("vflat,x,f_bul,f_gas", [log_vf[m], log_x, f_bul[m], f_gas[m]]),
470|     ("vflat,x,f_bul,f_gas,disk_conc",
471|      [log_vf[m], log_x, f_bul[m], f_gas[m], disk_conc[m]]),
472| ]
473|
474| print(f" {'制御変数':<40s} {'rho':>8s} {'p':>12s} {'|rho|変化':>10s}")
475| print(" " + "-" * 72)
476| prev_rho = None
477| for label, controls in steps:
478|     valid_controls = []
479|     for c in controls:
480|         if np.isfinite(c).all():
481|             valid_controls.append(c)
482|         else:
483|             mc = np.isfinite(c)
484|             if mc.sum() > 0.9 * len(c):
485|                 # ほぼ有効 -> NaNを中央値で埋める
486|                 c_filled = c.copy()
487|                 c_filled[~np.isfinite(c)] = np.nanmedian(c)
488|                 valid_controls.append(c_filled)
489|
490|     rho, p = partial_corr(log_hR[m], log_gc[m], valid_controls)
491|     change = f"{abs(rho) - abs(prev_rho):.4f}" if prev_rho is not None else "--"
492|     print(f" {label:<40s} {rho:.4f} p={p:.3e} {change:>10s}")
493|     prev_rho = rho
494|
495| # =====
496| # Test 7: 深MOND構造からの期待hR指数の導出
497| # =====

```

```

498| print("\n" + "=" * 50)
499| print("Test 7: hR指数の理論値 vs 観測値")
500| print("=" * 50)
501|
502| # 深MOND BTFR: gc = vflat^4 / (G M_bar)
503| # M_bar = L x Yd, ディスク luminosity L ~ Σ_0 L x hR^2
504| # Σ_0 = Yd x Σ_0 L -&gt; M_bar = Σ_0 / Yd x hR^2 x 2π... &lt;- Yd が入る
505| # gc = vflat^4 / (2π G Σ_0 hR^2)
506| # proxy = GΣ_0 = vflat^2/hR (定義)
507| # -&gt; gc = vflat^4 / (2π x vflat^2 x hR) = vflat^2/(2πhR) &lt;- hR^(-1)
508| # しかし幾何平均法則は gc ~ proxy^0.5 = (vflat^2/hR)^0.5 &lt;- hR^(-0.5)
509| #
510| # 矛盾? いや:
511| # gc = vflat^4/(GM_bar) は gc ~ vflat^4/M_bar
512| # gc ~ (proxy)^0.5 は gc ~ vflat/hR^0.5
513| # これらが同時成立 -&gt; vflat^4/M_bar ~ vflat/hR^0.5
514| # -&gt; vflat^3 ~ M_bar/hR^0.5
515| # -&gt; M_bar ~ vflat^3 x hR^0.5
516| # 一方 M_bar ~ Σ_0 hR^2 で Σ_0 = vflat^2/(G hR) (proxy定義)
517| # -&gt; M_bar ~ (vflat^2/hR) x hR^2 = vflat^2 x hR
518| # -&gt; vflat^3 x hR^0.5 ~ vflat^2 x hR -&gt; vflat ~ hR^0.5
519| # Test 6 で slope=0.545 -&gt; 自己無撞着
520|
521| # つまり:
522| # BTFR単独: gc ~ vflat^4/M_bar -&gt; hR指数は M_bar の hR 依存性で決まる
523| # M_bar ~ vflat^2 x hR (proxy定義から) -&gt; gc ~ vflat^2/hR -&gt; hR^(-1)
524| # しかし vflat ~ hR^0.5 を代入すると gc ~ hR/hR = hR^0 ???
525|
526| # もう少し丁寧に:
527| # gc ~ (vflat^2/hR)^alpha で, alpha の値を見る
528| # gc = vflat^4/(GM_bar), M_bar = 2π Σ_0 hR^2, GΣ_0 = vflat^2/hR
529| # -&gt; gc = vflat^4/(2π x (vflat^2/hR) x hR^2) = vflat^4/(2π vflat^2 hR) = vflat^2/(2πhR)
530| # -&gt; gc ~ vflat^2/hR = proxy^1
531| # 深MONDのBTFRでは alpha=1 のはず!
532| # しかし gc_deep フィットでは alpha=0.47 が出ている
533|
534| # 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、
535| # 大rでの BTFR とは異なる。有限rでの g_N(r) の構造が入る。
536|
537| print(" 理論的分析:")
538| print(" BTFR厳密 (大r極限): gc = vflat^4/(GM_bar)")
539| print(" M_bar = 2π Σ_0 hR^2, G Σ_0 = vflat^2/hR")
540| print(" -&gt; gc = vflat^2/(2π hR) -&gt; proxy^1.0")
541| print("")
542| print(" gc_deep フィット (回転曲線全体): alpha = 0.47")
543| 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")
544| print("")
545|
546| # 直接検証: gc_btfr (大r極限) vs gc_deep (全体フィット)
547| n_out = 3
548| gc_btfr_arr = []
549| for rec in records:
550|     rm = load_rotmod(rec["name"], ROTMOD_DIR)
551|     if rm is None:
552|         gc_btfr_arr.append(np.nan)
553|         continue
554|     r_kpc_i, _, Vgas_i, Vdisk_i, Vbul_i = rm
555|     gN_i = compute_gN(r_kpc_i, Vgas_i, Vdisk_i, Vbul_i)
556|     vf_ms = rec["vflat"] * 1e3
557|     n_o = max(3, len(r_kpc_i) // 3)
558|     r_out = r_kpc_i[-n_o:] * kpc_to_m
559|     gN_out = gN_i[-n_o:]
560|     valid_o = gN_out &gt; 0
561|     if valid_o.sum() &gt; 0:
562|         gc_btfr_i = np.median(vf_ms**4 / (r_out[valid_o]**2 * gN_out[valid_o]))
563|         gc_btfr_arr.append(gc_btfr_i)
564|     else:
565|         gc_btfr_arr.append(np.nan)
566|
567| gc_btfr = np.array(gc_btfr_arr)
568| log_gcb = np.log10(gc_btfr)
569| m_btfr = m & amp; np.isfinite(log_gcb)
570|
571| if m_btfr.sum() &gt; 10:
572|     sl_btfr = linregress(log_proxy[m_btfr], log_gcb[m_btfr])
573|     sl_deep = linregress(log_proxy[m], log_gcb[m])
574|     print(f" gc_btfr (大r) vs proxy: alpha = {sl_btfr.slope:.4f} +/- {sl_btfr.stderr:.4f}")
575|     print(f" gc_deep (全体) vs proxy: alpha = {sl_deep.slope:.4f} +/- {sl_deep.stderr:.4f}")
576|     print(f" 差: {sl_btfr.slope - sl_deep.slope:.4f}")
577|     print(f" -&gt; 有限の g_N 構造が alpha を {sl_btfr.slope:.3f} -&gt; {sl_deep.slope:.3f} に変える")
578|
579| # =====
580| # SUMMARY
581| # =====
582| print("\n" + "=" * 70)
583| print("SUMMARY: hR偏相関の分解")
584| 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) < 0.1 and abs(rho_eh) < 0.1:
599|     print(f"%n -&gt; 残差にもetaにもhR効果はほぼなし")
600|     print(f" -&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) > 0.1:
602|     print(f"%n -&gt; gc残差にhR効果あり (rho={rho_rh:+.3f}")
603|     print(f" -&gt; 有限x補正 or ディスク形状効果が寄与")
604|
605| if abs(rho_eh) > 0.15:
606|     print(f"%n -&gt; etaにhR依存性あり (rho={rho_eh:+.3f}")
607|     print(f" -&gt; Yd-hR相関 or 追加の物理的效果")
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

$x_{\text{outer}}$  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、 $\text{resid\_asymm}$ 、 $\text{gN\_curvature}$ 、 $\text{Vbar/Vobs}$ 、 $\text{flatness}$ 、 $\text{rmax/hR}$ ) で説明できるか検証する。

### 結果

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

### ソースコード

(580 行)

```
1| #!/usr/bin/env python3
2| """
3| sparc_hR_residual31.py
4| =====
5| hR偏相関の残り31% ( $\rho = -0.21$ ) の追求
6|
7| 既に否定: f_bul, f_gas, disk_conc, Yd(定数) -&gt; 効果なし
8| x_outer が 69% を説明済み
9|
10| 新しい候補:
11| (i) g_N プロファイルの高次形状パラメータ (曲率、勾配比)
12| (ii) rmax/hR (観測範囲の影響)
13| (iii) g_N 遷移幅 ( $g_N \sim a_0$  付近の急峻さ)
14| (iv) Vdisk プロファイル形状 (指数ディスクからの乖離度)
15| (v) 深MONDフィットの残差構造 (内側vs外側の系統差)
16|
17| テスト:
18| Test 1: 新しい形状パラメータの定義と計算
19| Test 2: 各パラメータ vs hR の相関
20| Test 3: 段階的偏相関 ( $x_{\text{outer}}$  + 新パラメータ)
21| Test 4: gc残差 (obs-deep)の新パラメータ依存性
22| Test 5: 「完全モデル」-- 何%まで説明可能か
23|
24| 実行: uv run --with scipy --with matplotlib python sparc_hR_residual31.py
25| """
26|
27| import numpy as np
28| from scipy.optimize import minimize_scalar
29| from scipy.stats import pearsonr, Linregress
30| from numpy.linalg import lstsq
31| from pathlib import Path
32| import csv
33| import sys
34|
35| # =====
36| a0 = 1.2e-10
37| kpc_to_m = 3.0857e19
38| kms2_to_ms2 = 1.0e6
39| Yd_default = 0.5
40| Yb_default = 0.7
41|
42| 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")
43| ROTMOD_DIR = BASE / "Rotmod_LTG"
44| GC_CSV = BASE / "sparc_gc.csv"
45|
46|
47| def load_gc_table(path):
48|     data = {}
49|     with open(path, "r") as f:
50|         reader = csv.DictReader(f)
51|         for row in reader:
52|             name = row["Galaxy"].strip()
53|             data[name] = {
54|                 "gc_obs": float(row["gc"]),
55|                 "vflat": float(row["Vflat"]),
56|                 "hR": float(row["hR"]),
57|             }
58|     return data
59|
60|
61| def load_rotmod(galaxy_name, rotmod_dir):
62|     fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
63|     if not fpath.exists():
```

```

64|         return None
65|     r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
66|     with open(fpath, "r") as f:
67|         for line in f:
68|             line = line.strip()
69|             if not line or line.startswith("#"):
70|                 continue
71|             parts = line.split()
72|             if len(parts) < 6:
73|                 continue
74|             r.append(float(parts[0]))
75|             vobs.append(float(parts[1]))
76|             errv.append(float(parts[2]))
77|             vgas.append(float(parts[3]))
78|             vdisk.append(float(parts[4]))
79|             vbul.append(float(parts[5]))
80|     return (np.array(r), np.array(vobs), np.array(errv),
81|           np.array(vgas), np.array(vdisk), np.array(vbul))
82|
83|
84| def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
85|     r_m = r_kpc * kpc_to_m
86|     conv = kms2_to_ms2 / r_m
87|     gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
88|          + Yb*np.abs(Vbul)*Vbul*conv)
89|     return np.abs(gN)
90|
91|
92| def partial_corr(x, y, controls):
93|     if len(controls) == 0:
94|         return pearsonr(x, y)
95|     C = np.column_stack(controls + [np.ones(len(x))])
96|     coef_x, _, _ = lstsq(C, x, rcond=None)
97|     res_x = x - C @ coef_x
98|     coef_y, _, _ = lstsq(C, y, rcond=None)
99|     res_y = y - C @ coef_y
100|     return pearsonr(res_x, res_y)
101|
102|
103| # =====
104| # 形状パラメータの計算
105| # =====
106| def compute_shape_params(r_kpc, Vobs, errV, Vgas, Vdisk, Vbul,
107|                          vflat, hR, gc_obs_ms2):
108|     """g_Nプロファイルの高次形状パラメータ"""
109|     gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)
110|     N = len(r_kpc)
111|     params = {}
112|
113|     # --- (1) rmax/hR: 観測範囲 ---
114|     params["rmax_hR"] = r_kpc[-1] / hR if hR > 0 else np.nan
115|
116|     # --- (2) g_N プロファイルの対数勾配 ---
117|     # d log(g_N) / d log(r) の中央値と変動
118|     log_r = np.log10(r_kpc)
119|     log_gN = np.log10(gN + 1e-30)
120|     valid = np.isfinite(log_gN) && np.isfinite(log_r) && (gN > 0)
121|
122|     if valid.sum() >= 5:
123|         dr = np.diff(log_r[valid])
124|         dg = np.diff(log_gN[valid])
125|         mask_dr = np.abs(dr) > 1e-10
126|         if mask_dr.sum() >= 3:
127|             grad = dg[mask_dr] / dr[mask_dr]
128|             r_mid = (log_r[valid[:-1]] + log_r[valid[1:]]) [mask_dr] / 2.0
129|
130|             # 外側半分勾配
131|             n_out = max(1, len(grad) // 2)
132|             params["grad_outer"] = np.median(grad[-n_out:])
133|
134|             # 内側半分勾配
135|             params["grad_inner"] = np.median(grad[:n_out])
136|
137|             # 勾配比 (inner/outer) -- ディスク形状の指標
138|             if abs(params["grad_outer"]) > 0.01:
139|                 params["grad_ratio"] = params["grad_inner"] / params["grad_outer"]
140|             else:
141|                 params["grad_ratio"] = np.nan
142|
143|             # 勾配の変動 (std) -- プロファイルの非単調性
144|             params["grad_std"] = np.std(grad)
145|         else:
146|             params["grad_outer"] = np.nan
147|             params["grad_inner"] = np.nan
148|             params["grad_ratio"] = np.nan
149|             params["grad_std"] = np.nan
150|     else:

```

```

151|     params["grad_outer"] = np.nan
152|     params["grad_inner"] = np.nan
153|     params["grad_ratio"] = np.nan
154|     params["grad_std"] = np.nan
155|
156| # --- (3) 遷移幅: gN が gc 付近を横切る半径範囲 ---
157| # g_N/gc が 0.3 -> 3.0 を横切る r の幅 (hR単位)
158| gc = gc_obs_ms2
159| if gc > 0:
160|     x_arr = gN / gc
161|     r_lo = np.nan # x = 3.0 の位置 (内側)
162|     r_hi = np.nan # x = 0.3 の位置 (外側)
163|
164|     for i in range(N-1):
165|         if x_arr[i] >= 3.0 and x_arr[i+1] < 3.0:
166|             # 線形補間
167|             frac = (3.0 - x_arr[i+1]) / (x_arr[i] - x_arr[i+1])
168|             r_lo = r_kpc[i+1] + frac * (r_kpc[i] - r_kpc[i+1])
169|             break
170|
171|     for i in range(N-1):
172|         if x_arr[i] >= 0.3 and x_arr[i+1] < 0.3:
173|             frac = (0.3 - x_arr[i+1]) / (x_arr[i] - x_arr[i+1])
174|             r_hi = r_kpc[i+1] + frac * (r_kpc[i] - r_kpc[i+1])
175|             break
176|
177|     if np.isfinite(r_lo) and np.isfinite(r_hi) and r_hi > r_lo:
178|         params["transition_width"] = (r_hi - r_lo) / hR
179|         params["transition_center"] = (r_hi + r_lo) / 2.0 / hR
180|     else:
181|         params["transition_width"] = np.nan
182|         params["transition_center"] = np.nan
183| else:
184|     params["transition_width"] = np.nan
185|     params["transition_center"] = np.nan
186|
187| # --- (4) Vdisk プロファイル形状: 指数ディスクからの乖離 ---
188| # 指数ディスク理論: Vdisk(r) ~ r x [I_0K_0 - I_1K_1]^(1/2)
189| # 乖離度 = Vdisk実測のピーク位置 / (2.2 x hR)
190| Vdisk_abs = np.abs(Vdisk)
191| if np.max(Vdisk_abs) > 0:
192|     idx_peak = np.argmax(Vdisk_abs)
193|     r_peak = r_kpc[idx_peak]
194|     params["peak_ratio"] = r_peak / (2.2 * hR) # 理想 = 1.0
195|
196|     # Vdisk の「形状集中度」: Vdisk(hR) / Vdisk_peak
197|     # hR での Vdisk を補間
198|     if r_kpc[-1] > hR and r_kpc[0] < hR:
199|         V_at_hR = np.interp(hR, r_kpc, Vdisk_abs)
200|         params["shape_conc"] = V_at_hR / np.max(Vdisk_abs)
201|     else:
202|         params["shape_conc"] = np.nan
203| else:
204|     params["peak_ratio"] = np.nan
205|     params["shape_conc"] = np.nan
206|
207| # --- (5) 深MONDフィット残差の内外比 ---
208| # gc_deep でフィットした V_pred の残差を内側 vs 外側で比較
209| def fit_gc_deep_full(r, gN, Vobs, errV):
210|     def chi2(lgc):
211|         gc = 10*lgc
212|         g_eff = np.sqrt(gN * gc)
213|         V_pred = np.sqrt(r * kpc_to_m * g_eff) * 1e-3
214|         e = np.where(errV > 0, errV, np.median(Vobs) * 0.1)
215|         return np.sum(((Vobs - V_pred) / e)**2)
216|     res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
217|     return 10**res.x
218|
219| gc_deep = fit_gc_deep_full(r_kpc, gN, Vobs, errV)
220| params["gc_deep"] = gc_deep
221|
222| g_eff_deep = np.sqrt(gN * gc_deep)
223| V_deep = np.sqrt(r_kpc * kpc_to_m * g_eff_deep) * 1e-3
224| e = np.where(errV > 0, errV, np.median(Vobs) * 0.1)
225| frac_resid = (Vobs - V_deep) / e
226|
227| n_half = N // 2
228| if n_half >= 3:
229|     params["resid_inner"] = np.median(frac_resid[:n_half])
230|     params["resid_outer"] = np.median(frac_resid[n_half:])
231|     params["resid_asymm"] = params["resid_inner"] - params["resid_outer"]
232| else:
233|     params["resid_inner"] = np.nan
234|     params["resid_outer"] = np.nan
235|     params["resid_asymm"] = np.nan
236|
237| # --- (6) x_outer (再計算) ---

```

```

238| n_out = max(3, N // 3)
239| params["x_outer"] = np.median(gN[-n_out:] / gc_deep) if gc_deep > 0 else np.nan
240|
241| # --- (7) g_N の曲率 (2次微分的) ---
242| if valid.sum() >= 7:
243|     # log(gN) vs log(r) の2次フィット -&gt; 曲率
244|     try:
245|         coeffs = np.polyfit(log_r[valid], log_gN[valid], 2)
246|         params["gN_curvature"] = coeffs[0] # 2次係数
247|     except:
248|         params["gN_curvature"] = np.nan
249| else:
250|     params["gN_curvature"] = np.nan
251|
252| # --- (8) Vbar/Vobs のフラット到達度 ---
253| # 外側での Vbar/Vobs = 「暗黒物質」支配度
254| Vbar = np.sqrt(np.abs(
255|     np.abs(Vgas)*Vgas + Yd_default*np.abs(Vdisk)*Vdisk
256|     + Yb_default*np.abs(Vbul)*Vbul)) * np.sign(
257|     np.abs(Vgas)*Vgas + Yd_default*np.abs(Vdisk)*Vdisk
258|     + Yb_default*np.abs(Vbul)*Vbul + 1e-30)
259|
260| Vbar_abs = np.abs(Vbar)
261| ratio_out = np.median(Vbar_abs[-n_out:] / (Vobs[-n_out:] + 1e-10))
262| params["Vbar_Vobs_outer"] = ratio_out
263|
264| # --- (9) フラット到達度: Vobs の外側変動 ---
265| V_out = Vobs[-n_out:]
266| params["flatness"] = np.std(V_out) / np.mean(V_out) if np.mean(V_out) > 0 else np.nan
267|
268| return params
269|
270|
271| # =====
272| def main():
273|     print("=" * 70)
274|     print("hR偏相関の残り31%: 高次形状パラメータによる分解")
275|     print("=" * 70)
276|
277|     gc_table = load_gc_table(GC_CSV)
278|
279|     records = []
280|     for gal_name, info in gc_table.items():
281|         rotmod = load_rotmod(gal_name, ROTMOD_DIR)
282|         if rotmod is None:
283|             continue
284|         r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
285|         if len(r_kpc) < 8:
286|             continue
287|
288|         gc_obs = info["gc_obs"]
289|         gc_obs_ms2 = gc_obs * a0 if gc_obs > 1e-5 else gc_obs
290|         hR = info["hR"]
291|         vflat = info["vflat"]
292|
293|         params = compute_shape_params(r_kpc, Vobs, errV, Vgas, Vdisk, Vbul,
294|             vflat, hR, gc_obs_ms2)
295|
296|         vflat_ms = vflat * 1e3
297|         hR_m = hR * kpc_to_m
298|         proxy = vflat_ms**2 / hR_m
299|         gc_geom = np.sqrt(a0 * proxy)
300|         eta = gc_obs_ms2 / gc_geom if gc_geom > 0 else np.nan
301|
302|         records.append({
303|             "name": gal_name,
304|             "gc_obs": gc_obs_ms2,
305|             "gc_deep": params["gc_deep"],
306|             "hR": hR,
307|             "vflat": vflat,
308|             "eta": eta,
309|             **params,
310|         })
311|
312|     N = len(records)
313|     print(f"Processed: {N}")
314|
315|     # 配列化
316|     def arr(key):
317|         return np.array([r[key] for r in records], dtype=float)
318|
319|     gc_obs = arr("gc_obs")
320|     gc_deep = arr("gc_deep")
321|     hR_a = arr("hR")
322|     vflat_a = arr("vflat")
323|     eta_a = arr("eta")
324|

```

```

325| log_gc = np.log10(gc_obs)
326| log_gcd = np.log10(gc_deep)
327| log_hR = np.log10(hR_a)
328| log_vf = np.log10(vflat_a)
329| log_eta = np.log10(np.abs(eta_a))
330|
331| x_outer = arr("x_outer")
332| rmax_hR = arr("rmax_hR")
333| grad_outer = arr("grad_outer")
334| grad_inner = arr("grad_inner")
335| grad_ratio = arr("grad_ratio")
336| grad_std = arr("grad_std")
337| trans_width = arr("transition_width")
338| trans_center = arr("transition_center")
339| peak_ratio = arr("peak_ratio")
340| shape_conc = arr("shape_conc")
341| resid_asymm = arr("resid_asymm")
342| gN_curvature = arr("gN_curvature")
343| Vbar_Vobs = arr("Vbar_Vobs_outer")
344| flatness = arr("flatness")
345| f_bul = arr("f_bul") if "f_bul" in records[0] else np.zeros(N)
346|
347| # f_bul を再計算
348| f_bul_arr = []
349| f_gas_arr = []
350| dc_arr = []
351| for rec in records:
352|     rm = load_rotmod(rec["name"], ROTMOD_DIR)
353|     if rm is None:
354|         f_bul_arr.append(np.nan); f_gas_arr.append(np.nan); dc_arr.append(np.nan)
355|         continue
356|     _, Vo, _, Vg, Vd, Vb = rm
357|     Vbar_sq = np.abs(Vg)*Vg + Yd_default*np.abs(Vd)*Vd + Yb_default*np.abs(Vb)*Vb
358|     va = np.abs(Vbar_sq)
359|     v = va &gt; 0
360|     f_bul_arr.append(np.median(Yb_default*Vb[v]**2 / va[v]) if v.sum()&gt;0 else 0)
361|     f_gas_arr.append(np.median(Vg[v]**2 / va[v]) if v.sum()&gt;0 else 0)
362|     dc_arr.append(np.max(np.abs(Vd))*np.sqrt(Yd_default)/rec["vflat"] if rec["vflat"]&gt;0 else np.nan)
363|
364| f_bul = np.array(f_bul_arr)
365| f_gas = np.array(f_gas_arr)
366| disk_conc = np.array(dc_arr)
367|
368| # 有効マスク
369| log_x = np.log10(x_outer + 1e-10)
370| m = (np.isfinite(log_gc) & np.isfinite(log_hR) & np.isfinite(log_vf)
371|      & np.isfinite(log_x) & np.isfinite(log_gcd) & (eta_a &gt; 0)
372|      & np.isfinite(log_eta))
373|
374| print(f"Valid: {m.sum()}")
375|
376| # =====
377| # Test 1: 新パラメータの基本統計
378| # =====
379| print("≠n" + "=" * 50)
380| print("Test 1: 新パラメータの基本統計")
381| print("=" * 50)
382|
383| new_params = {
384|     "rmax/hR": rmax_hR,
385|     "grad_outer": grad_outer,
386|     "grad_inner": grad_inner,
387|     "grad_ratio": grad_ratio,
388|     "grad_std": grad_std,
389|     "trans_width": trans_width,
390|     "trans_center": trans_center,
391|     "peak_ratio": peak_ratio,
392|     "shape_conc": shape_conc,
393|     "resid_asymm": resid_asymm,
394|     "gN_curvature": gN_curvature,
395|     "Vbar/Vobs_out": Vbar_Vobs,
396|     "flatness": flatness,
397| }
398|
399| for name, arr_p in new_params.items():
400|     valid = np.isfinite(arr_p) & m
401|     if valid.sum() &gt; 10:
402|         print(f" {name:20s}: N={valid.sum():3d}, "
403|               f"median={np.median(arr_p[valid]):+.3f}, "
404|               f"std={np.std(arr_p[valid]):.3f}")
405|
406| # =====
407| # Test 2: 各パラメータ vs hR, gc の相関
408| # =====
409| print("≠n" + "=" * 50)
410| print("Test 2: パラメータ vs hR, gc の相関")
411| print("=" * 50)

```

```

412|
413| print(f" {'パラメータ':&lt;20s} {'rho(hR)':&lt;10s} {'p(hR)':&lt;12s} "
414|       f"{'rho(gc)':&lt;10s} {'p(gc)':&lt;12s}")
415| print(" " + "-" * 66)
416|
417| for name, arr_p in new_params.items():
418|     valid = np.isfinite(arr_p) & m
419|     if valid.sum() & 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 & 0.01 else ""
423|         flag_g = "*" if pg & 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':&lt;8s} {'p':&lt;12s} {'Δ|rho|':&lt;10s} {'N':&lt;5s}")
441| print(" " + "-" * 58)
442|
443| results = []
444| for name, arr_p in new_params.items():
445|     valid = np.isfinite(arr_p) & m
446|     if valid.sum() & 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() & 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():&lt;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) & 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" 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, _, _ in results:
496|     all_controls.append(arr_filled[m])
497|
498| try:

```

```

499|     rho_all, p_all = partial_corr(log_hr[m], log_gc[m], all_controls)
500|     print(f" rho = {rho_all:+.4f}, p = {p_all:.3e}")
501|     print(f" ベースからの削減: {abs(rho_base) - abs(rho_all):.4f}")
502|     print(f" vflat のみからの削減率: "
503|           f"{(0.472 - abs(rho_all))/0.472*100:.1f}% "
504|           f"(x_outer: 69%, 新パラメータ: {(abs(rho_base) - abs(rho_all))/0.472*100:.1f}%)")
505| except Exception as e:
506|     print(f" ERROR: {e}")
507|
508| # =====
509| # Test 6: gc残差のrmax/hR依存性 (系統効果チェック)
510| # =====
511| print("\n" + "=" * 50)
512| print("Test 6: 観測的系統効果の検証")
513| print("=" * 50)
514|
515| resid = log_gc[m] - log_gcd[m]
516|
517| # rmax/hR vs 残差
518| rmax_valid = np.isfinite(rmax_hr[m])
519| if rmax_valid.sum() > 10:
520|     rho_rm, p_rm = pearsonr(rmax_hr[m][rmax_valid], resid[rmax_valid])
521|     print(f" rho(gc残差, rmax/hR) = {rho_rm:+.4f}, p = {p_rm:.3e}")
522|
523|     # 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
524|     rmax_med = np.median(rmax_hr[m][rmax_valid])
525|     hi = rmax_hr[m][rmax_valid] > rmax_med
526|     lo = ~hi
527|     print(f" rmax/hR &gt; {rmax_med:.1f}: 残差 median = {np.median(resid[rmax_valid][hi]):.4f}")
528|     print(f" rmax/hR &lt; {rmax_med:.1f}: 残差 median = {np.median(resid[rmax_valid][lo]):.4f}")
529|
530| # flatness vs 残差
531| flat_valid = np.isfinite(flatness[m])
532| if flat_valid.sum() > 10:
533|     rho_fl, p_fl = pearsonr(flatness[m][flat_valid], resid[flat_valid])
534|     print(f" rho(gc残差, flatness) = {rho_fl:+.4f}, p = {p_fl:.3e}")
535|
536| # =====
537| # Test 7: rmax/hR 制限下での偏相関
538| # =====
539| print("\n" + "=" * 50)
540| print("Test 7: rmax/hR 制限下での偏相関")
541| print("=" * 50)
542|
543| for rmax_cut in [5, 8, 10, 15]:
544|     m_cut = m &amp; (rmax_hr > rmax_cut)
545|     if m_cut.sum() > 20:
546|         rho_c, p_c = partial_corr(log_hr[m_cut], log_gc[m_cut],
547|                                   [log_vf[m_cut], log_x[m_cut]])
548|         print(f" rmax/hR &gt; {rmax_cut:2d}: N={m_cut.sum():3d}, "
549|               f"rho(hR|vflat, x) = {rho_c:+.4f}, p = {p_c:.3e}")
550|
551| # =====
552| # SUMMARY
553| # =====
554| print("\n" + "=" * 70)
555| print("SUMMARY")
556| print("=" * 70)
557|
558| print(f"\n |vflat: rho = -0.472")
559| print(f" |vflat, x_outer: rho = {rho_base:+.4f} (x_outer が 69% 説明)")
560|
561| if len(results) > 3:
562|     print(f" |vflat, x, top3: rho = {rho_top3:+.4f} "
563|           f"(追加 {(abs(rho_base) - abs(rho_top3))/0.472*100:.1f}% 説明)")
564|
565| try:
566|     print(f" |全パラメータ: rho = {rho_all:+.4f} "
567|           f"(追加 {(abs(rho_base) - abs(rho_all))/0.472*100:.1f}% 説明)")
568| except:
569|     pass
570|
571| print(f"\n 残り31%の内訳:")
572| if len(results) > 1:
573|     for name, rho, delta, _ in results[:5]:
574|         pct = -delta / 0.472 * 100
575|         print(f" {name:<20s}: {pct:+.1f}%")
576|
577|
578| if __name__ == "__main__":
579|     main()
580|

```

## 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)

```
1| #!/usr/bin/env python3
2| """
3| sparc_gc_method_systematics.py
4| =====
5| gc決定手法の系統効果: TA3 tanh gc_obs vs 深MOND gc_deep
6|
7| 問題:
8| 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。
9| rmax/hR &gt; 8 で rho=-0.507 と強化 -&gt; 手法依存性の可能性。
10|
11| 仮説:
12| 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。
13| 両者の差  $\Delta = \log(gc\_obs/gc\_deep)$  が hR と相関し、
14| これが残り31%の原因である可能性。
15|
16| テスト:
17| Test 1:  $\Delta = \log(gc\_obs/gc\_deep)$  の hR 依存性
18| Test 2: gc_obs, gc_deep それぞれの hR 偏相関の比較
19| Test 3: rmax/hR サブセットでの系統的比較
20| Test 4: gc_deep で幾何平均法則を再フィット -&gt; alpha の値
21| Test 5: gc_deep ベースでの hR 偏相関 -&gt; 31%が消えるか
22| Test 6: 完全MOND gc_full でも同じ検証
23| Test 7: 「外側フィット」vs「全域フィット」の gc の差
24|
25| 実行: uv run --with scipy --with matplotlib python sparc_gc_method_systematics.py
26| """
27|
28| import numpy as np
29| from scipy.optimize import minimize_scalar, minimize
30| from scipy.stats import pearsonr, linregress
31| from numpy.linalg import lstsq
32| from pathlib import Path
33| import csv
34| import sys
35|
36| # =====
37| a0 = 1.2e-10
38| kpc_to_m = 3.0857e19
39| kms2_to_ms2 = 1.0e6
40| Yd_default = 0.5
41| Yb_default = 0.7
42|
43| 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")
44| ROTMOD_DIR = BASE / "Rotmod_LTG"
45| GC_CSV = BASE / "sparc_gc.csv"
46|
47|
48| def load_gc_table(path):
49|     data = {}
50|     with open(path, "r") as f:
51|         reader = csv.DictReader(f)
52|         for row in reader:
53|             name = row["Galaxy"].strip()
54|             data[name] = {
55|                 "gc_obs": float(row["gc"]),
56|                 "vflat": float(row["Vflat"]),
57|                 "hR": float(row["hR"]),
58|             }
59|     return data
60|
61|
62| def load_rotmod(galaxy_name, rotmod_dir):
```

```

63| fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
64| if not fpath.exists():
65|     return None
66| r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
67| with open(fpath, "r") as f:
68|     for line in f:
69|         line = line.strip()
70|         if not line or line.startswith("#"):
71|             continue
72|         parts = line.split()
73|         if len(parts) < 6:
74|             continue
75|         r.append(float(parts[0]))
76|         vobs.append(float(parts[1]))
77|         errv.append(float(parts[2]))
78|         vgas.append(float(parts[3]))
79|         vdisk.append(float(parts[4]))
80|         vbul.append(float(parts[5]))
81|     return (np.array(r), np.array(vobs), np.array(errv),
82|           np.array(vgas), np.array(vdisk), np.array(vbul))
83|
84|
85| def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
86|     r_m = r_kpc * kpc_to_m
87|     conv = kms2_to_ms2 / r_m
88|     gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
89|          + Yb*np.abs(Vbul)*Vbul*conv)
90|     return np.abs(gN)
91|
92|
93| def partial_corr(x, y, controls):
94|     if len(controls) == 0:
95|         return pearsonr(x, y)
96|     C = np.column_stack(controls + [np.ones(len(x))])
97|     cx, r, _ = lstsq(C, x, rcond=None)
98|     cy, r, _ = lstsq(C, y, rcond=None)
99|     return pearsonr(x - C @ cx, y - C @ cy)
100|
101|
102| # =====
103| # 複数の gc フィット手法
104| # =====
105| def fit_gc_deep(r_kpc, gN, Vobs, errV, r_range="all"):
106|     """深MOND: g_eff = sqrt(g_N * gc)
107|     r_range: "all", "outer50", "outer30", "inner50"
108|     """
109|     N = len(r_kpc)
110|     if r_range == "outer50":
111|         sl = slice(N - max(5, N//2), N)
112|     elif r_range == "outer30":
113|         sl = slice(N - max(5, N//3), N)
114|     elif r_range == "inner50":
115|         sl = slice(0, max(5, N//2))
116|     else:
117|         sl = slice(0, N)
118|
119|     r_f = r_kpc[sl]
120|     V_f = Vobs[sl]
121|     e_f = errV[sl]
122|     gN_f = gN[sl]
123|     e_f = np.where(e_f > 0, e_f, np.median(Vobs) * 0.1)
124|
125|     def chi2(lgc):
126|         gc = 10**lgc
127|         g_eff = np.sqrt(gN_f * gc)
128|         V_pred = np.sqrt(r_f * kpc_to_m * g_eff) * 1e-3
129|         return np.sum(((V_f - V_pred) / e_f)**2)
130|
131|     res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
132|     return 10**res.x, res.fun
133|
134|
135| def fit_gc_full(r_kpc, gN, Vobs, errV, r_range="all"):
136|     """完全MOND: g_eff/gc = [x + sqrt(x^2+4x)]/2"""
137|     N = len(r_kpc)
138|     if r_range == "outer50":
139|         sl = slice(N - max(5, N//2), N)
140|     elif r_range == "outer30":
141|         sl = slice(N - max(5, N//3), N)
142|     elif r_range == "inner50":
143|         sl = slice(0, max(5, N//2))
144|     else:
145|         sl = slice(0, N)
146|
147|     r_f = r_kpc[sl]
148|     V_f = Vobs[sl]
149|     e_f = errV[sl]

```

```

150| gN_f = gN[sl]
151| e_f = np.where(e_f > 0, e_f, np.median(Vobs) * 0.1)
152|
153| def chi2(lgc):
154|     gc = 10**lgc
155|     x = gN_f / gc
156|     g_eff = gc * (x + np.sqrt(x**2 + 4*x)) / 2.0
157|     V_pred = np.sqrt(r_f * kpc_to_m * g_eff) * 1e-3
158|     return np.sum(((V_f - V_pred) / e_f)**2)
159|
160| res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
161| return 10**res.x, res.fun
162|
163|
164| def fit_gc_tanh(r_kpc, Vobs, errV, vflat):
165|     """TA3風 tanhフィット: V(r) = vflat x tanh(r/r_t)
166|     gc = vflat^2/(r_t x kpc_to_m) で近似
167|     """
168|     V_norm = Vobs / vflat
169|     e_norm = errV / vflat
170|     e_norm = np.where(e_norm > 0, e_norm, 0.1)
171|
172|     def chi2(log_rt):
173|         rt = 10**log_rt
174|         V_pred = np.tanh(r_kpc / rt)
175|         return np.sum(((V_norm - V_pred) / e_norm)**2)
176|
177|     res = minimize_scalar(chi2, bounds=(-2, 2), method='bounded')
178|     rt = 10**res.x # kpc
179|     gc_tanh = (vflat * 1e3)**2 / (rt * kpc_to_m)
180|     return gc_tanh, rt, res.fun
181|
182|
183| # =====
184| def main():
185|     print("=" * 70)
186|     print("gc決定手法の系統効果: TA3 vs 深MOND vs 完全MOND")
187|     print("=" * 70)
188|
189|     gc_table = load_gc_table(GC_CSV)
190|
191|     records = []
192|     for gal_name, info in gc_table.items():
193|         rotmod = load_rotmod(gal_name, ROTMOD_DIR)
194|         if rotmod is None:
195|             continue
196|         r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
197|         if len(r_kpc) < 8:
198|             continue
199|
200|         gc_obs = info["gc_obs"]
201|         gc_obs_ms2 = gc_obs * a0 if gc_obs > 1e-5 else gc_obs
202|         hR = info["hR"]
203|         vflat = info["vflat"]
204|
205|         gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)
206|
207|         # 各種 gc フィット
208|         gc_deep_all, chi2_da = fit_gc_deep(r_kpc, gN, Vobs, errV, "all")
209|         gc_deep_out, chi2_do = fit_gc_deep(r_kpc, gN, Vobs, errV, "outer50")
210|         gc_deep_in, chi2_di = fit_gc_deep(r_kpc, gN, Vobs, errV, "inner50")
211|         gc_deep_o30, _ = fit_gc_deep(r_kpc, gN, Vobs, errV, "outer30")
212|
213|         gc_full_all, chi2_fa = fit_gc_full(r_kpc, gN, Vobs, errV, "all")
214|         gc_full_out, chi2_fo = fit_gc_full(r_kpc, gN, Vobs, errV, "outer50")
215|
216|         gc_tanh, rt_tanh, chi2_t = fit_gc_tanh(r_kpc, Vobs, errV, vflat)
217|
218|         # x_outer
219|         n_out = max(3, len(r_kpc) // 3)
220|         x_outer = np.median(gN[-n_out:] / gc_deep_all) if gc_deep_all > 0 else np.nan
221|
222|         # rmax/hR
223|         rmax_hR = r_kpc[-1] / hR if hR > 0 else np.nan
224|
225|         # proxy
226|         vflat_ms = vflat * 1e3
227|         hR_m = hR * kpc_to_m
228|         proxy = vflat_ms**2 / hR_m
229|
230|         # 内外 gc 比
231|         gc_ratio_io = gc_deep_in / gc_deep_out if gc_deep_out > 0 else np.nan
232|
233|         records.append({
234|             "name": gal_name,
235|             "gc_obs": gc_obs_ms2,
236|             "gc_deep_all": gc_deep_all,

```

```

237|         "gc_deep_out": gc_deep_out,
238|         "gc_deep_in": gc_deep_in,
239|         "gc_deep_o30": gc_deep_o30,
240|         "gc_full_all": gc_full_all,
241|         "gc_full_out": gc_full_out,
242|         "gc_tanh": gc_tanh,
243|         "rt_tanh": rt_tanh,
244|         "gc_ratio_io": gc_ratio_io,
245|         "hR": hR,
246|         "vflat": vflat,
247|         "proxy": proxy,
248|         "x_outer": x_outer,
249|         "rmax_hR": rmax_hR,
250|     })
251|
252|     N = len(records)
253|     print(f"Processed: {N}")
254|
255|     # 配列化
256|     def arr(k): return np.array([r[k] for r in records], dtype=float)
257|
258|     gc_obs = arr("gc_obs")
259|     gc_deep = arr("gc_deep_all")
260|     gc_deep_o = arr("gc_deep_out")
261|     gc_deep_i = arr("gc_deep_in")
262|     gc_deep_o3 = arr("gc_deep_o30")
263|     gc_full = arr("gc_full_all")
264|     gc_full_o = arr("gc_full_out")
265|     gc_tanh = arr("gc_tanh")
266|     gc_ratio = arr("gc_ratio_io")
267|     hR_a = arr("hR")
268|     vflat_a = arr("vflat")
269|     proxy_a = arr("proxy")
270|     x_outer = arr("x_outer")
271|     rmax_hR = arr("rmax_hR")
272|
273|     log_gco = np.log10(gc_obs)
274|     log_gcd = np.log10(gc_deep)
275|     log_gcdo = np.log10(gc_deep_o)
276|     log_gcdi = np.log10(gc_deep_i)
277|     log_gcf = np.log10(gc_full)
278|     log_gcfo = np.log10(gc_full_o)
279|     log_gct = np.log10(gc_tanh)
280|     log_hR = np.log10(hR_a)
281|     log_vf = np.log10(vflat_a)
282|     log_pr = np.log10(proxy_a)
283|     log_x = np.log10(x_outer + 1e-10)
284|
285|     m = (np.isfinite(log_gco) & np.isfinite(log_gcd) & np.isfinite(log_hR)
286|          & np.isfinite(log_vf) & np.isfinite(log_x) & np.isfinite(log_gcdo)
287|          & np.isfinite(log_gcdi) & np.isfinite(log_gcf) & np.isfinite(log_gcfo)
288|          & np.isfinite(log_gct) & np.isfinite(rmax_hR))
289|
290|     print(f"Valid: {m.sum()}")
291|
292|     # =====
293|     # Test 1:  $\Delta = \log(\text{gc\_obs}/\text{gc\_X})$  の hR 依存性
294|     # =====
295|     print("≠n" + "-" * 50)
296|     print("Test 1: 手法間差  $\Delta$  の hR 依存性")
297|     print("=" * 50)
298|
299|     gc_methods = {
300|         "gc_deep(all)": log_gcd,
301|         "gc_deep(out50)": log_gcdo,
302|         "gc_deep(out30)": np.log10(gc_deep_o3),
303|         "gc_deep(in50)": log_gcdi,
304|         "gc_full(all)": log_gcf,
305|         "gc_full(out50)": log_gcfo,
306|         "gc_tanh": log_gct,
307|     }
308|
309|     print(f" {'手法':<20s} {' $\Delta$  median':>10s} {' $\rho(\Delta, \text{hR})$ ':>10s} {'p':>12s} "
310|           f"{' $\rho(\Delta, \text{hR}|\text{vf})$ ':>12s} {'p':>12s}")
311|     print(" " + "-" * 78)
312|
313|     for name, log_gc_x in gc_methods.items():
314|         mx = m & np.isfinite(log_gc_x)
315|         if mx.sum() < 20:
316|             continue
317|         delta = log_gco[mx] - log_gc_x[mx]
318|         rho_h, p_h = pearsonr(log_hR[mx], delta)
319|         rho_hv, p_hv = partial_corr(log_hR[mx], delta, [log_vf[mx]])
320|         print(f" {name:<20s} {np.median(delta):+.4f} {rho_h:+.4f} p={p_h:.3e} "
321|               f" {rho_hv:+.4f} p={p_hv:.3e}")
322|
323|     # =====

```

```

324| # Test 2: 各手法の gc での hR 偏相関
325| # =====
326| print("#n" + "-" * 50)
327| print("Test 2: gc(各手法) の hR 偏相関 |vflat, x_outer")
328| print("-" * 50)
329|
330| all_gc_methods = {
331|     "gc_obs (TA3)": log_gco,
332|     "gc_deep(all)": log_gcd,
333|     "gc_deep(out50)": log_gcdo,
334|     "gc_deep(in50)": log_gcdi,
335|     "gc_full(all)": log_gcf,
336|     "gc_full(out50)": log_gcfo,
337|     "gc_tanh": log_gct,
338| }
339|
340| print(f" 手法: <20s> { 'ρ|vf':&gt;8s} { 'ρ|vf,x':&gt;8s} { 'α(proxy)':&gt;10s} { 'R^2':&gt;8s}")
341| print("-" * 58)
342|
343| for name, log_gc_x in all_gc_methods.items():
344|     mx = m & np.isfinite(log_gc_x)
345|     if mx.sum() < 20:
346|         continue
347|     rho_vf, _ = partial_corr(log_hr[mx], log_gc_x[mx], [log_vf[mx]])
348|     rho_vfx, _ = partial_corr(log_hr[mx], log_gc_x[mx], [log_vf[mx], log_x[mx]])
349|     sl = linregress(log_pr[mx], log_gc_x[mx])
350|     print(f" {name}<20s> {rho_vf:+.4f} {rho_vfx:+.4f} {sl.slope:+.4f} {sl.rvalue**2:.4f}")
351|
352| # =====
353| # Test 3: rmax/hR サブセット比較
354| # =====
355| print("#n" + "-" * 50)
356| print("Test 3: rmax/hR サブセットでの比較")
357| print("-" * 50)
358|
359| for rcut in [5, 8, 10, 15]:
360|     mc = m & (rmax_hR > rcut)
361|     if mc.sum() < 15:
362|         continue
363|     print(f"#n rmax/hR > {rcut} (N={mc.sum()}:)")
364|
365|     for name, log_gc_x in [("gc_obs", log_gco), ("gc_deep(all)", log_gcd),
366|                           ("gc_full(all)", log_gcf), ("gc_tanh", log_gct)]:
367|         mx = mc & np.isfinite(log_gc_x)
368|         if mx.sum() < 10:
369|             continue
370|         rho_vf, p_vf = partial_corr(log_hr[mx], log_gc_x[mx], [log_vf[mx]])
371|         rho_vfx, p_vfx = partial_corr(log_hr[mx], log_gc_x[mx],
372|                                     [log_vf[mx], log_x[mx]])
373|         print(f" {name}<18s> |vf: {rho_vf:+.4f} (p={p_vf:.3e}) "
374|               f"|vfx: {rho_vfx:+.4f} (p={p_vfx:.3e})")
375|
376| # =====
377| # Test 4: gc_deep ベースで幾何平均法則を再フィット
378| # =====
379| print("#n" + "-" * 50)
380| print("Test 4: 各手法の幾何平均法則フィット")
381| print("-" * 50)
382|
383| for name, log_gc_x in all_gc_methods.items():
384|     mx = m & np.isfinite(log_gc_x)
385|     if mx.sum() < 20:
386|         continue
387|     # 単変量: log(gc) = α x log(proxy) + const
388|     sl = linregress(log_pr[mx], log_gc_x[mx])
389|     # 多変量: log(gc) = a x log(vflat) + b x log(hR) + const
390|     X = np.column_stack([log_vf[mx], log_hr[mx], np.ones(mx.sum())])
391|     coef, _, _ = lstsq(X, log_gc_x[mx], rcond=None)
392|     y_pred = X @ coef
393|     ss_r = np.sum((log_gc_x[mx] - y_pred)**2)
394|     ss_t = np.sum((log_gc_x[mx] - np.mean(log_gc_x[mx]))**2)
395|     R2 = 1 - ss_r / ss_t
396|
397|     print(f" {name}<20s>: α={sl.slope:.4f}+/{sl.stderr:.4f} "
398|           f"vf^{coef[0]:.3f} x hR^{coef[1]:.3f} R^2={R2:.4f}")
399|
400| # =====
401| # Test 5: 内側 vs 外側フィットの hR 依存性
402| # =====
403| print("#n" + "-" * 50)
404| print("Test 5: 内側 vs 外側フィットの差")
405| print("-" * 50)
406|
407| delta_io = log_gcdi[m] - log_gcdo[m]
408| rho_io_h, p_io_h = pearsonr(log_hr[m], delta_io)
409| rho_io_hv, p_io_hv = partial_corr(log_hr[m], delta_io, [log_vf[m]])
410| rho_io_hvx, p_io_hvx = partial_corr(log_hr[m], delta_io,

```

```

411|                                     [log_vf[m], log_x[m]])
412|
413| print(f" Δ(inner-outer) の hR 相関:")
414| print(f" raw: ρ = {rho_io_h:+.4f}, p = {p_io_h:.3e}")
415| print(f" |vflat: ρ = {rho_io_hv:+.4f}, p = {p_io_hv:.3e}")
416| print(f" |vflat,x: ρ = {rho_io_hvx:+.4f}, p = {p_io_hvx:.3e}")
417|
418| print(f"Δ(inner-outer) 統計:")
419| print(f" median = {np.median(delta_io):.4f}")
420| print(f" std = {np.std(delta_io):.4f}")
421| print(f" |Δ|>0.1: {np.mean(np.abs(delta_io) > 0.1):.1%}")
422| print(f" |Δ|>0.3: {np.mean(np.abs(delta_io) > 0.3):.1%}")
423|
424| # gc_obs は inner, outer どちらに近い?
425| d_obs_in = np.abs(log_gco[m] - log_gcdi[m])
426| d_obs_out = np.abs(log_gco[m] - log_gcdo[m])
427| d_obs_all = np.abs(log_gco[m] - log_gcd[m])
428| print(f"Δ gc_obs との距離:")
429| print(f" |gc_obs - gc_deep(all)| median = {np.median(d_obs_all):.4f}")
430| print(f" |gc_obs - gc_deep(out)| median = {np.median(d_obs_out):.4f}")
431| print(f" |gc_obs - gc_deep(in)| median = {np.median(d_obs_in):.4f}")
432|
433| # =====
434| # 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
435| # =====
436| print(f"Δn" + "=" * 50)
437| print(f"Test 6: hR偏相関が最小になる gc の定義")
438| print(f"=" * 50)
439|
440| # 線形結合: log(gc_best) = w x log(gc_deep_out) + (1-w) x log(gc_deep_in)
441| # ~&gt; wを最適化して hR偏相関を最小化
442|
443| def hR_partial_corr_for_weight(w):
444|     log_gc_w = w * log_gcdo[m] + (1-w) * log_gcdi[m]
445|     try:
446|         rho, _ = partial_corr(log_hR[m], log_gc_w, [log_vf[m], log_x[m]])
447|         return abs(rho)
448|     except:
449|         return 1.0
450|
451| from scipy.optimize import minimize_scalar as ms
452| res_w = ms(hR_partial_corr_for_weight, bounds=(-1, 2), method='bounded')
453| w_opt = res_w.x
454| rho_opt = res_w.fun
455|
456| log_gc_opt = w_opt * log_gcdo[m] + (1-w_opt) * log_gcdi[m]
457| rho_actual, p_actual = partial_corr(log_hR[m], log_gc_opt, [log_vf[m], log_x[m]])
458|
459| print(f" 最適重み: w_outer = {w_opt:.4f}")
460| print(f" 最小 |ρ|: {rho_opt:.4f} (sign: {rho_actual:+.4f}")
461| print(f" (gc_obs: ρ = -0.145, gc_deep: ρ = ??)")
462|
463| # gc_obs の暗黙的重みを推定
464| # gc_obs ~ w_obs x gc_deep_out + (1-w_obs) x gc_deep_in ?
465| # 最小二乗で推定
466| X_w = np.column_stack([log_gcdo[m], log_gcdi[m], np.ones(m.sum())])
467| coef_w, _, _ = lstsq(X_w, log_gco[m], rcond=None)
468| print(f"Δ gc_obs の分解:")
469| print(f" gc_obs ~ gc_deep_out^{coef_w[0]:.3f} x gc_deep_in^{coef_w[1]:.3f}")
470| print(f" 暗黙的外側重み: {coef_w[0]:.3f}")
471|
472| # =====
473| # Test 7: TA3 tanh fit の rt/hR 比とhR偏相関
474| # =====
475| print(f"Δn" + "=" * 50)
476| print(f"Test 7: tanh fit パラメータの構造")
477| print(f"=" * 50)
478|
479| rt = arr("rt_tanh")
480| log_rt = np.log10(rt)
481| rt_hR = rt / hR_a
482|
483| mr = m &amp; np.isfinite(log_rt) &amp; np.isfinite(rt_hR)
484|
485| if mr.sum() &gt; 20:
486|     print(f" rt/hR: median = {np.median(rt_hR[mr]):.3f}, "
487|           f"std = {np.std(rt_hR[mr]):.3f}")
488|
489|     rho_rth, p_rth = pearsonr(log_hR[mr], np.log10(rt_hR[mr]))
490|     print(f" ρ(log(rt/hR), log(hR)) = {rho_rth:+.4f}, p = {p_rth:.3e}")
491|
492|     # rt/hR を統制して hR 偏相関を見る
493|     rho_t, p_t = partial_corr(log_hR[mr], log_gco[mr],
494|                               [log_vf[mr], log_x[mr], np.log10(rt_hR[mr])])
495|     print(f" ρ(gc_obs, hR | vflat, x, rt/hR) = {rho_t:+.4f}, p = {p_t:.3e}")
496|
497|     rho_base_r, _ = partial_corr(log_hR[mr], log_gco[mr],

```

```

498|                                     [log_vf[mr], log_x[mr]])
499|     print(f" (ベースライン |vflat, x:  $\rho = \{\rho\_base\_r:+.4f\}$ )")
500|
501| # =====
502| # Test 8: gc_deep(out50) ベースでの偏相関構造
503| # =====
504| print("\n" + "=" * 50)
505| print("Test 8: gc_deep(out50) ベースの偏相関構造")
506| print("=" * 50)
507|
508| for label, lgc in [("gc_obs", log_gco), ("gc_deep(all)", log_gcd),
509|                  ("gc_deep(out50)", log_gcdo), ("gc_deep(in50)", log_gcdi),
510|                  ("gc_full(all)", log_gcf), ("gc_full(out50)", log_gcfo)]:
511|     mx = m & np.isfinite(lgc)
512|     if mx.sum() < 20:
513|         continue
514|     rho_raw, _ = pearsonr(log_hR[mx], lgc[mx])
515|     rho_vf, _ = partial_corr(log_hR[mx], lgc[mx], [log_vf[mx]])
516|     rho_vfx, _ = partial_corr(log_hR[mx], lgc[mx], [log_vf[mx], log_x[mx]])
517|     print(f" {label:<20s}: raw={rho_raw:+.4f} |vf={rho_vf:+.4f} |vfx={rho_vfx:+.4f}")
518|
519| # =====
520| # SUMMARY
521| # =====
522| print("\n" + "=" * 70)
523| print("SUMMARY")
524| print("=" * 70)
525|
526| print(f"\n gc決定手法によるhR偏相関の変化 (|vflat, x_outer):")
527| for label, lgc in all_gc_methods.items():
528|     mx = m & np.isfinite(lgc)
529|     if mx.sum() > 20:
530|         rho, p = partial_corr(log_hR[mx], lgc[mx], [log_vf[mx], log_x[mx]])
531|         print(f" {label:<20s}:  $\rho = \{\rho:+.4f\}$  ( $p=\{p:.3e\}$ ")
532|
533| print(f"\n hR偏相関を最小にする gc:")
534| print(f" 最適: gc_out^{w_opt:.3f} x gc_in^{1-w_opt:.3f} -&gt; | $\rho$ | = {rho_opt:.4f}")
535| print(f" gc_obs の暗黙的外側重み: {coef_w[0]:.3f}")
536|
537| print(f"\n 核心的問い: 31%は手法の系統効果か物理的效果か?")
538| print(f" -&gt; gc_deep ベースでの hR 偏相関を確認:")
539|
540| rho_deep_base, p_deep = partial_corr(log_hR[m], log_gcd[m],
541|                                     [log_vf[m], log_x[m]])
542| print(f" gc_deep(all) |vflat,x:  $\rho = \{\rho\_deep\_base:+.4f\}$  ( $p=\{p\_deep:.3e\}$ ")
543|
544| rho_obs_base, p_obs = partial_corr(log_hR[m], log_gco[m],
545|                                  [log_vf[m], log_x[m]])
546| print(f" gc_obs (TA3) |vflat,x:  $\rho = \{\rho\_obs\_base:+.4f\}$  ( $p=\{p\_obs:.3e\}$ ")
547|
548| if abs(rho_deep_base) < abs(rho_obs_base) * 0.5:
549|     print(f"\n [*] gc_deep では hR 偏相関が大幅に減少")
550|     print(f" -&gt; 31%の大部分は TA3 gc 決定の系統効果")
551| elif abs(rho_deep_base) > abs(rho_obs_base) * 0.8:
552|     print(f"\n [*] gc_deep でも hR 偏相関がほぼ同じ")
553|     print(f" -&gt; 31%は手法非依存の物理的效果")
554| else:
555|     print(f"\n [*] 部分的に手法依存、部分的に物理的")
556|     pct = (abs(rho_obs_base) - abs(rho_deep_base)) / abs(rho_obs_base) * 100
557|     print(f" -&gt; 手法効果: {pct:.0f}%, 物理的: {100-pct:.0f}%")
558|
559|
560| if __name__ == "__main__":
561|     main()
562|

```

## 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 ( $s_c$ ,  $r_{\text{plastic}}/hR$ ,  $f_p(gc)$ ,  $E_{\text{strain}}$ ,  $\text{excess\_strain}$ ,  $\text{central\_conc}$ ,  $\text{strain\_gradient}$ ,  $r_{\text{peak}}/hR$ ) 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  $x_{\text{outer}}$   $\rho = +0.527$  (共線)。統制で  $\rho$  悪化 ( $-0.145 \rightarrow -0.273$ )。

### ソースコード

(563 行)

```
1| #!/usr/bin/env python3
2| """
3| sparc_cond14_residual.py
4| =====
5| 条件14 (塑性領域) が残り31%のhR偏相関を説明するか
6|
7| 前セッションで否定されたもの:
8| -  $f_p(g_N \ \&lt; \ a_0)$  質量分率)  $\rho$ ;  $gc: f_p \sim 1$  で判別力なし
9| -  $gc \sim \sum \bar{a}^{(1/3)}$ :  $R^2 = 0.016$ 
10|
11| 今回の仮説 (弱い版) :
12| 塑性領域は「gc を決める主因」ではなく、
13| 深MOND gc に「微小な補正」を加える ( $R^2 \sim 0.04$ , 5%効果)。
14|
15| メカニズム:
16| (1) ディスク中心で  $g_N$  が高い  $\rho$ ; 歪み大  $\rho$ ; 塑性領域が集中
17| (2) 塑性領域の  $\rho_{\text{eff}} = u_{\text{strain}}/c^2$  が有効質量として作用
18| (3)  $\rho$ ;  $gc_{\text{in}}$   $\rho$ ;  $gc_{\text{out}}$  の系統差を生む
19| (4) hR が小さい銀河ほどバリオン集中  $\rho$ ; 中心  $g_N/a_0$  が高い  $\rho$ ; 塑性補正大
20|
21| 定量的予測:
22| - 「中心歪みパラメータ」 $s_c = g_N_{\text{peak}} / gc_{\text{obs}} \sim$  塑性度
23| -  $\Delta gc = gc_{\text{obs}} - gc_{\text{deep}} \sim s_c$  (塑性補正)
24| -  $s_c$  を統制すると hR 偏相関の残り31%が減少
25|
26| テスト:
27| Test 1: 中心歪みパラメータ  $s_c$  の定義と分布
28| Test 2:  $s_c$  vs  $gc_{\text{in}} - gc_{\text{out}}$  の相関 (塑性メカニズムの直接検証)
29| Test 3:  $s_c$  vs hR の相関 (hR依存性の経路確認)
30| Test 4:  $s_c$  を統制した hR 偏相関 (31%が減るか)
31| Test 5: 塑性補正モデル  $gc_{\text{corrected}} = gc_{\text{deep}} \times (1 + \beta \times s_c^\gamma)$  のフィット
32| Test 6:  $r_{\text{max}}/hR$   $\rho$ ; 8 サブセットでの検証
33| Test 7: 代替パラメータ (歪みエネルギー密度プロキシ)
34|
35| 実行: uv run --with scipy --with matplotlib python sparc_cond14_residual.py
36| """
37|
38| import numpy as np
39| from scipy.optimize import minimize_scalar, minimize
40| from scipy.stats import pearsonr, spearmanr, linregress
41| from numpy.linalg import lstsq
42| from pathlib import Path
43| import csv
44| import sys
45|
46| # =====
47| a0 = 1.2e-10
48| kpc_to_m = 3.0857e19
49| kms2_to_ms2 = 1.0e6
50| Yd_default = 0.5
51| Yb_default = 0.7
52|
53| BASE = Path(r"D:\ドキュメント\アントロピー\新膜宇宙論\これまでの軌跡\パイソン")
54| ROTMOD_DIR = BASE / "Rotmod_LTG"
55| GC_CSV = BASE / "sparc_gc.csv"
56|
57|
58| def load_gc_table(path):
59|     data = {}
60|     with open(path, "r") as f:
61|         reader = csv.DictReader(f)
62|         for row in reader:
```

```

63|         name = row["Galaxy"].strip()
64|         data[name] = {
65|             "gc_obs": float(row["gc"]),
66|             "vflat": float(row["Vflat"]),
67|             "hR": float(row["hR"]),
68|         }
69|     return data
70|
71|
72| def load_rotmod(galaxy_name, rotmod_dir):
73|     fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
74|     if not fpath.exists():
75|         return None
76|     r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
77|     with open(fpath, "r") as f:
78|         for line in f:
79|             line = line.strip()
80|             if not line or line.startswith("#"):
81|                 continue
82|             parts = line.split()
83|             if len(parts) < 6:
84|                 continue
85|             r.append(float(parts[0]))
86|             vobs.append(float(parts[1]))
87|             errv.append(float(parts[2]))
88|             vgas.append(float(parts[3]))
89|             vdisk.append(float(parts[4]))
90|             vbul.append(float(parts[5]))
91|     return (np.array(r), np.array(vobs), np.array(errv),
92|           np.array(vgas), np.array(vdisk), np.array(vbul))
93|
94|
95| def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
96|     r_m = r_kpc * kpc_to_m
97|     conv = kms2_to_ms2 / r_m
98|     gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
99|          + Yb*np.abs(Vbul)*Vbul*conv)
100|     return np.abs(gN)
101|
102|
103| def partial_corr(x, y, controls):
104|     if len(controls) == 0:
105|         return pearsonr(x, y)
106|     C = np.column_stack(controls + [np.ones(len(x))])
107|     cx, _, _ = lstsq(C, x, rcond=None)
108|     cy, _, _ = lstsq(C, y, rcond=None)
109|     return pearsonr(x - C @ cx, y - C @ cy)
110|
111|
112| def fit_gc_deep(r_kpc, gN, Vobs, errV, r_range="all"):
113|     N = len(r_kpc)
114|     if r_range == "outer50":
115|         sl = slice(N - max(5, N//2), N)
116|     elif r_range == "inner50":
117|         sl = slice(0, max(5, N//2))
118|     else:
119|         sl = slice(0, N)
120|     r_f, V_f, e_f, gN_f = r_kpc[sl], Vobs[sl], errV[sl], gN[sl]
121|     e_f = np.where(e_f > 0, e_f, np.median(Vobs) * 0.1)
122|
123|     def chi2(lgc):
124|         gc = 10**lgc
125|         V_pred = np.sqrt(r_f * kpc_to_m * np.sqrt(gN_f * gc)) * 1e-3
126|         return np.sum(((V_f - V_pred) / e_f)**2)
127|
128|     res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
129|     return 10**res.x
130|
131|
132| # =====
133| # 塑性パラメータの定義
134| # =====
135| def compute_plasticity_params(r_kpc, gN, gc_obs, hR):
136|     """条件14に基づく塑性パラメータ群"""
137|     params = {}
138|
139|     # --- (1) 中心歪みパラメータ s_c = g_N_peak / gc ---
140|     gN_peak = np.max(gN)
141|     params["s_c"] = gN_peak / gc_obs if gc_obs > 0 else np.nan
142|
143|     # --- (2) ピーク位置での歪み ---
144|     idx_peak = np.argmax(gN)
145|     params["r_peak_hR"] = r_kpc[idx_peak] / hR if hR > 0 else np.nan
146|
147|     # --- (3) 歪みエネルギー密度プロキシ ---
148|     # u_strain ~ ε^2 ~ (g_N/gc)^2 の積分
149|     # E_strain = int (g_N(r)/gc)^2 x r dr

```

```

150| x_arr = gN / gc_obs
151| if len(r_kpc) > 2:
152|     E_strain = np.trapz(x_arr**2 * r_kpc, r_kpc)
153|     params["E_strain"] = E_strain
154| else:
155|     params["E_strain"] = np.nan
156|
157| # --- (4) 「塑性半径」: g_N &gt; gc の領域の広さ ---
158| # r_plastic = g_N &gt; gc を満たす最大 r (hR単位)
159| above = r_kpc[gN &gt; gc_obs]
160| params["r_plastic_hR"] = above[-1] / hR if len(above) &gt; 0 and hR &gt; 0 else 0.0
161|
162| # --- (5) 塑性質量分率 (修正版) ---
163| # f_p = int_{g_N &gt; gc} g_N r dr / int g_N r dr
164| # g_N &gt; gc の領域の質量寄与の割合
165| total_int = np.trapz(gN * r_kpc, r_kpc)
166| if total_int &gt; 0:
167|     mask_plastic = gN &gt; gc_obs
168|     if mask_plastic.any():
169|         plastic_int = np.trapz(gN[mask_plastic] * r_kpc[mask_plastic],
170|                                r_kpc[mask_plastic])
171|         params["f_p_gc"] = plastic_int / total_int
172|     else:
173|         params["f_p_gc"] = 0.0
174| else:
175|     params["f_p_gc"] = np.nan
176|
177| # --- (6) 中心集中度: g_N(r < hR) / g_N(全体) ---
178| inner = r_kpc < hR
179| if inner.sum() &gt; 0 and len(r_kpc) &gt; 2:
180|     gN_inner = np.trapz(gN[inner] * r_kpc[inner], r_kpc[inner])
181|     gN_total = np.trapz(gN * r_kpc, r_kpc)
182|     params["central_conc"] = gN_inner / gN_total if gN_total &gt; 0 else np.nan
183| else:
184|     params["central_conc"] = np.nan
185|
186| # --- (7) 歪み勾配: d(g_N/gc)/d(r/hR) at r = hR ---
187| x_arr_norm = gN / gc_obs
188| r_norm = r_kpc / hR
189| # r/hR = 1 付近での勾配
190| mask_near = np.abs(r_norm - 1.0) < 0.5
191| if mask_near.sum() &gt;= 3:
192|     sl = linregress(r_norm[mask_near], x_arr_norm[mask_near])
193|     params["strain_gradient"] = sl.slope
194| else:
195|     params["strain_gradient"] = np.nan
196|
197| # --- (8) 「超過歪み」: max(0, g_N - gc) の積分 / (gc x hR^2) ---
198| excess = np.maximum(0, gN - gc_obs)
199| if len(r_kpc) &gt; 2 and gc_obs &gt; 0 and hR &gt; 0:
200|     E_excess = np.trapz(excess * r_kpc, r_kpc)
201|     params["excess_strain"] = E_excess / (gc_obs * (hR)**2)
202| else:
203|     params["excess_strain"] = np.nan
204|
205| return params
206|
207|
208| # =====
209| def main():
210|     print("=" * 70)
211|     print("条件14 (塑性領域) による残り31%の説明可能性")
212|     print("=" * 70)
213|
214|     gc_table = load_gc_table(GC_CSV)
215|
216|     records = []
217|     for gal_name, info in gc_table.items():
218|         rotmod = load_rotmod(gal_name, ROTMOD_DIR)
219|         if rotmod is None:
220|             continue
221|         r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
222|         if len(r_kpc) < 8:
223|             continue
224|
225|         gc_obs = info["gc_obs"]
226|         gc_obs_ms2 = gc_obs * a0 if gc_obs &gt; 1e-5 else gc_obs
227|         hR = info["hR"]
228|         vflat = info["vflat"]
229|
230|         gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)
231|
232|         # gc フィット
233|         gc_deep = fit_gc_deep(r_kpc, gN, Vobs, errV, "all")
234|         gc_deep_in = fit_gc_deep(r_kpc, gN, Vobs, errV, "inner50")
235|         gc_deep_out = fit_gc_deep(r_kpc, gN, Vobs, errV, "outer50")
236|

```

```

237|     # 塑性パラメータ
238|     plast = compute_plasticity_params(r_kpc, gN, gc_obs_ms2, hR)
239|
240|     # x_outer
241|     n_out = max(3, len(r_kpc) // 3)
242|     x_outer = np.median(gN[-n_out:] / gc_deep) if gc_deep > 0 else np.nan
243|
244|     # rmax/hR
245|     rmax_hR = r_kpc[-1] / hR if hR > 0 else np.nan
246|
247|     # 残差
248|     delta_gc = np.log10(gc_obs_ms2) - np.log10(gc_deep) if gc_deep > 0 else np.nan
249|     delta_io = np.log10(gc_deep_in) - np.log10(gc_deep_out) if gc_deep_out > 0 else np.nan
250|
251|     records.append({
252|         "name": gal_name,
253|         "gc_obs": gc_obs_ms2,
254|         "gc_deep": gc_deep,
255|         "gc_deep_in": gc_deep_in,
256|         "gc_deep_out": gc_deep_out,
257|         "delta_gc": delta_gc,
258|         "delta_io": delta_io,
259|         "hR": hR,
260|         "vflat": vflat,
261|         "x_outer": x_outer,
262|         "rmax_hR": rmax_hR,
263|         **plast,
264|     })
265|
266|     N = len(records)
267|     print(f"Processed: {N}")
268|
269|     def arr(k): return np.array([r[k] for r in records], dtype=float)
270|
271|     gc_obs = arr("gc_obs"); gc_deep = arr("gc_deep")
272|     hR_a = arr("hR"); vflat_a = arr("vflat")
273|     x_outer = arr("x_outer"); rmax_hR = arr("rmax_hR")
274|     delta_gc = arr("delta_gc"); delta_io = arr("delta_io")
275|
276|     s_c = arr("s_c")
277|     r_peak_hR = arr("r_peak_hR")
278|     E_strain = arr("E_strain")
279|     r_plastic_hR = arr("r_plastic_hR")
280|     f_p_gc = arr("f_p_gc")
281|     central_conc = arr("central_conc")
282|     strain_grad = arr("strain_gradient")
283|     excess_strain = arr("excess_strain")
284|
285|     log_gc = np.log10(gc_obs)
286|     log_gcd = np.log10(gc_deep)
287|     log_hR = np.log10(hR_a)
288|     log_vf = np.log10(vflat_a)
289|     log_x = np.log10(x_outer + 1e-10)
290|     log_sc = np.log10(s_c + 1e-10)
291|
292|     m = (np.isfinite(log_gc) & np.isfinite(log_hR) & np.isfinite(log_vf)
293|          & np.isfinite(log_x) & np.isfinite(log_gcd) & np.isfinite(log_sc)
294|          & np.isfinite(delta_gc))
295|
296|     print(f"Valid: {m.sum()}")
297|
298|     # =====
299|     # Test 1: 塑性パラメータの分布
300|     # =====
301|     print(f"%n" + "=" * 50)
302|     print("Test 1: 塑性パラメータの分布")
303|     print("=" * 50)
304|
305|     plast_params = {
306|         "s_c (gN_peak/gc)": s_c,
307|         "r_peak/hR": r_peak_hR,
308|         "E_strain": E_strain,
309|         "r_plastic/hR": r_plastic_hR,
310|         "f_p(gc)": f_p_gc,
311|         "central_conc": central_conc,
312|         "strain_gradient": strain_grad,
313|         "excess_strain": excess_strain,
314|     }
315|
316|     for name, arr_p in plast_params.items():
317|         v = np.isfinite(arr_p) & m
318|         if v.sum() > 10:
319|             print(f" {name:20s}: N={v.sum():3d} "
320|                   f"median={np.median(arr_p[v]):.4f} "
321|                   f"std={np.std(arr_p[v]):.4f} "
322|                   f"range=[{np.min(arr_p[v]):.3f}, {np.max(arr_p[v]):.3f}]")
323|

```

```

324| # =====
325| # Test 2: s_c vs gc_in - gc_out (塑性メカニズムの直接検証)
326| # =====
327| print("\n" + "=" * 50)
328| print("Test 2: 中心歪み s_c vs gc内外差")
329| print("=" * 50)
330|
331| m_io = m & np.isfinite(delta_io)
332| if m_io.sum() > 10:
333|     rho_sio, p_sio = pearsonr(log_sc[m_io], delta_io[m_io])
334|     print(f" rho(s_c, gc_in - gc_out) = {rho_sio:.4f}, p = {p_sio:.3e}")
335|     print(f" 予測: 正の相関 (s_c大 -& gc_inが相対的に高い)")
336|
337|     # s_c の四分位で delta_io を比較
338|     q25, q75 = np.percentile(s_c[m_io], [25, 75])
339|     lo = s_c[m_io] &lt; q25
340|     hi = s_c[m_io] &gt; q75
341|     print(f" s_c &lt; Q25 ({q25:.2f}): delta_io median = {np.median(delta_io[m_io][lo]:.4f}")
342|     print(f" s_c &gt; Q75 ({q75:.2f}): delta_io median = {np.median(delta_io[m_io][hi]:.4f}")
343|
344| # 全塑性パラメータ vs delta_io
345| print(f"\n 全塑性パラメータ vs delta_io:")
346| for name, arr_p in plast_params.items():
347|     v = m_io & np.isfinite(arr_p)
348|     if v.sum() > 15:
349|         # 対数化が適切なものは対数で
350|         if np.min(arr_p[v]) > 0:
351|             rho, p = pearsonr(np.log10(arr_p[v] + 1e-10), delta_io[v])
352|             print(f" {name:20s}: rho = {rho:.4f}, p = {p:.3e}")
353|         else:
354|             rho, p = pearsonr(arr_p[v], delta_io[v])
355|             print(f" {name:20s}: rho = {rho:.4f}, p = {p:.3e}")
356|
357| # =====
358| # Test 3: 塑性パラメータ vs hR の相関
359| # =====
360| print("\n" + "=" * 50)
361| print("Test 3: 塑性パラメータ vs hR, gc")
362| print("=" * 50)
363|
364| print(f" 'パラメータ':&lt;20s 'rho(hR)':&gt;10s 'p':&gt;12s "
365|       f" 'rho(gc)':&gt;10s 'p':&gt;12s")
366| print(" " + "-" * 66)
367| for name, arr_p in plast_params.items():
368|     v = m & np.isfinite(arr_p)
369|     if v.sum() > 15:
370|         if np.min(arr_p[v]) > 0:
371|             ap = np.log10(arr_p[v] + 1e-10)
372|         else:
373|             ap = arr_p[v]
374|         rh, ph = pearsonr(log_hR[v], ap)
375|         rg, pg = pearsonr(log_gc[v], ap)
376|         print(f" {name:&lt;20s} {rh:.4f} p={ph:.3e} {rg:.4f} p={pg:.3e}")
377|
378| # =====
379| # Test 4: 塑性パラメータを統制したhR偏相関 [*]核心[*]
380| # =====
381| print("\n" + "=" * 50)
382| print("Test 4: 塑性パラメータ統制後のhR偏相関")
383| print("=" * 50)
384|
385| base_controls = [log_vf[m], log_x[m]]
386| rho_base, p_base = partial_corr(log_hR[m], log_gc[m], base_controls)
387| print(f" ベースライン |vflat, x: rho = {rho_base:.4f}")
388|
389| print(f"\n '追加パラメータ':&lt;20s 'rho':&gt;8s 'Δ|rho|':&gt;10s")
390| print(" " + "-" * 40)
391|
392| best_delta = 0
393| best_name = ""
394| for name, arr_p in plast_params.items():
395|     v = m & np.isfinite(arr_p)
396|     if v.sum() &lt; m.sum() * 0.7:
397|         continue
398|     arr_filled = arr_p.copy()
399|     arr_filled[~np.isfinite(arr_filled)] = np.nanmedian(arr_p)
400|     if np.min(arr_filled[m]) > 0:
401|         ctrl = np.log10(arr_filled[m] + 1e-10)
402|     else:
403|         ctrl = arr_filled[m]
404|     try:
405|         rho_new, p_new = partial_corr(log_hR[m], log_gc[m],
406|                                       base_controls + [ctrl])
407|         delta = abs(rho_new) - abs(rho_base)
408|         print(f" {name:&lt;20s} {rho_new:.4f} {delta:.4f}")
409|         if delta &lt; best_delta:
410|             best_delta = delta

```

```

411|         best_name = name
412|     except:
413|         pass
414|
415| if best_delta < -0.01:
416|     print(f"最も効果的: {best_name} (Δ = {best_delta:+.4f})")
417| else:
418|     print(f"どの塑性パラメータも hR 偏相関を有意に減らさない")
419|
420| # =====
421| # Test 5: 塑性補正モデル
422| # =====
423| print(f"n" + "=" * 50)
424| print("Test 5: 塑性補正モデル gc_corr = gc_deep x (1 + beta x s_c^gamma)")
425| print("=" * 50)
426|
427| # gc_obs / gc_deep = 1 + β x s_c^γ をフィット
428| ratio = gc_obs[m] / gc_deep[m]
429| log_ratio = np.log10(ratio)
430|
431| # まず線形: log(gc_obs/gc_deep) = a x log(s_c) + b
432| sl_sc = linregress(log_sc[m], log_ratio)
433| print(f" log(gc_obs/gc_deep) vs log(s_c):")
434| print(f" slope = {sl_sc.slope:.4f} +/- {sl_sc.stderr:.4f}")
435| print(f" R<sup>2</sup> = {sl_sc.rvalue**2:.4f}")
436| print(f" p = {sl_sc.pvalue:.3e}")
437|
438| # s_c 補正後の gc
439| gc_corrected = gc_deep[m] * (s_c[m] ** sl_sc.slope) * 10**sl_sc.intercept
440|
441| log_gc = np.log10(gc_corrected)
442| rho_cc, p_cc = partial_corr(log_hR[m], log_gc,
443|                             [log_vf[m], log_x[m]])
444| print(f"補正後 gc の hR 偏相関 |vflat, x:")
445| print(f" rho = {rho_cc:+.4f} (ベースライン: {rho_base:+.4f})")
446| print(f" Δ|rho| = {abs(rho_cc) - abs(rho_base):+.4f}")
447|
448| # 他の塑性パラメータでも同様の補正
449| print(f"各塑性パラメータによる補正効果:")
450| for name, arr_p in plast_params.items():
451|     v = m & np.isfinite(arr_p) & (arr_p > 0)
452|     if v.sum() < m.sum() * 0.5:
453|         continue
454|     log_p = np.log10(arr_p[v] + 1e-10)
455|     sl = linregress(log_p, log_ratio[np.where(m)[0][v[m]]] if v.sum() == m.sum()
456|                    else np.nan)
457|     # 簡易版: m内でのフィット
458|     try:
459|         arr_m = arr_p[m]
460|         valid_m = arr_m > 0
461|         if valid_m.sum() > 20:
462|             log_pm = np.log10(arr_m[valid_m] + 1e-10)
463|             sl2 = linregress(log_pm, log_ratio[valid_m])
464|             gc_c2 = gc_deep[m].copy()
465|             gc_c2[valid_m] *= (arr_m[valid_m] ** sl2.slope) * 10**sl2.intercept
466|             log_gc2 = np.log10(gc_c2)
467|             if np.isfinite(log_gc2).sum() > 20:
468|                 m2 = np.isfinite(log_gc2)
469|                 rho2, _ = partial_corr(log_hR[m][m2], log_gc2[m2],
470|                                       [log_vf[m][m2], log_x[m][m2]])
471|                 delta2 = abs(rho2) - abs(rho_base)
472|                 print(f" {name:20s}: R2={sl2.rvalue**2:.4f}, "
473|                       f"rho_corr={rho2:+.4f}, delta={delta2:+.4f}")
474|     except:
475|         pass
476|
477| # =====
478| # Test 6: rmax/hR > 8 サブセット
479| # =====
480| print(f"n" + "=" * 50)
481| print("Test 6: rmax/hR > 8 サブセットでの塑性効果")
482| print("=" * 50)
483|
484| m8 = m & (rmax_hR > 8)
485| if m8.sum() > 15:
486|     rho_b8, _ = partial_corr(log_hR[m8], log_gc[m8],
487|                             [log_vf[m8], log_x[m8]])
488|     print(f"ベースライン (rmax/hR>8): rho = {rho_b8:+.4f} (N={m8.sum()})")
489|
490| for name, arr_p in [("s_c", s_c), ("excess_strain", excess_strain),
491|                   ("f_p(gc)", f_p_gc), ("central_conc", central_conc)]:
492|     v8 = m8 & np.isfinite(arr_p)
493|     if v8.sum() < 10:
494|         continue
495|     arr_f = arr_p.copy()
496|     arr_f[~np.isfinite(arr_f)] = np.nanmedian(arr_p)
497|     if np.min(arr_f[m8]) > 0:

```

```

498|         ctrl = np.log10(arr_f[m8] + 1e-10)
499|     else:
500|         ctrl = arr_f[m8]
501|     try:
502|         rho8, p8 = partial_corr(log_hR[m8], log_gc[m8],
503|                                 [log_vf[m8], log_x[m8], ctrl])
504|         delta8 = abs(rho8) - abs(rho_b8)
505|         print(f" +(name:20s): rho={rho8:+.4f}, delta={delta8:+.4f}")
506|     except:
507|         pass
508|
509| # =====
510| # Test 7: x_outer と s_c の独立性チェック
511| # =====
512| print("\n" + "=" * 50)
513| print("Test 7: x_outer と s_c の関係")
514| print("=" * 50)
515|
516| rho_xs, p_xs = pearsonr(log_x[m], log_sc[m])
517| print(f" rho(x_outer, s_c) = {rho_xs:+.4f}, p = {p_xs:.3e}")
518|
519| # s_c|x_outer の残差が hR 偏相関を説明するか
520| # つまり x_outer では捉えきれない塑性情報が s_c にあるか
521| rho_sc_hR, p_sc_hR = partial_corr(log_hR[m], log_gc[m],
522|                                   [log_vf[m], log_x[m], log_sc[m]])
523| print(f" rho(gc, hR | vflat, x, s_c) = {rho_sc_hR:+.4f}, p = {p_sc_hR:.3e}")
524| print(f" (ベースライン |vflat, x: {rho_base:+.4f})")
525| delta_sc = abs(rho_sc_hR) - abs(rho_base)
526| print(f" delta = {delta_sc:+.4f}")
527|
528| if delta_sc < -0.02:
529|     pct = -delta_sc / 0.472 * 100
530|     print(f" -> s_c が hR 偏相関の追加 {pct:.1f}% を説明")
531| else:
532|     print(f" -> s_c は x_outer 以上の説明力を持たない")
533|
534| # =====
535| # SUMMARY
536| # =====
537| print("\n" + "=" * 70)
538| print("SUMMARY: 条件14 (塑性領域) の残り31%への寄与")
539| print("=" * 70)
540|
541| print(f"\n hR偏相関:")
542| print(f" |vflat: rho = -0.472")
543| print(f" |vflat, x: rho = {rho_base:+.4f} (x が 69% 説明)")
544| print(f" |vflat, x, sc: rho = {rho_sc_hR:+.4f} (s_c 追加効果: {delta_sc:+.4f}")
545|
546| print(f"\n s_c vs gc内外差: rho = {rho_sio:+.4f}" if m_io.sum() > 10 else "")
547| print(f" s_c vs x_outer: rho = {rho_xs:+.4f}")
548|
549| print(f"\n 条件14の評価:")
550| if delta_sc < -0.03:
551|     print(f" -> 塑性パラメータが hR 偏相関の追加分を部分的に説明 [*]")
552|     print(f" -> 条件14は「微小補正」として機能する可能性あり")
553| elif abs(delta_sc) < 0.03:
554|     print(f" -> 塑性パラメータは x_outer と冗長 (独立な説明力なし)")
555|     print(f" -> 条件14は残り31%を説明しない")
556| else:
557|     print(f" -> 塑性パラメータの追加で hR 偏相関がむしろ増加")
558|     print(f" -> 条件14は残り31%と無関係")
559|
560|
561| if __name__ == "__main__":
562|     main()
563|

```

## 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)

```
1| #!/usr/bin/env python3
2| """
3| sparc_strain_gradient.py
4| =====
5| 歪みエネルギー勾配項による残り31%の説明
6|
7| 新しい着眼:
8| 膜の歪みエネルギーには2成分:
9|   u_local ~  $\epsilon^2$  > x_outer と同等 (既にテスト済み、69%説明)
10|  u_grad ~  $(d\epsilon/dr)^2$  > hR に独立に依存 (未テスト)
11|
12| 指数ディスクでは:
13|    $d\epsilon/dr \sim (dg_N/dr)/gc \sim g_N/(gc \times hR)$ 
14|    $u_{grad} \sim (g_N/(gc \times hR))^2$ 
15|   > 勾配エネルギー ~  $hR^{(-2)}$  の依存性
16|
17| 仮説: 勾配エネルギーが gc に微小補正を加える
18|   gc_eff = gc_deep x (1 +  $\lambda \times E_{grad} / E_0$ )
19|   E_grad = int  $(d\epsilon/dr)^2 r dr$ 
20|   E_0 は正規化定数
21|
22| テスト:
23|   Test 1:  $\epsilon(r)$  プロファイルの勾配計算
24|   Test 2:  $E_{grad}$  vs hR,  $x_{outer}$  の相関 (独立性チェック)
25|   Test 3:  $E_{grad}$  を統制した hR 偏相関
26|   Test 4: 勾配補正モデルの検証
27|   Test 5:  $E_{grad}$  vs  $E_{local}$  の独立性
28|   Test 6:  $r_{max}/hR$  > 8 サブセット
29|   Test 7: 偏相関の段階的分解 ( $x_{outer}$  >  $E_{grad}$ )
30|
31| 実行: uv run --with scipy --with matplotlib python sparc_strain_gradient.py
32| """
33|
34| import numpy as np
35| from scipy.optimize import minimize_scalar
36| from scipy.stats import pearsonr, linregress
37| from numpy.linalg import lstsq
38| from pathlib import Path
39| import csv
40| import sys
41|
42| # =====
43| a0 = 1.2e-10
44| kpc_to_m = 3.0857e19
45| kms2_to_ms2 = 1.0e6
46| Yd_default = 0.5
47| Yb_default = 0.7
48|
49| BASE = Path(r"D:\ドキュメント\アントロピー\新膜宇宙論\これまでの軌跡\パイソン")
50| ROTMOD_DIR = BASE / "Rotmod_LTG"
51| GC_CSV = BASE / "sparc_gc.csv"
52|
53|
54| def load_gc_table(path):
55|     data = {}
56|     with open(path, "r") as f:
57|         reader = csv.DictReader(f)
58|         for row in reader:
59|             name = row["Galaxy"].strip()
60|             data[name] = {
61|                 "gc_obs": float(row["gc"]),
62|                 "vflat": float(row["Vflat"]),
```

```

63|         "hR": float(row["hR"]),
64|     }
65|     return data
66|
67|
68| def load_rotmod(galaxy_name, rotmod_dir):
69|     fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
70|     if not fpath.exists():
71|         return None
72|     r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
73|     with open(fpath, "r") as f:
74|         for line in f:
75|             line = line.strip()
76|             if not line or line.startswith("#"):
77|                 continue
78|             parts = line.split()
79|             if len(parts) < 6:
80|                 continue
81|             r.append(float(parts[0]))
82|             vobs.append(float(parts[1]))
83|             errv.append(float(parts[2]))
84|             vgas.append(float(parts[3]))
85|             vdisk.append(float(parts[4]))
86|             vbul.append(float(parts[5]))
87|     return (np.array(r), np.array(vobs), np.array(errv),
88|           np.array(vgas), np.array(vdisk), np.array(vbul))
89|
90|
91| def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
92|     r_m = r_kpc * kpc_to_m
93|     conv = kms2_to_ms2 / r_m
94|     gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
95|          + Yb*np.abs(Vbul)*Vbul*conv)
96|     return np.abs(gN)
97|
98|
99| def partial_corr(x, y, controls):
100|     if len(controls) == 0:
101|         return pearsonr(x, y)
102|     C = np.column_stack(controls + [np.ones(len(x))])
103|     cx, _, _ = lstsq(C, x, rcond=None)
104|     cy, _, _ = lstsq(C, y, rcond=None)
105|     return pearsonr(x - C @ cx, y - C @ cy)
106|
107|
108| def fit_gc_deep(r_kpc, gN, Vobs, errV):
109|     e = np.where(errV > 0, errV, np.median(Vobs) * 0.1)
110|     def chi2(lgc):
111|         gc = 10**lgc
112|         V_pred = np.sqrt(r_kpc * kpc_to_m * np.sqrt(gN * gc)) * 1e-3
113|         return np.sum(((Vobs - V_pred) / e)**2)
114|     res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
115|     return 10**res.x
116|
117|
118| # =====
119| # ε(r) と勾配エネルギーの計算
120| # =====
121| def compute_epsilon_profile(r_kpc, gN, gc):
122|     """平衡ε プロファイル
123|     平衡条件:  $(c - 1 + \epsilon^2)/(1 - \epsilon) = g_N/gc = x$ 
124|     ->  $\epsilon = [-x + \sqrt{(x+2)^2 - 4c}] / 2$  where  $c = gc/a_0$ 
125|
126|     深MOND近似 ( $x \ll 1$ ):  $\epsilon \approx \sqrt{c} - 1 + x/(2\sqrt{c}) + \dots$ 
127|     ->  $\epsilon$  はほぼ定数 +  $x$  に比例する小さな変動
128|
129|     しかし  $c \ll 1$  の銀河では上の解は複素数になる可能性。
130|     代替: 深MOND で  $g_{\text{eff}} = \sqrt{g_N x gc}$  から
131|      $\epsilon$  を 「 $g_{\text{eff}}/gc - 1$  の関数」として定義:
132|      $y = g_{\text{eff}}/gc$  とすると  $\mu(y) = g_N/g_{\text{eff}}$  ->  $\epsilon$  は  $\mu$  の逆関数的量
133|
134|     実用的定義:  $\epsilon(r) = \sqrt{g_N(r)/gc}$  (深MOND限定の正規化歪み)
135|     """
136|     x = gN / gc
137|     epsilon = np.sqrt(x) # 深MOND近似での歪みパラメータ
138|     return epsilon
139|
140|
141| def compute_strain_energies(r_kpc, gN, gc, hR):
142|     """歪みエネルギーの局所項と勾配項を計算"""
143|     eps = compute_epsilon_profile(r_kpc, gN, gc)
144|     r_norm = r_kpc / hR # hR 正規化
145|
146|     N = len(r_kpc)
147|     result = {}
148|
149|     # --- 局所エネルギー:  $E_{\text{local}} = \int \epsilon^2 x r dr$  ---

```

```

150| if N > 2:
151|     result["E_local"] = np.trapz(eps**2 * r_kpc, r_kpc)
152| else:
153|     result["E_local"] = np.nan
154|
155| # --- 勾配エネルギー: E_grad = int (dε/dr)^2 x r dr ---
156| if N > 3:
157|     dr = np.diff(r_kpc)
158|     deps = np.diff(eps)
159|     mask = dr > 0
160|     if mask.sum() > 2:
161|         dedr = deps[mask] / dr[mask] # dε/dr [1/kpc]
162|         r_mid = (r_kpc[:-1] + r_kpc[1:])[mask] / 2.0
163|         result["E_grad"] = np.trapz(dedr**2 * r_mid, r_mid)
164|
165|         # hR正規化版: (dε/d(r/hR))^2 = (hR x dε/dr)^2
166|         dedr_norm = dedr * hR
167|         result["E_grad_norm"] = np.trapz(dedr_norm**2 * r_mid, r_mid)
168|
169|         # 勾配のピーク値
170|         result["grad_peak"] = np.max(np.abs(dedr))
171|         result["grad_peak_norm"] = np.max(np.abs(dedr_norm))
172|
173|         # 勾配ピーク位置 / hR
174|         idx_gp = np.argmax(np.abs(dedr))
175|         result["r_grad_peak_hR"] = r_mid[idx_gp] / hR
176|
177|         # 勾配の外側 vs 内側の比
178|         n_half = len(dedr) // 2
179|         if n_half >= 2:
180|             result["grad_outer_inner"] = (
181|                 np.median(np.abs(dedr[n_half:])) /
182|                 (np.median(np.abs(dedr[:n_half])) + 1e-30))
183|         else:
184|             result["grad_outer_inner"] = np.nan
185|     else:
186|         for k in ["E_grad", "E_grad_norm", "grad_peak", "grad_peak_norm",
187|                 "r_grad_peak_hR", "grad_outer_inner"]:
188|             result[k] = np.nan
189| else:
190|     for k in ["E_grad", "E_grad_norm", "grad_peak", "grad_peak_norm",
191|             "r_grad_peak_hR", "grad_outer_inner"]:
192|         result[k] = np.nan
193|
194| # --- 勾配/局所比: E_grad/E_local ---
195| if np.isfinite(result.get("E_grad", np.nan)) and result.get("E_local", 0) > 0:
196|     result["grad_local_ratio"] = result["E_grad"] / result["E_local"]
197| else:
198|     result["grad_local_ratio"] = np.nan
199|
200| # --- 理論的予測: E_grad_theory ~ (gN_peak/(gcxhR))^2 x hR^3 ---
201| gN_peak = np.max(gN)
202| result["E_grad_theory"] = (gN_peak / (gc * hR * kpc_to_m))**2 * (hR * kpc_to_m)**3
203|
204| # --- 正規化勾配エネルギー密度: e_grad = E_grad / (hR^2 x gc^2) ---
205| # 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
206| if np.isfinite(result.get("E_grad", np.nan)):
207|     result["e_grad_density"] = result["E_grad"] / ((hR)**2 * gc**2 + 1e-30)
208| else:
209|     result["e_grad_density"] = np.nan
210|
211| return result
212|
213|
214| # =====
215| def main():
216|     print("=" * 70)
217|     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")
218|     print("=" * 70)
219|
220|     gc_table = load_gc_table(GC_CSV)
221|
222|     records = []
223|     for gal_name, info in gc_table.items():
224|         rotmod = load_rotmod(gal_name, ROTMOD_DIR)
225|         if rotmod is None:
226|             continue
227|         r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
228|         if len(r_kpc) < 8:
229|             continue
230|
231|         gc_obs = info["gc_obs"]
232|         gc_obs_ms2 = gc_obs * a0 if gc_obs > 1e-5 else gc_obs
233|         hR = info["hR"]
234|         vflat = info["vflat"]
235|
236|         gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)

```

```

237|         gc_deep = fit_gc_deep(r_kpc, gN, Vobs, errV)
238|
239|         # 歪みエネルギー (gc_obs ベースと gc_deep ベースの両方)
240|         strain_obs = compute_strain_energies(r_kpc, gN, gc_obs_ms2, hR)
241|         strain_deep = compute_strain_energies(r_kpc, gN, gc_deep, hR)
242|
243|         # x_outer
244|         n_out = max(3, len(r_kpc) // 3)
245|         x_outer = np.median(gN[-n_out:] / gc_deep) if gc_deep > 0 else np.nan
246|
247|         rmax_hR = r_kpc[-1] / hR if hR > 0 else np.nan
248|
249|         records.append({
250|             "name": gal_name,
251|             "gc_obs": gc_obs_ms2,
252|             "gc_deep": gc_deep,
253|             "hR": hR, "vflat": vflat,
254|             "x_outer": x_outer, "rmax_hR": rmax_hR,
255|             # gc_deep ベースの勾配量 (循環を避ける)
256|             "E_grad_d": strain_deep.get("E_grad", np.nan),
257|             "E_grad_norm_d": strain_deep.get("E_grad_norm", np.nan),
258|             "E_local_d": strain_deep.get("E_local", np.nan),
259|             "grad_peak_d": strain_deep.get("grad_peak", np.nan),
260|             "grad_peak_norm_d": strain_deep.get("grad_peak_norm", np.nan),
261|             "grad_local_ratio_d": strain_deep.get("grad_local_ratio", np.nan),
262|             "r_grad_peak_hR_d": strain_deep.get("r_grad_peak_hR", np.nan),
263|             "grad_oi_d": strain_deep.get("grad_outer_inner", np.nan),
264|             "e_grad_density_d": strain_deep.get("e_grad_density", np.nan),
265|             # gc_obs ベース (循環の可能性あり、参考用)
266|             "E_grad_o": strain_obs.get("E_grad", np.nan),
267|             "grad_local_ratio_o": strain_obs.get("grad_local_ratio", np.nan),
268|         })
269|
270|     N = len(records)
271|     print(f"Processed: {N}")
272|
273|     def arr(k): return np.array([r[k] for r in records], dtype=float)
274|
275|     gc_obs = arr("gc_obs"); gc_deep = arr("gc_deep")
276|     hR_a = arr("hR"); vflat_a = arr("vflat")
277|     x_outer = arr("x_outer"); rmax_hR = arr("rmax_hR")
278|
279|     log_gc = np.log10(gc_obs); log_gcd = np.log10(gc_deep)
280|     log_hR = np.log10(hR_a); log_vf = np.log10(vflat_a)
281|     log_x = np.log10(x_outer + 1e-10)
282|
283|     # 勾配量 (gc_deep ベース)
284|     E_grad = arr("E_grad_d")
285|     E_grad_norm = arr("E_grad_norm_d")
286|     E_local = arr("E_local_d")
287|     grad_peak = arr("grad_peak_d")
288|     grad_peak_norm = arr("grad_peak_norm_d")
289|     grad_local = arr("grad_local_ratio_d")
290|     r_grad_hR = arr("r_grad_peak_hR_d")
291|     grad_oi = arr("grad_oi_d")
292|     e_grad_dens = arr("e_grad_density_d")
293|
294|     m = (np.isfinite(log_gc) & np.isfinite(log_hR) & np.isfinite(log_vf)
295|          & np.isfinite(log_x) & np.isfinite(log_gcd)
296|          & np.isfinite(E_grad) & (E_grad > 0)
297|          & np.isfinite(E_local) & (E_local > 0))
298|
299|     print(f"Valid: {m.sum()}")
300|
301|     log_Eg = np.log10(E_grad + 1e-30)
302|     log_Egn = np.log10(E_grad_norm + 1e-30)
303|     log_El = np.log10(E_local + 1e-30)
304|     log_glr = np.log10(grad_local + 1e-30)
305|     log_gp = np.log10(grad_peak + 1e-30)
306|     log_gpn = np.log10(grad_peak_norm + 1e-30)
307|
308|     # =====
309|     # Test 1: 勾配パラメータの基本統計
310|     # =====
311|     print("\n" + "=" * 50)
312|     print("Test 1: 勾配パラメータの基本統計")
313|     print("=" * 50)
314|
315|     grad_params = {
316|         "E_grad": E_grad,
317|         "E_grad_norm": E_grad_norm,
318|         "E_local": E_local,
319|         "grad_peak": grad_peak,
320|         "grad_peak_norm": grad_peak_norm,
321|         "E_grad/E_local": grad_local,
322|         "r_grad_peak/hR": r_grad_hR,
323|         "grad_outer/inner": grad_oi,

```

```

324|     "e_grad_density": e_grad_dens,
325| }
326|
327| for name, ap in grad_params.items():
328|     v = np.isfinite(ap) & m & (ap > 0)
329|     if v.sum() > 10:
330|         print(f" {name:20s}: N={v.sum():3d} "
331|               f"median={np.median(ap[v]):.4e} "
332|               f"std/mean={np.std(ap[v])/np.mean(ap[v]):.3f}")
333|
334| # =====
335| # Test 2: 勾配量 vs hR, x_outer の相関 [*]核心[*]
336| # =====
337| print("\n" + "=" * 50)
338| print("Test 2: 勾配量 vs hR, x_outer, gc の相関")
339| print("=" * 50)
340|
341| print(f" 'パラメータ':<20s} {'rho(hR)':>10s} {'rho(x)':>10s} "
342|       f"{'rho(gc)':>10s} {'rho(hR|x)':>10s}")
343| print(" " + "-" * 64)
344|
345| key_params = {}
346| for name, ap in grad_params.items():
347|     v = np.isfinite(ap) & m & (ap > 0)
348|     if v.sum() < 20:
349|         continue
350|     log_ap = np.log10(ap[v] + 1e-30)
351|
352|     rh, _ = pearsonr(log_hR[v], log_ap)
353|     rx, _ = pearsonr(log_x[v], log_ap)
354|     rg, _ = pearsonr(log_gc[v], log_ap)
355|     # hR|x_outer 偏相関: 勾配量が x を超える hR 情報を持つか
356|     rhx, phx = partial_corr(log_hR[v], log_ap, [log_x[v]])
357|
358|     print(f" {name:<20s} {rh:+.4f} {rx:+.4f} {rg:+.4f} {rhx:+.4f}")
359|     key_params[name] = (ap, log_ap, v, rhx)
360|
361| # =====
362| # Test 3: 勾配量を統制した hR 偏相関 [*][*]核心[*][*]
363| # =====
364| print("\n" + "=" * 50)
365| print("Test 3: 勾配量統制後の hR 偏相関")
366| print("=" * 50)
367|
368| base_controls = [log_vf[m], log_x[m]]
369| rho_base, p_base = partial_corr(log_hR[m], log_gc[m], base_controls)
370| print(f" ベースライン |vflat, x: rho = {rho_base:+.4f}")
371|
372| print(f"\n '追加パラメータ':<20s} {'rho':>8s} {'rho'|>10s} {'p':>12s}")
373| print(" " + "-" * 52)
374|
375| best_delta = 0; best_name = ""; best_ctrl = None
376| for name, ap in grad_params.items():
377|     v = np.isfinite(ap) & m & (ap > 0)
378|     if v.sum() < m.sum() * 0.7:
379|         continue
380|     ap_filled = ap.copy()
381|     ap_filled[(np.isfinite(ap) & (ap > 0))] = np.nanmedian(ap[ap > 0])
382|     ctrl = np.log10(ap_filled[m] + 1e-30)
383|
384|     try:
385|         rho_new, p_new = partial_corr(log_hR[m], log_gc[m],
386|                                     base_controls + [ctrl])
387|         delta = abs(rho_new) - abs(rho_base)
388|         print(f" {name:<20s} {rho_new:+.4f} {delta:+.4f} p={p_new:.3e}")
389|         if delta < best_delta:
390|             best_delta = delta; best_name = name; best_ctrl = ctrl
391|     except:
392|         pass
393|
394| if best_delta < -0.01:
395|     print(f"\n [*] 最も効果的: {best_name} ( $\Delta = {best\_delta:+.4f}$ )")
396|     pct = -best_delta / 0.472 * 100
397|     print(f" -> hR偏相関の追加 {pct:.1f}% を説明")
398| else:
399|     print(f"\n 勾配パラメータも hR 偏相関を有意に減らさない")
400|
401| # =====
402| # Test 4: E_grad/E_local 比の独立性
403| # =====
404| print("\n" + "=" * 50)
405| print("Test 4: E_grad/E_local 比の独立性")
406| print("=" * 50)
407|
408| v_glr = np.isfinite(grad_local) & m & (grad_local > 0)
409| if v_glr.sum() > 20:
410|     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) & m & (ap > 0)
436|     if v.sum() < 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 > 8 サブセット
448| # =====
449| print("\n" + "=" * 50)
450| print("Test 6: rmax/hR > 8 サブセット")
451| print("=" * 50)
452|
453| m8 = m & (rmax_hR > 8)
454| if m8.sum() > 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>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 & np.isfinite(ap) & (ap > 0)
463|         if v8.sum() < 10:
464|             continue
465|         ap_f = ap.copy()
466|         ap_f[~(np.isfinite(ap) & (ap > 0))] = np.nanmedian(ap[ap > 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 > x > 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) & (glr_filled > 0))] = np.nanmedian(
486|     grad_local[np.isfinite(grad_local) & (grad_local > 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":

```

```

498|     steps.append(
499|         (f"|vflat, x, {best_name}", [log_vf[m], log_x[m], best_ctrl])
500|     )
501|     # 両方同時
502|     steps.append(
503|         (f"|vflat, x, ratio, {best_name}",
504|          [log_vf[m], log_x[m], log_glr_all, best_ctrl])
505|     )
506|
507| print(f" {'統制変数':<40s} {'rho':<8s} {'p':<12s}")
508| print(" " + "-" * 62)
509| for label, ctrl in steps:
510|     try:
511|         rho_s, p_s = partial_corr(log_hR[m], log_gc[m], ctrl)
512|         print(f" {label:<40s} {rho_s:+.4f} p={p_s:.3e}")
513|     except Exception as e:
514|         print(f" {label:<40s} ERROR: {e}")
515|
516| # =====
517| # Test 8: 理論的スケーリングの検証
518| # =====
519| print("#n" + "-" * 50)
520| print("Test 8: 理論的スケーリング")
521| print("#" * 50)
522|
523| # E_grad ~ hR^a の a を測定
524| v_eg = np.isfinite(E_grad) & m & (E_grad > 0)
525| if v_eg.sum() > 20:
526|     sl_eg = linregress(log_hR[v_eg], np.log10(E_grad[v_eg]))
527|     print(f" E_grad vs hR: slope = {sl_eg.slope:.4f} +/- {sl_eg.stderr:.4f}")
528|     print(f" 理論予測: E_grad ~ hR^(-2+3) = hR^1 (次元から)")
529|
530| # E_local ~ hR^a
531| v_el = np.isfinite(E_local) & m & (E_local > 0)
532| if v_el.sum() > 20:
533|     sl_el = linregress(log_hR[v_el], np.log10(E_local[v_el]))
534|     print(f" E_local vs hR: slope = {sl_el.slope:.4f} +/- {sl_el.stderr:.4f}")
535|
536| # E_grad/E_local ~ hR^a -& aが負なら hR 小でgradient支配
537| if v_glr.sum() > 20:
538|     print(f" E_grad/E_local vs hR: slope = {sl.slope:.4f} (再掲)")
539|     if sl.slope < -0.5:
540|         print(f" -& hR が小さい銀河ほど勾配項が支配的 [*]")
541|
542| # =====
543| # SUMMARY
544| # =====
545| print("#n" + "-" * 70)
546| print("SUMMARY")
547| print("#" * 70)
548|
549| print(f"#n hR偏相関の段階的分解:")
550| print(f" |vflat: rho = -0.472")
551| print(f" |vflat, x: rho = {rho_base:+.4f} (x_outer 69%)")
552| if best_delta < -0.01:
553|     rho_best, _ = partial_corr(log_hR[m], log_gc[m],
554|                               base_controls + [best_ctrl])
555|     print(f" |vflat, x, {best_name}: rho = {rho_best:+.4f} "
556|           f"(追加 {-best_delta/0.472*100:.1f}%)")
557|
558| print(f"#n 勾配エネルギー仮説の評価:")
559| if best_delta < -0.02:
560|     print(f" -& 勾配項が hR 偏相関の追加分を部分的に説明 [*]")
561|     print(f" -& 膜の歪み勾配エネルギーは物理的に妥当な候補")
562| elif best_delta < -0.01:
563|     print(f" -& わずかな効果あり ({-best_delta/0.472*100:.1f}%) だが決定的ではない")
564| else:
565|     print(f" -& 勾配項も残り31%を説明しない")
566|
567| if v_glr.sum() > 20 and sl.slope < -1.0:
568|     print(f"#n ただし E_grad/E_local ~ hR^{sl.slope:.1f} は物理的に興味深い:")
569|     print(f" 小さい hR の銀河で勾配項が相対的に重要 -& hR 依存性の定性的起源")
570|
571|
572| if __name__ == "__main__":
573|     main()
574|

```

## 9. トートロジーチェック

ファイル: sparc\_tautology\_check.py

### 解析目的

$E_{\text{grad\_norm}} = hR^2 \times E_{\text{grad}}$  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:  
 $E_{\text{grad}}$  単独、ブラシーボ ( $E_{\text{local}} \times hR^2$ )、ランダム  $\times$   
 $hR^2$ 、分解、 $hR$  シャッフル (1000回)、 $E_{\text{grad}}/E_{\text{local}}$  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  $\alpha = 1.75$ 。  $E_{\text{grad\_norm}}$  26% を撤回。

### ソースコード

(493 行)

```
1| #!/usr/bin/env python3
2| """
3| sparc_tautology_check.py
4| =====
5| 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
6|
7| 問題:
8| E_grad_norm = int(hR x dε/dr)^2 r dr -> 定義に hR が含まれる
9| hR 偏相関を統制すると ρ = -0.145 -> +0.023 に消失
10| -> hR^2 を掛けただけのトートロジーか?
11|
12| テスト:
13| Test 1: E_grad (非正規化、hR非含有) 単独での偏相関
14| Test 2: ブラシーボ -- E_local x hR^2 で同じ効果が出るか
15| Test 3: ブラシーボ -- ランダム量 x hR^2 で同じ効果が出るか
16| Test 4: E_grad_norm = hR^2 x G(r) と分解、G(r) の独立寄与
17| Test 5: hR シャッフルテスト (1000回)
18| Test 6: E_grad/E_local (hR 非含有) の精密評価
19| Test 7: 「任意の X x hR^α」で最適 α を探索 -> α=2 が特別か
20|
21| 実行: uv run --with scipy --with matplotlib python sparc_tautology_check.py
22| """
23|
24| import numpy as np
25| from scipy.optimize import minimize_scalar
26| from scipy.stats import pearsonr
27| from numpy.linalg import lstsq
28| from pathlib import Path
29| import csv
30| import sys
31|
32| a0 = 1.2e-10
33| kpc_to_m = 3.0857e19
34| kms2_to_ms2 = 1.0e6
35| Yd_default = 0.5
36| Yb_default = 0.7
37|
38| 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")
39| ROTMOD_DIR = BASE / "Rotmod_LTG"
40| GC_CSV = BASE / "sparc_gc.csv"
41|
42|
43| def load_gc_table(path):
44|     data = {}
45|     with open(path, "r") as f:
46|         reader = csv.DictReader(f)
47|         for row in reader:
48|             name = row["Galaxy"].strip()
49|             data[name] = {
50|                 "gc_obs": float(row["gc"]),
51|                 "vflat": float(row["Vflat"]),
52|                 "hR": float(row["hR"]),
53|             }
54|     return data
55|
56|
57| def load_rotmod(galaxy_name, rotmod_dir):
58|     fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
59|     if not fpath.exists():
60|         return None
61|     r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
62|     with open(fpath, "r") as f:
```

```

63|         for line in f:
64|             line = line.strip()
65|             if not line or line.startswith("#"):
66|                 continue
67|             parts = line.split()
68|             if len(parts) < 6:
69|                 continue
70|             r.append(float(parts[0]))
71|             vobs.append(float(parts[1]))
72|             errv.append(float(parts[2]))
73|             vgas.append(float(parts[3]))
74|             vdisk.append(float(parts[4]))
75|             vbul.append(float(parts[5]))
76|         return (np.array(r), np.array(vobs), np.array(errv),
77|                np.array(vgas), np.array(vdisk), np.array(vbul))
78|
79|
80| def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
81|     r_m = r_kpc * kpc_to_m
82|     conv = kms2_to_ms2 / r_m
83|     gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
84|           + Yb*np.abs(Vbul)*Vbul*conv)
85|     return np.abs(gN)
86|
87|
88| def partial_corr(x, y, controls):
89|     if len(controls) == 0:
90|         return pearsonr(x, y)
91|     C = np.column_stack(controls + [np.ones(len(x))])
92|     cx, _, _ = lstsq(C, x, rcond=None)
93|     cy, _, _ = lstsq(C, y, rcond=None)
94|     return pearsonr(x - C @ cx, y - C @ cy)
95|
96|
97| def fit_gc_deep(r_kpc, gN, Vobs, errV):
98|     e = np.where(errV > 0, errV, np.median(Vobs) * 0.1)
99|     def chi2(lgc):
100|         gc = 10**lgc
101|         V_pred = np.sqrt(r_kpc * kpc_to_m * np.sqrt(gN * gc)) * 1e-3
102|         return np.sum(((Vobs - V_pred) / e)**2)
103|     res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
104|     return 10**res.x
105|
106|
107| def compute_strain_energies(r_kpc, gN, gc, hR):
108|     eps = np.sqrt(gN / gc)
109|     N = len(r_kpc)
110|     result = {"E_local": np.nan, "E_grad": np.nan, "E_grad_norm": np.nan}
111|
112|     if N < 4:
113|         return result
114|
115|     result["E_local"] = np.trapz(eps**2 * r_kpc, r_kpc)
116|
117|     dr = np.diff(r_kpc)
118|     deps = np.diff(eps)
119|     mask = dr > 0
120|     if mask.sum() < 2:
121|         return result
122|
123|     dedr = deps[mask] / dr[mask]
124|     r_mid = (r_kpc[:-1] + r_kpc[1:])[mask] / 2.0
125|
126|     result["E_grad"] = np.trapz(dedr**2 * r_mid, r_mid)
127|     dedr_norm = dedr * hR
128|     result["E_grad_norm"] = np.trapz(dedr_norm**2 * r_mid, r_mid)
129|
130|     return result
131|
132|
133| # =====
134| def main():
135|     print("=" * 70)
136|     print("トートロジーチェック: E_grad_norm の hR 含有効果")
137|     print("=" * 70)
138|
139|     gc_table = load_gc_table(GC_CSV)
140|
141|     records = []
142|     for gal_name, info in gc_table.items():
143|         rotmod = load_rotmod(gal_name, ROTMOD_DIR)
144|         if rotmod is None:
145|             continue
146|         r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
147|         if len(r_kpc) < 8:
148|             continue
149|

```

```

150|         gc_obs = info["gc_obs"]
151|         gc_obs_ms2 = gc_obs * a0 if gc_obs > 1e-5 else gc_obs
152|         hR = info["hR"]
153|         vflat = info["vflat"]
154|
155|         gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)
156|         gc_deep = fit_gc_deep(r_kpc, gN, Vobs, errV)
157|
158|         strain = compute_strain_energies(r_kpc, gN, gc_deep, hR)
159|
160|         n_out = max(3, len(r_kpc) // 3)
161|         x_outer = np.median(gN[-n_out:] / gc_deep) if gc_deep > 0 else np.nan
162|
163|         records.append({
164|             "gc_obs": gc_obs_ms2, "gc_deep": gc_deep,
165|             "hR": hR, "vflat": vflat, "x_outer": x_outer,
166|             "E_local": strain["E_local"],
167|             "E_grad": strain["E_grad"],
168|             "E_grad_norm": strain["E_grad_norm"],
169|         })
170|
171|     N = len(records)
172|
173|     def arr(k): return np.array([r[k] for r in records], dtype=float)
174|
175|     gc_obs = arr("gc_obs"); hR_a = arr("hR"); vflat_a = arr("vflat")
176|     x_outer = arr("x_outer")
177|     E_local = arr("E_local"); E_grad = arr("E_grad"); E_grad_norm = arr("E_grad_norm")
178|
179|     log_gc = np.log10(gc_obs)
180|     log_hR = np.log10(hR_a)
181|     log_vf = np.log10(vflat_a)
182|     log_x = np.log10(x_outer + 1e-10)
183|
184|     m = (np.isfinite(log_gc) & np.isfinite(log_hR) & np.isfinite(log_vf)
185|          & np.isfinite(log_x) & np.isfinite(E_grad) & (E_grad > 0)
186|          & np.isfinite(E_local) & (E_local > 0)
187|          & np.isfinite(E_grad_norm) & (E_grad_norm > 0))
188|
189|     print(f"Valid: {m.sum()}")
190|
191|     base = [log_vf[m], log_x[m]]
192|     rho_base, p_base = partial_corr(log_hR[m], log_gc[m], base)
193|     print(f"%nべースライン |vflat, x: rho = {rho_base:+.4f}, p = {p_base:.3e}")
194|
195|     # =====
196|     # Test 1: E_grad (非正規化、hR非含有)
197|     # =====
198|     print("%n" + "=" * 50)
199|     print("Test 1: E_grad (非正規化、hR非含有)")
200|     print("=" * 50)
201|
202|     log_Eg = np.log10(E_grad[m])
203|     rho_Eg, p_Eg = partial_corr(log_hR[m], log_gc[m], base + [log_Eg])
204|     print(f" |vflat, x, E_grad: rho = {rho_Eg:+.4f}, p = {p_Eg:.3e}")
205|     print(f" Δ|rho| = {abs(rho_Eg) - abs(rho_base):+.4f}")
206|
207|     # E_grad_norm (比較)
208|     log_Egn = np.log10(E_grad_norm[m])
209|     rho_Egn, p_Egn = partial_corr(log_hR[m], log_gc[m], base + [log_Egn])
210|     print(f" |vflat, x, E_grad_norm: rho = {rho_Egn:+.4f}, p = {p_Egn:.3e}")
211|     print(f" Δ|rho| = {abs(rho_Egn) - abs(rho_base):+.4f}")
212|
213|     diff = abs(rho_Eg) - abs(rho_Egn)
214|     print(f"%n E_grad vs E_grad_norm の差: {diff:+.4f}")
215|     if abs(rho_Eg) > abs(rho_base) * 0.9:
216|         print(" -> E_grad(非正規化)は hR偏相関をほとんど減らさない")
217|         print(" -> E_grad_normの効果は hR^2正規化に依存")
218|     else:
219|         print(f" -> E_grad(非正規化)も偏相関を {(abs(rho_base)-abs(rho_Eg))/abs(rho_base)*100:.1f}% 削減")
220|
221|     # =====
222|     # Test 2: プラシーボ -- E_local x hR^2
223|     # =====
224|     print("%n" + "=" * 50)
225|     print("Test 2: プラシーボ -- E_local x hR^2")
226|     print("=" * 50)
227|
228|     placebo1 = E_local[m] * hR_a[m]**2
229|     log_p1 = np.log10(placebo1 + 1e-30)
230|     rho_p1, p_p1 = partial_corr(log_hR[m], log_gc[m], base + [log_p1])
231|     print(f" |vflat, x, E_local*hR^2: rho = {rho_p1:+.4f}, p = {p_p1:.3e}")
232|     print(f" Δ|rho| = {abs(rho_p1) - abs(rho_base):+.4f}")
233|
234|     # 純粹 hR^2 だけ
235|     log_hR2 = 2 * log_hR[m]
236|     rho_hR2, p_hR2 = partial_corr(log_hR[m], log_gc[m], base + [log_hR2])

```

```

237| print(f"%n |vflat, x, hR^2: rho = {rho_hR2:+.4f}, p = {p_hR2:.3e}")
238| print(f" (hR^2 は hR の単調変換なので、hR統制と同義のはず)")
239|
240| # E_local x hR^n for various n
241| print(f"%n E_local x hR^n のスキャン:")
242| for n in [0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0]:
243|     pn = E_local[m] * hR_a[m]**n
244|     log_pn = np.log10(pn + 1e-30)
245|     rho_pn, _ = partial_corr(log_hR[m], log_gc[m], base + [log_pn])
246|     tag = " &lt;-- E_grad_norm相当" if abs(n - 2.0) &lt; 0.01 else ""
247|     print(f"      n={n:.1f}: rho = {rho_pn:+.4f}{tag}")
248|
249| # =====
250| # Test 3: ランダム量 x hR^2
251| # =====
252| print("%n" + "=" * 50)
253| print("Test 3: ランダム量 x hR^2 のブラシーボ")
254| print("=" * 50)
255|
256| np.random.seed(42)
257| n_trials = 5
258| rho_randoms = []
259| for i in range(n_trials):
260|     rand_qty = np.random.lognormal(0, 1, m.sum())
261|     placebo_rand = rand_qty * hR_a[m]**2
262|     log_pr = np.log10(placebo_rand + 1e-30)
263|     rho_r, _ = partial_corr(log_hR[m], log_gc[m], base + [log_pr])
264|     rho_randoms.append(rho_r)
265|     print(f" Trial {i+1}: rho = {rho_r:+.4f}")
266|
267| print(f" Mean: {np.mean(rho_randoms):+.4f}, std: {np.std(rho_randoms):.4f}")
268| print(f" E_grad_norm: {rho_Egn:+.4f}")
269|
270| if abs(np.mean(rho_randoms)) &lt; abs(rho_base) * 0.3:
271|     print(" -&gt; ランダム x hR^2 でも偏相関がほぼ消える")
272|     print(" -&gt; [*] hR^2 を掛けるだけで偏相関が消える (トートロジー)")
273|     tautology_random = True
274| else:
275|     print(" -&gt; ランダム x hR^2 では偏相関が残る")
276|     print(" -&gt; E_grad_norm の物理的内容が寄与している")
277|     tautology_random = False
278|
279| # =====
280| # Test 4: E_grad_norm = hR^2 x G(r) の分解
281| # =====
282| print("%n" + "=" * 50)
283| print("Test 4: E_grad_norm = hR^2 x G(r) の分解")
284| print("=" * 50)
285|
286| # G(r) = E_grad_norm / hR^2 = E_grad (定義から)
287| # つまり G(r) = E_grad そのもの
288| print(" E_grad_norm = hR^2 x int(dε/dr)^2 r dr = hR^2 x E_grad")
289| print(" つまり G(r) = E_grad (hR非含有成分)")
290| print(f" E_grad のみでの偏相関: rho = {rho_Eg:+.4f} (Test 1 再掲)")
291| print(f" E_grad_norm での偏相関: rho = {rho_Egn:+.4f}")
292| print(f" 差: {abs(rho_Eg) - abs(rho_Egn):+.4f}")
293| print(f"%n -&gt; E_grad_norm の効果 = E_grad の効果 + hR^2 の効果")
294|
295| # E_grad と hR^2 の両方を同時統制
296| rho_both, p_both = partial_corr(log_hR[m], log_gc[m],
297|                                 base + [log_Eg, log_hR2])
298| print(f" |vflat, x, E_grad, hR^2: rho = {rho_both:+.4f}")
299| print(f" (E_grad + hR^2 同時 = E_grad_norm と同等のはず)")
300|
301| # =====
302| # Test 5: hR シャッフルテスト (1000回)
303| # =====
304| print("%n" + "=" * 50)
305| print("Test 5: hR シャッフルテスト (1000回)")
306| print("=" * 50)
307|
308| # 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
309| np.random.seed(123)
310| n_shuffle = 1000
311| rho_shuffled = []
312| for _ in range(n_shuffle):
313|     hR_shuf = hR_a[m].copy()
314|     np.random.shuffle(hR_shuf)
315|     Egn_shuf = E_grad[m] * hR_shuf**2
316|     log_shuf = np.log10(Egn_shuf + 1e-30)
317|     rho_s, _ = partial_corr(log_hR[m], log_gc[m], base + [log_shuf])
318|     rho_shuffled.append(rho_s)
319|
320| rho_shuffled = np.array(rho_shuffled)
321| print(f" Shuffled: mean = {np.mean(rho_shuffled):+.4f}, "
322|       f"std = {np.std(rho_shuffled):.4f}")
323| print(f" Shuffled: median = {np.median(rho_shuffled):+.4f}")

```

```

324| print(f" Shuffled: 5th-95th = [{np.percentile(rho_shuffled,5):+.4f}, "
325|       f"{np.percentile(rho_shuffled,95):+.4f}]")
326| print(f" Real E_grad_norm: rho = {rho_Egn:+.4f}")
327|
328| # E_grad_norm の rho が shuffled 分布の何パーセントスタイルか
329| pct = np.mean(rho_shuffled &lt;= rho_Egn) * 100
330| print(f" Real は shuffled の {pct:.1f} パーセントスタイル")
331|
332| if pct &gt; 5 and pct < 95:
333|     print(" -&gt; Real は shuffled と区別できない")
334|     print(" -&gt; [*] E_grad_norm の偏相関消失は hR^2 x (何か) の構造的効果")
335|     tautology_shuffle = True
336| else:
337|     print(f" -&gt; Real ({rho_Egn:+.4f}) は shuffled 分布の外")
338|     print(f" -&gt; E_grad_norm の物理的内容 (E_grad の gN 構造) が寄与")
339|     tautology_shuffle = False
340|
341| # =====
342| # Test 6: E_grad/E_local (hR 非含有) の精密評価
343| # =====
344| print("%n" + "=" * 50)
345| print("Test 6: E_grad/E_local (hR 非含有) の精密評価")
346| print("=" * 50)
347|
348| ratio = E_grad[m] / E_local[m]
349| log_ratio = np.log10(ratio + 1e-30)
350|
351| rho_ratio, p_ratio = partial_corr(log_hR[m], log_gc[m], base + [log_ratio])
352| print(f" |vflat, x, E_grad/E_local: rho = {rho_ratio:+.4f}, p = {p_ratio:.3e}")
353| print(f" Δ|rho| = {abs(rho_ratio) - abs(rho_base):+.4f}")
354| pct_explained = (abs(rho_base) - abs(rho_ratio)) / 0.472 * 100
355| print(f" hR偏相関全体に対する説明率: {pct_explained:.1f}%")
356|
357| # E_grad/E_local が hR と独立に gc と相関するか
358| rho_ratio_gc, p_rg = partial_corr(log_ratio, log_gc[m], [log_hR[m], log_vf[m]])
359| print(f" rho(ratio, gc | hR, vflat) = {rho_ratio_gc:+.4f}, p = {p_rg:.3e}")
360|
361| # E_grad/E_local と hR の相関
362| rho_ratio_hR, p_rh = pearsonr(log_hR[m], log_ratio)
363| print(f" rho(ratio, hR) = {rho_ratio_hR:+.4f}")
364|
365| # E_grad/E_local と x_outer の相関
366| rho_ratio_x, p_rx = pearsonr(log_x[m], log_ratio)
367| print(f" rho(ratio, x_outer) = {rho_ratio_x:+.4f}")
368|
369| # =====
370| # Test 7: 「X x hR^alpha」で最適alphaを探索
371| # =====
372| print("%n" + "=" * 50)
373| print("Test 7: E_grad x hR^alpha で最適alpha")
374| print("=" * 50)
375|
376| alphas = np.arange(-2.0, 4.1, 0.25)
377| rho_alpha = []
378| for a in alphas:
379|     qty = E_grad[m] * hR_a[m]**a
380|     log_qty = np.log10(qty + 1e-30)
381|     rho_a, _ = partial_corr(log_hR[m], log_gc[m], base + [log_qty])
382|     rho_alpha.append(rho_a)
383|     if abs(a - 0) &lt; 0.01 or abs(a - 2) &lt; 0.01 or abs(rho_a) &lt; 0.05:
384|         print(f" alpha={a:+.2f}: rho = {rho_a:+.4f}")
385|
386| rho_alpha = np.array(rho_alpha)
387| best_idx = np.argmin(np.abs(rho_alpha))
388| best_alpha = alphas[best_idx]
389| best_rho = rho_alpha[best_idx]
390| print(f"%n 最適 alpha = {best_alpha:.2f}: rho = {best_rho:+.4f}")
391| print(f" alpha=0 (E_grad のみ): rho = {rho_alpha[alphas==0][0] if 0 in alphas else 'N/A'}")
392| print(f" alpha=2 (E_grad_norm): rho = {rho_alpha[np.argmin(np.abs(alphas-2))]:+.4f}")
393|
394| # alphaが0付近で最適なら物理的、2付近なら hR^2 効果
395| if abs(best_alpha) &lt; 0.5:
396|     print(" -&gt; 最適alphaが0付近 -&gt; E_grad自体に物理的説明力あり")
397| elif abs(best_alpha - 2) &lt; 0.5:
398|     print(" -&gt; 最適alphaが2付近 -&gt; hR^2正規化が本質 (トートロジー寄り)")
399| else:
400|     print(f" -&gt; 最適alphaが{best_alpha:.1f} -&gt; 中間的")
401|
402| # =====
403| # Test 8: log(hR) を直接統制変数に追加 (自明なテスト)
404| # =====
405| print("%n" + "=" * 50)
406| print("Test 8: hR 自体を統制 (参照テスト)")
407| print("=" * 50)
408|
409| # 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
410| rho_hR_ctrl, p_hR_ctrl = partial_corr(log_hR[m], log_gc[m],

```

```

411|                                     base + [log_hR[m]])
412| print(f" |vflat, x, hR: rho = {rho_hR_ctrl:+.4f}")
413| print(f" (hR を直接統制 -&gt; 偏相関は定義上ゼロ)")
414| print(f" E_grad_norm で rho={rho_Egn:+.4f} はこれにどれだけ近いか:")
415| print(f" -&gt; {abs(rho_Egn)/abs(rho_hR_ctrl)*100:.0f}% of hR直接統制")
416|     if abs(rho_hR_ctrl) &gt; 0.001
417|         else f" -&gt; hR直接統制もゼロ、E_grad_normもゼロ")
418|
419| # =====
420| # SUMMARY
421| # =====
422| print("\n" + "=" * 70)
423| print("SUMMARY: トートロジー判定")
424| print("=" * 70)
425|
426| print(f"\n 偏相関の比較:")
427| print(f" |vflat, x:                rho = {rho_base:+.4f} (ベースライン)")
428| print(f" |vflat, x, E_grad:          rho = {rho_Eg:+.4f} (hR非含有)")
429| print(f" |vflat, x, E_grad/E_local:    rho = {rho_ratio:+.4f} (hR非含有)")
430| print(f" |vflat, x, E_local*hR^2:      rho = {rho_p1:+.4f} (プラシーボ)")
431| print(f" |vflat, x, E_grad_norm:       rho = {rho_Egn:+.4f} (hR含有)")
432| print(f" |vflat, x, hR:              rho = {rho_hR_ctrl:+.4f} (hR直接)")
433|
434| print(f"\n シャッフルテスト:")
435| print(f" shuffled mean = {np.mean(rho_shuffled):+.4f}")
436| print(f" real = {rho_Egn:+.4f}")
437| print(f" percentile = {pct:.1f}%")
438|
439| print(f"\n 最適 alpha = {best_alpha:.2f}")
440|
441| print(f"\n ■ 判定:")
442|
443| # 判定ロジック
444| Eg_effective = abs(rho_Eg) &lt; abs(rho_base) * 0.7
445| ratio_effective = abs(rho_ratio) &lt; abs(rho_base) * 0.7
446| placebo_works = abs(rho_p1) &lt; abs(rho_base) * 0.3
447| shuffle_indistinguishable = (pct &gt; 5 and pct &lt; 95)
448|
449| if placebo_works and shuffle_indistinguishable:
450|     print(" [*] トートロジー確定")
451|     print("   E_local*hR^2 (プラシーボ) で同じ効果が出る")
452|     print("   シャッフルでも区別できない")
453|     print("   -&gt; E_grad_normの26%はhR^2正規化のアーティファクト")
454|     if ratio_effective:
455|         print(f"   ただし E_grad/E_local (hR非含有) で rho={rho_ratio:+.4f}")
456|         pct_real = (abs(rho_base) - abs(rho_ratio)) / 0.472 * 100
457|         print(f"   -&gt; 真の物理的効果: {pct_real:.1f}% (勾配/局所比)")
458|     else:
459|         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")
460| elif not placebo_works and not shuffle_indistinguishable:
461|     print(" [*] 物理的効果確定")
462|     print("   プラシーボでは効果なし、シャッフルでも区別できる")
463|     print("   -&gt; E_grad_normの26%は真の物理的効果")
464| else:
465|     print(" [*] 部分的トートロジー")
466|     print(f"   プラシーボ効果: {'あり' if placebo_works else 'なし'}")
467|     print(f"   シャッフル区別: {'不能' if shuffle_indistinguishable else '可能'}")
468|     if Eg_effective:
469|         pct_Eg = (abs(rho_base) - abs(rho_Eg)) / 0.472 * 100
470|         print(f"   E_grad(hR非含有)の物理的寄与: {pct_Eg:.1f}%")
471|     if ratio_effective:
472|         pct_real = (abs(rho_base) - abs(rho_ratio)) / 0.472 * 100
473|         print(f"   E_grad/E_local(hR非含有)の物理的寄与: {pct_real:.1f}%")
474|
475| # 最終的な hR 偏相関分解
476| print(f"\n ■ hR偏相関の最終分解:")
477| print(f" x_outer (MOND遷移): 69%")
478| if ratio_effective:
479|     pct_real = (abs(rho_base) - abs(rho_ratio)) / 0.472 * 100
480|     print(f"   E_grad/E_local (勾配比, hR非含有): {pct_real:.1f}%")
481|     print(f"   残差: {100 - 69 - pct_real:.1f}%")
482| elif Eg_effective:
483|     pct_Eg = (abs(rho_base) - abs(rho_Eg)) / 0.472 * 100
484|     print(f"   E_grad (勾配, hR非含有): {pct_Eg:.1f}%")
485|     print(f"   残差: {100 - 69 - pct_Eg:.1f}%")
486| else:
487|     print(f"   勾配項 (hR非含有): 効果なし")
488|     print(f"   残差: 31% (未説明)")
489|
490|
491| if __name__ == "__main__":
492|     main()
493|

```

## 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)

```
1| #!/usr/bin/env python3
2| """
3| sparc_grad_ratio_detail.py
4| =====
5| E_grad/E_local の精密評価
6|
7| トートロジーチェックで確定:
8| - E_grad_norm の26%はhR2トートロジー (撤回)
9| - E_grad/E_local (hR非含有) で18%は示唆的だがp=0.49
10|
11| 本スクリプトの目的:
12| E_grad/E_local の物理的寄与をあらゆる角度から精査し、
13| 「弱い本物」か「ノイズ」かを判定する。
14|
15| テスト:
16| Test 1: E_grad/E_local の偏相関の詳細 (p値、CI、効果量)
17| Test 2: rmax/hR サブセットでの有意性
18| Test 3: ブートストラップ信頼区間 (10000回)
19| Test 4: E_grad/E_local のhR・x_outerとの独立性の精密評価
20| Test 5: E_grad/E_local の物理的内容の分解 (何がratioを駆動するか)
21| Test 6: gc残差(obs-deep)との相関の精密評価
22| Test 7: E_grad/E_local をさらに分解: 勾配の形状 vs 振幅
23| Test 8: permutation test (E_grad/E_localの偏相関のp値を正確に計算)
24|
25| 実行: uv run --with scipy --with matplotlib python sparc_grad_ratio_detail.py
26| """
27|
28| import numpy as np
29| from scipy.optimize import minimize_scalar
30| from scipy.stats import pearsonr, spearmanr, linregress
31| from numpy.linalg import lstsq
32| from pathlib import Path
33| import csv
34| import sys
35|
36| a0 = 1.2e-10
37| kpc_to_m = 3.0857e19
38| kms2_to_ms2 = 1.0e6
39| Yd_default = 0.5
40| Yb_default = 0.7
41|
42| 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")
43| ROTMOD_DIR = BASE / "Rotmod_LTG"
44| GC_CSV = BASE / "sparc_gc.csv"
45|
46|
47| def load_gc_table(path):
48|     data = {}
49|     with open(path, "r") as f:
50|         reader = csv.DictReader(f)
51|         for row in reader:
52|             name = row["Galaxy"].strip()
53|             data[name] = {
54|                 "gc_obs": float(row["gc"]),
55|                 "vflat": float(row["Vflat"]),
56|                 "hR": float(row["hR"]),
57|             }
58|     return data
59|
60|
61| def load_rotmod(galaxy_name, rotmod_dir):
62|     fpath = rotmod_dir / f"{galaxy_name}_rotmod.dat"
```

```

63|     if not fpath.exists():
64|         return None
65|     r, vobs, errv, vgas, vdisk, vbul = [], [], [], [], [], []
66|     with open(fpath, "r") as f:
67|         for line in f:
68|             line = line.strip()
69|             if not line or line.startswith("#"):
70|                 continue
71|             parts = line.split()
72|             if len(parts) < 6:
73|                 continue
74|             r.append(float(parts[0]))
75|             vobs.append(float(parts[1]))
76|             errv.append(float(parts[2]))
77|             vgas.append(float(parts[3]))
78|             vdisk.append(float(parts[4]))
79|             vbul.append(float(parts[5]))
80|     return (np.array(r), np.array(vobs), np.array(errv),
81|           np.array(vgas), np.array(vdisk), np.array(vbul))
82|
83|
84| def compute_gN(r_kpc, Vgas, Vdisk, Vbul, Yd=0.5, Yb=0.7):
85|     r_m = r_kpc * kpc_to_m
86|     conv = kms2_to_ms2 / r_m
87|     gN = (np.abs(Vgas)*Vgas*conv + Yd*np.abs(Vdisk)*Vdisk*conv
88|           + Yb*np.abs(Vbul)*Vbul*conv)
89|     return np.abs(gN)
90|
91|
92| def partial_corr(x, y, controls):
93|     if len(controls) == 0:
94|         return pearsonr(x, y)
95|     C = np.column_stack(controls + [np.ones(len(x))])
96|     cx, _, _ = lstsq(C, x, rcond=None)
97|     cy, _, _ = lstsq(C, y, rcond=None)
98|     rx = x - C @ cx
99|     ry = y - C @ cy
100|     return pearsonr(rx, ry)
101|
102|
103| def partial_corr_residuals(x, y, controls):
104|     """偏相関の残差を返す (ブートストラップ用) """
105|     if len(controls) == 0:
106|         return x, y
107|     C = np.column_stack(controls + [np.ones(len(x))])
108|     cx, _, _ = lstsq(C, x, rcond=None)
109|     cy, _, _ = lstsq(C, y, rcond=None)
110|     return x - C @ cx, y - C @ cy
111|
112|
113| def fit_gc_deep(r_kpc, gN, Vobs, errV):
114|     e = np.where(errV > 0, errV, np.median(Vobs) * 0.1)
115|     def chi2(lgc):
116|         gc = 10**lgc
117|         V_pred = np.sqrt(r_kpc * kpc_to_m * np.sqrt(gN * gc)) * 1e-3
118|         return np.sum(((Vobs - V_pred) / e)**2)
119|     res = minimize_scalar(chi2, bounds=(-12, -8), method='bounded')
120|     return 10**res.x
121|
122|
123| def compute_strain(r_kpc, gN, gc, hR):
124|     eps = np.sqrt(gN / gc)
125|     N = len(r_kpc)
126|     if N < 4:
127|         return np.nan, np.nan, np.nan, None, None
128|
129|     E_local = np.trapz(eps**2 * r_kpc, r_kpc)
130|
131|     dr = np.diff(r_kpc)
132|     deps = np.diff(eps)
133|     mask = dr > 0
134|     if mask.sum() < 2:
135|         return E_local, np.nan, np.nan, None, None
136|
137|     dedr = deps[mask] / dr[mask]
138|     r_mid = (r_kpc[:-1] + r_kpc[1:])[mask] / 2.0
139|     E_grad = np.trapz(dedr**2 * r_mid, r_mid)
140|
141|     ratio = E_grad / E_local if E_local > 0 else np.nan
142|
143|     return E_local, E_grad, ratio, dedr, r_mid
144|
145|
146| # =====
147| def main():
148|     print("=" * 70)
149|     print("E_grad/E_local の精密評価")

```

```

150| print("=" * 70)
151|
152| gc_table = load_gc_table(GC_CSV)
153|
154| records = []
155| for gal_name, info in gc_table.items():
156|     rotmod = load_rotmod(gal_name, ROTMOD_DIR)
157|     if rotmod is None:
158|         continue
159|     r_kpc, Vobs, errV, Vgas, Vdisk, Vbul = rotmod
160|     if len(r_kpc) < 8:
161|         continue
162|
163|     gc_obs = info["gc_obs"]
164|     gc_obs_ms2 = gc_obs * a0 if gc_obs > 1e-5 else gc_obs
165|     hR = info["hR"]
166|     vflat = info["vflat"]
167|
168|     gN = compute_gN(r_kpc, Vgas, Vdisk, Vbul)
169|     gc_deep = fit_gc_deep(r_kpc, gN, Vobs, errV)
170|
171|     E_local, E_grad, ratio, dedr, r_mid = compute_strain(r_kpc, gN, gc_deep, hR)
172|
173|     n_out = max(3, len(r_kpc) // 3)
174|     x_outer = np.median(gN[-n_out:] / gc_deep) if gc_deep > 0 else np.nan
175|     rmax_hR = r_kpc[-1] / hR if hR > 0 else np.nan
176|
177|     # 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)
178|     grad_shape = {}
179|     if dedr is not None and len(dedr) > 4:
180|         # 勾配の集中度: ピーク付近 vs 全体
181|         abs_dedr = np.abs(dedr)
182|         peak_idx = np.argmax(abs_dedr)
183|         grad_shape["grad_peak_frac"] = abs_dedr[peak_idx] / (np.mean(abs_dedr) + 1e-30)
184|
185|         # 勾配の非対称性: 内側 vs 外側
186|         n_half = len(dedr) // 2
187|         if n_half >= 2:
188|             grad_shape["grad_asymm"] = (
189|                 np.mean(abs_dedr[:n_half]) / (np.mean(abs_dedr[n_half:]) + 1e-30))
190|
191|         # 勾配ピーク位置/hR
192|         grad_shape["grad_peak_r_hR"] = r_mid[peak_idx] / hR if hR > 0 else np.nan
193|
194|         # 勾配の2次モーメント (幅)
195|         total = np.sum(abs_dedr)
196|         if total > 0:
197|             r_mean = np.sum(abs_dedr * r_mid) / total
198|             r_var = np.sum(abs_dedr * (r_mid - r_mean)**2) / total
199|             grad_shape["grad_width_hR"] = np.sqrt(r_var) / hR if hR > 0 else np.nan
200|         else:
201|             grad_shape["grad_width_hR"] = np.nan
202|     else:
203|         grad_shape = {"grad_peak_frac": np.nan, "grad_asymm": np.nan,
204|                       "grad_peak_r_hR": np.nan, "grad_width_hR": np.nan}
205|
206|     # バルジ/ガス割合
207|     Vbar_sq = (np.abs(Vgas)*Vgas + Yd_default*np.abs(Vdisk)*Vdisk
208|               + Yb_default*np.abs(Vbul)*Vbul)
209|     va = np.abs(Vbar_sq)
210|     v = va > 0
211|     f_bul = np.median(Yb_default*Vbul[v]**2 / va[v]) if v.sum() > 0 else 0
212|     f_gas = np.median(Vgas[v]**2 / va[v]) if v.sum() > 0 else 0
213|
214|     records.append({
215|         "gc_obs": gc_obs_ms2, "gc_deep": gc_deep,
216|         "hR": hR, "vflat": vflat,
217|         "x_outer": x_outer, "rmax_hR": rmax_hR,
218|         "E_local": E_local, "E_grad": E_grad, "ratio": ratio,
219|         "f_bul": f_bul, "f_gas": f_gas,
220|         **grad_shape,
221|     })
222|
223| N = len(records)
224| def arr(k): return np.array([r[k] for r in records], dtype=float)
225|
226| gc_obs = arr("gc_obs"); gc_deep = arr("gc_deep")
227| hR_a = arr("hR"); vflat_a = arr("vflat")
228| x_outer = arr("x_outer"); rmax_hR = arr("rmax_hR")
229| E_local = arr("E_local"); E_grad = arr("E_grad"); ratio = arr("ratio")
230| f_bul = arr("f_bul"); f_gas = arr("f_gas")
231| grad_pf = arr("grad_peak_frac"); grad_as = arr("grad_asymm")
232| grad_pr = arr("grad_peak_r_hR"); grad_wd = arr("grad_width_hR")
233|
234| log_gc = np.log10(gc_obs); log_gcd = np.log10(gc_deep)
235| log_hR = np.log10(hR_a); log_vf = np.log10(vflat_a)
236| log_x = np.log10(x_outer + 1e-10)

```

```

237| log_ratio = np.log10(ratio + 1e-30)
238|
239| m = (np.isfinite(log_gc) & np.isfinite(log_hR) & np.isfinite(log_vf)
240|      & np.isfinite(log_x) & np.isfinite(log_ratio)
241|      & np.isfinite(E_grad) & (E_grad > 0)
242|      & np.isfinite(E_local) & (E_local > 0))
243|
244| print(f"Valid: {m.sum()}")
245|
246| base = [log_vf[m], log_x[m]]
247| rho_base, p_base = partial_corr(log_hR[m], log_gc[m], base)
248| print(f"ベースライン |vflat, x: rho = {rho_base:+.4f}, p = {p_base:.3e}")
249|
250| # =====
251| # Test 1: E_grad/E_local 偏相関の詳細
252| # =====
253| print("\n" + "=" * 50)
254| print("Test 1: E_grad/E_local 偏相関の詳細")
255| print("=" * 50)
256|
257| rho_r, p_r = partial_corr(log_hR[m], log_gc[m], base + [log_ratio[m]])
258| print(f" |vflat, x, ratio: rho = {rho_r:+.4f}, p = {p_r:.3e}")
259| delta_pct = (abs(rho_base) - abs(rho_r)) / 0.472 * 100
260| print(f" hR偏相関全体に対する削減: {delta_pct:.1f}%")
261|
262| # Spearman
263| # 残差を取ってからSpearman
264| rx_hR, rx_gc = partial_corr_residuals(log_hR[m], log_gc[m], base)
265| rho_sp_base, p_sp_base = spearmanr(rx_hR, rx_gc)
266| rx_hR2, rx_gc2 = partial_corr_residuals(log_hR[m], log_gc[m], base + [log_ratio[m]])
267| rho_sp_r, p_sp_r = spearmanr(rx_hR2, rx_gc2)
268| print(f"\n Spearman (残差):")
269| print(f"   base: rho_s = {rho_sp_base:+.4f}, p = {p_sp_base:.3e}")
270| print(f" +ratio: rho_s = {rho_sp_r:+.4f}, p = {p_sp_r:.3e}")
271|
272| # 効果量 (Cohen's f^2)
273| R2_base = rho_base**2
274| R2_full = rho_r**2
275| # partial R^2 の変化
276| delta_R2 = R2_base - R2_full
277| print(f"\n 効果量:")
278| print(f"   partial R^2 base: {R2_base:.4f}")
279| print(f"   partial R^2 +ratio: {R2_full:.4f}")
280| print(f"   ΔR^2: {delta_R2:.4f}")
281|
282| # =====
283| # Test 2: rmax/hR サブセット
284| # =====
285| print("\n" + "=" * 50)
286| print("Test 2: rmax/hR サブセットでの ratio 効果")
287| print("=" * 50)
288|
289| for rcut in [3, 5, 8, 10]:
290|     mc = m & (rmax_hR > rcut)
291|     if mc.sum() < 15:
292|         continue
293|     rb, pb = partial_corr(log_hR[mc], log_gc[mc], [log_vf[mc], log_x[mc]])
294|     rr, pr = partial_corr(log_hR[mc], log_gc[mc],
295|                          [log_vf[mc], log_x[mc], log_ratio[mc]])
296|     delta = abs(rb) - abs(rr)
297|     print(f" rmax/hR > {rcut:2d} (N={mc.sum():3d}): "
298|           f"base={rb:+.4f}, +ratio={rr:+.4f}, Δ|ρ|={delta:+.4f}, p={pr:.3e}")
299|
300| # =====
301| # Test 3: ブートストラップ信頼区間
302| # =====
303| print("\n" + "=" * 50)
304| print("Test 3: ブートストラップ信頼区間 (5000回)")
305| print("=" * 50)
306|
307| np.random.seed(42)
308| n_boot = 5000
309| n_samp = m.sum()
310| idx_all = np.where(m)[0]
311|
312| rho_boot_base = []
313| rho_boot_ratio = []
314| delta_boot = []
315|
316| for _ in range(n_boot):
317|     idx = np.random.choice(n_samp, n_samp, replace=True)
318|     try:
319|         rb, _ = partial_corr(log_hR[m][idx], log_gc[m][idx],
320|                             [log_vf[m][idx], log_x[m][idx]])
321|         rr, _ = partial_corr(log_hR[m][idx], log_gc[m][idx],
322|                             [log_vf[m][idx], log_x[m][idx], log_ratio[m][idx]])
323|         rho_boot_base.append(rb)

```

```

324|         rho_boot_ratio.append(rr)
325|         delta_boot.append(abs(rb) - abs(rr))
326|     except:
327|         pass
328|
329| rho_boot_base = np.array(rho_boot_base)
330| rho_boot_ratio = np.array(rho_boot_ratio)
331| delta_boot = np.array(delta_boot)
332|
333| print(f" 成功: {len(delta_boot)}/{n_boot}")
334| print(f"%n rho(base) の 95% CI: [{np.percentile(rho_boot_base, 2.5):.4f}, "
335|       f"{np.percentile(rho_boot_base, 97.5):.4f}]")
336| print(f" rho(+ratio) の 95% CI: [{np.percentile(rho_boot_ratio, 2.5):.4f}, "
337|       f"{np.percentile(rho_boot_ratio, 97.5):.4f}]")
338| print(f"%n Δ|rho| の分布:")
339| print(f" mean = {np.mean(delta_boot):.4f}")
340| print(f" median = {np.median(delta_boot):.4f}")
341| print(f" 95% CI = [{np.percentile(delta_boot, 2.5):.4f}, "
342|       f"{np.percentile(delta_boot, 97.5):.4f}]")
343| print(f" P(Δ > 0) = {np.mean(delta_boot > 0):.3f}")
344|
345| if np.percentile(delta_boot, 2.5) > 0:
346|     print(" -> 95% CI が正の領域にある -> ratio の改善は有意")
347| elif np.mean(delta_boot > 0) > 0.8:
348|     print(" -> P(Δ>0)>80% -> ratio の改善は probable だが決定的でない")
349| else:
350|     print(" -> 改善はノイズの範囲")
351|
352| # =====
353| # Test 4: ratio と hR, x_outer の独立性
354| # =====
355| print("%n" + "=" * 50)
356| print("Test 4: ratio の独立性")
357| print("=" * 50)
358|
359| rho_rh, p_rh = pearsonr(log_hR[m], log_ratio[m])
360| rho_rx, p_rx = pearsonr(log_x[m], log_ratio[m])
361| rho_rv, p_rv = pearsonr(log_vf[m], log_ratio[m])
362| rho_rhx, p_rhx = partial_corr(log_hR[m], log_ratio[m], [log_x[m]])
363| rho_rvx, p_rvx = partial_corr(log_x[m], log_ratio[m], [log_hR[m]])
364|
365| print(f" ratio vs hR: rho = {rho_rh:.4f}, p = {p_rh:.3e}")
366| print(f" ratio vs x: rho = {rho_rx:.4f}, p = {p_rx:.3e}")
367| print(f" ratio vs vflat: rho = {rho_rv:.4f}, p = {p_rv:.3e}")
368| print(f" ratio vs hR|x: rho = {rho_rhx:.4f}, p = {p_rhx:.3e}")
369| print(f" ratio vs x|hR: rho = {rho_rvx:.4f}, p = {p_rvx:.3e}")
370|
371| # ratio が gc と独立に相関するか (hR, vflat 統制後)
372| rho_rgc, p_rgc = partial_corr(log_ratio[m], log_gc[m],
373|                               [log_hR[m], log_vf[m]])
374| print(f"%n ratio vs gc | hR, vflat: rho = {rho_rgc:.4f}, p = {p_rgc:.3e}")
375|
376| rho_rgcx, p_rgcx = partial_corr(log_ratio[m], log_gc[m],
377|                               [log_hR[m], log_vf[m], log_x[m]])
378| print(f" ratio vs gc | hR, vflat, x: rho = {rho_rgcx:.4f}, p = {p_rgcx:.3e}")
379|
380| # =====
381| # Test 5: ratio を駆動する物理量
382| # =====
383| print("%n" + "=" * 50)
384| print("Test 5: ratio を駆動する物理量")
385| print("=" * 50)
386|
387| # E_grad/E_local = int(dε/dr)^2rdr / int ε^2rdr
388| # 次元的に: ratio ~ 1/L^2 (Lは特徴的長さスケール)
389| # 何が ratio を駆動するか
390|
391| # ratio vs hR のスケーリング
392| sl = linregress(log_hR[m], log_ratio[m])
393| print(f" ratio vs hR: slope = {sl.slope:.4f} +/- {sl.stderr:.4f}")
394| print(f" (理論予測: slope ~ -1 to -2)")
395|
396| # ratio vs gN_peak/gc_deep
397| gN_peak = []
398| for rec in records:
399|     rm = load_rotmod(rec.get("name", ""), ROTMOD_DIR) if "name" in rec else None
400|     if rm is None:
401|         # name を使えないので代替計算
402|         gN_peak.append(np.nan)
403|     else:
404|         gN_i = compute_gN(rm[0], rm[3], rm[4], rm[5])
405|         gN_peak.append(np.max(gN_i))
406| # name がないので records から再計算
407| # 簡略化: E_local/hR^2 ~ ε^2 & gN/gc ~ x_average
408| # ratio ~ 1/hR^2 x (勾配形状) -> x と hR の組み合わせ
409|
410| print(f"%n ratio の分散分解:")

```

```

411| # ratio = f(hR, x, 形状パラメータ)
412| X = np.column_stack([log_hR[m], log_x[m], np.ones(m.sum())])
413| coef, _, _, _ = lstsq(X, log_ratio[m], rcond=None)
414| y_pred = X @ coef
415| R2_hx = 1 - np.sum((log_ratio[m] - y_pred)**2) / np.sum((log_ratio[m] - np.mean(log_ratio[m]))**2)
416| print(f" R^2(ratio ~ hR + x): {R2_hx:.4f}")
417| print(f" ratio ~ hR^{coef[0]:.3f} x x^{coef[1]:.3f}")
418| print(f" -&gt; ratio の {R2_hx*100:.1f}% は hR と x で説明される")
419| print(f" -&gt; 残り {(1-R2_hx)*100:.1f}% が「独立な物理情報」")
420|
421| # =====
422| # Test 6: gc残差 vs ratio の精密評価
423| # =====
424| print("\n" + "=" * 50)
425| print("Test 6: gc残差(obs-deep) vs ratio")
426| print("=" * 50)
427|
428| resid = log_gc[m] - log_gcd[m]
429|
430| rho_res_r, p_res_r = pearsonr(log_ratio[m], resid)
431| print(f" raw: rho(resid, ratio) = {rho_res_r:+.4f}, p = {p_res_r:.3e}")
432|
433| rho_res_rx, p_res_rx = partial_corr(log_ratio[m], resid, [log_x[m]])
434| print(f" |x: rho(resid, ratio | x) = {rho_res_rx:+.4f}, p = {p_res_rx:.3e}")
435|
436| rho_res_rhx, p_res_rhx = partial_corr(log_ratio[m], resid,
437|                                     [log_x[m], log_hR[m]])
438| print(f" |x,hR: rho(resid, ratio | x,hR) = {rho_res_rhx:+.4f}, p = {p_res_rhx:.3e}")
439|
440| rho_res_full, p_res_full = partial_corr(log_ratio[m], resid,
441|                                       [log_x[m], log_hR[m], log_vf[m]])
442| print(f" |x,hR,vf: rho(resid, ratio | x,hR,vf) = {rho_res_full:+.4f}, p = {p_res_full:.3e}")
443|
444| # =====
445| # Test 7: 勾配形状パラメータの独立寄与
446| # =====
447| print("\n" + "=" * 50)
448| print("Test 7: 勾配形状パラメータの独立寄与")
449| print("=" * 50)
450|
451| shape_params = {
452|     "grad_peak_frac": grad_pf,
453|     "grad_asymm": grad_as,
454|     "grad_peak_r/hR": grad_pr,
455|     "grad_width/hR": grad_wd,
456| }
457|
458| for name, sp_arr in shape_params.items():
459|     v = m & np.isfinite(sp_arr) & (sp_arr > 0)
460|     if v.sum() < 20:
461|         continue
462|     log_sp = np.log10(sp_arr[v] + 1e-10)
463|
464|     # hR偏相関への寄与
465|     rho_s, p_s = partial_corr(log_hR[v], log_gc[v],
466|                             [log_vf[v], log_x[v], log_sp])
467|     delta = abs(rho_s) - abs(rho_base)
468|     # gc残差との相関
469|     resid_v = log_gc[v] - log_gcd[v]
470|     rho_sr, p_sr = partial_corr(log_sp, resid_v,
471|                               [log_x[v], log_hR[v]])
472|     print(f" {name:20s}: hR偏相関 rho={rho_s:+.4f} ( $\Delta$ ={delta:+.4f}), "
473|           f"resid|x,hR: rho={rho_sr:+.4f}")
474|
475| # =====
476| # Test 8: Permutation test (正確なp値)
477| # =====
478| print("\n" + "=" * 50)
479| print("Test 8: Permutation test (2000回)")
480| print("=" * 50)
481|
482| # ratio をシャッフルして偏相関の分布を作り、
483| # 実測の偏相関改善が偶然で起きる確率を計算
484| np.random.seed(777)
485| n_perm = 2000
486|
487| # 実測の改善量
488| real_delta = abs(rho_base) - abs(rho_r)
489|
490| perm_deltas = []
491| for _ in range(n_perm):
492|     ratio_shuf = log_ratio[m].copy()
493|     np.random.shuffle(ratio_shuf)
494|     try:
495|         rho_perm, _ = partial_corr(log_hR[m], log_gc[m],
496|                                   base + [ratio_shuf])
497|         perm_delta = abs(rho_base) - abs(rho_perm)

```

```

498|         perm_deltas.append(perm_delta)
499|     except:
500|         pass
501|
502| perm_deltas = np.array(perm_deltas)
503| p_perm = np.mean(perm_deltas >= real_delta)
504|
505| print(f" 実測  $\Delta|\rho| = \{real\_delta:.4f\}")
506| print(f"  Permutation: mean =  $\{np.mean(perm\_deltas):.4f\}$ , "
507|       f"std =  $\{np.std(perm\_deltas):.4f\}")
508| print(f"  Permutation p-value =  $\{p\_perm:.4f\}")
509| print(f"  95th percentile =  $\{np.percentile(perm\_deltas, 95):.4f\}")
510|
511| if p_perm < 0.05:
512|     print("  -> p < 0.05: ratio の改善は有意 [*]")
513| elif p_perm < 0.10:
514|     print("  -> 0.05 < p < 0.10: marginally significant")
515| else:
516|     print("  -> p >= 0.10: ratio の改善はノイズ")
517|
518| # =====
519| # SUMMARY
520| # =====
521| print("\n" + "=" * 70)
522| print("SUMMARY: E_grad/E_local の最終評価")
523| print("=" * 70)
524|
525| print(f"\n  偏相関:")
526| print(f"    base |vflat,x:      rho =  $\{rho\_base:+.4f\}")
527| print(f"    +ratio:                rho =  $\{rho\_r:+.4f\}")
528| print(f"     $\Delta|\rho| = \{real\_delta:+.4f\}$  ( $\{\delta\_pct:.1f\}$ % of total)")
529| print(f"    Pearson p =  $\{p\_r:.3e\}")
530| print(f"    Permutation p =  $\{p\_perm:.4f\}")
531|
532| print(f"\n  ブートストラップ:")
533| print(f"     $P(\Delta > 0) = \{np.mean(delta\_boot > 0):.3f\}")
534| print(f"    95% CI of  $\Delta = [\{np.percentile(delta\_boot, 2.5):.4f\}$ , "
535|      f" $\{np.percentile(delta\_boot, 97.5):.4f\}]"$ )
536|
537| print(f"\n  独立性:")
538| print(f"    ratio の  $\{R2\_hx*100:.1f\}$ % は hR+x で説明")
539| print(f"    ratio vs gc | hR,vflat,x: rho =  $\{rho\_rgcx:+.4f\}")
540|
541| print(f"\n  gc残差との相関:")
542| print(f"    ratio vs resid | x,hR,vf: rho =  $\{rho\_res\_full:+.4f\}")
543|
544| print(f"\n  ■ 最終判定:")
545| if p_perm < 0.05 and np.mean(delta_boot > 0) > 0.9:
546|     print("  [*] E_grad/E_local は統計的に有意な物理的效果")
547|     print(f"  -> hR偏相関の追加  $\{\delta\_pct:.0f\}$ % を説明")
548| elif p_perm < 0.10 or np.mean(delta_boot > 0) > 0.7:
549|     print("   $\Delta$  E_grad/E_local は示唆的だが決定的でない")
550|     print(f"  -> 方向は正しいが統計力不足 (N= $\{m.sum()\}$ )")
551| else:
552|     print("  x E_grad/E_local の効果はノイズ")
553|     print("  -> 勾配エネルギー仮説は支持されない")
554|
555| print(f"\n  ■ hR偏相関の最終分解:")
556| print(f"    x_outer (MOND遷移): 69%")
557| if p_perm < 0.10:
558|     print(f"    E_grad/E_local (勾配比):  $\{\delta\_pct:.0f\}$ % (p_perm= $\{p\_perm:.3f\}$ )")
559|     print(f"    残差:  $\{100-69-\delta\_pct:.0f\}$ %")
560| else:
561|     print(f"    E_grad/E_local: 効果不明 ( $\{\delta\_pct:.0f\}$ %, p= $\{p\_perm:.3f\}$ )")
562|     print(f"    残差: 31%")
563|
564|
565| if __name__ == "__main__":
566|     main()
567|$$$$$$$$$$$ 
```

## 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:  $m/s^2$  ( $a_0$ 単位の場合はスクリプト内で自動変換)
- r: kpc -> m (1 kpc =  $3.0857 \times 10^{19}$  m)
- V: km/s -> m/s ( $\times 10^3$ )
- $a_0 = 1.2 \times 10^{-10} m/s^2$
- Yd = 0.5 (デフォルト)、Yb = 0.7 (デフォルト)