

膜宇宙論 観測データ解析スクリプト集

V-1 / V-1改 / V-1b / N-1 Layer 2 / 2b / 3 / 3fix

著者: 坂口 忍 (坂口製麺所) | 2026年4月

本書は2026年4月9-10日セッションで使用した全7本の解析スクリプトの目的・構造・全文を収録する。再検証時にはこの文書からスクリプトを抽出して実行できる。

実行環境: Claude Code (VS Code拡張) on Windows。コマンド: `uv run --with scipy --with matplotlib python [script]`。必要データ: `Rotmod_LTG/*.dat, phase1/sparc_results.csv, TA3_gc_independent.csv`。

目次

#	ID	スクリプト	目的	行数	結論
1	V-1	sparc_fp_verification.py	$f_p = (g_N < a_0$ の質量分率) から $c=1-f_p$ を構成し、...	--	FAIL: $f_p \sim 1.0$ (全銀河)、相関なし。機構否定...
2	V-1改	sparc_fp_v1kai.py	幾何平均法則と BTFR の連立から導かれる $gc \sim \text{Sigma_bar}^{\sim}(1...$	--	FAIL: Sigma_bar 直接では $R^2=0.016$ (...)
3	V-1b	sparc_gc_vs_Mstar.py	gc vs vflat, hR, M_bar の7つの単変量回帰 + 多変量分解...	--	決定的発見: $gc \sim vflat^{1.10} \times hR^{(-...$
4	N-1 Layer 2	sparc_N1_layer2.py	hR の偏相関 ($\rho=-0.356$) が MOND 遷移域の補正効果で説明される...	--	R^2 改善 (+0.130) は循環アーティファクト (Laye...
5	N-1 Layer 2b	sparc_N1_layer2b.py	Layer 2 の R^2 改善が循環アーティファクトかを検証。gc-free ...	--	Layer 2 の改善は循環 (corr_gc vs gc: ...)
6	N-1 Layer 3	sparc_N1_layer3.py	式(5') + 式(5) の連立から $gc \sim \text{Sigma_bar}$ が予測されるの...	--	Test 1: BTFR残差 vs hR $\rho=-0.16...$
7	N-1 Layer 3 修正版	sparc_N1_layer3_fix.py	Layer 3 初版の eta 計算バグ (gc一乗/二乗取り違え) を修正し、et...	--	$\eta=0.736$ (妥当)。 $et a \sim Yd^{(-0.41)}...$

1. [V-1] f_p 閾値機構の数値検証

項目	内容
ファイル名	sparc_fp_verification.py
行数	542
検証ID	V-1

解析目的

$f_p = (g_N < a_0 \text{ の質量分率})$ から $c=1-f_p$ を構成し、観測された g_c との相関を検証する。条件14の最初の定量的テスト。

主要テスト

- [*] f_p vs g_c の Spearman 相関 (期待: $\rho < -0.6$)
- [*] sqrt スケーリング slope (期待: 0.5)
- [*] Binney-Tremaine 係数 (期待: $g_{\text{peak}}/GS0 \sim 1.8$)

結論

FAIL: $f_p \sim 1.0$ (全銀河)、相関なし。機構否定。

出力ファイル

fig_fp_vs_gc.png, fig_c_vs_sqrt_scaling.png, fig_fp_mechanism_summary.png, fp_verification_results.csv

スクリプト全文

```
1 | # -*- coding: utf-8 -*-
2 | """
3 | V-1: 条件14 定量的機構の数値検証
4 | =====
5 | SPARC 175銀河の  $g_N(r)$  プロファイルから  $f_p$  (塑性領域質量分率) を
6 | 数値計算し、観測された  $g_c$  との相関を検証する。
7 |
8 | 実行: uv run --with scipy --with matplotlib python sparc_fp_verification.py
9 |
10 | 必要ファイル:
11 | - Rotmod_LTG/*.dat (SPARC回転曲線)
12 | - phase1/sparc_results.csv (フィット結果: vflat, Yd, rs, gc等)
13 | - TA3_gc_independent.csv (gc_over_a0)
14 |
15 | 出力:
16 | - fp_verification_results.csv
17 | - fig_fp_vs_gc.png
18 | - fig_c_vs_sqrt_scaling.png
19 | - fig_fp_mechanism_summary.png
20 | - コンソールに統計サマリー
21 |
22 | 著者: 坂口 忍 (坂口製麺所)
23 | 日付: 2026年4月
24 | """
25 | import os, sys, glob, warnings
26 | import numpy as np
27 | from scipy import stats
28 |
29 | # --- matplotlib setup (IPAGothic for Japanese labels) ---
30 | import matplotlib
31 | matplotlib.use('Agg')
32 | import matplotlib.pyplot as plt
33 | from matplotlib import font_manager as _fm
34 | for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
35 |            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
36 |            r'C:\Windows\Fonts\msgothic.ttc']:
37 |     try: _fm.fontManager.addfont(_fp)
38 |     except: pass
39 | # Try IPAGothic first, fall back to MS Gothic on Windows
40 | for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
41 |     try:
42 |         plt.rcParams['font.family'] = fontname
43 |         break
44 |     except:
45 |         continue
46 | plt.rcParams['axes.unicode_minus'] = False
47 |
48 | warnings.filterwarnings('ignore')
49 |
50 | # =====
51 | # 0. Constants
```

```

52 | # =====
53 | a0 = 1.2e-10 # m/s^2 MOND acceleration
54 | G_SI = 6.674e-11 # m^3 kg^-1 s^-2
55 | kpc_m = 3.0857e19 # 1 kpc in meters
56 | Msun = 1.989e30 # kg
57 |
58 | # =====
59 | # 1. Paths -- adjust to your local layout
60 | # =====
61 | BASE = os.path.dirname(os.path.abspath(__file__))
62 | ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
63 | PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
64 | TA3 = os.path.join(BASE, 'TA3_gc_independent.csv')
65 |
66 | for p, label in [(ROTMOD, 'Rotmod_LTG'), (PHASE1, 'sparc_results.csv'), (TA3, 'TA3_gc_independent.csv')]:
67 |     if not os.path.exists(p):
68 |         print(f'[ERROR] {label} not found: {p}')
69 |         print(f' -> BASE = {BASE}')
70 |         sys.exit(1)
71 |
72 | # =====
73 | # 2. Load pipeline results
74 | # =====
75 | def load_csv_flexible(path):
76 |     """Load CSV handling both comma and whitespace delimiters."""
77 |     with open(path, 'r', encoding='utf-8-sig') as f:
78 |         header = f.readline().strip()
79 |         sep = ',' if ',' in header else None
80 |         data = {}
81 |         with open(path, 'r', encoding='utf-8-sig') as f:
82 |             cols = [c.strip() for c in f.readline().strip().split(sep)]
83 |             rows = []
84 |             for line in f:
85 |                 line = line.strip()
86 |                 if not line: continue
87 |                 parts = line.split(sep)
88 |                 rows.append([p.strip() for p in parts])
89 |             for i, col in enumerate(cols):
90 |                 vals = []
91 |                 for row in rows:
92 |                     if i < len(row):
93 |                         try: vals.append(float(row[i]))
94 |                         except: vals.append(row[i])
95 |                     else:
96 |                         vals.append(np.nan)
97 |                 data[col] = vals
98 |             return data
99 |
100 | print('[1] Loading pipeline results...')
101 | phase1 = load_csv_flexible(PHASE1)
102 | ta3 = load_csv_flexible(TA3)
103 |
104 | # Build lookup: galaxy_name -> {vflat, epsilon_d, gc_over_a0, ...}
105 | # Identify name column
106 | def find_name_col(data):
107 |     for candidate in ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY']:
108 |         if candidate in data:
109 |             return candidate
110 |     # First column with string values
111 |     for k, v in data.items():
112 |         if isinstance(v[0], str):
113 |             return k
114 |     return list(data.keys())[0]
115 |
116 | p1_name_col = find_name_col(phase1)
117 | ta3_name_col = find_name_col(ta3)
118 |
119 | galaxy_info = {}
120 | for i, name in enumerate(phase1[p1_name_col]):
121 |     name = str(name).strip()
122 |     info = {}
123 |     for k in phase1:
124 |         if k == p1_name_col: continue
125 |         try: info[k] = float(phase1[k][i])
126 |         except: info[k] = phase1[k][i]
127 |     galaxy_info[name] = info
128 |
129 | # Merge TA3 gc_over_a0
130 | for i, name in enumerate(ta3[ta3_name_col]):
131 |     name = str(name).strip()
132 |     if name in galaxy_info:
133 |         for k in ta3:
134 |             if k == ta3_name_col: continue
135 |             try: galaxy_info[name][k] = float(ta3[k][i])
136 |             except: galaxy_info[name][k] = ta3[k][i]
137 |
138 | print(f' phase1: {len(phase1[p1_name_col])} galaxies')
139 | print(f' TA3: {len(ta3[ta3_name_col])} galaxies')

```

```

140 |
141 | # =====
142 | # 3. Core computation: f_p from g_N(r) profile
143 | # =====
144 | def compute_fp(radii_kpc, vgas, vdisk, vbul, upsilon_d, upsilon_b=0.7):
145 |     """
146 |     Compute f_p = mass fraction where g_N(r) < a0.
147 |
148 |     For an exponential disk, Sigma(r) ~ exp(-r/hR). We approximate
149 |     the local mass weight as proportional to |V_bar(r)|^2 * r
150 |     (since g_N ~ V_bar^2/r, and dM ~ Sigma 2pi r dr ~ g_N r^2 dr / G).
151 |
152 |     Actually, the simplest mass weight for a thin disk is:
153 |     dM/dr = 2 pi r Sigma(r)
154 |     We approximate Sigma(r) from the disk velocity:
155 |     V_disk^2(r) = 4 pi G Sigma_0 h_R y^2 [I0 K0 - I1 K1]
156 |     Since we don't have Sigma(r) directly, we use:
157 |     g_N(r) = V_bar^2(r) / r [centripetal acceleration from baryons]
158 |     and weight by the mass enclosed in each radial annulus.
159 |
160 |     For simplicity, we use mass weight ~ |V_disk(r)|^2 as proxy for
161 |     Sigma(r) (valid for exponential disk).
162 |
163 |     Parameters
164 |     -----
165 |     radii_kpc : array, radii in kpc
166 |     vgas, vdisk, vbul : arrays, velocity components in km/s
167 |     upsilon_d : float, disk mass-to-light ratio
168 |     upsilon_b : float, bulge mass-to-light ratio (default 0.7)
169 |
170 |     Returns
171 |     -----
172 |     f_p : float, plastic mass fraction
173 |     g_N_array : array, g_N(r) in m/s^2
174 |     """
175 |     r_m = radii_kpc * kpc_m # convert to meters
176 |
177 |     # Baryonic velocity: V_bar^2 = Vgas^2 + Yd * Vdisk^2 + Yb * Vbul^2
178 |     # Note: SPARC files give Vdisk for Yd=1, so scale by sqrt(Yd)
179 |     vbar2 = (vgas**2
180 |             + upsilon_d * np.sign(vdisk) * vdisk**2
181 |             + upsilon_b * np.sign(vbul) * vbul**2)
182 |     # Handle sign convention: V can be negative (counter-rotating)
183 |     vbar2 = np.abs(vbar2)
184 |
185 |     # g_N(r) = V_bar^2 / r
186 |     g_N = np.zeros_like(r_m)
187 |     mask = r_m > 0
188 |     g_N[mask] = (vbar2[mask] * 1e6) / r_m[mask] # km/s -> m/s: *1e3, squared: *1e6
189 |
190 |     # Mass weight proxy: for exponential disk, dM/dr ~ Sigma(r) * r
191 |     # Sigma(r) is proportional to the disk contribution:
192 |     # We use |Vdisk(r)|^2 as proxy for Sigma(r) * r (since Vdisk^2 ~ 4piG Sigma hR * Bessel)
193 |     # A simpler and more robust proxy: weight = max(|Vdisk|^2, |Vgas|^2) * delta_r
194 |     weight = np.abs(vdisk**2) * upsilon_d + np.abs(vgas**2)
195 |     weight = np.maximum(weight, 1e-6)
196 |
197 |     # Compute f_p = sum(weight where g_N < a0) / sum(weight)
198 |     plastic_mask = g_N < a0
199 |     total_weight = np.sum(weight)
200 |     if total_weight <= 0:
201 |         return np.nan, g_N
202 |
203 |     f_p = np.sum(weight[plastic_mask]) / total_weight
204 |     return f_p, g_N
205 |
206 |
207 | def load_rotmod(filepath):
208 |     """Load SPARC rotation curve file."""
209 |     rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = [], [], [], [], [], [], [], []
210 |     with open(filepath, 'r') as f:
211 |         for line in f:
212 |             line = line.strip()
213 |             if not line or line.startswith("#"): continue
214 |             parts = line.split()
215 |             if len(parts) < 6: continue
216 |             try:
217 |                 rad.append(float(parts[0]))
218 |                 vobs.append(float(parts[1]))
219 |                 errv.append(float(parts[2]))
220 |                 vgas.append(float(parts[3]))
221 |                 vdisk.append(float(parts[4]))
222 |                 vbul.append(float(parts[5]))
223 |                 if len(parts) > 6: sbdisk.append(float(parts[6]))
224 |                 else: sbdisk.append(0.0)
225 |                 if len(parts) > 7: sbbul.append(float(parts[7]))
226 |                 else: sbbul.append(0.0)
227 |             except ValueError:

```

```

228 |         continue
229 |     return (np.array(rad), np.array(vobs), np.array(errv),
230 |           np.array(vgas), np.array(vdisk), np.array(vbul),
231 |           np.array(sbdisk), np.array(sbbul))
232 |
233 |
234 | # =====
235 | # 4. Process all galaxies
236 | # =====
237 | print('[2] Processing SPARC galaxies...')
238 |
239 | results = []
240 | rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
241 | print(f' Found {len(rotmod_files)} rotmod files')
242 |
243 | # Detect key column names in phase1
244 | def get_key(info, candidates, default=None):
245 |     for c in candidates:
246 |         if c in info:
247 |             return info[c]
248 |     return default
249 |
250 | for fpath in rotmod_files:
251 |     gname = os.path.splitext(os.path.basename(fpath))[0]
252 |     gname_clean = gname.replace('_rotmod', '').strip()
253 |
254 |     # Find matching pipeline entry
255 |     info = None
256 |     for key in [gname_clean, gname, gname_clean.upper(), gname_clean.lower()]:
257 |         if key in galaxy_info:
258 |             info = galaxy_info[key]
259 |             break
260 |     if info is None:
261 |         continue
262 |
263 |     # Get Yd (optimal) and gc
264 |     ud = get_key(info, ['upsilon_d', 'Upsilon_d', 'Ud', 'ud', 'Yd'], None)
265 |     gc_a0 = get_key(info, ['gc_over_a0', 'gc/a0', 'gc_ratio'], None)
266 |     vflat_val = get_key(info, ['vflat', 'Vflat', 'v_flat'], None)
267 |
268 |     if ud is None or gc_a0 is None or vflat_val is None:
269 |         continue
270 |
271 |     try:
272 |         ud = float(ud)
273 |         gc_a0 = float(gc_a0)
274 |         vflat_val = float(vflat_val)
275 |     except (ValueError, TypeError):
276 |         continue
277 |
278 |     if np.isnan(ud) or np.isnan(gc_a0) or gc_a0 <= 0:
279 |         continue
280 |
281 |     # Load rotation curve
282 |     try:
283 |         rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
284 |     except:
285 |         continue
286 |
287 |     if len(rad) < 3:
288 |         continue
289 |
290 |     # Compute f_p
291 |     f_p, g_N = compute_fp(rad, vgas, vdisk, vbul, ud)
292 |     if np.isnan(f_p):
293 |         continue
294 |
295 |     c_membrane = 1.0 - f_p
296 |
297 |     # Compute G*Sigma0 proxy = vflat^2 / hR
298 |     # Estimate hR from rpk / 2.15 (pipeline convention)
299 |     vds = np.sqrt(np.maximum(ud, 0.01)) * np.abs(vdisk)
300 |     rpk_idx = np.argmax(vds)
301 |     rpk = rad[rpk_idx]
302 |     if rpk < 0.01 or rpk >= rad.max() * 0.9:
303 |         # Edge filter (same as pipeline)
304 |         continue
305 |     hR = rpk / 2.15 # kpc
306 |     hR_m = hR * kpc_m
307 |     vflat_ms = vflat_val * 1e3
308 |     GS0 = vflat_ms**2 / hR_m # m/s^2
309 |
310 |     # Theoretical prediction: c ~ sqrt(G*Sigma0 / a0)
311 |     sqrt_ratio = np.sqrt(GS0 / a0)
312 |
313 |     results.append({
314 |         'galaxy': gname_clean,
315 |         'f_p': f_p,

```

```

316 |         'c_membrane': c_membrane,
317 |         'gc_over_a0': gc_a0,
318 |         'log_gc_a0': np.log10(gc_a0),
319 |         'vflat': vflat_val,
320 |         'Yd': ud,
321 |         'hR_kpc': hR,
322 |         'GS0': GS0,
323 |         'log_GS0_a0': np.log10(GS0 / a0),
324 |         'sqrt_GS0_a0': sqrt_ratio,
325 |         'n_points': len(rad),
326 |         'g_N_max': np.max(g_N),
327 |         'g_N_min': np.min(g_N[g_N > 0]) if np.any(g_N > 0) else 0,
328 |     })
329 |
330 | N = len(results)
331 | print(f' Successfully processed: {N} galaxies')
332 |
333 | if N < 10:
334 |     print('[ERROR] Too few galaxies. Check file paths and column names.')
```

```

335 |     print(f' Sample galaxy_info keys: {list(galaxy_info.keys())[:5]}')
336 |     if galaxy_info:
337 |         sample = list(galaxy_info.values())[0]
338 |         print(f' Sample info keys: {list(sample.keys())}')
339 |     sys.exit(1)
340 |
341 | # =====
342 | # 5. Statistical analysis
343 | # =====
344 | print(f' %n' + '='*60)
345 | print('V-1: f_p vs gc correlation analysis')
346 | print('='*60)
347 |
348 | fp_arr = np.array([r['f_p'] for r in results])
349 | c_arr = np.array([r['c_membrane'] for r in results])
350 | gc_arr = np.array([r['gc_over_a0'] for r in results])
351 | log_gc = np.array([r['log_gc_a0'] for r in results])
352 | sqrt_arr = np.array([r['sqrt_GS0_a0'] for r in results])
353 | log_GS0 = np.array([r['log_GS0_a0'] for r in results])
354 |
355 | # --- Test 1: f_p vs log(gc/a0) ---
356 | rho1, p1 = stats.spearmanr(fp_arr, log_gc)
357 | print(f' %n[Test 1] f_p vs log(gc/a0):')
358 | print(f' Spearman rho = {rho1:.3f}, p = {p1:.2e}')
359 | print(f' Expected: strong negative correlation (rho < -0.6)')
360 |
361 | # --- Test 2: c = 1-f_p vs gc/a0 ---
362 | rho2, p2 = stats.spearmanr(c_arr, gc_arr)
363 | print(f' %n[Test 2] c = 1-f_p vs gc/a0:')
364 | print(f' Spearman rho = {rho2:.3f}, p = {p2:.2e}')
365 |
366 | # --- Test 3: c vs sqrt(G*Sigma0/a0) -- the key prediction ---
367 | rho3, p3 = stats.spearmanr(c_arr, sqrt_arr)
368 | print(f' %n[Test 3] c = 1-f_p vs sqrt(G*Sigma0/a0):')
369 | print(f' Spearman rho = {rho3:.3f}, p = {p3:.2e}')
370 | print(f' This is the KEY test: c should scale as sqrt(G*Sigma0/a0)')
371 |
372 | # --- Test 4: log(c) vs 0.5*log(G*Sigma0/a0) -- slope test ---
373 | valid = (c_arr > 0) & (sqrt_arr > 0)
374 | log_c = np.log10(c_arr[valid])
375 | log_sqrt = np.log10(sqrt_arr[valid])
376 | log_GS0_v = log_GS0[valid]
377 |
378 | slope, intercept, r_val, p_slope, se_slope = stats.linregress(log_GS0_v, log_c)
379 | print(f' %n[Test 4] log(c) vs log(G*Sigma0/a0) regression:')
380 | print(f' slope = {slope:.3f} +/- {se_slope:.3f}')
381 | print(f' Expected slope = 0.5 (sqrt scaling)')
382 | print(f' p(slope=0.5) = {2*(1-stats.t.cdf(abs(slope-0.5)/se_slope, df=sum(valid)-2)):.3f}')
383 | print(f' R^2 = {r_val**2:.3f}')
384 | print(f' intercept = {intercept:.3f} (= log(eta) effectively)')
385 |
386 | # --- Derived eta ---
387 | eta_eff = 10**intercept
388 | print(f' %n[Derived] eta_eff = 10^{intercept:.3f} = {eta_eff:.3f}')
389 |
390 | # --- Summary statistics ---
391 | print(f' %n--- Summary ---')
392 | print(f' N = {N}')
393 | print(f' f_p: median = {np.median(fp_arr):.3f}, '
394 |       f' IQR = [{np.percentile(fp_arr, 25):.3f}, {np.percentile(fp_arr, 75):.3f}]')
395 | print(f' c = 1-f_p: median = {np.median(c_arr):.3f}, '
396 |       f' IQR = [{np.percentile(c_arr, 25):.3f}, {np.percentile(c_arr, 75):.3f}]')
397 | print(f' gc/a0: median = {np.median(gc_arr):.3f}')
398 |
399 | # Binney & Tremaine check: g_peak vs 1.8*G*Sigma0
400 | g_N_max_arr = np.array([r['g_N_max'] for r in results])
401 | GS0_arr = np.array([r['GS0'] for r in results])
402 | ratio_peak = g_N_max_arr / GS0_arr
403 | print(f' %n[BT check] g_N_max / (G*Sigma0): median = {np.median(ratio_peak):.2f} (expect ~1.8)')
```

```

404 |
405 | # =====
406 | # 6. Figures
407 | # =====
408 | print('\n[3] Generating figures...')
409 |
410 | # --- Fig 1: f_p vs log(gc/a0) ---
411 | fig, ax = plt.subplots(1, 1, figsize=(7, 5))
412 | ax.scatter(fp_arr, log_gc, s=12, alpha=0.6, c=log_GS0, cmap='viridis', edgecolors='none')
413 | ax.set_xlabel('f_p (g_N < a0 mass fraction)', fontsize=11)
414 | ax.set_ylabel('log(gc / a0)', fontsize=11)
415 | ax.set_title('V-1: f_p vs gc (N={N}, rho={rho1:.3f}, p={p1:.1e})', fontsize=12)
416 | cb = plt.colorbar(ax.collections[0], ax=ax, label='log(G*Sigma0/a0)')
417 | # Add trend line
418 | z = np.polyfit(fp_arr, log_gc, 1)
419 | xline = np.linspace(0, 1, 100)
420 | ax.plot(xline, np.polyval(z, xline), 'r--', lw=1.5, label=f'linear fit: slope={z[0]:.2f}')
421 | ax.legend(fontsize=9)
422 | ax.grid(True, alpha=0.3)
423 | fig.tight_layout()
424 | fig.savefig(os.path.join(BASE, 'fig_fp_vs_gc.png'), dpi=150)
425 | print(' -> fig_fp_vs_gc.png')
426 |
427 | # --- Fig 2: c = 1-f_p vs sqrt(G*Sigma0/a0) with slope test ---
428 | fig, axes = plt.subplots(1, 2, figsize=(13, 5))
429 |
430 | ax = axes[0]
431 | ax.scatter(sqrt_arr, c_arr, s=12, alpha=0.6, c='steelblue', edgecolors='none')
432 | ax.set_xlabel('sqrt(G*Sigma0 / a0)', fontsize=11)
433 | ax.set_ylabel('c = 1 - f_p', fontsize=11)
434 | ax.set_title('c vs sqrt scaling (rho={rho3:.3f})', fontsize=12)
435 | # 1:1 reference (with eta normalization)
436 | xref = np.linspace(0, np.max(sqrt_arr)*1.1, 100)
437 | ax.plot(xref, eta_eff * xref, 'r-', lw=1.5, label=f'c = {eta_eff:.2f} * sqrt(GS0/a0)')
438 | ax.legend(fontsize=9)
439 | ax.grid(True, alpha=0.3)
440 |
441 | ax = axes[1]
442 | ax.scatter(log_GS0_v, log_c, s=12, alpha=0.6, c='darkorange', edgecolors='none')
443 | ax.set_xlabel('log(G*Sigma0 / a0)', fontsize=11)
444 | ax.set_ylabel('log(c)', fontsize=11)
445 | ax.set_title('Slope test: {slope:.3f} +/- {se_slope:.3f} (expect 0.5)', fontsize=12)
446 | xfit = np.linspace(log_GS0_v.min(), log_GS0_v.max(), 100)
447 | ax.plot(xfit, slope*xfit + intercept, 'r-', lw=1.5, label=f'fit: slope={slope:.3f}')
448 | ax.plot(xfit, 0.5*xfit + intercept, 'b--', lw=1.0, label='theory: slope=0.5')
449 | ax.legend(fontsize=9)
450 | ax.grid(True, alpha=0.3)
451 |
452 | fig.tight_layout()
453 | fig.savefig(os.path.join(BASE, 'fig_c_vs_sqrt_scaling.png'), dpi=150)
454 | print(' -> fig_c_vs_sqrt_scaling.png')
455 |
456 | # --- Fig 3: 4-panel summary ---
457 | fig, axes = plt.subplots(2, 2, figsize=(12, 10))
458 |
459 | # Panel A: f_p distribution
460 | ax = axes[0, 0]
461 | ax.hist(fp_arr, bins=30, color='steelblue', alpha=0.7, edgecolor='white')
462 | ax.axvline(np.median(fp_arr), color='red', ls='--', label=f'median={np.median(fp_arr):.2f}')
463 | ax.set_xlabel('f_p', fontsize=11)
464 | ax.set_ylabel('Count', fontsize=11)
465 | ax.set_title('A: f_p distribution', fontsize=12)
466 | ax.legend()
467 |
468 | # Panel B: c vs gc/a0
469 | ax = axes[0, 1]
470 | ax.scatter(c_arr, gc_arr, s=12, alpha=0.6, c='steelblue', edgecolors='none')
471 | ax.set_xlabel('c = 1 - f_p', fontsize=11)
472 | ax.set_ylabel('gc / a0', fontsize=11)
473 | ax.set_title('B: c vs gc/a0 (rho={rho2:.3f})', fontsize=12)
474 | # Plot c*a0 line (gc = c*a0 means gc/a0 = c)
475 | xline = np.linspace(0, 1, 100)
476 | ax.plot(xline, xline, 'r--', lw=1, label='gc/a0 = c (exact match)')
477 | ax.legend(fontsize=9)
478 | ax.grid(True, alpha=0.3)
479 |
480 | # Panel C: g_peak / G*Sigma0 distribution (BT check)
481 | ax = axes[1, 0]
482 | ax.hist(ratio_peak[np.isfinite(ratio_peak)], bins=30, color='darkorange', alpha=0.7, edgecolor='white')
483 | ax.axvline(1.8, color='red', ls='--', lw=2, label='BT prediction: 1.8')
484 | ax.axvline(np.median(ratio_peak), color='blue', ls='--', label=f'median={np.median(ratio_peak):.2f}')
485 | ax.set_xlabel('g_N_max / (G*Sigma0)', fontsize=11)
486 | ax.set_ylabel('Count', fontsize=11)
487 | ax.set_title('C: Binney-Tremaine check', fontsize=12)
488 | ax.legend(fontsize=9)
489 |
490 | # Panel D: residual from sqrt law
491 | ax = axes[1, 1]

```

```

492 | if np.sum(valid) > 5:
493 |     residual = log_c - (0.5 * log_GS0_v + intercept)
494 |     ax.hist(residual, bins=25, color='green', alpha=0.7, edgecolor='white')
495 |     ax.axvline(0, color='red', ls='--')
496 |     ax.set_xlabel('log(c) - [0.5*log(GS0/a0) + const]', fontsize=11)
497 |     ax.set_ylabel('Count', fontsize=11)
498 |     ax.set_title('f'D: Residual from sqrt law (sigma={np.std(residual):.3f} dex)', fontsize=12)
499 |
500 | fig.suptitle('f'V-1: Condition-14 Mechanism Verification (N={N})', fontsize=14, y=1.01)
501 | fig.tight_layout()
502 | fig.savefig(os.path.join(BASE, 'fig_fp_mechanism_summary.png'), dpi=150)
503 | print(' -> fig_fp_mechanism_summary.png')
504 |
505 | # =====
506 | # 7. Save results
507 | # =====
508 | outcsv = os.path.join(BASE, 'fp_verification_results.csv')
509 | with open(outcsv, 'w', encoding='utf-8') as f:
510 |     cols = ['galaxy', 'f_p', 'c_membrane', 'gc_over_a0', 'log_gc_a0',
511 |            'vflat', 'Yd', 'hR_kpc', 'GS0', 'log_GS0_a0', 'sqrt_GS0_a0',
512 |            'g_N_max', 'g_N_min', 'n_points']
513 |     f.write(','.join(cols) + '\n')
514 |     for r in results:
515 |         f.write(','.join(str(r[c]) for c in cols) + '\n')
516 | print(f'\n[4] Results saved: {outcsv}')
517 |
518 | # =====
519 | # 8. Final verdict
520 | # =====
521 | print('\n' + '='*60)
522 | print('VERDICT')
523 | print('='*60)
524 |
525 | verdict_fp_gc = 'PASS' if (rho1 < -0.5 and p1 < 0.001) else 'FAIL'
526 | verdict_slope = 'PASS' if abs(slope - 0.5) < 2*se_slope else 'MARGINAL' if abs(slope-0.5) < 3*se_slope else 'FAIL'
527 | verdict_bt = 'PASS' if 1.0 < np.median(ratio_peak) < 3.0 else 'CHECK'
528 |
529 | print(f' [V-1a] f_p vs gc correlation:      {verdict_fp_gc} (rho={rho1:.3f}, p={p1:.1e})')
530 | print(f' [V-1b] sqrt scaling slope:        {verdict_slope} (slope={slope:.3f}+/-{se_slope:.3f}, expect 0.5)')
531 | print(f' [V-1c] BT coefficient check:      {verdict_bt} (median ratio={np.median(ratio_peak):.2f}, expect ~1.8)')
532 | print(f' [V-2] eta effective:              {eta_eff:.3f}')
533 | print()
534 |
535 | if verdict_fp_gc == 'PASS' and verdict_slope in ('PASS', 'MARGINAL'):
536 |     print(' >>> Condition-14 mechanism SUPPORTED: f_p determines gc via sqrt scaling')
537 |     print(f' >>> c = 1-f_p ~ {eta_eff:.2f} * sqrt(G*Sigma0/a0)')
538 | else:
539 |     print(' >>> Results require further investigation')
540 |     print(f' >>> f_p-gc: {verdict_fp_gc}, slope: {verdict_slope}')
541 |
542 | print('\n[DONE]')

```

2. [V-1改] gc vs 真のバリオン面密度 Σ_{bar} の検証

項目	内容
ファイル名	sparc_fp_v1kai.py
行数	549
検証ID	V-1改

解析目的

幾何平均法則と BTFR の連立から導かれる $gc \propto \Sigma_{\text{bar}}^{1/3}$ を、BT式で直接計算した Σ_{bar} を用いて検証する。5つの方法 (proxy, Vdisk peak BT, disk+gas BT, BTFR逆算, cross-check) を同時テスト。

主要テスト

- [*] proxy v_{flat}^2/hR でのスロープ=0.5 (ベースライン確認)
- [*] Vdisk peak BT (gc非依存) でのスロープ=1/3
- [*] disk+gas BT でのスロープ=1/3
- [*] 1/3 vs 0.5 のどちらが真の Σ_{bar} に適合するか

結論

FAIL: Σ_{bar} 直接では $R^2=0.016$ (無相関)。proxy のみ $\alpha=0.548$ 。

出力ファイル

fig_v1kai_slope_test.png, fig_v1kai_4panel.png, v1kai_results.csv

スクリプト全文

```
1 | # -*- coding: utf-8 -*-
2 | """
3 | V-1改: 条件14 機構再検討 -- gc  $\propto \Sigma_{\text{bar}}^{1/3}$  の検証
4 | =====
5 | 幾何平均法則  $gc = \eta \sqrt{a_0 \cdot \text{GS0\_proxy}}$  と  $\text{BTFR } v_{\text{flat}}^4 = gc \cdot G \cdot M_{\text{bar}}$ 
6 | を組み合わせると、真のバリオン面密度  $\Sigma_{\text{bar}}$  との関係は:
7 |
8 |  $gc = \eta \cdot a_0^{2/3} \cdot (G \cdot \Sigma_{\text{bar}})^{1/3}$ 
9 |
10 | すなわち  $\log(gc/a_0)$  vs  $\log(G \cdot \Sigma_{\text{bar}}/a_0)$  のスロープ = 1/3 が予測される。
11 |
12 | 本スクリプトは SPARC 175銀河でこれを検証する。
13 |
14 | 実行: uv run --with scipy --with matplotlib python sparc_fp_v1kai.py
15 |
16 | 必要ファイル:
17 | - Rotmod_LTG/*.dat
18 | - phase1/sparc_results.csv
19 | - TA3_gc_independent.csv
20 |
21 | 出力:
22 | - fig_v1kai_slope_test.png (主要結果: 1/3スロープ検定)
23 | - fig_v1kai_4panel.png (4パネルサマリー)
24 | - v1kai_results.csv
25 | - コンソールに統計サマリー
26 |
27 | 著者: 坂口 忍 (坂口製麺所)
28 | 日付: 2026年4月
29 | """
30 | import os, sys, glob, warnings
31 | import numpy as np
32 | from scipy import stats
33 |
34 | import matplotlib
35 | matplotlib.use('Agg')
36 | import matplotlib.pyplot as plt
37 | from matplotlib import font_manager as _fm
38 | for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
39 |            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
40 |            r'C:\Windows\Fonts\msgothic.ttc']:
41 |     try: _fm.fontManager.addfont(_fp)
42 |     except: pass
43 | for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
44 |     try:
45 |         plt.rcParams['font.family'] = fontname
46 |         break
47 |     except:
48 |         continue
```

```

49 | plt.rcParams['axes.unicode_minus'] = False
50 | warnings.filterwarnings('ignore')
51 |
52 | # =====
53 | # 0. Constants
54 | # =====
55 | a0 = 1.2e-10 # m/s^2
56 | G_SI = 6.674e-11 # m^3 kg^-1 s^-2
57 | kpc_m = 3.0857e19 # m
58 | Msun = 1.989e30 # kg
59 | pc_m = 3.0857e16 # m
60 |
61 | # Milgrom critical surface density
62 | Sigma_crit = a0 / G_SI # kg/m^2
63 | Sigma_crit_Msun_pc2 = Sigma_crit / Msun * pc_m**2
64 | print(f'Milgrom critical surface density: {Sigma_crit_Msun_pc2:.0f} M_sun/pc^2')
65 |
66 | # =====
67 | # 1. Paths
68 | # =====
69 | BASE = os.path.dirname(os.path.abspath(__file__))
70 | ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
71 | PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
72 | TA3 = os.path.join(BASE, 'TA3_gc_independent.csv')
73 |
74 | for p, label in [(ROTMOD, 'Rotmod LTG'), (PHASE1, 'sparc_results.csv'),
75 |                 (TA3, 'TA3_gc_independent.csv')]:
76 |     if not os.path.exists(p):
77 |         print(f'[ERROR] {label} not found: {p}')
78 |         sys.exit(1)
79 |
80 | # =====
81 | # 2. Load CSV helper
82 | # =====
83 | def load_csv(path):
84 |     with open(path, 'r', encoding='utf-8-sig') as f:
85 |         header = f.readline().strip()
86 |         sep = ',' if ',' in header else None
87 |         data = {}
88 |         with open(path, 'r', encoding='utf-8-sig') as f:
89 |             cols = [c.strip() for c in f.readline().strip().split(sep)]
90 |             rows = []
91 |             for line in f:
92 |                 line = line.strip()
93 |                 if not line:
94 |                     continue
95 |                 rows.append([p.strip() for p in line.split(sep)])
96 |             for i, col in enumerate(cols):
97 |                 vals = []
98 |                 for row in rows:
99 |                     if i < len(row):
100 |                         try: vals.append(float(row[i]))
101 |                         except: vals.append(row[i])
102 |                     else:
103 |                         vals.append(np.nan)
104 |                 data[col] = vals
105 |             return data
106 |
107 | def find_name_col(data):
108 |     for c in ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY']:
109 |         if c in data:
110 |             return c
111 |     for k, v in data.items():
112 |         if isinstance(v[0], str):
113 |             return k
114 |     return list(data.keys())[0]
115 |
116 | def get_key(info, candidates, default=None):
117 |     for c in candidates:
118 |         if c in info:
119 |             try: return float(info[c])
120 |             except: return info[c]
121 |     return default
122 |
123 | # =====
124 | # 3. Load pipeline data
125 | # =====
126 | print('[1] Loading pipeline data...')
127 | phase1 = load_csv(PHASE1)
128 | ta3 = load_csv(TA3)
129 |
130 | p1_nc = find_name_col(phase1)
131 | ta3_nc = find_name_col(ta3)
132 |
133 | galaxy_info = {}
134 | for i, name in enumerate(phase1[p1_nc]):
135 |     name = str(name).strip()
136 |     info = {}

```

```

137 |     for k in phase1:
138 |         if k == p1_nc: continue
139 |         try:     info[k] = float(phase1[k][i])
140 |         except: info[k] = phase1[k][i]
141 |     galaxy_info[name] = info
142 |
143 | for i, name in enumerate(ta3[ta3_nc]):
144 |     name = str(name).strip()
145 |     if name in galaxy_info:
146 |         for k in ta3:
147 |             if k == ta3_nc: continue
148 |             try:     galaxy_info[name][k] = float(ta3[k][i])
149 |             except: galaxy_info[name][k] = ta3[k][i]
150 |
151 | print(f' phase1: {len(phase1[p1_nc])} galaxies')
152 | print(f' TA3:   {len(ta3[ta3_nc])} galaxies')
153 |
154 | # =====
155 | # 4. Load rotmod and compute Sigma_bar
156 | # =====
157 | def load_rotmod(filepath):
158 |     cols = [[] for _ in range(8)]
159 |     with open(filepath, 'r') as f:
160 |         for line in f:
161 |             line = line.strip()
162 |             if not line or line.startswith('#'):
163 |                 continue
164 |             parts = line.split()
165 |             if len(parts) < 6:
166 |                 continue
167 |             try:
168 |                 for j in range(min(len(parts), 8)):
169 |                     cols[j].append(float(parts[j]))
170 |                 for j in range(len(parts), 8):
171 |                     cols[j].append(0.0)
172 |             except ValueError:
173 |                 continue
174 |     return tuple(np.array(c) for c in cols)
175 |
176 |
177 | print('[2] Processing galaxies...')
178 | results = []
179 | rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
180 | print(f' Found {len(rotmod_files)} rotmod files')
181 |
182 | for fpath in rotmod_files:
183 |     gname = os.path.splitext(os.path.basename(fpath))[0]
184 |     gname_clean = gname.replace('_rotmod', '').strip()
185 |
186 |     info = None
187 |     for key in [gname_clean, gname, gname_clean.upper(), gname_clean.lower()]:
188 |         if key in galaxy_info:
189 |             info = galaxy_info[key]
190 |             break
191 |     if info is None:
192 |         continue
193 |
194 |     # Required quantities
195 |     ud = get_key(info, ['upsilon_d', 'Upsilon_d', 'Ud', 'ud', 'Yd'])
196 |     gc_a0 = get_key(info, ['gc_over_a0', 'gc/a0', 'gc_ratio'])
197 |     vflat = get_key(info, ['vflat', 'Vflat', 'v_flat'])
198 |
199 |     if ud is None or gc_a0 is None or vflat is None:
200 |         continue
201 |     if np.isnan(ud) or np.isnan(gc_a0) or gc_a0 <= 0 or vflat <= 0:
202 |         continue
203 |
204 |     # Load rotation curve
205 |     try:
206 |         rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
207 |     except:
208 |         continue
209 |     if len(rad) < 3:
210 |         continue
211 |
212 |     # --- Compute h_R from r_pk (pipeline convention) ---
213 |     vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
214 |     rpk_idx = np.argmax(vds)
215 |     rpk = rad[rpk_idx]
216 |     if rpk < 0.01 or rpk >= rad.max() * 0.9:
217 |         continue
218 |     hR_kpc = rpk / 2.15
219 |
220 |     # --- Method A: Sigma_bar from SBdisk (if available) ---
221 |     # SBdisk in SPARC is face-on surface brightness in L_sun/pc^2 (3.6um)
222 |     # Sigma_bar = Yd * SBdisk_central
223 |     # SBdisk in rotmod is mag/arcsec^2 at each radius.
224 |     # Central SB = extrapolation or use max(SBdisk)

```

```

225 | # Actually: SBdisk column in rotmod is in L_sun/pc^2
226 | # Use the disk luminosity profile to get total L_disk
227 |
228 | # --- Method B: Sigma_bar from Vdisk profile (more robust) ---
229 | # For exponential disk: V_disk_peak^2 ~ pi*G*Sigma0_bar * hR * f(2.2)
230 | # f(2.2) = 2.2^2 * [I0*K0 - I1*K1] evaluated at y=1.1
231 | # This gives Sigma0_bar from V_disk_peak directly.
232 | # But simpler: M_disk = integral of disk mass
233 | # M_disk ~ Yd * L_disk
234 | # Use: V_disk^2(r) relates to Sigma(r) directly
235 |
236 | # --- Method C (cleanest): Use vflat + hR to get GS0_proxy,
237 | # then compute Sigma_bar by removing gc contamination ---
238 | # GS0_proxy = vflat^2 / hR [this is what the geometric mean law uses]
239 | # From the derivation: GS0_proxy = sqrt(2pi * gc * G * Sigma_bar)
240 | # So: G*Sigma_bar = GS0_proxy^2 / (2pi * gc)
241 | # = (vflat^2/hR)^2 / (2pi * gc_a0 * a0)
242 |
243 | hR_m = hR_kpc * kpc_m
244 | vflat_ms = vflat * 1e3 # km/s -> m/s
245 | GS0_proxy = vflat_ms**2 / hR_m # m/s^2
246 |
247 | gc = gc_a0 * a0 # m/s^2
248 |
249 | # Method C: derived from BTFR decontamination
250 | G_Sigma_bar_C = GS0_proxy**2 / (2 * np.pi * gc) # m/s^2
251 |
252 | # --- Method D (fully independent): Sigma_bar from V_disk peak ---
253 | # V_disk_peak = max(sqrt(Yd) * |Vdisk|)
254 | # For exponential disk: V_disk_peak^2 = 0.56 * pi * G * Sigma0_bar * hR
255 | # (Binney & Tremaine eq 2.165, evaluated at r=2.2hR)
256 | # => G * Sigma0_bar = V_disk_peak^2 / (0.56 * pi * hR)
257 | vdisk_peak = np.max(np.sqrt(max(ud, 0.01)) * np.abs(vdisk))
258 | vdisk_peak_ms = vdisk_peak * 1e3
259 | G_Sigma0_bar_D = vdisk_peak_ms**2 / (0.56 * np.pi * hR_m) # m/s^2
260 |
261 | # --- Method E: total Sigma_bar = disk + gas ---
262 | # Gas contribution: V_gas_peak
263 | vgas_peak_ms = np.max(np.abs(vgas)) * 1e3
264 | # Approximate total baryonic: add gas disk
265 | # G_Sigma_bar_total ~ G_Sigma0_bar_D + G_Sigma_gas
266 | # For gas: same exponential approximation
267 | G_Sigma_gas = vgas_peak_ms**2 / (0.56 * np.pi * hR_m) if vgas_peak_ms > 0 else 0
268 | G_Sigma_bar_E = G_Sigma0_bar_D + G_Sigma_gas
269 |
270 | # Store all methods
271 | results.append({
272 |     'galaxy': gname_clean,
273 |     'gc_over_a0': gc_a0,
274 |     'log_gc_a0': np.log10(gc_a0),
275 |     'vflat': vflat,
276 |     'Yd': ud,
277 |     'hR_kpc': hR_kpc,
278 |     'GS0_proxy': GS0_proxy,
279 |     'log_GS0_proxy': np.log10(GS0_proxy / a0),
280 |     # Method C: BTFR decontaminated
281 |     'G_Sbar_C': G_Sigma_bar_C,
282 |     'log_GSbar_C': np.log10(G_Sigma_bar_C / a0) if G_Sigma_bar_C > 0 else np.nan,
283 |     # Method D: V_disk peak (BT formula)
284 |     'G_Sbar_D': G_Sigma0_bar_D,
285 |     'log_GSbar_D': np.log10(G_Sigma0_bar_D / a0) if G_Sigma0_bar_D > 0 else np.nan,
286 |     # Method E: disk + gas
287 |     'G_Sbar_E': G_Sigma_bar_E,
288 |     'log_GSbar_E': np.log10(G_Sigma_bar_E / a0) if G_Sigma_bar_E > 0 else np.nan,
289 |     # For diagnostics
290 |     'vdisk_peak': vdisk_peak,
291 |     'vgas_peak': np.max(np.abs(vgas)),
292 | })
293 |
294 | N = len(results)
295 | print(f' Processed: {N} galaxies')
296 |
297 | if N < 10:
298 |     print('[ERROR] Too few galaxies.')
299 |     sys.exit(1)
300 |
301 | # =====
302 | # 5. Statistical analysis
303 | # =====
304 | log_gc = np.array([r['log_gc_a0'] for r in results])
305 |
306 | # --- Method D: V_disk peak (fully independent of gc) ---
307 | log_GSbar_D = np.array([r['log_GSbar_D'] for r in results])
308 | valid_D = np.isfinite(log_GSbar_D) & np.isfinite(log_gc)
309 |
310 | # --- Method E: disk + gas ---
311 | log_GSbar_E = np.array([r['log_GSbar_E'] for r in results])
312 | valid_E = np.isfinite(log_GSbar_E) & np.isfinite(log_gc)

```

```

313 |
314 | # --- Method C: BTFR decontaminated (circular but diagnostic) ---
315 | log_GSbar_C = np.array([r['log_GSbar_C'] for r in results])
316 | valid_C = np.isfinite(log_GSbar_C) & np.isfinite(log_gc)
317 |
318 | # --- Original proxy (for comparison) ---
319 | log_GS0 = np.array([r['log_GS0_proxy'] for r in results])
320 | valid_0 = np.isfinite(log_GS0) & np.isfinite(log_gc)
321 |
322 | print('%n' + '='*70)
323 | print('V-1 revised: gc vs Sigma_bar scaling test')
324 | print('='*70)
325 |
326 | def do_regression(x, y, mask, label, expected_slope):
327 |     """Run regression and report."""
328 |     xv, yv = x[mask], y[mask]
329 |     n = len(xv)
330 |     if n < 5:
331 |         print(f'%n[{label}] N={n} -- too few points')
332 |         return None
333 |
334 |     slope, intercept, r_val, p_val, se = stats.linregress(xv, yv)
335 |     rho_s, p_s = stats.spearmanr(xv, yv)
336 |
337 |     # Test slope = expected
338 |     t_stat = abs(slope - expected_slope) / se
339 |     p_slope = 2 * (1 - stats.t.cdf(t_stat, df=n-2))
340 |
341 |     print(f'%n[{label}] N = {n}')
342 |     print(f' slope = {slope:.4f} +/- {se:.4f}')
343 |     print(f' expected = {expected_slope}')
344 |     print(f' p(slope={expected_slope}) = {p_slope:.4f}')
345 |     print(f' R^2 = {r_val**2:.4f}')
346 |     print(f' Spearman = {rho_s:.4f}, p = {p_s:.2e}')
347 |     print(f' intercept = {intercept:.4f} (eta% = 10^{intercept:.3f} = {10**intercept:.4f})')
348 |
349 |     verdict = 'PASS' if p_slope > 0.05 else 'MARGINAL' if p_slope > 0.01 else 'FAIL'
350 |     print(f' VERDICT: {verdict}')
351 |
352 |     return {
353 |         'slope': slope, 'se': se, 'intercept': intercept,
354 |         'r2': r_val**2, 'rho': rho_s, 'p_spearman': p_s,
355 |         'p_slope': p_slope, 'n': n, 'verdict': verdict,
356 |         'eta_prime': 10**intercept
357 |     }
358 |
359 | print('%n--- Test 0: Original proxy (alpha=0.5 expected, baseline) ---')
360 | r0 = do_regression(log_GS0, log_gc, valid_0,
361 |                   'GS0_proxy = vflatt^2/hR^2, 0.5)
362 |
363 | print('%n--- Test D: V_disk peak Sigma_bar (1/3 expected, INDEPENDENT) ---')
364 | rD = do_regression(log_GSbar_D, log_gc, valid_D,
365 |                   'G*Sigma_bar from Vdisk_peak (BT)', 1/3)
366 |
367 | print('%n--- Test E: disk+gas Sigma_bar (1/3 expected, INDEPENDENT) ---')
368 | rE = do_regression(log_GSbar_E, log_gc, valid_E,
369 |                   'G*Sigma_bar disk+gas (BT)', 1/3)
370 |
371 | print('%n--- Test C: BTFR-decontaminated (1/3 expected, CIRCULAR CHECK) ---')
372 | rC = do_regression(log_GSbar_C, log_gc, valid_C,
373 |                   'G*Sigma_bar BTFR-decontam (circular)', 1/3)
374 |
375 | # --- Additional: test slope=0.5 for Method D/E ---
376 | print('%n--- Cross-check: does 0.5 work for true Sigma_bar? ---')
377 | rD5 = do_regression(log_GSbar_D, log_gc, valid_D,
378 |                   'G*Sigma_bar (BT) vs slope=0.5', 0.5)
379 | rE5 = do_regression(log_GSbar_E, log_gc, valid_E,
380 |                   'G*Sigma_bar disk+gas vs slope=0.5', 0.5)
381 |
382 | # --- Diagnostic: distribution of Sigma_bar/Sigma_crit ---
383 | G_Sbar_D_arr = np.array([r['G_Sbar_D'] for r in results])
384 | Sbar_ratio = G_Sbar_D_arr / (G_SI * Sigma_crit) # dimensionless
385 | print(f'%n--- Diagnostic ---')
386 | print(f' Sigma_bar / Sigma_crit (Milgrom):')
387 | print(f' median = {np.nanmedian(Sbar_ratio):.4f}')
388 | print(f' IQR = [{np.nanpercentile(Sbar_ratio, 25):.4f}, '
389 |        f' {np.nanpercentile(Sbar_ratio, 75):.4f}']')
390 | print(f' range = [{np.nanmin(Sbar_ratio):.4f}, {np.nanmax(Sbar_ratio):.4f}']')
391 | print(f' (ALL << 1 confirms SPARC galaxies are low surface brightness dominated)')
392 |
393 | # =====
394 | # 6. Figures
395 | # =====
396 | print('%n[3] Generating figures...')
397 |
398 | # --- Fig 1: Main slope test (D and E) ---
399 | fig, axes = plt.subplots(1, 3, figsize=(17, 5))
400 |

```

```

401 | for ax, logx, mask, reg, title, expected in [
402 |     (axes[0], log_GS0, valid_0, r0,
403 |      'Proxy: vflat^2/hR (baseline)', 0.5),
404 |     (axes[1], log_GSbar_D, valid_D, rD,
405 |      'V_disk peak (BT, independent)', 1/3),
406 |     (axes[2], log_GSbar_E, valid_E, rE,
407 |      'Disk+Gas (BT, independent)', 1/3),
408 | ]:
409 |     if reg is None:
410 |         continue
411 |         xv, yv = logx[mask], log_gc[mask]
412 |         ax.scatter(xv, yv, s=10, alpha=0.5, c='steelblue', edgecolors='none')
413 |         xfit = np.linspace(xv.min(), xv.max(), 100)
414 |         ax.plot(xfit, reg['slope']*xfit + reg['intercept'], 'r-', lw=2,
415 |                label=f'fit: slope={reg["slope"]:.3f}+/-{reg["se"]:.3f}')
416 |         ax.plot(xfit, expected*xfit + reg['intercept'], 'b--', lw=1.5,
417 |                label=f'theory: slope={expected:.3f}')
418 |         ax.set_xlabel('log(G*Sigma / a0)', fontsize=10)
419 |         ax.set_ylabel('log(gc / a0)', fontsize=10)
420 |         ax.set_title(f'{title} (slope={expected})={reg["p_slope"]:.3f} [{reg["verdict"]}]',
421 |                    fontsize=10)
422 |         ax.legend(fontsize=8, loc='upper left')
423 |         ax.grid(True, alpha=0.3)
424 |
425 | fig.suptitle(f'V-1 revised: gc vs Sigma_bar scaling (N={N})', fontsize=13)
426 | fig.tight_layout()
427 | fig.savefig(os.path.join(BASE, 'fig_v1kai_slope_test.png'), dpi=150)
428 | print(' -> fig_v1kai_slope_test.png')
429 |
430 | # --- Fig 2: 4-panel summary ---
431 | fig, axes = plt.subplots(2, 2, figsize=(12, 10))
432 |
433 | # A: Sigma_bar distribution
434 | ax = axes[0, 0]
435 | vals = log_GSbar_D[valid_D]
436 | ax.hist(vals, bins=30, color='steelblue', alpha=0.7, edgecolor='white')
437 | ax.axvline(0, color='red', ls='--', lw=2, label='G*Sigma_bar = a0')
438 | ax.set_xlabel('log(G*Sigma_bar / a0)', fontsize=10)
439 | ax.set_ylabel('Count', fontsize=10)
440 | ax.set_title('A: True baryon surface density', fontsize=11)
441 | ax.legend(fontsize=9)
442 |
443 | # B: slope comparison across methods
444 | ax = axes[0, 1]
445 | methods = []
446 | slopes = []
447 | errors = []
448 | colors_bar = []
449 | for label, reg, col in [
450 |     ('Proxy (vflat^2/hR)', r0, 'steelblue'),
451 |     ('Vdisk peak (BT)', rD, 'darkorange'),
452 |     ('Disk+Gas (BT)', rE, 'green'),
453 |     ('BTFR decontam (circular)', rC, 'grey'),
454 | ]:
455 |     if reg is not None:
456 |         methods.append(label)
457 |         slopes.append(reg['slope'])
458 |         errors.append(reg['se'])
459 |         colors_bar.append(col)
460 |
461 | x_pos = np.arange(len(methods))
462 | ax.bar(x_pos, slopes, yerr=errors, color=colors_bar, alpha=0.7,
463 |        capsize=5, edgecolor='black', linewidth=0.5)
464 | ax.axhline(0.5, color='blue', ls='--', lw=1.5, label='0.5 (proxy prediction)')
465 | ax.axhline(1/3, color='red', ls='--', lw=1.5, label='1/3 (Sigma_bar prediction)')
466 | ax.set_xticks(x_pos)
467 | ax.set_xticklabels(methods, fontsize=8)
468 | ax.set_ylabel('Slope', fontsize=10)
469 | ax.set_title('B: Slope comparison', fontsize=11)
470 | ax.legend(fontsize=8)
471 | ax.grid(True, alpha=0.3, axis='y')
472 |
473 | # C: Residuals from 1/3 law (Method D)
474 | ax = axes[1, 0]
475 | if rD is not None:
476 |     xv = log_GSbar_D[valid_D]
477 |     yv = log_gc[valid_D]
478 |     resid = yv - (1/3 * xv + rD['intercept'])
479 |     ax.hist(resid, bins=25, color='darkorange', alpha=0.7, edgecolor='white')
480 |     ax.axvline(0, color='red', ls='--')
481 |     ax.set_xlabel('Residual from 1/3 law [dex]', fontsize=10)
482 |     ax.set_ylabel('Count', fontsize=10)
483 |     ax.set_title(f'C: Residual (sigma={np.std(resid):.3f} dex)', fontsize=11)
484 |
485 | # D: Residuals from 0.5 law (proxy, for comparison)
486 | ax = axes[1, 1]
487 | if r0 is not None:
488 |     xv = log_GS0[valid_0]

```

```

489 | yv = log_gc[valid_0]
490 | resid0 = yv - (0.5 * xv + r0['intercept'])
491 | ax.hist(resid0, bins=25, color='steelblue', alpha=0.7, edgecolor='white')
492 | ax.axvline(0, color='red', ls='--')
493 | ax.set_xlabel('Residual from 0.5 law (proxy) [dex]', fontsize=10)
494 | ax.set_ylabel('Count', fontsize=10)
495 | ax.set_title(f'D: Proxy residual (sigma={np.std(resid0):.3f} dex)', fontsize=11)
496 |
497 | fig.suptitle(f'V-1 revised summary (N={N})', fontsize=13, y=1.01)
498 | fig.tight_layout()
499 | fig.savefig(os.path.join(BASE, 'fig_v1kai_4panel.png'), dpi=150)
500 | print(' -> fig_v1kai_4panel.png')
501 |
502 | # =====
503 | # 7. Save CSV
504 | # =====
505 | outcsv = os.path.join(BASE, 'v1kai_results.csv')
506 | cols_out = ['galaxy', 'gc_over_a0', 'log_gc_a0', 'vflat', 'Yd', 'hR_kpc',
507 |            'GS0_proxy', 'log_GS0_proxy',
508 |            'G_Sbar_D', 'log_GSbar_D', 'G_Sbar_E', 'log_GSbar_E',
509 |            'G_Sbar_C', 'log_GSbar_C',
510 |            'vdisk_peak', 'vgas_peak']
511 | with open(outcsv, 'w', encoding='utf-8') as f:
512 |     f.write(','.join(cols_out) + '\n')
513 |     for r in results:
514 |         f.write(','.join(str(r.get(c, '')) for c in cols_out) + '\n')
515 | print(f'\n[4] Saved: {outcsv}')
516 |
517 | # =====
518 | # 8. Final Summary
519 | # =====
520 | print('\n' + '='*70)
521 | print('FINAL SUMMARY')
522 | print('='*70)
523 |
524 | print('\n Method          | slope +/- SE   | expected | p(match) | R^2   | verdict')
525 | print(' ' + '-'*95)
526 | for label, reg, expected in [
527 |     ('Proxy vflat^2/hR (baseline)', r0, 0.5),
528 |     ('Vdisk peak BT (independent)', rD, 1/3),
529 |     ('Disk+Gas BT (independent)', rE, 1/3),
530 |     ('BTFR decontam (circular)', rC, 1/3),
531 |     ('Vdisk peak BT vs 0.5', rD5, 0.5),
532 |     ('Disk+Gas BT vs 0.5', rE5, 0.5),
533 | ]:
534 |     if reg is None:
535 |         continue
536 |     print(f' {label:30s} | {reg["slope"]:.4f} +/- {reg["se"]:.4f} | '
537 |           f'{expected:.4f} | {reg["p_slope"]:.4f} | '
538 |           f'{reg["r2"]:.4f} | {reg["verdict"]}')
539 |
540 | print('\n Key question: Is the slope with TRUE Sigma_bar closer to 1/3 or 0.5?')
541 | if rD is not None and rD5 is not None:
542 |     if rD['p_slope'] > rD5['p_slope']:
543 |         print(' >>> 1/3 is BETTER fit for true Sigma_bar')
544 |     elif rD5['p_slope'] > rD['p_slope']:
545 |         print(' >>> 0.5 is BETTER fit for true Sigma_bar')
546 |     else:
547 |         print(' >>> Inconclusive')
548 |
549 | print('\n[DONE]')

```

3. [V-1b] gc の駆動力の特定 -- 多変量分解と偏相関

項目	内容
ファイル名	sparc_gc_vs_Mstar.py
行数	545
検証ID	V-1b

解析目的

gc vs vflat, hR, M_bar の7つの単変量回帰 + 多変量分解 + 偏相関を実施し、gc を駆動する物理量を特定する。proxy = vflat²/hR の二成分 (vflat, hR) の独立な寄与を分離する。

主要テスト

- [*] gc vs proxy (alpha=0.5 ベースライン)
- [*] gc vs M_bar直接 (gc非依存、質量の予測力)
- [*] gc vs Sigma_bar直接 (V-1改の再確認)
- [*] gc vs vflat 単独 / hR 単独 (個別寄与)
- [*] 多変量: gc ~ vflat^a x hR^b の a, b 推定
- [*] p(a=1.0), p(b=-0.5) でproxy分解を検定
- [*] 偏相関: vflat|hR, hR|vflat の独立性

結論

決定的発見: gc ~ vflat^{1.10} x hR^(-0.56), R²=0.529. p(vflat=1.0)=0.24, p(hR=-0.5)=0.49. M_bar直接 R²=0.021. 質量はgcを駆動しない。

出力ファイル

fig_gc_vs_Mstar.png, gc_vs_Mstar_results.csv

スクリプト全文

```
1 | # -*- coding: utf-8 -*-
2 | """
3 | V-1b: gc vs M* 直接回帰 -- gcの駆動力はバリオン総質量か?
4 | =====
5 | V-1改で Sigma_bar との相関が崩壊 (R^2=0.016) したのに対し、
6 | T-5では log M* が gc と最強相関 (r=+0.80) を示す。
7 |
8 | 本スクリプトは gc vs M* の直接回帰を行い:
9 | (1) gc ∝ M*^beta のbetaを推定
10 | (2) M* vs Sigma_bar vs vflat^2/hR のどれが gc を最もよく予測するか比較
11 | (3) hR の寄与を分離 (M* と hR の独立な効果)
12 |
13 | 実行: uv run --with scipy --with matplotlib python sparc_gc_vs_Mstar.py
14 |
15 | 必要ファイル:
16 | - Rotmod_LTG/*.dat
17 | - phase1/sparc_results.csv
18 | - TA3_gc_independent.csv
19 |
20 | 著者: 坂口 忍 (坂口製麺所)
21 | 日付: 2026年4月
22 | """
23 | import os, sys, glob, warnings
24 | import numpy as np
25 | from scipy import stats
26 |
27 | import matplotlib
28 | matplotlib.use('Agg')
29 | import matplotlib.pyplot as plt
30 | from matplotlib import font_manager as _fm
31 | for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
32 |            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
33 |            r'C:\Windows\Fonts\msgothic.ttc']:
34 |     try: _fm.fontManager.addfont(_fp)
35 |     except: pass
36 | for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
37 |     try:
38 |         plt.rcParams['font.family'] = fontname
39 |         break
40 |     except: continue
41 | plt.rcParams['axes.unicode_minus'] = False
```

```

42 | warnings.filterwarnings('ignore')
43 |
44 | # =====
45 | # Constants
46 | # =====
47 | a0    = 1.2e-10
48 | G_SI  = 6.674e-11
49 | kpc_m = 3.0857e19
50 | Msun  = 1.989e30
51 | pc_m  = 3.0857e16
52 |
53 | # =====
54 | # Paths
55 | # =====
56 | BASE = os.path.dirname(os.path.abspath(__file__))
57 | ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
58 | PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
59 | TA3    = os.path.join(BASE, 'TA3_gc_independent.csv')
60 |
61 | for p, label in [(ROTMOD, 'Rotmod_LTG'), (PHASE1, 'sparc_results.csv'), (TA3, 'TA3_gc_independent.csv')]:
62 |     if not os.path.exists(p):
63 |         print(f'[ERROR] {label} not found: {p}'); sys.exit(1)
64 |
65 | # =====
66 | # CSV loader
67 | # =====
68 | def load_csv(path):
69 |     with open(path, 'r', encoding='utf-8-sig') as f:
70 |         header = f.readline().strip()
71 |         sep = ',' if ',' in header else None
72 |         data = {}
73 |         with open(path, 'r', encoding='utf-8-sig') as f:
74 |             cols = [c.strip() for c in f.readline().strip().split(sep)]
75 |             rows = []
76 |             for line in f:
77 |                 line = line.strip()
78 |                 if not line: continue
79 |                 rows.append([p.strip() for p in line.split(sep)])
80 |             for i, col in enumerate(cols):
81 |                 vals = []
82 |                 for row in rows:
83 |                     if i < len(row):
84 |                         try: vals.append(float(row[i]))
85 |                             except: vals.append(row[i])
86 |                         else: vals.append(np.nan)
87 |                 data[col] = vals
88 |             return data
89 |
90 | def find_name_col(data):
91 |     for c in ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY']:
92 |         if c in data: return c
93 |     for k, v in data.items():
94 |         if isinstance(v[0], str): return k
95 |     return list(data.keys())[0]
96 |
97 | def get_key(info, candidates, default=None):
98 |     for c in candidates:
99 |         if c in info:
100 |             try: return float(info[c])
101 |                 except: return info[c]
102 |     return default
103 |
104 | # =====
105 | # Load data
106 | # =====
107 | print('[1] Loading data...')
108 | phase1 = load_csv(PHASE1)
109 | ta3     = load_csv(TA3)
110 | p1_nc  = find_name_col(phase1)
111 | ta3_nc = find_name_col(ta3)
112 |
113 | galaxy_info = {}
114 | for i, name in enumerate(phase1[p1_nc]):
115 |     name = str(name).strip()
116 |     info = {}
117 |     for k in phase1:
118 |         if k == p1_nc: continue
119 |         try: info[k] = float(phase1[k][i])
120 |             except: info[k] = phase1[k][i]
121 |     galaxy_info[name] = info
122 |
123 | for i, name in enumerate(ta3[ta3_nc]):
124 |     name = str(name).strip()
125 |     if name in galaxy_info:
126 |         for k in ta3:
127 |             if k == ta3_nc: continue
128 |             try: galaxy_info[name][k] = float(ta3[k][i])
129 |                 except: galaxy_info[name][k] = ta3[k][i]

```

```

130 |
131 | # =====
132 | # Load rotmod and compute quantities
133 | # =====
134 | def load_rotmod(filepath):
135 |     cols = [[] for _ in range(8)]
136 |     with open(filepath, 'r') as f:
137 |         for line in f:
138 |             line = line.strip()
139 |             if not line or line.startswith('#'): continue
140 |             parts = line.split()
141 |             if len(parts) < 6: continue
142 |             try:
143 |                 for j in range(min(len(parts),8)):
144 |                     cols[j].append(float(parts[j]))
145 |                 for j in range(len(parts),8):
146 |                     cols[j].append(0.0)
147 |             except ValueError: continue
148 |     return tuple(np.array(c) for c in cols)
149 |
150 | print('[2] Processing galaxies...')
151 | results = []
152 | rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
153 |
154 | for fpath in rotmod_files:
155 |     gname = os.path.splitext(os.path.basename(fpath))[0].replace('_rotmod','').strip()
156 |     info = None
157 |     for key in [gname, gname.upper(), gname.lower()]:
158 |         if key in galaxy_info:
159 |             info = galaxy_info[key]; break
160 |     if info is None: continue
161 |
162 |     ud = get_key(info, ['upsilon_d', 'Upsilon_d', 'Ud', 'ud', 'Yd'])
163 |     gc_a0 = get_key(info, ['gc_over_a0', 'gc/a0', 'gc_ratio'])
164 |     vflat = get_key(info, ['vflat', 'Vflat', 'v_flat'])
165 |
166 |     if ud is None or gc_a0 is None or vflat is None: continue
167 |     if np.isnan(ud) or np.isnan(gc_a0) or gc_a0<=0 or vflat<=0: continue
168 |
169 |     try:
170 |         rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
171 |     except: continue
172 |     if len(rad) < 3: continue
173 |
174 |     # hR from rpk
175 |     vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
176 |     rpk_idx = np.argmax(vds)
177 |     rpk = rad[rpk_idx]
178 |     if rpk < 0.01 or rpk >= rad.max()*0.9: continue
179 |     hR_kpc = rpk / 2.15
180 |     hR_m = hR_kpc * kpc_m
181 |
182 |     vflat_ms = vflat * 1e3
183 |     GS0_proxy = vflat_ms**2 / hR_m
184 |
185 |     # --- M_star estimation ---
186 |     # Method 1: from BTFR -- M_bar = vflat^4 / (gc * G)
187 |     gc_si = gc_a0 * a0
188 |     M_bar_btfr = vflat_ms**4 / (gc_si * G_SI) # kg
189 |     M_bar_btfr_sun = M_bar_btfr / Msun
190 |
191 |     # Method 2: from disk velocity profile (gc-independent)
192 |     # M_disk = Yd * L_disk
193 |     # For exponential disk: V_disk_peak^2 = 0.56*pi*G*Sigma0*hR
194 |     # => Sigma0 = V_disk_peak^2 / (0.56*pi*G*hR)
195 |     # => M_disk = 2*pi*Sigma0*hR^2 = 2*V_disk_peak^2*hR / (0.56*G)
196 |     vdisk_peak_ms = np.max(np.sqrt(max(ud,0.01)) * np.abs(vdisk)) * 1e3
197 |     M_disk_direct = 2.0 * vdisk_peak_ms**2 * hR_m / (0.56 * G_SI)
198 |     M_disk_direct_sun = M_disk_direct / Msun
199 |
200 |     # Gas mass (same approach)
201 |     vgas_peak_ms = np.max(np.abs(vgas)) * 1e3
202 |     M_gas_direct = 2.0 * vgas_peak_ms**2 * hR_m / (0.56 * G_SI) if vgas_peak_ms > 0 else 0
203 |     M_gas_direct_sun = M_gas_direct / Msun
204 |
205 |     M_bar_direct_sun = M_disk_direct_sun + M_gas_direct_sun
206 |
207 |     # Sigma_bar (gc-independent)
208 |     Sigma_bar = M_disk_direct / (2 * np.pi * hR_m**2) # kg/m^2
209 |     G_Sigma_bar = G_SI * Sigma_bar # m/s^2
210 |
211 |     results.append({
212 |         'galaxy': gname,
213 |         'gc_over_a0': gc_a0,
214 |         'log_gc': np.log10(gc_a0),
215 |         'vflat': vflat,
216 |         'Yd': ud,
217 |         'hR_kpc': hR_kpc,

```

```

218 |         'GS0_proxy':      GS0_proxy,
219 |         'log_GS0':        np.log10(GS0_proxy / a0),
220 |         # BTFR mass (circular)
221 |         'log_Mbar_btfr':  np.log10(M_bar_btfr_sun) if M_bar_btfr_sun>0 else np.nan,
222 |         # Direct mass (independent)
223 |         'M_disk_sun':     M_disk_direct_sun,
224 |         'M_gas_sun':      M_gas_direct_sun,
225 |         'M_bar_direct_sun': M_bar_direct_sun,
226 |         'log_Mbar_direct': np.log10(M_bar_direct_sun) if M_bar_direct_sun>0 else np.nan,
227 |         'log_Mdisk':      np.log10(M_disk_direct_sun) if M_disk_direct_sun>0 else np.nan,
228 |         # Surface density (independent)
229 |         'G_Sigma_bar':    G_Sigma_bar,
230 |         'log_GSbar':      np.log10(G_Sigma_bar/a0) if G_Sigma_bar>0 else np.nan,
231 |         # Individual components for decomposition
232 |         'log_vflat':      np.log10(vflat),
233 |         'log_hR':         np.log10(hR_kpc),
234 |         'vdisk_peak':     vdisk_peak_ms/1e3,
235 |     })
236 |
237 | N = len(results)
238 | print(f' Processed: {N} galaxies')
239 | if N < 10:
240 |     print('[ERROR] Too few galaxies.');
```

```

241 |
242 | # =====
243 | # Arrays
244 | # =====
245 | log_gc    = np.array([r['log_gc'] for r in results])
246 | log_GS0   = np.array([r['log_GS0'] for r in results])
247 | log_Mbar  = np.array([r['log_Mbar_direct'] for r in results])
248 | log_Mdisk = np.array([r['log_Mdisk'] for r in results])
249 | log_GSbar = np.array([r['log_GSbar'] for r in results])
250 | log_vflat = np.array([r['log_vflat'] for r in results])
251 | log_hR    = np.array([r['log_hR'] for r in results])
252 | log_Mbtfr = np.array([r['log_Mbar_btfr'] for r in results])
253 |
254 | # =====
255 | # Regression helper
256 | # =====
257 | def regtest(x, y, label, expected_slopes=None):
258 |     mask = np.isfinite(x) & np.isfinite(y)
259 |     xv, yv = x[mask], y[mask]
260 |     n = len(xv)
261 |     if n < 10:
262 |         print(f' [{label}] N={n} too few'); return None
263 |
264 |     slope, intercept, r_val, p_val, se = stats.linregress(xv, yv)
265 |     rho_s, p_s = stats.spearmanr(xv, yv)
266 |
267 |     print(f' %n [{label}] N={n}')
268 |     print(f' slope = {slope:.4f} +/- {se:.4f}')
269 |     print(f' R^2 = {r_val**2:.4f}')
270 |     print(f' Spearman = {rho_s:.4f} (p={p_s:.2e})')
```

```

271 |
272 |     if expected_slopes:
273 |         for es in expected_slopes:
274 |             t = abs(slope - es) / se
275 |             p = 2*(1-stats.t.cdf(t, df=n-2))
276 |             v = 'PASS' if p>0.05 else 'MARGINAL' if p>0.01 else 'FAIL'
277 |             print(f' p(slope={es:.4f}) = {p:.4f} [{v}]')
```

```

278 |
279 |     return {'slope':slope, 'se':se, 'intercept':intercept,
280 |            'r2':r_val**2, 'rho':rho_s, 'p_s':p_s, 'n':n,
281 |            'xv':xv, 'yv':yv}
282 |
283 | # =====
284 | # Tests
285 | # =====
286 | print(f' %n + '= *70)
287 | print(f' V-lb: gc vs M* direct regression')
288 | print(f' '= *70)
289 |
290 | print(f' %n--- A: Baseline proxy (vflat^2/hR) ---')
291 | rA = regtest(log_GS0, log_gc, 'proxy vflat^2/hR', [0.5])
292 |
293 | print(f' %n--- B: gc vs M_bar_direct (gc-independent, BT) ---')
294 | rB = regtest(log_Mbar, log_gc, 'M_bar direct (BT)', [1/3, 1/4, 0.2, 0.125])
295 |
296 | print(f' %n--- C: gc vs M_disk only (gc-independent) ---')
297 | rC = regtest(log_Mdisk, log_gc, 'M_disk only', [1/3, 1/4, 0.2, 0.125])
298 |
299 | print(f' %n--- D: gc vs Sigma_bar (gc-independent, V-1kai recheck) ---')
300 | rD = regtest(log_GSbar, log_gc, 'G*Sigma_bar (BT)', [1/3, 0.5])
301 |
302 | print(f' %n--- E: gc vs vflat only ---')
303 | rE = regtest(log_vflat, log_gc, 'vflat only', [1.0, 2.0])
304 |
305 | print(f' %n--- F: gc vs hR only ---')
```

```

306 | rF = regtest(log_hR, log_gc, 'hR only', [-0.5, -1.0])
307 |
308 | print('\n--- G: gc vs M_bar BTFR (circular, diagnostic) ---')
309 | rG = regtest(log_Mbtf, log_gc, 'M_bar BTFR (circular)', [1/3, 1/4])
310 |
311 | # =====
312 | # Multivariate: gc = a * log(vflat) + b * log(hR) + c
313 | # =====
314 | print('\n--- H: Multivariate log(gc) = a*log(vflat) + b*log(hR) + c ---')
315 | mask = np.isfinite(log_gc) & np.isfinite(log_vflat) & np.isfinite(log_hR)
316 | X = np.column_stack([log_vflat[mask], log_hR[mask], np.ones(mask.sum())])
317 | y = log_gc[mask]
318 | # OLS
319 | beta, residuals, rank, sv = np.linalg.lstsq(X, y, rcond=None)
320 | y_pred = X @ beta
321 | ss_res = np.sum((y - y_pred)**2)
322 | ss_tot = np.sum((y - y.mean())**2)
323 | r2_multi = 1 - ss_res/ss_tot
324 | n_multi = len(y)
325 | # Standard errors
326 | if len(residuals) > 0:
327 |     mse = residuals[0] / (n_multi - 3)
328 | else:
329 |     mse = ss_res / (n_multi - 3)
330 | se_beta = np.sqrt(np.diag(mse * np.linalg.inv(X.T @ X)))
331 |
332 | print(f' N = {n_multi}')
333 | print(f' log(gc) = {beta[0]:.4f}*log(vflat) + {beta[1]:.4f}*log(hR) + {beta[2]:.4f}')
334 | print(f' SE:      {se_beta[0]:.4f}          {se_beta[1]:.4f}          {se_beta[2]:.4f}')
335 | print(f' R^2 = {r2_multi:.4f}')
336 | print(f' vflat exponent: {beta[0]:.3f} +/- {se_beta[0]:.3f}')
337 | print(f' hR exponent:    {beta[1]:.3f} +/- {se_beta[1]:.3f}')
338 |
339 | # Test: is it consistent with proxy (vflat^2/hR)^0.5 = vflat^1 * hR^(-0.5)?
340 | print(f'\n Proxy decomposition test:')
341 | print(f' (vflat^2/hR)^0.5 predicts: vflat^1.0, hR^-0.5')
342 | print(f' Observed:          vflat^{beta[0]:.3f}, hR^{beta[1]:.3f}')
343 | t_vf = abs(beta[0] - 1.0) / se_beta[0]
344 | p_vf = 2*(1-stats.t.cdf(t_vf, df=n_multi-3))
345 | t_hr = abs(beta[1] - (-0.5)) / se_beta[1]
346 | p_hr = 2*(1-stats.t.cdf(t_hr, df=n_multi-3))
347 | print(f' p(vflat=1.0) = {p_vf:.4f}')
348 | print(f' p(hR=-0.5) = {p_hr:.4f}')
349 |
350 | # =====
351 | # Partial correlation: gc vs hR controlling for vflat
352 | # =====
353 | print('\n--- I: Partial correlations ---')
354 | # gc vs hR | vflat
355 | mask_p = np.isfinite(log_gc) & np.isfinite(log_vflat) & np.isfinite(log_hR)
356 | gc_p = log_gc[mask_p]
357 | vf_p = log_vflat[mask_p]
358 | hr_p = log_hR[mask_p]
359 |
360 | # Residualize gc and hR against vflat
361 | _, _, r_gc_vf, _, _ = stats.linregress(vf_p, gc_p)
362 | res_gc = gc_p - (stats.linregress(vf_p, gc_p)[0]*vf_p + stats.linregress(vf_p, gc_p)[1])
363 | res_hr = hr_p - (stats.linregress(vf_p, hr_p)[0]*vf_p + stats.linregress(vf_p, hr_p)[1])
364 | rho_partial, p_partial = stats.spearmanr(res_gc, res_hr)
365 | print(f' gc vs hR | vflat: rho_partial = {rho_partial:.4f}, p = {p_partial:.4e}')
366 | print(f' (Does hR contribute BEYOND vflat?)')
367 |
368 | # gc vs vflat | hR
369 | res_gc2 = gc_p - (stats.linregress(hr_p, gc_p)[0]*hr_p + stats.linregress(hr_p, gc_p)[1])
370 | res_vf2 = vf_p - (stats.linregress(hr_p, vf_p)[0]*hr_p + stats.linregress(hr_p, vf_p)[1])
371 | rho_partial2, p_partial2 = stats.spearmanr(res_gc2, res_vf2)
372 | print(f' gc vs vflat | hR: rho_partial = {rho_partial2:.4f}, p = {p_partial2:.4e}')
373 | print(f' (Does vflat contribute BEYOND hR?)')
374 |
375 | # =====
376 | # Figures
377 | # =====
378 | print('\n[3] Generating figures...')
379 |
380 | fig, axes = plt.subplots(2, 3, figsize=(17, 11))
381 |
382 | # A: proxy baseline
383 | ax = axes[0,0]
384 | if rA:
385 |     ax.scatter(rA['xv'], rA['yv'], s=10, alpha=0.5, c='steelblue', edgecolors='none')
386 |     xf = np.linspace(rA['xv'].min(), rA['xv'].max(), 100)
387 |     ax.plot(xf, rA['slope']*xf+rA['intercept'], 'r-', lw=2,
388 |            label=f'slope={rA["slope"]:.3f}, R^2={rA["r2"]:.3f}')
389 |     ax.plot(xf, 0.5*xf+rA['intercept'], 'b--', lw=1, label='theory: 0.5')
390 |     ax.set_xlabel('log(vflat^2/hR / a0)')
391 |     ax.set_ylabel('log(gc/a0)')
392 |     ax.set_title('A: Proxy (baseline)')
393 |     ax.legend(fontsize=8)

```

```

394 |     ax.grid(True, alpha=0.3)
395 |
396 | # B: M_bar direct
397 | ax = axes[0,1]
398 | if rB:
399 |     ax.scatter(rB['xv'], rB['yv'], s=10, alpha=0.5, c='darkorange', edgecolors='none')
400 |     xf = np.linspace(rB['xv'].min(), rB['xv'].max(), 100)
401 |     ax.plot(xf, rB['slope']*xf+rB['intercept'], 'r-', lw=2,
402 |            label=f'slope={rB["slope"]:.3f}, R^2={rB["r2"]:.3f}')
403 |     for es, col, ls in [(1/3,'blue','--'),(1/4,'green','--'),(0.125,'purple',':')]:
404 |         ax.plot(xf, es*xf+rB['intercept'], color=col, ls=ls, lw=1,
405 |                label=f'{es:.3f}')
406 |     ax.set_xlabel('log(M_bar / M_sun) [direct, gc-free]')
407 |     ax.set_ylabel('log(gc/a0)')
408 |     ax.set_title('B: M_bar direct (BT)')
409 |     ax.legend(fontsize=7)
410 |     ax.grid(True, alpha=0.3)
411 |
412 | # C: Sigma_bar
413 | ax = axes[0,2]
414 | if rD:
415 |     ax.scatter(rD['xv'], rD['yv'], s=10, alpha=0.5, c='green', edgecolors='none')
416 |     xf = np.linspace(rD['xv'].min(), rD['xv'].max(), 100)
417 |     ax.plot(xf, rD['slope']*xf+rD['intercept'], 'r-', lw=2,
418 |            label=f'slope={rD["slope"]:.3f}, R^2={rD["r2"]:.3f}')
419 |     ax.set_xlabel('log(G*Sigma_bar / a0) [direct]')
420 |     ax.set_ylabel('log(gc/a0)')
421 |     ax.set_title('C: Sigma_bar (recheck)')
422 |     ax.legend(fontsize=8)
423 |     ax.grid(True, alpha=0.3)
424 |
425 | # D: vflat only
426 | ax = axes[1,0]
427 | if rE:
428 |     ax.scatter(rE['xv'], rE['yv'], s=10, alpha=0.5, c='crimson', edgecolors='none')
429 |     xf = np.linspace(rE['xv'].min(), rE['xv'].max(), 100)
430 |     ax.plot(xf, rE['slope']*xf+rE['intercept'], 'r-', lw=2,
431 |            label=f'slope={rE["slope"]:.3f}, R^2={rE["r2"]:.3f}')
432 |     ax.set_xlabel('log(vflat / km/s)')
433 |     ax.set_ylabel('log(gc/a0)')
434 |     ax.set_title('D: vflat only')
435 |     ax.legend(fontsize=8)
436 |     ax.grid(True, alpha=0.3)
437 |
438 | # E: hR only
439 | ax = axes[1,1]
440 | if rF:
441 |     ax.scatter(rF['xv'], rF['yv'], s=10, alpha=0.5, c='teal', edgecolors='none')
442 |     xf = np.linspace(rF['xv'].min(), rF['xv'].max(), 100)
443 |     ax.plot(xf, rF['slope']*xf+rF['intercept'], 'r-', lw=2,
444 |            label=f'slope={rF["slope"]:.3f}, R^2={rF["r2"]:.3f}')
445 |     ax.set_xlabel('log(hR / kpc)')
446 |     ax.set_ylabel('log(gc/a0)')
447 |     ax.set_title('E: hR only')
448 |     ax.legend(fontsize=8)
449 |     ax.grid(True, alpha=0.3)
450 |
451 | # F: R^2 comparison bar chart
452 | ax = axes[1,2]
453 | labels_bar = []
454 | r2_bar = []
455 | cols_bar = []
456 | for label, reg, col in [
457 |     ('Proxy* $\nu$ vflat^2/hR', rA, 'steelblue'),
458 |     ('M_bar*(direct)', rB, 'darkorange'),
459 |     ('Sigma_bar*(direct)', rD, 'green'),
460 |     ('vflat*nonly', rE, 'crimson'),
461 |     ('hR*nonly', rF, 'teal'),
462 |     ('M_bar*(BTFR circ)', rG, 'grey'),
463 | ]:
464 |     if reg:
465 |         labels_bar.append(label)
466 |         r2_bar.append(reg['r2'])
467 |         cols_bar.append(col)
468 |
469 | # Add multivariate
470 | labels_bar.append('vflat+hR* $\nu$ multivar')
471 | r2_bar.append(r2_multi)
472 | cols_bar.append('purple')
473 |
474 | xpos = np.arange(len(labels_bar))
475 | ax.bar(xpos, r2_bar, color=cols_bar, alpha=0.7, edgecolor='black', linewidth=0.5)
476 | ax.set_xticks(xpos)
477 | ax.set_xticklabels(labels_bar, fontsize=8)
478 | ax.set_ylabel('R^2', fontsize=11)
479 | ax.set_title('F: Predictive power comparison', fontsize=11)
480 | ax.grid(True, alpha=0.3, axis='y')
481 | for i, v in enumerate(r2_bar):

```

```

482 |     ax.text(i, v+0.01, f'{v:.3f}', ha='center', fontsize=8)
483 |
484 | fig.suptitle(f'V-1b: What drives gc? (N={N})', fontsize=14, y=1.01)
485 | fig.tight_layout()
486 | fig.savefig(os.path.join(BASE, 'fig_gc_vs_Mstar.png'), dpi=150)
487 | print(' -> fig_gc_vs_Mstar.png')
488 |
489 | # =====
490 | # Save CSV
491 | # =====
492 | outcsv = os.path.join(BASE, 'gc_vs_Mstar_results.csv')
493 | cols_out = ['galaxy', 'gc_over_a0', 'log_gc', 'vflat', 'Yd', 'hR_kpc',
494 |            'log_GS0', 'log_Mbar_direct', 'log_Mdisk', 'log_GSbar',
495 |            'log_vflat', 'log_hR', 'log_Mbar_btfr', 'vdisk_peak']
496 | with open(outcsv, 'w', encoding='utf-8') as f:
497 |     f.write(','.join(cols_out)+'\n')
498 |     for r in results:
499 |         f.write(','.join(str(r.get(c,'')) for c in cols_out)+'\n')
500 | print(f'\n[4] Saved: {outcsv}')
501 |
502 | # =====
503 | # Final Summary
504 | # =====
505 | print('\n'+' '*70)
506 | print('FINAL SUMMARY')
507 | print('='*70)
508 |
509 | print(f'\n {"Predictor":<25s} | {"slope":>8s} +/- {"SE":>6s} | {"R^2":>6s} | {"Spearman":>8s}')
510 | print(' ' + '*'*70)
511 | for label, reg in [
512 |     ('Proxy vflat^2/hR', rA),
513 |     ('M_bar direct (BT)', rB),
514 |     ('M_disk only', rC),
515 |     ('G*Sigma_bar (BT)', rD),
516 |     ('vflat only', rE),
517 |     ('hR only', rF),
518 |     ('M_bar BTFR (circ)', rG),
519 | ]:
520 |     if reg:
521 |         print(f' {label:<25s} | {reg["slope"]:>8.4f} +/- {reg["se"]:>6.4f} | '
522 |               f'{reg["r2"]:>6.4f} | {reg["rho"]:>8.4f}')
523 | print(f' {"vflat+hR multivar":<25s} | {"--":>8s} {"--":>6s} | {r2_multi:>6.4f} | {"--":>8s}')
524 |
525 | print(f'\n Multivariate decomposition:')
526 | print(f' gc ~ vflat^{beta[0]:.3f} * hR^{beta[1]:.3f}')
527 | print(f' Proxy predicts: vflat^1.0 * hR^-0.5')
528 | print(f' p(vflat=1.0) = {p_vf:.4f}')
529 | print(f' p(hR=-0.5) = {p_hr:.4f}')
530 |
531 | print(f'\n Partial correlations:')
532 | print(f' gc vs hR | vflat: rho={rho_partial:.4f}, p={p_partial:.4e}')
533 | print(f' gc vs vflat | hR: rho={rho_partial2:.4f}, p={p_partial2:.4e}')
534 |
535 | print('\n KEY QUESTION: Is gc driven by M* (mass) or proxy (vflat^2/hR)?')
536 | if rB and rA:
537 |     if rB['r2'] > rA['r2'] * 0.8:
538 |         print(' >>> M* has comparable predictive power -> mass is the driver')
539 |     elif rB['r2'] < 0.1:
540 |         print(' >>> M* (direct) has NO predictive power -> proxy encodes something else')
541 |         print(' >>> gc depends on vflat (which includes DM/membrane effect)')
542 |     else:
543 |         print(' >>> Intermediate case -> further investigation needed')
544 |
545 | print('\n[DONE]')

```

4. [N-1 Layer 2] MOND遷移域補正と hR 偏相関の起源

項目	内容
ファイル名	sparc_N1_layer2.py
行数	592
検証ID	N-1 Layer 2

解析目的

hR の偏相関 ($\rho=-0.356$) が MOND 遷移域の補正効果で説明されるかを検証。 $x_{\text{outer}} = g_{\text{N}}(r_{\text{outer}})/g_{\text{c}}$ を計算し、遷移補正因子を多変量モデルに組み込む。

主要テスト

- [*] x_{outer} 分布 (SPARC銀河がどの程度MOND領域にいるか)
- [*] x_{outer} vs hR の相関
- [*] 補正因子を含む多変量での R^2 改善
- [*] hR偏相関の削減率

結論

R^2 改善 (+0.130) は循環アーティファクト (Layer 2b で判明)。hR 偏相関の36%削減も見かけ上。

出力ファイル

fig_N1_layer2.png, N1_layer2_results.csv

スクリプト全文

```
1 | # -*- coding: utf-8 -*-
2 | """
3 | N-1 第二層検証: MOND遷移域補正と hR 偏相関の起源
4 | =====
5 | 式(5)完全形  $g_{\text{obs}} = (g_{\text{N}} + \sqrt{g_{\text{N}}^2 + 4*gc*g_{\text{N}}}) / 2$  において、
6 | MOND極限  $g_{\text{obs}} \sim \sqrt{gc*g_{\text{N}}}$  は  $g_{\text{N}} \ll gc$  のとき成立する。
7 |
8 | 仮説: hR の偏相関 ( $\rho=-0.356$ ) は、コンパクト銀河 (小hR) で
9 |  $g_{\text{N}}(r_{\text{outer}})/g_{\text{c}}$  が大きく MOND極限からのずれが大きいに起因する。
10 |
11 | 検証:
12 | (1) 各銀河で  $x_{\text{outer}} = g_{\text{N}}(r_{\text{outer}})/g_{\text{c}}$  を計算
13 | (2)  $x_{\text{outer}}$  と hR の相関を確認
14 | (3) MOND遷移域補正を式(5')に組み込み、hR偏相関が消えるか検証
15 | (4) 補正後の残差構造を確認
16 |
17 | 実行: uv run --with scipy --with matplotlib python sparc_N1_layer2.py
18 |
19 | 必要ファイル:
20 | - Rotmod_LTG/*.dat
21 | - phase1/sparc_results.csv
22 | - TA3_gc_independent.csv
23 |
24 | 著者: 坂口 忍 (坂口製麺所)
25 | 日付: 2026年4月
26 | """
27 | import os, sys, glob, warnings
28 | import numpy as np
29 | from scipy import stats
30 |
31 | import matplotlib
32 | matplotlib.use('Agg')
33 | import matplotlib.pyplot as plt
34 | from matplotlib import font_manager as _fm
35 | for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
36 |            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
37 |            r'C:\Windows\Fonts\msgothic.ttc']:
38 |     try: _fm.fontManager.addfont(_fp)
39 |     except: pass
40 | for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
41 |     try:
42 |         plt.rcParams['font.family'] = fontname
43 |         break
44 |     except: continue
45 | plt.rcParams['axes.unicode_minus'] = False
46 | warnings.filterwarnings('ignore')
47 |
48 | # =====
```

```

49 | # Constants
50 | # =====
51 | a0    = 1.2e-10
52 | G_SI  = 6.674e-11
53 | kpc_m = 3.0857e19
54 |
55 | # =====
56 | # Paths
57 | # =====
58 | BASE = os.path.dirname(os.path.abspath(__file__))
59 | ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
60 | PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
61 | TA3    = os.path.join(BASE, 'TA3_gc_independent.csv')
62 |
63 | for p, label in [(ROTMOD, 'Rotmod_LTG'), (PHASE1, 'sparc_results.csv'),
64 |                 (TA3, 'TA3_gc_independent.csv')]:
65 |     if not os.path.exists(p):
66 |         print(f'[ERROR] {label} not found: {p}'); sys.exit(1)
67 |
68 | # =====
69 | # CSV / rotmod loaders (same as previous scripts)
70 | # =====
71 | def load_csv(path):
72 |     with open(path, 'r', encoding='utf-8-sig') as f:
73 |         header = f.readline().strip()
74 |         sep = ',' if ',' in header else None
75 |         data = {}
76 |         with open(path, 'r', encoding='utf-8-sig') as f:
77 |             cols = [c.strip() for c in f.readline().strip().split(sep)]
78 |             rows = []
79 |             for line in f:
80 |                 line = line.strip()
81 |                 if not line: continue
82 |                 rows.append([p.strip() for p in line.split(sep)])
83 |             for i, col in enumerate(cols):
84 |                 vals = []
85 |                 for row in rows:
86 |                     if i < len(row):
87 |                         try: vals.append(float(row[i]))
88 |                             except: vals.append(row[i])
89 |                         else: vals.append(np.nan)
90 |                 data[col] = vals
91 |             return data
92 |
93 | def find_name_col(data):
94 |     for c in ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY']:
95 |         if c in data: return c
96 |     for k, v in data.items():
97 |         if isinstance(v[0], str): return k
98 |     return list(data.keys())[0]
99 |
100 | def get_key(info, candidates, default=None):
101 |     for c in candidates:
102 |         if c in info:
103 |             try: return float(info[c])
104 |             except: return info[c]
105 |     return default
106 |
107 | def load_rotmod(filepath):
108 |     cols = [[] for _ in range(8)]
109 |     with open(filepath, 'r') as f:
110 |         for line in f:
111 |             line = line.strip()
112 |             if not line or line.startswith('#'): continue
113 |             parts = line.split()
114 |             if len(parts) < 6: continue
115 |             try:
116 |                 for j in range(min(len(parts), 8)):
117 |                     cols[j].append(float(parts[j]))
118 |                 for j in range(len(parts), 8):
119 |                     cols[j].append(0.0)
120 |             except ValueError: continue
121 |     return tuple(np.array(c) for c in cols)
122 |
123 | # =====
124 | # Load data
125 | # =====
126 | print('[1] Loading data...')
127 | phase1 = load_csv(PHASE1)
128 | ta3    = load_csv(TA3)
129 | p1_nc  = find_name_col(phase1)
130 | ta3_nc = find_name_col(ta3)
131 |
132 | galaxy_info = {}
133 | for i, name in enumerate(phase1[p1_nc]):
134 |     name = str(name).strip()
135 |     info = {}
136 |     for k in phase1:

```

```

137 |         if k == p1_nc: continue
138 |         try: info[k] = float(phase1[k][i])
139 |         except: info[k] = phase1[k][i]
140 |     galaxy_info[name] = info
141 |
142 | for i, name in enumerate(ta3[ta3_nc]):
143 |     name = str(name).strip()
144 |     if name in galaxy_info:
145 |         for k in ta3:
146 |             if k == ta3_nc: continue
147 |             try: galaxy_info[name][k] = float(ta3[k][i])
148 |             except: galaxy_info[name][k] = ta3[k][i]
149 |
150 | # =====
151 | # Process galaxies
152 | # =====
153 | print('[2] Processing galaxies...')
154 | results = []
155 | rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
156 |
157 | for fpath in rotmod_files:
158 |     gname = os.path.splitext(os.path.basename(fpath))[0].replace('_rotmod', '').strip()
159 |     info = None
160 |     for key in [gname, gname.upper(), gname.lower()]:
161 |         if key in galaxy_info:
162 |             info = galaxy_info[key]; break
163 |     if info is None: continue
164 |
165 |     ud = get_key(info, ['upsilon_d', 'Upsilon_d', 'Ud', 'ud', 'Yd'])
166 |     gc_a0 = get_key(info, ['gc_over_a0', 'gc/a0', 'gc_ratio'])
167 |     vflat = get_key(info, ['vflat', 'Vflat', 'v_flat'])
168 |
169 |     if ud is None or gc_a0 is None or vflat is None: continue
170 |     if np.isnan(ud) or np.isnan(gc_a0) or gc_a0<=0 or vflat<=0: continue
171 |
172 |     try:
173 |         rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
174 |     except: continue
175 |     if len(rad) < 3: continue
176 |
177 |     # hR from rpk
178 |     vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
179 |     rpk_idx = np.argmax(vds)
180 |     rpk = rad[rpk_idx]
181 |     if rpk < 0.01 or rpk >= rad.max()*0.9: continue
182 |     hR_kpc = rpk / 2.15
183 |     hR_m = hR_kpc * kpc_m
184 |
185 |     # g_N(r) from baryonic components
186 |     r_m = rad * kpc_m
187 |     vbar2 = (vgas**2 + ud * np.sign(vdisk)*vdisk**2
188 |             + 0.7 * np.sign(vbul)*vbul**2)
189 |     vbar2 = np.abs(vbar2)
190 |     g_N = np.zeros_like(r_m)
191 |     mask_r = r_m > 0
192 |     g_N[mask_r] = vbar2[mask_r] * 1e6 / r_m[mask_r]
193 |
194 |     gc_si = gc_a0 * a0
195 |
196 |     # --- Key quantity: x_outer = g_N(r_outer) / gc ---
197 |     # Use outer 3 points average for stability
198 |     n_outer = min(3, len(rad))
199 |     g_N_outer = np.mean(g_N[-n_outer:])
200 |     x_outer = g_N_outer / gc_si if gc_si > 0 else np.nan
201 |
202 |     # --- g_N at various radii ---
203 |     # At r = 2.2 hR (peak region)
204 |     r_peak_target = 2.2 * hR_kpc
205 |     idx_peak = np.argmin(np.abs(rad - r_peak_target))
206 |     g_N_peak = g_N[idx_peak]
207 |     x_peak = g_N_peak / gc_si if gc_si > 0 else np.nan
208 |
209 |     # --- MOND transition correction ---
210 |     # Full formula: g_obs = (g_N + sqrt(g_N^2 + 4*gc*g_N)) / 2
211 |     # Deep MOND: g_obs_MOND ~ sqrt(gc * g_N)
212 |     # Ratio: g_obs_full / g_obs_MOND = correction factor
213 |     # At r_outer:
214 |     if g_N_outer > 0 and gc_si > 0:
215 |         g_obs_full = (g_N_outer + np.sqrt(g_N_outer**2 + 4*gc_si*g_N_outer)) / 2
216 |         g_obs_mond = np.sqrt(gc_si * g_N_outer)
217 |         correction = g_obs_full / g_obs_mond # >1 always, larger for larger x
218 |     else:
219 |         correction = np.nan
220 |
221 |     # --- vflat from full formula vs MOND limit ---
222 |     # vflat_obs^2 / r_outer = g_obs_full
223 |     # vflat_mond^2 / r_outer = g_obs_mond = sqrt(gc * g_N)
224 |     # So vflat_obs / vflat_mond = sqrt(correction)

```

```

225 | # And (vflat_obs^2/hR) / (vflat_mond^2/hR) = correction
226 | # The proxy GS0 = vflat^2/hR includes this correction factor
227 |
228 | vflat_ms = vflat * 1e3
229 | GS0_proxy = vflat_ms**2 / hR_m
230 |
231 | # --- Corrected proxy: remove transition effect ---
232 | # GS0_corrected = GS0_proxy / correction^2
233 | # (because vflat^2 ~ g_obs * r, and correction = g_obs_full/g_obs_mond)
234 | # Actually: vflat^2/r = g_obs at r_outer
235 | # proxy = vflat^2/hR = g_obs(r_outer) * r_outer/hR
236 | # MOND proxy = sqrt(gc*g_N(r_outer)) * r_outer/hR
237 | # So proxy/MOND_proxy = correction
238 | # To get "MOND-corrected proxy": divide by correction
239 |
240 | if np.isfinite(correction) and correction > 0:
241 |     GS0_corrected = GS0_proxy / correction
242 | else:
243 |     GS0_corrected = np.nan
244 |
245 | # Also compute a per-point mean correction
246 | corrections_all = np.ones_like(g_N)
247 | for i_r in range(len(g_N)):
248 |     if g_N[i_r] > 0 and gc_si > 0:
249 |         gf = (g_N[i_r] + np.sqrt(g_N[i_r]**2 + 4*gc_si*g_N[i_r])) / 2
250 |         gm = np.sqrt(gc_si * g_N[i_r])
251 |         corrections_all[i_r] = gf / gm
252 | mean_correction = np.mean(corrections_all[g_N > 0])
253 |
254 | results.append({
255 |     'galaxy':      gname,
256 |     'gc_over_a0': gc_a0,
257 |     'log_gc':     np.log10(gc_a0),
258 |     'vflat':     vflat,
259 |     'Yd':        ud,
260 |     'hR_kpc':    hR_kpc,
261 |     'log_hR':    np.log10(hR_kpc),
262 |     'log_vflat': np.log10(vflat),
263 |     'GS0_proxy': GS0_proxy,
264 |     'log_GS0':   np.log10(GS0_proxy/a0),
265 |     'x_outer':   x_outer,
266 |     'log_x_outer': np.log10(x_outer) if x_outer > 0 else np.nan,
267 |     'x_peak':    x_peak,
268 |     'g_N_outer': g_N_outer,
269 |     'g_N_peak':  g_N_peak,
270 |     'correction': correction,
271 |     'log_correction': np.log10(correction) if np.isfinite(correction) and correction>0 else np.nan,
272 |     'mean_correction': mean_correction,
273 |     'GS0_corrected': GS0_corrected,
274 |     'log_GS0_corr': np.log10(GS0_corrected/a0) if np.isfinite(GS0_corrected) and GS0_corrected>0 else np.nan,
275 |     'r_outer_kpc': rad[-1],
276 |     'r_outer_over_hR': rad[-1] / hR_kpc if hR_kpc > 0 else np.nan,
277 | })
278 |
279 | N = len(results)
280 | print(f' Processed: {N} galaxies')
281 | if N < 10:
282 |     print('[ERROR] Too few galaxies.');
```

```

283 | sys.exit(1)
284 | # =====
285 | # Arrays
286 | # =====
287 | log_gc     = np.array([r['log_gc'] for r in results])
288 | log_hR    = np.array([r['log_hR'] for r in results])
289 | log_vflat = np.array([r['log_vflat'] for r in results])
290 | log_GS0   = np.array([r['log_GS0'] for r in results])
291 | x_outer   = np.array([r['x_outer'] for r in results])
292 | log_x_outer = np.array([r['log_x_outer'] for r in results])
293 | correction = np.array([r['correction'] for r in results])
294 | log_corr   = np.array([r['log_correction'] for r in results])
295 | log_GS0_c  = np.array([r['log_GS0_corr'] for r in results])
296 | r_over_hR = np.array([r['r_outer_over_hR'] for r in results])
297 |
298 | # =====
299 | # Analysis
300 | # =====
301 | print(f'#{N} + ' = *70)
302 | print('N-1 Layer 2: MOND transition correction analysis')
303 | print(' = *70)
304 |
305 | # --- A: x_outer distribution ---
306 | valid = np.isfinite(x_outer) & (x_outer > 0)
307 | print(f'#{N}[A] x_outer = g_N(r_outer)/gc distribution (N={np.sum(valid)})')
308 | print(f'   median = {np.nanmedian(x_outer[valid]):.4f}')
309 | print(f'   IQR = [{np.nanpercentile(x_outer[valid], 25):.4f}, '
310 |       f' {np.nanpercentile(x_outer[valid], 75):.4f}']')
311 | print(f'   range = [{np.nanmin(x_outer[valid]):.4f}, {np.nanmax(x_outer[valid]):.4f}']')
312 | print(f'   fraction x_outer > 0.1 (significant transition): '

```

```

313 |     f' {np.sum(x_outer[valid]>0.1)/np.sum(valid):.1%}'
314 | print(f'     fraction x_outer > 0.5 (strong transition): '
315 |     f' {np.sum(x_outer[valid]>0.5)/np.sum(valid):.1%}'
316 |
317 | # --- B: x_outer vs hR correlation ---
318 | mask_b = np.isfinite(log_x_outer) & np.isfinite(log_hR)
319 | rho_xh, p_xh = stats.spearmanr(log_x_outer[mask_b], log_hR[mask_b])
320 | print(f'#{n[B]} x_outer vs hR correlation')
321 | print(f'     Spearman rho(log x_outer, log hR) = {rho_xh:.4f}, p = {p_xh:.2e}')
322 | print(f'     Expected: negative (compact galaxies have larger x_outer)')
323 |
324 | slope_xh, int_xh, r_xh, _, se_xh = stats.linregress(
325 |     log_hR[mask_b], log_x_outer[mask_b])
326 | print(f'     Linear: log(x_outer) = {slope_xh:.3f}*log(hR) + {int_xh:.3f}, '
327 |     f' R^2={r_xh**2:.3f}')
328 |
329 | # --- C: correction factor distribution ---
330 | valid_c = np.isfinite(correction)
331 | print(f'#{n[C]} Correction factor g_obs_full / g_obs_MOND (N={np.sum(valid_c)})')
332 | print(f'     median = {np.nanmedian(correction[valid_c]):.4f}')
333 | print(f'     IQR = [{np.nanpercentile(correction[valid_c],25):.4f}, '
334 |     f' {np.nanpercentile(correction[valid_c],75):.4f}']')
335 | print(f'     range = [{np.nanmin(correction[valid_c]):.4f}, '
336 |     f' {np.nanmax(correction[valid_c]):.4f}']')
337 |
338 | # --- D: Does correction correlate with hR? ---
339 | mask_d = np.isfinite(log_corr) & np.isfinite(log_hR)
340 | rho_ch, p_ch = stats.spearmanr(log_corr[mask_d], log_hR[mask_d])
341 | print(f'#{n[D]} Correction vs hR')
342 | print(f'     Spearman rho(log correction, log hR) = {rho_ch:.4f}, p = {p_ch:.2e}')
343 |
344 | # --- E: Original proxy regression (baseline) ---
345 | mask_e = np.isfinite(log_GS0) & np.isfinite(log_gc)
346 | slope_0, int_0, r_0, _, se_0 = stats.linregress(log_GS0[mask_e], log_gc[mask_e])
347 | print(f'#{n[E]} Baseline: log(gc) vs log(proxy/a0)')
348 | print(f'     slope = {slope_0:.4f} +/- {se_0:.4f}, R^2 = {r_0**2:.4f}')
349 |
350 | # --- F: Corrected proxy regression ---
351 | mask_f = np.isfinite(log_GS0_c) & np.isfinite(log_gc)
352 | if np.sum(mask_f) > 10:
353 |     slope_c, int_c, r_c, _, se_c = stats.linregress(log_GS0_c[mask_f], log_gc[mask_f])
354 |     print(f'#{n[F]} Corrected proxy: log(gc) vs log(proxy_corrected/a0)')
355 |     print(f'     slope = {slope_c:.4f} +/- {se_c:.4f}, R^2 = {r_c**2:.4f}')
356 |     print(f'     (Compare with baseline R^2 = {r_0**2:.4f})')
357 | else:
358 |     slope_c, r_c, se_c = np.nan, np.nan, np.nan
359 |     print(f'#{n[F]} Too few valid corrected points ({np.sum(mask_f)})')
360 |
361 | # --- G: Multivariate with correction ---
362 | # log(gc) = a*log(vflat) + b*log(hR) + c -- original
363 | # log(gc) = a*log(vflat) + b*log(hR) + d*log(correction) + c -- with correction
364 | mask_g = (np.isfinite(log_gc) & np.isfinite(log_vflat) &
365 |     np.isfinite(log_hR) & np.isfinite(log_corr))
366 |
367 | print(f'#{n[G]} Multivariate analysis (N={np.sum(mask_g)})')
368 |
369 | # G1: without correction (reproduce V-1b)
370 | X1 = np.column_stack([log_vflat[mask_g], log_hR[mask_g], np.ones(mask_g.sum())])
371 | y = log_gc[mask_g]
372 | beta1, res1, _, _ = np.linalg.lstsq(X1, y, rcond=None)
373 | ss_res1 = np.sum((y - X1 @ beta1)**2)
374 | ss_tot = np.sum((y - y.mean())**2)
375 | r2_1 = 1 - ss_res1/ss_tot
376 | mse1 = ss_res1 / (len(y)-3)
377 | se1 = np.sqrt(np.diag(mse1 * np.linalg.inv(X1.T @ X1)))
378 |
379 | print(f' G1: log(gc) = {beta1[0]:.4f}*log(vflat) + {beta1[1]:.4f}*log(hR) + {beta1[2]:.4f}')
380 | print(f'     SE:     {se1[0]:.4f}           {se1[1]:.4f}')
381 | print(f'     R^2 = {r2_1:.4f}')
382 |
383 | # G2: with correction
384 | X2 = np.column_stack([log_vflat[mask_g], log_hR[mask_g], log_corr[mask_g],
385 |     np.ones(mask_g.sum())])
386 | beta2, res2, _, _ = np.linalg.lstsq(X2, y, rcond=None)
387 | ss_res2 = np.sum((y - X2 @ beta2)**2)
388 | r2_2 = 1 - ss_res2/ss_tot
389 | mse2 = ss_res2 / (len(y)-4)
390 | se2 = np.sqrt(np.diag(mse2 * np.linalg.inv(X2.T @ X2)))
391 |
392 | print(f'#{n G2}: log(gc) = {beta2[0]:.4f}*log(vflat) + {beta2[1]:.4f}*log(hR) '
393 |     f' + {beta2[2]:.4f}*log(corr) + {beta2[3]:.4f}')
394 | print(f'     SE:     {se2[0]:.4f}           {se2[1]:.4f} '
395 |     f' {se2[2]:.4f}')
396 | print(f'     R^2 = {r2_2:.4f} (vs {r2_1:.4f} without correction)')
397 |
398 | # Is correction term significant?
399 | t_corr = abs(beta2[2]) / se2[2]
400 | p_corr = 2*(1-stats.t.cdf(t_corr, df=len(y)-4))

```

```

401 | print(f'      p(correction coeff = 0) = {p_corr:.4f}')
402 |
403 | # Did hR coefficient change?
404 | print(f' %n hR coefficient comparison:')
405 | print(f'      Without correction: {beta[1]:.4f} +/- {se1[1]:.4f}')
406 | print(f'      With correction:   {beta2[1]:.4f} +/- {se2[1]:.4f}')
407 | delta_hR = abs(beta2[1]) - abs(beta1[1])
408 | print(f'      |hR coeff| change: {delta_hR:+.4f}')
409 | if abs(beta2[1]) < abs(beta1[1]) * 0.5:
410 |     print(f'      >>> hR effect ABSORBED by correction (>50% reduction)')
411 | elif abs(beta2[1]) < abs(beta1[1]) * 0.8:
412 |     print(f'      >>> hR effect PARTIALLY absorbed ({(1-abs(beta2[1])/abs(beta1[1]))*100:.0f}% reduction)')
413 | else:
414 |     print(f'      >>> hR effect NOT absorbed by correction')
415 |
416 | # --- H: Partial correlations with correction controlled ---
417 | print(f' %n[H] Partial correlations controlling for correction')
418 |
419 | # gc vs hR | vflat, correction
420 | X_ctrl = np.column_stack([log_vflat[mask_g], log_corr[mask_g]])
421 | y_gc = log_gc[mask_g]
422 | y_hR = log_hR[mask_g]
423 |
424 | # Residualize gc and hR against (vflat, correction)
425 | beta_gc = np.linalg.lstsq(np.column_stack([X_ctrl, np.ones(len(X_ctrl))]),
426 |                          y_gc, rcond=None)[0]
427 | res_gc = y_gc - np.column_stack([X_ctrl, np.ones(len(X_ctrl))]) @ beta_gc
428 |
429 | beta_hR = np.linalg.lstsq(np.column_stack([X_ctrl, np.ones(len(X_ctrl))]),
430 |                          y_hR, rcond=None)[0]
431 | res_hR = y_hR - np.column_stack([X_ctrl, np.ones(len(X_ctrl))]) @ beta_hR
432 |
433 | rho_partial_new, p_partial_new = stats.spearmanr(res_gc, res_hR)
434 | print(f' gc vs hR | (vflat, correction): rho = {rho_partial_new:.4f}, p = {p_partial_new:.4e}')
435 |
436 | # Original partial for comparison
437 | X_ctrl0 = log_vflat[mask_g].reshape(-1,1)
438 | beta_gc0 = np.linalg.lstsq(np.column_stack([X_ctrl0, np.ones(len(X_ctrl0))]),
439 |                          y_gc, rcond=None)[0]
440 | res_gc0 = y_gc - np.column_stack([X_ctrl0, np.ones(len(X_ctrl0))]) @ beta_gc0
441 | beta_hR0 = np.linalg.lstsq(np.column_stack([X_ctrl0, np.ones(len(X_ctrl0))]),
442 |                          y_hR, rcond=None)[0]
443 | res_hR0 = y_hR - np.column_stack([X_ctrl0, np.ones(len(X_ctrl0))]) @ beta_hR0
444 | rho_partial_orig, p_partial_orig = stats.spearmanr(res_gc0, res_hR0)
445 |
446 | print(f' gc vs hR | vflat only (original): rho = {rho_partial_orig:.4f}, p = {p_partial_orig:.4e}')
447 | print(f' Reduction: {rho_partial_orig:.4f} -> {rho_partial_new:.4f} '
448 |       f'({(1-abs(rho_partial_new)/abs(rho_partial_orig))*100:.1f}% )')
449 |
450 | # =====
451 | # Figures
452 | # =====
453 | print(f' %n[3] Generating figures...')
454 |
455 | fig, axes = plt.subplots(2, 3, figsize=(17, 11))
456 |
457 | # A: x_outer distribution
458 | ax = axes[0, 0]
459 | xo_valid = x_outer[np.isfinite(x_outer) & (x_outer > 0)]
460 | ax.hist(np.log10(xo_valid), bins=30, color='steelblue', alpha=0.7, edgecolor='white')
461 | ax.axvline(np.log10(0.1), color='orange', ls='--', lw=2, label='x=0.1 (10% transition)')
462 | ax.axvline(np.log10(0.5), color='red', ls='--', lw=2, label='x=0.5 (strong transition)')
463 | ax.set_xlabel('log(x_outer) = log(g_N(r_outer)/gc)')
464 | ax.set_ylabel('Count')
465 | ax.set_title('A: MOND departure parameter')
466 | ax.legend(fontsize=8)
467 |
468 | # B: x_outer vs hR
469 | ax = axes[0, 1]
470 | mb = np.isfinite(log_x_outer) & np.isfinite(log_hR)
471 | ax.scatter(log_hR[mb], log_x_outer[mb], s=10, alpha=0.5, c='steelblue', edgecolors='none')
472 | xf = np.linspace(log_hR[mb].min(), log_hR[mb].max(), 100)
473 | ax.plot(xf, slope_xh*xf + int_xh, 'r-', lw=2,
474 |        label=f' rho={rho_xh:.3f}, p={p_xh:.1e}')
475 | ax.set_xlabel('log(hR / kpc)')
476 | ax.set_ylabel('log(x_outer)')
477 | ax.set_title('B: x_outer vs hR')
478 | ax.legend(fontsize=8)
479 | ax.grid(True, alpha=0.3)
480 |
481 | # C: correction factor vs hR
482 | ax = axes[0, 2]
483 | md = np.isfinite(log_corr) & np.isfinite(log_hR)
484 | ax.scatter(log_hR[md], log_corr[md], s=10, alpha=0.5, c='darkorange', edgecolors='none')
485 | ax.set_xlabel('log(hR / kpc)')
486 | ax.set_ylabel('log(correction factor)')
487 | ax.set_title(f'C: Correction vs hR (rho={rho_ch:.3f})')
488 | ax.grid(True, alpha=0.3)

```

```

489 |
490 | # D: Original vs corrected proxy
491 | ax = axes[1, 0]
492 | mf = np.isfinite(log_GS0) & np.isfinite(log_gc)
493 | ax.scatter(log_GS0[mf], log_gc[mf], s=10, alpha=0.3, c='steelblue',
494 |           edgecolors='none', label=f'Original R^2={r_0**2:.3f}')
495 | mfc = np.isfinite(log_GS0_c) & np.isfinite(log_gc)
496 | if np.sum(mfc) > 5:
497 |     ax.scatter(log_GS0_c[mfc], log_gc[mfc], s=10, alpha=0.3, c='red',
498 |             edgecolors='none', label=f'Corrected R^2={r_c**2:.3f}')
499 | ax.set_xlabel('log(proxy / a0)')
500 | ax.set_ylabel('log(gc / a0)')
501 | ax.set_title('D: Original vs corrected proxy')
502 | ax.legend(fontsize=8)
503 | ax.grid(True, alpha=0.3)
504 |
505 | # E: hR partial correlation before/after
506 | ax = axes[1, 1]
507 | ax.scatter(res_hR0, res_gc0, s=10, alpha=0.3, c='steelblue', edgecolors='none',
508 |           label=f'Original: rho={rho_partial_orig:.3f}')
509 | ax.scatter(res_hR, res_gc, s=10, alpha=0.3, c='red', edgecolors='none',
510 |           label=f'+ correction: rho={rho_partial_new:.3f}')
511 | ax.set_xlabel('hR residual (after controlling)')
512 | ax.set_ylabel('gc residual (after controlling)')
513 | ax.set_title('E: hR partial correlation')
514 | ax.legend(fontsize=8)
515 | ax.grid(True, alpha=0.3)
516 |
517 | # F: R^2 comparison
518 | ax = axes[1, 2]
519 | labels = ['vflat+hR%N(original)', 'vflat+hR%N+correction', 'proxy%Noriginal',
520 |          'proxy%Ncorrected']
521 | r2s = [r2_1, r2_2, r_0**2, r_c**2 if np.isfinite(r_c) else 0]
522 | cols = ['steelblue', 'darkorange', 'green', 'red']
523 | ax.bar(range(len(labels)), r2s, color=cols, alpha=0.7, edgecolor='black', linewidth=0.5)
524 | ax.set_xticks(range(len(labels)))
525 | ax.set_xticklabels(labels, fontsize=8)
526 | ax.set_ylabel('R^2')
527 | ax.set_title('F: Model comparison')
528 | for i, v in enumerate(r2s):
529 |     ax.text(i, v+0.01, f'{v:.3f}', ha='center', fontsize=9)
530 | ax.grid(True, alpha=0.3, axis='y')
531 |
532 | fig.suptitle(f'N-1 Layer 2: MOND transition correction (N={N})', fontsize=14, y=1.01)
533 | fig.tight_layout()
534 | fig.savefig(os.path.join(BASE, 'fig_N1_layer2.png'), dpi=150)
535 | print(' -> fig_N1_layer2.png')
536 |
537 | # =====
538 | # Save CSV
539 | # =====
540 | outcsv = os.path.join(BASE, 'N1_layer2_results.csv')
541 | cols_out = ['galaxy', 'gc_over_a0', 'log_gc', 'vflat', 'hR_kpc', 'log_hR', 'log_vflat',
542 |            'x_outer', 'log_x_outer', 'correction', 'log_correction',
543 |            'GS0_proxy', 'log_GS0', 'GS0_corrected', 'log_GS0_corr',
544 |            'r_outer_kpc', 'r_outer_over_hR']
545 | with open(outcsv, 'w', encoding='utf-8') as f:
546 |     f.write(','.join(cols_out)+'\n')
547 |     for r in results:
548 |         f.write(','.join(str(r.get(c, '')) for c in cols_out)+'\n')
549 | print(f' -> {outcsv}')
550 |
551 | # =====
552 | # Verdict
553 | # =====
554 | print(f'\n' + '='*70)
555 | print('VERDICT')
556 | print('='*70)
557 |
558 | print(f'\n x_outer median: {np.nanmedian(x_outer[valid]):.4f}')
559 | if np.nanmedian(x_outer[valid]) < 0.05:
560 |     print(' >>> Most galaxies are deep in MOND regime (x_outer << 1)')
561 |     print(' >>> Transition correction is negligible for most galaxies')
562 |     print(' >>> hR partial correlation is NOT explained by transition effect')
563 | elif np.nanmedian(x_outer[valid]) < 0.3:
564 |     print(' >>> Moderate MOND departure for typical galaxies')
565 |     print(' >>> Transition correction may partially explain hR effect')
566 | else:
567 |     print(' >>> Significant MOND departure - transition correction is important')
568 |
569 | print(f'\n hR partial correlation:')
570 | print(f'   Original (gc vs hR | vflat):           rho = {rho_partial_orig:.4f}')
571 | print(f'   After correction (gc vs hR | vflat,corr): rho = {rho_partial_new:.4f}')
572 | reduction = (1-abs(rho_partial_new)/abs(rho_partial_orig))*100
573 | print(f'   Reduction: {reduction:.1f}%')
574 |
575 | if reduction > 70:
576 |     print(' >>> hR effect EXPLAINED by MOND transition correction')

```

```
577 |     print(' >>> N-1 Layer 2: RESOLVED')
578 | elif reduction > 30:
579 |     print(' >>> hR effect PARTIALLY explained by transition correction')
580 |     print(' >>> N-1 Layer 2: PARTIAL')
581 | else:
582 |     print(' >>> hR effect NOT explained by transition correction')
583 |     print(' >>> N-1 Layer 2: hR contribution is INDEPENDENT of MOND transition')
584 |
585 | print(f'#{n} Correction coefficient in multivariate:')
586 | print(f'   coeff = {beta2[2]:.4f} +/- {se2[2]:.4f}, p = {p_corr:.4f}')
587 | if p_corr < 0.05:
588 |     print(' >>> Correction is statistically significant in multivariate model')
589 | else:
590 |     print(' >>> Correction is NOT significant in multivariate model')
591 |
592 | print(f'#{n}[DONE]')
```

5. [N-1 Layer 2b] 循環性チェック + Yd-hR 隠れた相関

項目	内容
ファイル名	sparc_N1_layer2b.py
行数	654
検証ID	N-1 Layer 2b

解析目的

Layer 2 の R^2 改善が循環アーティファクトかを検証。gc-free correction (gc \rightarrow a0 固定) で循環を除去。Yd と hR の隠れた相関も同時検証。

主要テスト

- [*] Yd vs hR の相関 (隠れた共変動の有無)
- [*] corr_gc (循環) vs corr_a0 (gc-free) の gc との相関比較
- [*] gc-free correction での多変量 R^2
- [*] hR 偏相関: vflat+Yd+corr_a0 統制後の残存値
- [*] 4モデル AIC 比較

結論

Layer 2 の改善は循環 (corr_gc vs gc: rho=-0.756)。gc-free では Delta- R^2 =+0.014。hR 偏相関は全変数統制後も rho=-0.312。Yd は独立予測力 (dAIC=-16.5)。

出力ファイル

fig_N1_layer2b.png, N1_layer2b_results.csv

スクリプト全文

```
1 | # -*- coding: utf-8 -*-
2 | """
3 | N-1 Layer 2b: 循環性チェック + Yd-hR 隠れた相関
4 | =====
5 | 検証1: Yd vs hR の相関 -- hR偏相関がYd変動の代理か
6 | 検証2: gc非依存correction (gc->a0固定) で循環を除去した純粋な遷移効果
7 |
8 | 実行: uv run --with scipy --with matplotlib python sparc_N1_layer2b.py
9 |
10 | 必要ファイル:
11 | - Rotmod_LTG/*.dat
12 | - phase1/sparc_results.csv
13 | - TA3_gc_independent.csv
14 |
15 | 著者: 坂口 忍 (坂口製麺所)
16 | 日付: 2026年4月
17 | """
18 | import os, sys, glob, warnings
19 | import numpy as np
20 | from scipy import stats
21 |
22 | import matplotlib
23 | matplotlib.use('Agg')
24 | import matplotlib.pyplot as plt
25 | from matplotlib import font_manager as _fm
26 | for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
27 |            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
28 |            r'C:\Windows\Fonts\msgothic.ttc']:
29 |     try: _fm.fontManager.addfont(_fp)
30 |     except: pass
31 | for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
32 |     try:
33 |         plt.rcParams['font.family'] = fontname
34 |         break
35 |     except: continue
36 | plt.rcParams['axes.unicode_minus'] = False
37 | warnings.filterwarnings('ignore')
38 |
39 | a0 = 1.2e-10
40 | G_SI = 6.674e-11
41 | kpc_m = 3.0857e19
42 |
43 | BASE = os.path.dirname(os.path.abspath(__file__))
44 | ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
45 | PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
```

```

46 | TA3 = os.path.join(BASE, 'TA3_gc_independent.csv')
47 |
48 | for p, label in [(ROTMOD, 'Rotmod_LTG'), (PHASE1, 'sparc_results.csv'),
49 |                 (TA3, 'TA3_gc_independent.csv')]:
50 |     if not os.path.exists(p):
51 |         print(f'[ERROR] {label} not found: {p}'); sys.exit(1)
52 |
53 | # =====
54 | # Loaders (same as previous)
55 | # =====
56 | def load_csv(path):
57 |     with open(path, 'r', encoding='utf-8-sig') as f:
58 |         header = f.readline().strip()
59 |         sep = ',' if ',' in header else None
60 |         data = {}
61 |         with open(path, 'r', encoding='utf-8-sig') as f:
62 |             cols = [c.strip() for c in f.readline().strip().split(sep)]
63 |             rows = []
64 |             for line in f:
65 |                 line = line.strip()
66 |                 if not line: continue
67 |                 rows.append([p.strip() for p in line.split(sep)])
68 |             for i, col in enumerate(cols):
69 |                 vals = []
70 |                 for row in rows:
71 |                     if i < len(row):
72 |                         try: vals.append(float(row[i]))
73 |                         except: vals.append(row[i])
74 |                     else: vals.append(np.nan)
75 |                 data[col] = vals
76 |             return data
77 |
78 | def find_name_col(data):
79 |     for c in ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY']:
80 |         if c in data: return c
81 |     for k, v in data.items():
82 |         if isinstance(v[0], str): return k
83 |     return list(data.keys())[0]
84 |
85 | def get_key(info, candidates, default=None):
86 |     for c in candidates:
87 |         if c in info:
88 |             try: return float(info[c])
89 |             except: return info[c]
90 |     return default
91 |
92 | def load_rotmod(filepath):
93 |     cols = [[] for _ in range(8)]
94 |     with open(filepath, 'r') as f:
95 |         for line in f:
96 |             line = line.strip()
97 |             if not line or line.startswith('#'): continue
98 |             parts = line.split()
99 |             if len(parts) < 6: continue
100 |            try:
101 |                for j in range(min(len(parts), 8)):
102 |                    cols[j].append(float(parts[j]))
103 |                for j in range(len(parts), 8):
104 |                    cols[j].append(0.0)
105 |            except ValueError: continue
106 |            return tuple(np.array(c) for c in cols)
107 |
108 | # =====
109 | # Load data
110 | # =====
111 | print('[1] Loading data...')
112 | phase1 = load_csv(PHASE1)
113 | ta3 = load_csv(TA3)
114 | p1_nc = find_name_col(phase1)
115 | ta3_nc = find_name_col(ta3)
116 |
117 | galaxy_info = {}
118 | for i, name in enumerate(phase1[p1_nc]):
119 |     name = str(name).strip()
120 |     info = {}
121 |     for k in phase1:
122 |         if k == p1_nc: continue
123 |         try: info[k] = float(phase1[k][i])
124 |         except: info[k] = phase1[k][i]
125 |     galaxy_info[name] = info
126 |
127 | for i, name in enumerate(ta3[ta3_nc]):
128 |     name = str(name).strip()
129 |     if name in galaxy_info:
130 |         for k in ta3:
131 |             if k == ta3_nc: continue
132 |             try: galaxy_info[name][k] = float(ta3[k][i])
133 |             except: galaxy_info[name][k] = ta3[k][i]

```

```

134 |
135 | # =====
136 | # Process galaxies
137 | # =====
138 | print('[2] Processing galaxies...')
139 | results = []
140 | rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
141 |
142 | for fpath in rotmod_files:
143 |     gname = os.path.splitext(os.path.basename(fpath))[0].replace('_rotmod', '').strip()
144 |     info = None
145 |     for key in [gname, gname.upper(), gname.lower()]:
146 |         if key in galaxy_info:
147 |             info = galaxy_info[key]; break
148 |     if info is None: continue
149 |
150 |     ud = get_key(info, ['upsilon_d', 'Upsilon_d', 'Ud', 'ud', 'Yd'])
151 |     gc_a0 = get_key(info, ['gc_over_a0', 'gc/a0', 'gc_ratio'])
152 |     vflat = get_key(info, ['vflat', 'Vflat', 'v_flat'])
153 |
154 |     if ud is None or gc_a0 is None or vflat is None: continue
155 |     if np.isnan(ud) or np.isnan(gc_a0) or gc_a0<=0 or vflat<=0 or ud<=0: continue
156 |
157 |     try:
158 |         rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
159 |     except: continue
160 |     if len(rad) < 3: continue
161 |
162 |     vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
163 |     rpk_idx = np.argmax(vds)
164 |     rpk = rad[rpk_idx]
165 |     if rpk < 0.01 or rpk >= rad.max()*0.9: continue
166 |     hR_kpc = rpk / 2.15
167 |     hR_m = hR_kpc * kpc_m
168 |
169 |     # g_N at outer points
170 |     r_m = rad * kpc_m
171 |     vbar2 = np.abs(vgas**2 + ud * np.sign(vdisk)*vdisk**2
172 |                 + 0.7 * np.sign(vbul)*vbul**2)
173 |     g_N = np.zeros_like(r_m)
174 |     mask_r = r_m > 0
175 |     g_N[mask_r] = vbar2[mask_r] * 1e6 / r_m[mask_r]
176 |
177 |     gc_si = gc_a0 * a0
178 |     vflat_ms = vflat * 1e3
179 |     GS0_proxy = vflat_ms**2 / hR_m
180 |
181 |     # outer g_N (average of last 3 points)
182 |     n_out = min(3, len(rad))
183 |     g_N_outer = np.mean(g_N[-n_out:])
184 |
185 |     # --- Correction WITH gc (original, potentially circular) ---
186 |     if g_N_outer > 0 and gc_si > 0:
187 |         gf = (g_N_outer + np.sqrt(g_N_outer**2 + 4*gc_si*g_N_outer)) / 2
188 |         gm = np.sqrt(gc_si * g_N_outer)
189 |         corr_gc = gf / gm
190 |     else:
191 |         corr_gc = np.nan
192 |
193 |     # --- Correction WITHOUT gc (gc -> a0, breaks circularity) ---
194 |     if g_N_outer > 0:
195 |         gf_a0 = (g_N_outer + np.sqrt(g_N_outer**2 + 4*a0*g_N_outer)) / 2
196 |         gm_a0 = np.sqrt(a0 * g_N_outer)
197 |         corr_a0 = gf_a0 / gm_a0
198 |     else:
199 |         corr_a0 = np.nan
200 |
201 |     # --- Per-point mean corrections (both versions) ---
202 |     corrs_gc_all = []
203 |     corrs_a0_all = []
204 |     for i_r in range(len(g_N)):
205 |         if g_N[i_r] > 0:
206 |             gf1 = (g_N[i_r] + np.sqrt(g_N[i_r]**2 + 4*gc_si*g_N[i_r])) / 2
207 |             gm1 = np.sqrt(gc_si * g_N[i_r])
208 |             corrs_gc_all.append(gf1/gm1)
209 |
210 |             gf2 = (g_N[i_r] + np.sqrt(g_N[i_r]**2 + 4*a0*g_N[i_r])) / 2
211 |             gm2 = np.sqrt(a0 * g_N[i_r])
212 |             corrs_a0_all.append(gf2/gm2)
213 |
214 |     mean_corr_gc = np.mean(corrs_gc_all) if corrs_gc_all else np.nan
215 |     mean_corr_a0 = np.mean(corrs_a0_all) if corrs_a0_all else np.nan
216 |
217 |     results.append({
218 |         'galaxy': gname,
219 |         'gc_a0': gc_a0,
220 |         'log_gc': np.log10(gc_a0),
221 |         'vflat': vflat,

```

```

222 |         'Yd':          ud,
223 |         'log_Yd':     np.log10(ud),
224 |         'hR_kpc':    hR_kpc,
225 |         'log_hR':    np.log10(hR_kpc),
226 |         'log_vflat': np.log10(vflat),
227 |         'GS0_proxy': GS0_proxy,
228 |         'log_GS0':   np.log10(GS0_proxy/a0),
229 |         'g_N_outer': g_N_outer,
230 |         'corr_gc':   corr_gc,
231 |         'log_corr_gc': np.log10(corr_gc) if np.isfinite(corr_gc) and corr_gc>0 else np.nan,
232 |         'corr_a0':   corr_a0,
233 |         'log_corr_a0': np.log10(corr_a0) if np.isfinite(corr_a0) and corr_a0>0 else np.nan,
234 |         'mean_corr_gc': mean_corr_gc,
235 |         'mean_corr_a0': mean_corr_a0,
236 |     })
237 |
238 | N = len(results)
239 | print(f' Processed: {N} galaxies')
240 | if N < 10:
241 |     print(' [ERROR] Too few. '); sys.exit(1)
242 |
243 | # =====
244 | # Arrays
245 | # =====
246 | log_gc      = np.array([r['log_gc'] for r in results])
247 | log_hR     = np.array([r['log_hR'] for r in results])
248 | log_vflat  = np.array([r['log_vflat'] for r in results])
249 | log_GS0    = np.array([r['log_GS0'] for r in results])
250 | log_Yd     = np.array([r['log_Yd'] for r in results])
251 | Yd_arr     = np.array([r['Yd'] for r in results])
252 | log_corr_gc = np.array([r['log_corr_gc'] for r in results])
253 | log_corr_a0 = np.array([r['log_corr_a0'] for r in results])
254 |
255 | # =====
256 | # PART 1: Yd vs hR correlation
257 | # =====
258 | print(f' %n' + '='*70)
259 | print(' PART 1: Yd vs hR hidden correlation')
260 | print('='*70)
261 |
262 | mask1 = np.isfinite(log_Yd) & np.isfinite(log_hR)
263 | rho_Yd_hR, p_Yd_hR = stats.spearmanr(log_Yd[mask1], log_hR[mask1])
264 | sL_yh, int_yh, r_yh, p_yh, se_yh = stats.linregress(log_hR[mask1], log_Yd[mask1])
265 | print(f' %n [1a] Yd vs hR:')
266 | print(f' Spearman rho = {rho_Yd_hR:.4f}, p = {p_Yd_hR:.2e}')
267 | print(f' Linear: log(Yd) = {sL_yh:.3f}*log(hR) + {int_yh:.3f}, R^2={r_yh**2:.3f}')
268 |
269 | # Yd vs vflat
270 | mask1v = np.isfinite(log_Yd) & np.isfinite(log_vflat)
271 | rho_Yd_vf, p_Yd_vf = stats.spearmanr(log_Yd[mask1v], log_vflat[mask1v])
272 | print(f' %n [1b] Yd vs vflat:')
273 | print(f' Spearman rho = {rho_Yd_vf:.4f}, p = {p_Yd_vf:.2e}')
274 |
275 | # Yd vs gc
276 | mask1g = np.isfinite(log_Yd) & np.isfinite(log_gc)
277 | rho_Yd_gc, p_Yd_gc = stats.spearmanr(log_Yd[mask1g], log_gc[mask1g])
278 | print(f' %n [1c] Yd vs gc:')
279 | print(f' Spearman rho = {rho_Yd_gc:.4f}, p = {p_Yd_gc:.2e}')
280 |
281 | # Partial: gc vs hR | (vflat, Yd)
282 | mask_p1 = np.isfinite(log_gc) & np.isfinite(log_hR) & np.isfinite(log_vflat) & np.isfinite(log_Yd)
283 | X_ctrl = np.column_stack([log_vflat[mask_p1], log_Yd[mask_p1], np.ones(mask_p1.sum())])
284 | y_gc = log_gc[mask_p1]
285 | y_hR = log_hR[mask_p1]
286 |
287 | b_gc = np.linalg.lstsq(X_ctrl, y_gc, rcond=None)[0]
288 | res_gc = y_gc - X_ctrl @ b_gc
289 | b_hR = np.linalg.lstsq(X_ctrl, y_hR, rcond=None)[0]
290 | res_hR = y_hR - X_ctrl @ b_hR
291 | rho_hR_Yd, p_hR_Yd = stats.spearmanr(res_gc, res_hR)
292 |
293 | # Original partial (vflat only)
294 | X_ctrl0 = np.column_stack([log_vflat[mask_p1], np.ones(mask_p1.sum())])
295 | b_gc0 = np.linalg.lstsq(X_ctrl0, y_gc, rcond=None)[0]
296 | res_gc0 = y_gc - X_ctrl0 @ b_gc0
297 | b_hR0 = np.linalg.lstsq(X_ctrl0, y_hR, rcond=None)[0]
298 | res_hR0 = y_hR - X_ctrl0 @ b_hR0
299 | rho_hR_orig, p_hR_orig = stats.spearmanr(res_gc0, res_hR0)
300 |
301 | print(f' %n [1d] hR partial correlations:')
302 | print(f' gc vs hR | vflat: rho = {rho_hR_orig:.4f}, p = {p_hR_orig:.4e}')
303 | print(f' gc vs hR | (vflat, Yd): rho = {rho_hR_Yd:.4f}, p = {p_hR_Yd:.4e}')
304 | red_Yd = (1 - abs(rho_hR_Yd)/abs(rho_hR_orig))*100 if abs(rho_hR_orig)>0 else 0
305 | print(f' Reduction by adding Yd: {red_Yd:.1f}%')
306 |
307 | # Multivariate: gc = a*vflat + b*hR + c*Yd + d
308 | X_m3 = np.column_stack([log_vflat[mask_p1], log_hR[mask_p1], log_Yd[mask_p1],
309 | np.ones(mask_p1.sum())])

```

```

310 | b3, _, _ = np.linalg.lstsq(X_m3, y_gc, rcond=None)
311 | ss3 = np.sum((y_gc - X_m3 @ b3)**2)
312 | ss_tot = np.sum((y_gc - y_gc.mean())**2)
313 | r2_3 = 1 - ss3/ss_tot
314 | mse3 = ss3 / (len(y_gc)-4)
315 | se3 = np.sqrt(np.diag(mse3 * np.linalg.inv(X_m3.T @ X_m3)))
316 |
317 | print(f' #n [1e] Multivariate: vflat + hR + Yd')
318 | print(f'      log(gc) = {b3[0]:.4f}*log(vflat) + {b3[1]:.4f}*log(hR) '
319 |       f' + {b3[2]:.4f}*log(Yd) + {b3[3]:.4f}')
320 | print(f'      SE:      {se3[0]:.4f}          {se3[1]:.4f}
321 |       f' {se3[2]:.4f}')
322 | print(f'      R^2 = {r2_3:.4f}')
323 | t_yd = abs(b3[2]) / se3[2]
324 | p_yd_coeff = 2*(1-stats.t.cdf(t_yd, df=len(y_gc)-4))
325 | print(f'      p(Yd coeff = 0) = {p_yd_coeff:.4f}')
326 |
327 | # Compare with vflat+hR only (same mask)
328 | X_m2 = np.column_stack([log_vflat[mask_p1], log_hR[mask_p1], np.ones(mask_p1.sum())])
329 | b2, _, _ = np.linalg.lstsq(X_m2, y_gc, rcond=None)
330 | ss2 = np.sum((y_gc - X_m2 @ b2)**2)
331 | r2_2 = 1 - ss2/ss_tot
332 | print(f'      Compare: vflat+hR only R^2 = {r2_2:.4f}')
333 | print(f'      Delta R^2 from Yd: +{r2_3 - r2_2:.4f}')
334 |
335 | # =====
336 | # PART 2: gc-free correction (circularity removal)
337 | # =====
338 | print(f' #n ' + ' *70)
339 | print('PART 2: gc-free correction (gc -> a0)')
340 | print(' *70)
341 |
342 | # Compare distributions
343 | valid_c = np.isfinite(log_corr_gc) & np.isfinite(log_corr_a0)
344 | print(f' #n [2a] Correction distributions (N={np.sum(valid_c)}):')
345 | print(f'      With gc: median = {np.nanmedian(np.array([r["corr_gc"] for r in results])[valid_c]):.4f}')
346 | print(f'      With a0: median = {np.nanmedian(np.array([r["corr_a0"] for r in results])[valid_c]):.4f}')
347 |
348 | # Correlation between two corrections
349 | rho_cc, p_cc = stats.spearmanr(log_corr_gc[valid_c], log_corr_a0[valid_c])
350 | print(f' #n [2b] corr_gc vs corr_a0: rho = {rho_cc:.4f}, p = {p_cc:.2e}')
351 |
352 | # corr_a0 vs gc (should be LESS correlated than corr_gc vs gc if circularity matters)
353 | mask_cg = np.isfinite(log_corr_gc) & np.isfinite(log_gc)
354 | rho_cg_gc, p_cg_gc = stats.spearmanr(log_corr_gc[mask_cg], log_gc[mask_cg])
355 | mask_ca = np.isfinite(log_corr_a0) & np.isfinite(log_gc)
356 | rho_ca_gc, p_ca_gc = stats.spearmanr(log_corr_a0[mask_ca], log_gc[mask_ca])
357 | print(f' #n [2c] Circularity check:')
358 | print(f'      corr_gc vs gc: rho = {rho_cg_gc:.4f}, p = {p_cg_gc:.2e} (circular)')
359 | print(f'      corr_a0 vs gc: rho = {rho_ca_gc:.4f}, p = {p_ca_gc:.2e} (gc-free)')
360 |
361 | # Multivariate with gc-free correction
362 | mask_2m = (np.isfinite(log_gc) & np.isfinite(log_vflat) &
363 |           np.isfinite(log_hR) & np.isfinite(log_corr_a0))
364 |
365 | print(f' #n [2d] Multivariate with gc-free correction (N={np.sum(mask_2m)}')
366 |
367 | y2 = log_gc[mask_2m]
368 | ss_tot2 = np.sum((y2 - y2.mean())**2)
369 |
370 | # Without correction
371 | X_nc = np.column_stack([log_vflat[mask_2m], log_hR[mask_2m], np.ones(mask_2m.sum())])
372 | b_nc, _, _ = np.linalg.lstsq(X_nc, y2, rcond=None)
373 | r2_nc = 1 - np.sum((y2 - X_nc @ b_nc)**2)/ss_tot2
374 | mse_nc = np.sum((y2 - X_nc @ b_nc)**2) / (len(y2)-3)
375 | se_nc = np.sqrt(np.diag(mse_nc * np.linalg.inv(X_nc.T @ X_nc)))
376 |
377 | print(f'      G1: vflat+hR: R^2 = {r2_nc:.4f}')
378 | print(f'      vflat^{b_nc[0]:.3f}+/{-}{se_nc[0]:.3f}, hR^{b_nc[1]:.3f}+/{-}{se_nc[1]:.3f}')
379 |
380 | # With corr_gc (circular)
381 | X_cg = np.column_stack([log_vflat[mask_2m], log_hR[mask_2m],
382 |                       log_corr_gc[mask_2m], np.ones(mask_2m.sum())])
383 | b_cg, _, _ = np.linalg.lstsq(X_cg, y2, rcond=None)
384 | r2_cg = 1 - np.sum((y2 - X_cg @ b_cg)**2)/ss_tot2
385 | mse_cg = np.sum((y2 - X_cg @ b_cg)**2) / (len(y2)-4)
386 | se_cg = np.sqrt(np.diag(mse_cg * np.linalg.inv(X_cg.T @ X_cg)))
387 | t_cg = abs(b_cg[2]) / se_cg[2]
388 | p_cg = 2*(1-stats.t.cdf(t_cg, df=len(y2)-4))
389 |
390 | print(f' #n      G2: vflat+hR+corr_gc (circular): R^2 = {r2_cg:.4f}')
391 | print(f'      vflat^{b_cg[0]:.3f}, hR^{b_cg[1]:.3f}, corr_gc^{b_cg[2]:.3f}+/{-}{se_cg[2]:.3f}')
392 | print(f'      p(corr_gc=0) = {p_cg:.4f}')
393 |
394 | # With corr_a0 (gc-free)
395 | X_ca = np.column_stack([log_vflat[mask_2m], log_hR[mask_2m],
396 |                       log_corr_a0[mask_2m], np.ones(mask_2m.sum())])
397 | b_ca, _, _ = np.linalg.lstsq(X_ca, y2, rcond=None)

```

```

398 | r2_ca = 1 - np.sum((y2 - X_ca @ b_ca)**2)/ss_tot2
399 | mse_ca = np.sum((y2 - X_ca @ b_ca)**2) / (len(y2)-4)
400 | se_ca = np.sqrt(np.diag(mse_ca * np.linalg.inv(X_ca.T @ X_ca)))
401 | t_ca = abs(b_ca[2]) / se_ca[2]
402 | p_ca = 2*(1-stats.t.cdf(t_ca, df=len(y2)-4))
403 |
404 | print(f' #n G3: vflat+hR+corr_a0 (gc-free): R^2 = {r2_ca:.4f}')
405 | print(f' vflat^{b_ca[0]:.3f}+/{-se_ca[0]:.3f}, hR^{b_ca[1]:.3f}+/{-se_ca[1]:.3f}, '
406 | f'corr_a0^{b_ca[2]:.3f}+/{-se_ca[2]:.3f}')
407 | print(f' p(corr_a0=0) = {p_ca:.4f}')
408 |
409 | # hR partial with gc-free correction controlled
410 | X_ctrl2 = np.column_stack([log_vflat[mask_2m], log_corr_a0[mask_2m],
411 | np.ones(mask_2m.sum())])
412 | b_gc2 = np.linalg.lstsq(X_ctrl2, y2, rcond=None)[0]
413 | res_gc2 = y2 - X_ctrl2 @ b_gc2
414 | y_hR2 = log_hR[mask_2m]
415 | b_hR2 = np.linalg.lstsq(X_ctrl2, y_hR2, rcond=None)[0]
416 | res_hR2 = y_hR2 - X_ctrl2 @ b_hR2
417 | rho_hR_a0, p_hR_a0 = stats.spearmanr(res_gc2, res_hR2)
418 |
419 | # Original partial (same mask)
420 | X_ctrl0b = np.column_stack([log_vflat[mask_2m], np.ones(mask_2m.sum())])
421 | b_gc0b = np.linalg.lstsq(X_ctrl0b, y2, rcond=None)[0]
422 | res_gc0b = y2 - X_ctrl0b @ b_gc0b
423 | b_hR0b = np.linalg.lstsq(X_ctrl0b, y_hR2, rcond=None)[0]
424 | res_hR0b = y_hR2 - X_ctrl0b @ b_hR0b
425 | rho_hR_0b, p_hR_0b = stats.spearmanr(res_gc0b, res_hR0b)
426 |
427 | print(f' #n [2e] hR partial correlations:')
428 | print(f' gc vs hR | vflat: rho = {rho_hR_0b:.4f}')
429 | print(f' gc vs hR | (vflat, corr_gc): rho = (from Layer2: -0.227)')
430 | print(f' gc vs hR | (vflat, corr_a0): rho = {rho_hR_a0:.4f}, p = {p_hR_a0:.4e}')
431 | red_a0 = (1 - abs(rho_hR_a0)/abs(rho_hR_0b))*100 if abs(rho_hR_0b)>0 else 0
432 | print(f' Reduction (gc-free correction): {red_a0:.1f}%')
433 |
434 | # =====
435 | # PART 3: Full 4-variable model
436 | # =====
437 | print(f' #n + ' + '*70)
438 | print('PART 3: Full model comparison')
439 | print(' ' + '*70)
440 |
441 | mask_all = (np.isfinite(log_gc) & np.isfinite(log_vflat) & np.isfinite(log_hR)
442 | & np.isfinite(log_Yd) & np.isfinite(log_corr_a0))
443 |
444 | y_all = log_gc[mask_all]
445 | ss_all = np.sum((y_all - y_all.mean())**2)
446 |
447 | models = {}
448 |
449 | # M1: vflat + hR
450 | X_1 = np.column_stack([log_vflat[mask_all], log_hR[mask_all], np.ones(mask_all.sum())])
451 | b_1 = np.linalg.lstsq(X_1, y_all, rcond=None)[0]
452 | r2_m1 = 1 - np.sum((y_all - X_1 @ b_1)**2)/ss_all
453 | models['vflat+hR'] = {'r2': r2_m1, 'k': 2, 'beta': b_1}
454 |
455 | # M2: vflat + hR + Yd
456 | X_2 = np.column_stack([log_vflat[mask_all], log_hR[mask_all], log_Yd[mask_all],
457 | np.ones(mask_all.sum())])
458 | b_2 = np.linalg.lstsq(X_2, y_all, rcond=None)[0]
459 | r2_m2 = 1 - np.sum((y_all - X_2 @ b_2)**2)/ss_all
460 | mse_m2 = np.sum((y_all - X_2 @ b_2)**2) / (len(y_all)-4)
461 | se_m2 = np.sqrt(np.diag(mse_m2 * np.linalg.inv(X_2.T @ X_2)))
462 | models['vflat+hR+Yd'] = {'r2': r2_m2, 'k': 3, 'beta': b_2, 'se': se_m2}
463 |
464 | # M3: vflat + hR + corr_a0
465 | X_3 = np.column_stack([log_vflat[mask_all], log_hR[mask_all], log_corr_a0[mask_all],
466 | np.ones(mask_all.sum())])
467 | b_3 = np.linalg.lstsq(X_3, y_all, rcond=None)[0]
468 | r2_m3 = 1 - np.sum((y_all - X_3 @ b_3)**2)/ss_all
469 | mse_m3 = np.sum((y_all - X_3 @ b_3)**2) / (len(y_all)-4)
470 | se_m3 = np.sqrt(np.diag(mse_m3 * np.linalg.inv(X_3.T @ X_3)))
471 | models['vflat+hR+corr_a0'] = {'r2': r2_m3, 'k': 3, 'beta': b_3, 'se': se_m3}
472 |
473 | # M4: vflat + hR + Yd + corr_a0
474 | X_4 = np.column_stack([log_vflat[mask_all], log_hR[mask_all], log_Yd[mask_all],
475 | log_corr_a0[mask_all], np.ones(mask_all.sum())])
476 | b_4 = np.linalg.lstsq(X_4, y_all, rcond=None)[0]
477 | r2_m4 = 1 - np.sum((y_all - X_4 @ b_4)**2)/ss_all
478 | mse_m4 = np.sum((y_all - X_4 @ b_4)**2) / (len(y_all)-5)
479 | se_m4 = np.sqrt(np.diag(mse_m4 * np.linalg.inv(X_4.T @ X_4)))
480 | models['vflat+hR+Yd+corr_a0'] = {'r2': r2_m4, 'k': 4, 'beta': b_4, 'se': se_m4}
481 |
482 | n_all = mask_all.sum()
483 | print(f' #n N = {n_all}')
484 | print(f' #n {"Model":<28s} | {"R^2":>6s} | {"AIC":>8s} | k')
485 | print(' ' + '*55)

```

```

486 | for name, m in models.items():
487 |     k = m['k']
488 |     r2 = m['r2']
489 |     ss_r = (1-r2) * ss_all
490 |     aic = n_all * np.log(ss_r/n_all) + 2*(k+1)
491 |     m['aic'] = aic
492 |     print(f' {name:<28s} | {r2:>6.4f} | {aic:>8.1f} | {k}')
493 |
494 | aic_base = models['vflat+hR']['aic']
495 | print(f' %n dAIC vs baseline (vflat+hR):')
496 | for name, m in models.items():
497 |     print(f' {name:<28s}: dAIC = {m["aic"]-aic_base:>+7.1f}')
498 |
499 | print(f' %n Full model (M4) coefficients:')
500 | print(f' vflat: {b_4[0]:.4f} +/- {se_m4[0]:.4f}')
501 | print(f' hR: {b_4[1]:.4f} +/- {se_m4[1]:.4f}')
502 | print(f' Yd: {b_4[2]:.4f} +/- {se_m4[2]:.4f}')
503 | print(f' corr_a0: {b_4[3]:.4f} +/- {se_m4[3]:.4f}')
504 |
505 | # Final partial: hR after everything
506 | X_fin = np.column_stack([log_vflat[mask_all], log_Yd[mask_all],
507 |                          log_corr_a0[mask_all], np.ones(mask_all.sum())])
508 | b_fin_gc = np.linalg.lstsq(X_fin, y_all, rcond=None)[0]
509 | res_fin_gc = y_all - X_fin @ b_fin_gc
510 | y_hR_all = log_hR[mask_all]
511 | b_fin_hR = np.linalg.lstsq(X_fin, y_hR_all, rcond=None)[0]
512 | res_fin_hR = y_hR_all - X_fin @ b_fin_hR
513 | rho_fin, p_fin = stats.spearmanr(res_fin_gc, res_fin_hR)
514 |
515 | print(f' %n [Final] gc vs hR | (vflat, Yd, corr_a0):')
516 | print(f' rho = {rho_fin:.4f}, p = {p_fin:.4e}')
517 | print(f' Original (| vflat only): rho = {rho_hR_orig:.4f}')
518 | total_red = (1 - abs(rho_fin)/abs(rho_hR_orig))*100 if abs(rho_hR_orig)>0 else 0
519 | print(f' Total reduction: {total_red:.1f}%)
520 |
521 | # =====
522 | # Figures
523 | # =====
524 | print(f' %n[3] Generating figures...')
525 | fig, axes = plt.subplots(2, 3, figsize=(17, 11))
526 |
527 | # 1: Yd vs hR
528 | ax = axes[0, 0]
529 | m = np.isfinite(log_Yd) & np.isfinite(log_hR)
530 | ax.scatter(log_hR[m], log_Yd[m], s=12, alpha=0.5, c='steelblue', edgecolors='none')
531 | xf = np.linspace(log_hR[m].min(), log_hR[m].max(), 100)
532 | ax.plot(xf, s_l_yh*xf + int_yh, 'r-', lw=2, label=f' rho={rho_Yd_hR:.3f}, p={p_Yd_hR:.1e}')
533 | ax.set_xlabel('log(hR / kpc)')
534 | ax.set_ylabel('log(Yd)')
535 | ax.set_title('1: Yd vs hR correlation')
536 | ax.legend(fontsize=8)
537 | ax.grid(True, alpha=0.3)
538 |
539 | # 2: corr_gc vs corr_a0
540 | ax = axes[0, 1]
541 | vc = np.isfinite(log_corr_gc) & np.isfinite(log_corr_a0)
542 | ax.scatter(log_corr_a0[vc], log_corr_gc[vc], s=12, alpha=0.5, c='darkorange', edgecolors='none')
543 | lim = [min(log_corr_a0[vc].min(), log_corr_gc[vc].min()),
544 |        max(log_corr_a0[vc].max(), log_corr_gc[vc].max())]
545 | ax.plot(lim, lim, 'k--', lw=1, label='1:1')
546 | ax.set_xlabel('log(corr_a0) [gc-free]')
547 | ax.set_ylabel('log(corr_gc) [circular]')
548 | ax.set_title(f'2: Two corrections (rho={rho_cc:.3f})')
549 | ax.legend(fontsize=8)
550 | ax.grid(True, alpha=0.3)
551 |
552 | # 3: corr_a0 vs gc (should be weaker)
553 | ax = axes[0, 2]
554 | ax.scatter(log_corr_a0[mask_ca], log_gc[mask_ca], s=12, alpha=0.5, c='green', edgecolors='none')
555 | ax.set_xlabel('log(corr_a0) [gc-free]')
556 | ax.set_ylabel('log(gc/a0)')
557 | ax.set_title(f'3: corr_a0 vs gc (rho={rho_ca_gc:.3f})')
558 | ax.grid(True, alpha=0.3)
559 |
560 | # 4: R^2 comparison
561 | ax = axes[1, 0]
562 | names = list(models.keys())
563 | r2s = [models[n]['r2'] for n in names]
564 | cols = ['steelblue', 'darkorange', 'green', 'red']
565 | ax.bar(range(len(names)), r2s, color=cols[:len(names)], alpha=0.7,
566 |        edgecolor='black', linewidth=0.5)
567 | ax.set_xticks(range(len(names)))
568 | ax.set_xticklabels([n.replace('+','+ %n') for n in names], fontsize=7)
569 | ax.set_ylabel('R^2')
570 | ax.set_title('4: Model comparison')
571 | for i, v in enumerate(r2s):
572 |     ax.text(i, v+0.01, f'{v:.3f}', ha='center', fontsize=8)
573 | ax.grid(True, alpha=0.3, axis='y')

```

```

574 |
575 | # 5: hR partial correlation progression
576 | ax = axes[1, 1]
577 | stages = ['vflat', 'vflat,Yd', 'vflat,corr_a0', 'vflat,Yd,corr_a0']
578 | rhos = [rho_hR_orig, rho_hR_Yd, rho_hR_a0, rho_fin]
579 | barcolors = ['steelblue', 'darkorange', 'green', 'red']
580 | ax.bar(range(len(stages)), [abs(r) for r in rhos], color=barcolors, alpha=0.7,
581 |       edgecolor='black', linewidth=0.5)
582 | ax.set_xticks(range(len(stages)))
583 | ax.set_xticklabels(stages, fontsize=7, rotation=15)
584 | ax.set_ylabel('|rho| (hR partial)')
585 | ax.set_title('5: hR partial reduction')
586 | for i,v in enumerate(rhos):
587 |     ax.text(i, abs(v)+0.01, f'{v:.3f}', ha='center', fontsize=8)
588 | ax.axhline(0.1, color='grey', ls=':', label='negligible threshold')
589 | ax.legend(fontsize=7)
590 | ax.grid(True, alpha=0.3, axis='y')
591 |
592 | # 6: Residual from full model
593 | ax = axes[1, 2]
594 | resid_full = y_all - X_4 @ b_4
595 | ax.hist(resid_full, bins=25, color='purple', alpha=0.7, edgecolor='white')
596 | ax.axvline(0, color='red', ls='--')
597 | ax.set_xlabel('Residual [dex]')
598 | ax.set_ylabel('Count')
599 | ax.set_title('6: Full model residual (sigma={np.std(resid_full):.3f})')
600 |
601 | fig.suptitle('N-1 Layer 2b: Circularity + Yd check (N={N})', fontsize=14, y=1.01)
602 | fig.tight_layout()
603 | fig.savefig(os.path.join(BASE, 'fig_N1_layer2b.png'), dpi=150)
604 | print(' -> fig_N1_layer2b.png')
605 |
606 | # =====
607 | # Save CSV
608 | # =====
609 | outcsv = os.path.join(BASE, 'N1_layer2b_results.csv')
610 | cols_out = ['galaxy', 'gc_a0', 'log_gc', 'vflat', 'Yd', 'log_Yd', 'hR_kpc', 'log_hR',
611 |           'log_vflat', 'log_GS0', 'corr_gc', 'log_corr_gc', 'corr_a0', 'log_corr_a0']
612 | with open(outcsv, 'w', encoding='utf-8') as f:
613 |     f.write(''.join(cols_out)+'\n')
614 |     for r in results:
615 |         f.write(''.join(str(r.get(c,'')) for c in cols_out)+'\n')
616 | print(f' -> {outcsv}')
617 |
618 | # =====
619 | # Final Verdict
620 | # =====
621 | print(' %n' + '='*70)
622 | print(' FINAL VERDICT')
623 | print(' ='*70)
624 |
625 | print(f' %n PART 1: Yd-hR correlation')
626 | if abs(rho_Yd_hR) > 0.3 and p_Yd_hR < 0.01:
627 |     print(f' >>> Yd and hR ARE correlated (rho={rho_Yd_hR:.3f})')
628 |     print(f' >>> hR partial may partly reflect Yd variation')
629 | else:
630 |     print(f' >>> Yd and hR are NOT strongly correlated (rho={rho_Yd_hR:.3f})')
631 |     print(f' >>> hR effect is INDEPENDENT of Yd')
632 |
633 | print(f' %n PART 2: Circularity')
634 | print(f' corr_gc vs gc: rho = {rho_cg_gc:.3f} (circular)')
635 | print(f' corr_a0 vs gc: rho = {rho_ca_gc:.3f} (gc-free)')
636 | if abs(rho_ca_gc) < abs(rho_cg_gc) * 0.5:
637 |     print(f' >>> Layer 2 R^2 boost was LARGELY circular')
638 | elif abs(rho_ca_gc) < abs(rho_cg_gc) * 0.8:
639 |     print(f' >>> Layer 2 R^2 boost was PARTIALLY circular')
640 | else:
641 |     print(f' >>> Layer 2 R^2 boost was GENUINE (not circular)')
642 |
643 | print(f' %n PART 3: hR partial after all controls')
644 | print(f' Original: rho = {rho_hR_orig:.4f}')
645 | print(f' Final: rho = {rho_fin:.4f}')
646 | print(f' Reduction: {total_red:.1f}%')
647 | if abs(rho_fin) < 0.1:
648 |     print(f' >>> hR effect FULLY EXPLAINED by Yd + transition')
649 | elif abs(rho_fin) < 0.2:
650 |     print(f' >>> hR effect MOSTLY explained ({total_red:.0f}%)')
651 | else:
652 |     print(f' >>> hR effect PARTIALLY explained ({total_red:.0f}%), residual remains')
653 |
654 | print(' %n[DONE]')

```

6. [N-1 Layer 3] 自己無撞着方程式の代数的帰結 (Test 1,2,4,5)

項目	内容
ファイル名	sparc_N1_layer3.py
行数	659
検証ID	N-1 Layer 3

解析目的

式(5') + 式(5) の連立から $gc \sim \Sigma_{\text{bar}}$ が予測されるのに gc vs Σ_{bar} が無相関となる矛盾の源を特定。BTFR残差、BTFR品質、 c_{fit} vs gc を検証。【注意】Test 3 (eta構造) にバグあり。修正版は `layer3_fix.py`。

主要テスト

- [*] Test 1: BTFR残差 = $v_{\text{flat}}^4 / (G \times M_{\text{bar}})$ vs hR
- [*] Test 2: gc vs Σ_{bar} (V-1改の再確認)
- [*] Test 4: v_{flat}^4 vs M_{bar} (BTFR成立度)
- [*] Test 5: c_{fit} (経験式8) vs gc

結論

Test 1: BTFR残差 vs hR $\rho = -0.168$ (弱い)。Test 4: BTFR slope = 1.095, $p(1.0) = 0.262$ (成立)。Test 3 はバグのため `layer3_fix.py` で再実行。

出力ファイル

`fig_N1_layer3.png`, `N1_layer3_results.csv`

スクリプト全文

```
1 | # -*- coding: utf-8 -*-
2 | """
3 | N-1 Layer 3: U(ε;c)からの導出 -- 自己無撞着方程式の代数的帰結の検証
4 | =====
5 | 式(5') + 式(5) の連立から  $gc = \eta \times 2\pi \times a \times G \times \Sigma_{\text{bar}}$  が導出される。
6 | しかし V-1改で  $gc$  vs  $\Sigma_{\text{bar}}$  は  $R^2 = 0.016$ 。この矛盾の源を特定する。
7 |
8 | テスト1: BTFR残差 =  $v_{\text{flat}} / (G \times M_{\text{bar\_direct}})$  vs  $hR$ 
9 |   → BTFRに $hR$ 依存性があるか
10 |
11 | テスト2:  $gc$  vs  $M_{\text{bar}} / hR^2$  (=  $\Sigma_{\text{bar}}$ ) の直接回帰
12 |   → 代数的予測  $gc \propto \Sigma_{\text{bar}}$  の再確認
13 |
14 | テスト3:  $\eta_{\text{eff}} = gc / (\eta \times a \times v_{\text{flat}}^2 / hR)$  の残差構造
15 |   →  $\eta$  が  $\Sigma_{\text{bar}}$  依存性を持ち相殺しているか
16 |
17 | テスト4: BTFR成立度 --  $v_{\text{flat}}^4$  vs  $gc \times G \times M_{\text{bar}}$  の直接確認
18 |   → BTFR自体がどの程度成立しているか
19 |
20 | テスト5: U(ε;c) の曲率  $U''(\epsilon_{\text{eq}};c)$  と  $gc$  の関係
21 |   →  $c$  を経験式(8)から計算し、 $U''$  と  $gc$  の相関を確認
22 |
23 | 実行: uv run --with scipy --with matplotlib python sparc_N1_layer3.py
24 |
25 | 著者: 坂口 忍 (坂口製麺所)
26 | 日付: 2026年4月
27 | """
28 | import os, sys, glob, warnings
29 | import numpy as np
30 | from scipy import stats
31 |
32 | import matplotlib
33 | matplotlib.use('Agg')
34 | import matplotlib.pyplot as plt
35 | from matplotlib import font_manager as _fm
36 | for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
37 |            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
38 |            r'C:\Windows\Fonts\msgothic.ttc']:
39 |     try: _fm.fontManager.addfont(_fp)
40 |     except: pass
41 | for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
42 |     try:
43 |         plt.rcParams['font.family'] = fontname
44 |         break
45 |     except: continue
```

```

46 | plt.rcParams['axes.unicode_minus'] = False
47 | warnings.filterwarnings('ignore')
48 |
49 | a0 = 1.2e-10
50 | G_SI = 6.674e-11
51 | kpc_m = 3.0857e19
52 | Msun = 1.989e30
53 | pc_m = 3.0857e16
54 |
55 | BASE = os.path.dirname(os.path.abspath(__file__))
56 | ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
57 | PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
58 | TA3 = os.path.join(BASE, 'TA3_gc_independent.csv')
59 |
60 | for p, label in [(ROTMOD, 'Rotmod_LTG'), (PHASE1, 'sparc_results.csv'),
61 |                 (TA3, 'TA3_gc_independent.csv')]:
62 |     if not os.path.exists(p):
63 |         print(f'[ERROR] {label} not found: {p}'); sys.exit(1)
64 |
65 | # =====
66 | # Loaders
67 | # =====
68 | def load_csv(path):
69 |     with open(path, 'r', encoding='utf-8-sig') as f:
70 |         header = f.readline().strip()
71 |         sep = ',' if ',' in header else None
72 |         data = {}
73 |         with open(path, 'r', encoding='utf-8-sig') as f:
74 |             cols = [c.strip() for c in f.readline().strip().split(sep)]
75 |             rows = []
76 |             for line in f:
77 |                 line = line.strip()
78 |                 if not line: continue
79 |                 rows.append([p.strip() for p in line.split(sep)])
80 |             for i, col in enumerate(cols):
81 |                 vals = []
82 |                 for row in rows:
83 |                     if i < len(row):
84 |                         try: vals.append(float(row[i]))
85 |                         except: vals.append(row[i])
86 |                         else: vals.append(np.nan)
87 |                 data[col] = vals
88 |             return data
89 |
90 | def find_name_col(data):
91 |     for c in ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY']:
92 |         if c in data: return c
93 |     for k, v in data.items():
94 |         if isinstance(v[0], str): return k
95 |     return list(data.keys())[0]
96 |
97 | def get_key(info, candidates, default=None):
98 |     for c in candidates:
99 |         if c in info:
100 |             try: return float(info[c])
101 |             except: return info[c]
102 |     return default
103 |
104 | def load_rotmod(filepath):
105 |     cols = [[] for _ in range(8)]
106 |     with open(filepath, 'r') as f:
107 |         for line in f:
108 |             line = line.strip()
109 |             if not line or line.startswith('#'): continue
110 |             parts = line.split()
111 |             if len(parts) < 6: continue
112 |             try:
113 |                 for j in range(min(len(parts), 8)):
114 |                     cols[j].append(float(parts[j]))
115 |                 for j in range(len(parts), 8):
116 |                     cols[j].append(0.0)
117 |             except ValueError: continue
118 |     return tuple(np.array(c) for c in cols)
119 |
120 | # =====
121 | # U(ε; c) functions
122 | # =====
123 | def U(eps, c):
124 |     """U(ε; c) = -ε - ε2/2 - c*ln(1-ε)"""
125 |     if eps >= 1.0 or eps < 0:
126 |         return np.nan
127 |     return -eps - eps**2/2.0 - c * np.log(1.0 - eps)
128 |
129 | def U_prime(eps, c):
130 |     """U'(ε; c) = -1 - ε + c/(1-ε)"""
131 |     if eps >= 1.0:
132 |         return np.nan
133 |     return -1.0 - eps + c/(1.0 - eps)

```

```

134 |
135 | def U_double_prime(eps, c):
136 |     """U' (ε;c) = -1 + c/(1-ε)2"""
137 |     if eps >= 1.0:
138 |         return np.nan
139 |     return -1.0 + c / (1.0 - eps)**2
140 |
141 | def epsilon_eq(x, c):
142 |     """ε_eq(x,c) = (-x + sqrt((x+2)2 - 4c)) / 2"""
143 |     disc = (x + 2.0)**2 - 4.0*c
144 |     if disc < 0:
145 |         return np.nan
146 |     return (-x + np.sqrt(disc)) / 2.0
147 |
148 | # =====
149 | # Load data
150 | # =====
151 | print('[1] Loading data...')
152 | phase1 = load_csv(PHASE1)
153 | ta3     = load_csv(TA3)
154 | p1_nc  = find_name_col(phase1)
155 | ta3_nc = find_name_col(ta3)
156 |
157 | galaxy_info = {}
158 | for i, name in enumerate(phase1[p1_nc]):
159 |     name = str(name).strip()
160 |     info = {}
161 |     for k in phase1:
162 |         if k == p1_nc: continue
163 |         try: info[k] = float(phase1[k][i])
164 |         except: info[k] = phase1[k][i]
165 |     galaxy_info[name] = info
166 |
167 | for i, name in enumerate(ta3[ta3_nc]):
168 |     name = str(name).strip()
169 |     if name in galaxy_info:
170 |         for k in ta3:
171 |             if k == ta3_nc: continue
172 |             try: galaxy_info[name][k] = float(ta3[k][i])
173 |             except: galaxy_info[name][k] = ta3[k][i]
174 |
175 | # =====
176 | # Process
177 | # =====
178 | print('[2] Processing galaxies...')
179 | results = []
180 | rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
181 |
182 | for fpath in rotmod_files:
183 |     gname = os.path.splitext(os.path.basename(fpath))[0].replace('_rotmod','').strip()
184 |     info = None
185 |     for key in [gname, gname.upper(), gname.lower()]:
186 |         if key in galaxy_info:
187 |             info = galaxy_info[key]; break
188 |     if info is None: continue
189 |
190 |     ud = get_key(info, ['upsilon_d', 'Upsilon_d', 'Ud', 'ud', 'Yd'])
191 |     gc_a0 = get_key(info, ['gc_over_a0', 'gc/a0', 'gc_ratio'])
192 |     vflat = get_key(info, ['vflat', 'Vflat', 'v_flat'])
193 |     # T-type for c_fit calculation
194 |     ttype = get_key(info, ['T', 't_type', 'T_type', 'ttype'])
195 |     # Stellar mass
196 |     log_mstar = get_key(info, ['log_Mstar', 'logMstar', 'log_M*'])
197 |     # Surface brightness
198 |     log_sb = get_key(info, ['log_SBdisk', 'logSBdisk', 'SBdisk'])
199 |
200 |     if ud is None or gc_a0 is None or vflat is None: continue
201 |     if np.isnan(ud) or np.isnan(gc_a0) or gc_a0<=0 or vflat<=0 or ud<=0: continue
202 |
203 |     try:
204 |         rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
205 |     except: continue
206 |     if len(rad) < 3: continue
207 |
208 |     vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
209 |     rpk_idx = np.argmax(vds)
210 |     rpk = rad[rpk_idx]
211 |     if rpk < 0.01 or rpk >= rad.max()*0.9: continue
212 |     hR_kpc = rpk / 2.15
213 |     hR_m = hR_kpc * kpc_m
214 |
215 |     vflat_ms = vflat * 1e3
216 |     gc_si = gc_a0 * a0
217 |     GS0_proxy = vflat_ms**2 / hR_m
218 |
219 |     # M_bar direct (BT formula, gc-independent)
220 |     vdisk_peak_ms = np.max(np.sqrt(max(ud, 0.01)) * np.abs(vdisk)) * 1e3
221 |     M_disk = 2.0 * vdisk_peak_ms**2 * hR_m / (0.56 * G_SI)

```

```

222 | vgas_peak_ms = np.max(np.abs(vgas)) * 1e3
223 | M_gas = 2.0 * vgas_peak_ms**2 * hR_m / (0.56 * G_SI) if vgas_peak_ms > 0 else 0
224 | M_bar_direct = M_disk + M_gas
225 | M_bar_sun = M_bar_direct / Msun
226 |
227 | # Sigma_bar direct
228 | Sigma_bar = M_disk / (2 * np.pi * hR_m**2) # kg/m^2
229 | G_Sigma_bar = G_SI * Sigma_bar
230 |
231 | # --- Test 1: BTFR residual ---
232 | # BTFR: vflat^4 = gc * G * M_bar
233 | # Residual: R_BTFR = vflat^4 / (G * M_bar_direct)
234 | # If BTFR is exact, R_BTFR = gc
235 | if M_bar_direct > 0:
236 |     R_BTFR = vflat_ms**4 / (G_SI * M_bar_direct)
237 |     log_R_BTFR = np.log10(R_BTFR / a0)
238 | else:
239 |     R_BTFR = np.nan
240 |     log_R_BTFR = np.nan
241 |
242 | # BTFR ratio: should be gc if BTFR exact
243 | btfr_ratio = R_BTFR / gc_si if gc_si > 0 and np.isfinite(R_BTFR) else np.nan
244 |
245 | # --- Test 3: η residual ---
246 | # From proxy: gc = η^2 × a☉ × vflat^2/hR
247 | # So η^2 = gc / (a☉ × vflat^2/hR) = gc / (a☉ × GS0_proxy)
248 | eta_sq = gc_si / (a0 * GS0_proxy) if GS0_proxy > 0 else np.nan
249 | log_eta_sq = np.log10(eta_sq) if np.isfinite(eta_sq) and eta_sq > 0 else np.nan
250 |
251 | # --- Test 5: c_fit from empirical formula (8) ---
252 | # log c_fit = -0.894 + 0.278*log(M*/10^9 Msun) - 0.046*T + 0.153*log(SBdisk)
253 | c_fit = np.nan
254 | eps_eq_val = np.nan
255 | U_pp = np.nan
256 |
257 | if (log_mstar is not None and ttype is not None and log_sb is not None
258 |     and not np.isnan(log_mstar) and not np.isnan(ttype) and not np.isnan(log_sb)):
259 |     # log_mstar in SPARC is log10(M*/Msun), need log10(M*/10^9 Msun) = log_mstar - 9
260 |     log_m9 = log_mstar - 9.0
261 |     log_c_fit = -0.894 + 0.278*log_m9 - 0.046*ttype + 0.153*log_sb
262 |     c_fit = 10**log_c_fit
263 |
264 |     # Compute ε_eq at x = g_N/gc ≈ 0.5 (typical transition)
265 |     # Use x=0.5 as representative
266 |     x_rep = 0.5
267 |     if 0 < c_fit < 10:
268 |         eps_eq_val = epsilon_eq(x_rep, min(c_fit, 0.999))
269 |         if np.isfinite(eps_eq_val) and 0 < eps_eq_val < 1:
270 |             U_pp = U_double_prime(eps_eq_val, min(c_fit, 0.999))
271 |
272 | results.append({
273 |     'galaxy': gname,
274 |     'gc_a0': gc_a0,
275 |     'log_gc': np.log10(gc_a0),
276 |     'vflat': vflat,
277 |     'Yd': ud,
278 |     'log_Yd': np.log10(ud),
279 |     'hR_kpc': hR_kpc,
280 |     'log_hR': np.log10(hR_kpc),
281 |     'log_vflat': np.log10(vflat),
282 |     'GS0_proxy': GS0_proxy,
283 |     'log_GS0': np.log10(GS0_proxy/a0),
284 |     'M_bar_sun': M_bar_sun,
285 |     'log_Mbar': np.log10(M_bar_sun) if M_bar_sun > 0 else np.nan,
286 |     'G_Sigma_bar': G_Sigma_bar,
287 |     'log_GSbar': np.log10(G_Sigma_bar/a0) if G_Sigma_bar > 0 else np.nan,
288 |     'R_BTFR': R_BTFR,
289 |     'log_R_BTFR': log_R_BTFR,
290 |     'btfr_ratio': btfr_ratio,
291 |     'log_btfr_ratio': np.log10(btfr_ratio) if np.isfinite(btfr_ratio) and btfr_ratio > 0 else np.nan,
292 |     'eta_sq': eta_sq,
293 |     'log_eta_sq': log_eta_sq,
294 |     'ttype': ttype if ttype is not None else np.nan,
295 |     'log_mstar': log_mstar if log_mstar is not None else np.nan,
296 |     'log_sb': log_sb if log_sb is not None else np.nan,
297 |     'c_fit': c_fit,
298 |     'log_c_fit': np.log10(c_fit) if np.isfinite(c_fit) and c_fit > 0 else np.nan,
299 |     'eps_eq': eps_eq_val,
300 |     'U_pp': U_pp,
301 | })
302 |
303 | N = len(results)
304 | print(f' Processed: {N} galaxies')
305 | if N < 10:
306 |     print('[ERROR] Too few. '); sys.exit(1)
307 |
308 | # =====
309 | # Arrays

```

```

310 | # =====
311 | log_gc      = np.array([r['log_gc'] for r in results])
312 | log_hR      = np.array([r['log_hR'] for r in results])
313 | log_vflat   = np.array([r['log_vflat'] for r in results])
314 | log_GS0     = np.array([r['log_GS0'] for r in results])
315 | log_Mbar    = np.array([r['log_Mbar'] for r in results])
316 | log_GSbar   = np.array([r['log_GSbar'] for r in results])
317 | log_R_BTFR  = np.array([r['log_R_BTFR'] for r in results])
318 | log_btfr_r  = np.array([r['log_btfr_ratio'] for r in results])
319 | log_eta_sq  = np.array([r['log_eta_sq'] for r in results])
320 | log_Yd      = np.array([r['log_Yd'] for r in results])
321 | log_c_fit   = np.array([r['log_c_fit'] for r in results])
322 | U_pp_arr    = np.array([r['U_pp'] for r in results])
323 | c_fit_arr   = np.array([r['c_fit'] for r in results])
324 |
325 | # =====
326 | # Test 1: BTFR residual vs hR
327 | # =====
328 | print('\n' + '*'*70)
329 | print('TEST 1: BTFR residual vs hR')
330 | print('='*70)
331 |
332 | mask1 = np.isfinite(log_btfr_r) & np.isfinite(log_hR)
333 | print(f'\n N = {np.sum(mask1)}')
334 |
335 | # BTFR ratio distribution
336 | br = np.array([r['btfr_ratio'] for r in results])
337 | br_valid = br[~np.isfinite(br) & (br > 0)]
338 | print(f' BTFR ratio = vflat^4/(G*M_bar_direct*gc):')
339 | print(f' median = {np.nanmedian(br_valid):.2f}')
340 | print(f' IQR = [{np.nanpercentile(br_valid,25):.2f}, {np.nanpercentile(br_valid,75):.2f}']')
341 | print(f' (should be ~1.0 if BTFR exact with M_bar_direct)')
342 |
343 | # BTFR residual vs hR
344 | rho_br_hR, p_br_hR = stats.spearmanr(log_btfr_r[mask1], log_hR[mask1])
345 | sl_br, int_br, r_br, _, se_br = stats.linregress(log_hR[mask1], log_btfr_r[mask1])
346 | print(f'\n log(BTFR_ratio) vs log(hR):')
347 | print(f' Spearman rho = {rho_br_hR:.4f}, p = {p_br_hR:.2e}')
348 | print(f' slope = {sl_br:.3f} +/- {se_br:.3f}, R^2 = {r_br**2:.3f}')
349 |
350 | # Also check: does R_BTFR predict gc?
351 | mask1b = np.isfinite(log_R_BTFR) & np.isfinite(log_gc)
352 | sl_rb, int_rb, r_rb, _, se_rb = stats.linregress(log_R_BTFR[mask1b], log_gc[mask1b])
353 | rho_rb, p_rb = stats.spearmanr(log_R_BTFR[mask1b], log_gc[mask1b])
354 | print(f'\n log(R_BTFR/a0) vs log(gc/a0):')
355 | print(f' slope = {sl_rb:.3f} +/- {se_rb:.3f}, R^2 = {r_rb**2:.3f}')
356 | print(f' Spearman rho = {rho_rb:.4f}')
357 | print(f' (should be slope=1, R^2=1 if BTFR exact)')
358 |
359 | # =====
360 | # Test 2: gc vs Σ_bar (re-confirm V-1改)
361 | # =====
362 | print('\n' + '*'*70)
363 | print('TEST 2: gc vs Sigma_bar (re-confirmation)')
364 | print('='*70)
365 |
366 | mask2 = np.isfinite(log_GSbar) & np.isfinite(log_gc)
367 | sl2, int2, r2, _, se2 = stats.linregress(log_GSbar[mask2], log_gc[mask2])
368 | rho2, p2 = stats.spearmanr(log_GSbar[mask2], log_gc[mask2])
369 | print(f'\n N = {np.sum(mask2)}')
370 | print(f' slope = {sl2:.4f} +/- {se2:.4f}, R^2 = {r2**2:.4f}')
371 | print(f' Spearman rho = {rho2:.4f}, p = {p2:.2e}')
372 | print(f' (V-1改 reported: slope=0.080, R^2=0.016)')
373 |
374 | # =====
375 | # Test 3: η residual structure
376 | # =====
377 | print('\n' + '*'*70)
378 | print('TEST 3: eta residual structure')
379 | print('='*70)
380 |
381 | mask3 = np.isfinite(log_eta_sq)
382 | print(f'\n N = {np.sum(mask3)}')
383 | print(f' log(eta^2) distribution:')
384 | print(f' median = {np.nanmedian(log_eta_sq[mask3]):.4f}')
385 | print(f' std = {np.nanstd(log_eta_sq[mask3]):.4f} dex')
386 | print(f' eta = 10^(median/2) = {10**(np.nanmedian(log_eta_sq[mask3])/2):.4f}')
387 |
388 | # Does η correlate with Σ_bar? If so, η cancels the Σ_bar dependence
389 | mask3s = np.isfinite(log_eta_sq) & np.isfinite(log_GSbar)
390 | rho_eta_S, p_eta_S = stats.spearmanr(log_eta_sq[mask3s], log_GSbar[mask3s])
391 | sl_eS, int_eS, r_eS, _, se_eS = stats.linregress(log_GSbar[mask3s], log_eta_sq[mask3s])
392 | print(f'\n log(eta^2) vs log(G*Sigma_bar/a0):')
393 | print(f' Spearman rho = {rho_eta_S:.4f}, p = {p_eta_S:.2e}')
394 | print(f' slope = {sl_eS:.3f} +/- {se_eS:.3f}, R^2 = {r_eS**2:.3f}')
395 | print(f' If eta cancels Sigma_bar: expect slope ~ -1.0')
396 |
397 | # η vs hR

```

```

398 | mask3h = np.isfinite(log_eta_sq) & np.isfinite(log_hR)
399 | rho_eta_hR, p_eta_hR = stats.spearmanr(log_eta_sq[mask3h], log_hR[mask3h])
400 | print(f'N log(eta^2) vs log(hR):')
401 | print(f' Spearman rho = {rho_eta_hR:.4f}, p = {p_eta_hR:.2e}')
402 |
403 | # η vs Yd (known from 5-5: eta ~ Yd^-0.44)
404 | mask3y = np.isfinite(log_eta_sq) & np.isfinite(log_Yd)
405 | rho_eta_Yd, p_eta_Yd = stats.spearmanr(log_eta_sq[mask3y], log_Yd[mask3y])
406 | sl_eY, int_eY, r_eY, se_eY = stats.linregress(log_Yd[mask3y], log_eta_sq[mask3y])
407 | print(f'N log(eta^2) vs log(Yd):')
408 | print(f' Spearman rho = {rho_eta_Yd:.4f}, p = {p_eta_Yd:.2e}')
409 | print(f' slope = {sl_eY:.3f} +/- {se_eY:.3f}')
410 | print(f' (5-5 reports eta ~ Yd^-0.44, so expect slope ~ -0.88 for eta^2)')
411 |
412 | # =====
413 | # Test 4: BTFR quality check
414 | # =====
415 | print(f'N + ' = '*70)
416 | print('TEST 4: BTFR direct check')
417 | print(' = '*70)
418 |
419 | # vflat^4 vs M_bar_direct (gc-free)
420 | log_vf4 = 4 * log_vflat # log(vflat^4)
421 | mask4 = np.isfinite(log_vf4) & np.isfinite(log_Mbar)
422 | sl4, int4, r4, se4 = stats.linregress(log_Mbar[mask4], log_vf4[mask4])
423 | rho4, p4 = stats.spearmanr(log_Mbar[mask4], log_vf4[mask4])
424 | print(f'N N = {np.sum(mask4)}')
425 | print(f' log(vflat^4) vs log(M_bar_direct):')
426 | print(f' slope = {sl4:.3f} +/- {se4:.3f} (expect 1.0 for BTFR)')
427 | print(f' R^2 = {r4**2:.3f}')
428 | print(f' Spearman rho = {rho4:.4f}')
429 |
430 | # p(slope=1)
431 | t4 = abs(sl4 - 1.0) / se4
432 | p4s = 2*(1-stats.t.cdf(t4, df=np.sum(mask4)-2))
433 | print(f' p(slope=1.0) = {p4s:.4f}')
434 |
435 | # BTFR with hR correction: vflat^4 vs M_bar * hR^b
436 | # Multivariate: log(vflat^4) = a*log(M_bar) + b*log(hR) + c
437 | mask4m = np.isfinite(log_vf4) & np.isfinite(log_Mbar) & np.isfinite(log_hR)
438 | X4 = np.column_stack([log_Mbar[mask4m], log_hR[mask4m], np.ones(mask4m.sum())])
439 | y4 = log_vf4[mask4m]
440 | b4, _, _ = np.linalg.lstsq(X4, y4, rcond=None)
441 | ss4 = np.sum((y4 - X4 @ b4)**2)
442 | ss4t = np.sum((y4 - y4.mean())**2)
443 | r2_4m = 1 - ss4/ss4t
444 | mse4m = ss4 / (len(y4)-3)
445 | se4m = np.sqrt(np.diag(mse4m * np.linalg.inv(X4.T @ X4)))
446 |
447 | print(f'N Multivariate BTFR: log(vflat^4) = {b4[0]:.3f}*log(M_bar) + {b4[1]:.3f}*log(hR)')
448 | print(f' SE: {se4m[0]:.3f}, {se4m[1]:.3f}')
449 | print(f' R^2 = {r2_4m:.3f} (vs univariate {r4**2:.3f})')
450 | t_hR4 = abs(b4[1]) / se4m[1]
451 | p_hR4 = 2*(1-stats.t.cdf(t_hR4, df=len(y4)-3))
452 | print(f' p(hR coeff = 0) = {p_hR4:.4f}')
453 | print(f' -> hR adds to BTFR? {"YES" if p_hR4 < 0.05 else "NO"}')
454 |
455 | # =====
456 | # Test 5: U' (ε_eq; c_fit) vs gc
457 | # =====
458 | print(f'N + ' = '*70)
459 | print('TEST 5: U' (ε_eq; c_fit) vs gc')
460 | print(' = '*70)
461 |
462 | mask5 = np.isfinite(log_c_fit) & np.isfinite(log_gc) & np.isfinite(log_pp_arr)
463 | n5 = np.sum(mask5)
464 | print(f'N N with valid c_fit and U' : {n5}')
465 |
466 | if n5 > 10:
467 |     # c_fit vs gc
468 |     rho_cg, p_cg = stats.spearmanr(log_c_fit[mask5], log_gc[mask5])
469 |     sl_cg, int_cg, r_cg, se_cg = stats.linregress(log_c_fit[mask5], log_gc[mask5])
470 |     print(f'N log(c_fit) vs log(gc/a0):')
471 |     print(f' Spearman rho = {rho_cg:.4f}, p = {p_cg:.2e}')
472 |     print(f' slope = {sl_cg:.3f} +/- {se_cg:.3f}, R^2 = {r_cg**2:.3f}')
473 |     print(f' (gc = c*a0 predicts slope=1.0)')
474 |     t5 = abs(sl_cg - 1.0) / se_cg
475 |     p5s = 2*(1-stats.t.cdf(t5, df=n5-2))
476 |     print(f' p(slope=1.0) = {p5s:.4f}')
477 |
478 |     # U' vs gc
479 |     log_Upp = np.log10(np.abs(U_pp_arr[mask5]))
480 |     sign_Upp = np.sign(U_pp_arr[mask5])
481 |     rho_ug, p_ug = stats.spearmanr(log_Upp, log_gc[mask5])
482 |     print(f'N log|U'| vs log(gc/a0):')
483 |     print(f' Spearman rho = {rho_ug:.4f}, p = {p_ug:.2e}')
484 |     print(f' U' > 0 fraction: {np.sum(sign_Upp > 0)/len(sign_Upp):.1%}')
485 |     print(f' (U' > 0 means stable equilibrium)')

```

```

486 |
487 |     # c_fit distribution
488 |     cf_valid = c_fit_arr[np.isfinite(c_fit_arr) & (c_fit_arr > 0)]
489 |     print(f'N c_fit distribution:')
490 |     print(f'     median = {np.nanmedian(cf_valid):.3f}')
491 |     print(f'     IQR = [{np.nanpercentile(cf_valid,25):.3f}, {np.nanpercentile(cf_valid,75):.3f}]')
492 |     print(f'     fraction c_fit > 1: {np.sum(cf_valid > 1)/len(cf_valid):.1%}')
493 | else:
494 |     print(' Too few galaxies with c_fit data. Skipping.')
495 |     rho_cg = r_cg = np.nan
496 |
497 | # =====
498 | # Figures
499 | # =====
500 | print(f'N[3] Generating figures...')
501 | fig, axes = plt.subplots(2, 3, figsize=(17, 11))
502 |
503 | # 1: BTFR ratio distribution
504 | ax = axes[0, 0]
505 | br_log = log_btfr_r[np.isfinite(log_btfr_r)]
506 | ax.hist(br_log, bins=30, color='steelblue', alpha=0.7, edgecolor='white')
507 | ax.axvline(0, color='red', ls='--', lw=2, label='BTFR exact (ratio=1)')
508 | ax.set_xlabel('log(vflat^4 / (G*M_bar*gc))')
509 | ax.set_ylabel('Count')
510 | ax.set_title('1: BTFR ratio (should be ~0)')
511 | ax.legend(fontsize=8)
512 |
513 | # 2: BTFR residual vs hR
514 | ax = axes[0, 1]
515 | m = np.isfinite(log_btfr_r) & np.isfinite(log_hR)
516 | ax.scatter(log_hR[m], log_btfr_r[m], s=10, alpha=0.5, c='steelblue', edgecolors='none')
517 | xf = np.linspace(log_hR[m].min(), log_hR[m].max(), 100)
518 | ax.plot(xf, sl_br*xf + int_br, 'r-', lw=2,
519 |         label=f'rho={rho_br_hR:.3f}, slope={sl_br:.3f}')
520 | ax.set_xlabel('log(hR / kpc)')
521 | ax.set_ylabel('log(BTFR ratio)')
522 | ax.set_title('2: BTFR residual vs hR')
523 | ax.legend(fontsize=8)
524 | ax.grid(True, alpha=0.3)
525 |
526 | # 3: eta^2 vs Sigma_bar
527 | ax = axes[0, 2]
528 | m3 = np.isfinite(log_eta_sq) & np.isfinite(log_GSbar)
529 | ax.scatter(log_GSbar[m3], log_eta_sq[m3], s=10, alpha=0.5, c='darkorange', edgecolors='none')
530 | xf = np.linspace(log_GSbar[m3].min(), log_GSbar[m3].max(), 100)
531 | ax.plot(xf, sl_eS*xf + int_eS, 'r-', lw=2,
532 |         label=f'slope={sl_eS:.3f} (cancel=-1.0)')
533 | ax.plot(xf, -1.0*xf + int_eS, 'b--', lw=1, label='exact cancellation')
534 | ax.set_xlabel('log(G*Sigma_bar / a0)')
535 | ax.set_ylabel('log(eta^2)')
536 | ax.set_title('3: Does eta cancel Sigma_bar?')
537 | ax.legend(fontsize=8)
538 | ax.grid(True, alpha=0.3)
539 |
540 | # 4: BTFR direct
541 | ax = axes[1, 0]
542 | m4 = np.isfinite(log_Mbar) & np.isfinite(log_vf4)
543 | ax.scatter(log_Mbar[m4], log_vf4[m4], s=10, alpha=0.5, c='green', edgecolors='none')
544 | xf = np.linspace(log_Mbar[m4].min(), log_Mbar[m4].max(), 100)
545 | ax.plot(xf, sl4*xf + int4, 'r-', lw=2, label=f'slope={sl4:.3f}, R^2={r4**2:.3f}')
546 | ax.plot(xf, 1.0*xf + int4, 'b--', lw=1, label='BTFR: slope=1')
547 | ax.set_xlabel('log(M_bar / M_sun) [direct]')
548 | ax.set_ylabel('log(vflat^4)')
549 | ax.set_title('4: BTFR quality')
550 | ax.legend(fontsize=8)
551 | ax.grid(True, alpha=0.3)
552 |
553 | # 5: c_fit vs gc
554 | ax = axes[1, 1]
555 | m5 = np.isfinite(log_c_fit) & np.isfinite(log_gc)
556 | if np.sum(m5) > 5:
557 |     ax.scatter(log_c_fit[m5], log_gc[m5], s=10, alpha=0.5, c='purple', edgecolors='none')
558 |     xf = np.linspace(log_c_fit[m5].min(), log_c_fit[m5].max(), 100)
559 |     if np.isfinite(r_cg):
560 |         ax.plot(xf, sl_cg*xf + int_cg, 'r-', lw=2,
561 |                 label=f'slope={sl_cg:.3f}, R^2={r_cg**2:.3f}')
562 |     ax.plot(xf, 1.0*xf, 'b--', lw=1, label='gc=c*a0 (slope=1)')
563 |     ax.set_xlabel('log(c_fit)')
564 |     ax.set_ylabel('log(gc/a0)')
565 |     ax.set_title('5: c_fit vs gc (eq.8 -> gc=c*a0?)')
566 |     ax.legend(fontsize=8)
567 |     ax.grid(True, alpha=0.3)
568 |
569 | # 6: Summary R^2 bar chart
570 | ax = axes[1, 2]
571 | labels = ['proxy%alpha=0.5', 'M_bar%ndirect', 'Sigma_bar%ndirect',
572 |          'R_BTFR%N=vf^4/GM', 'c_fit%N(eq.8)']
573 | r2s = [

```

```

574 | stats.linregress(log_GS0[np.isfinite(log_GS0)&np.isfinite(log_gc)],
575 |                 log_gc[np.isfinite(log_GS0)&np.isfinite(log_gc)])[2]**2,
576 | stats.linregress(log_Mbar[np.isfinite(log_Mbar)&np.isfinite(log_gc)],
577 |                 log_gc[np.isfinite(log_Mbar)&np.isfinite(log_gc)])[2]**2,
578 | r2**2 if np.isfinite(r2) else 0,
579 | r_rb**2 if np.isfinite(r_rb) else 0,
580 | r_cg**2 if np.isfinite(r_cg) else 0,
581 | ]
582 | cols = ['steelblue', 'darkorange', 'green', 'crimson', 'purple']
583 | ax.bar(range(len(labels)), r2s, color=cols, alpha=0.7, edgecolor='black', linewidth=0.5)
584 | ax.set_xticks(range(len(labels)))
585 | ax.set_xticklabels(labels, fontsize=8)
586 | ax.set_ylabel('R^2')
587 | ax.set_title('6: gc prediction comparison')
588 | for i, v in enumerate(r2s):
589 |     ax.text(i, v+0.01, f'{v:.3f}', ha='center', fontsize=8)
590 | ax.grid(True, alpha=0.3, axis='y')
591 |
592 | fig.suptitle(f'N-1 Layer 3: Self-consistency algebra (N={N})', fontsize=14, y=1.01)
593 | fig.tight_layout()
594 | fig.savefig(os.path.join(BASE, 'fig_N1_layer3.png'), dpi=150)
595 | print(' -> fig_N1_layer3.png')
596 |
597 | # =====
598 | # Save CSV
599 | # =====
600 | outcsv = os.path.join(BASE, 'N1_layer3_results.csv')
601 | cols_out = ['galaxy', 'gc_a0', 'log_gc', 'vflat', 'Yd', 'hR_kpc', 'log_hR',
602 |            'log_GS0', 'log_Mbar', 'log_GSbar', 'log_R_BTFR', 'log_btfr_ratio',
603 |            'log_eta_sq', 'c_fit', 'log_c_fit', 'eps_eq', 'U_pp']
604 | with open(outcsv, 'w', encoding='utf-8') as f:
605 |     f.write(','.join(cols_out)+'\n')
606 |     for r in results:
607 |         f.write(','.join(str(r.get(c, '')) for c in cols_out)+'\n')
608 | print(f' -> {outcsv}')
609 |
610 | # =====
611 | # Verdict
612 | # =====
613 | print(f'\n' + '='*70)
614 | print('VERDICT')
615 | print('='*70)
616 |
617 | print(f'\n TEST 1: BTFR residual vs hR')
618 | if abs(rho_br_hR) > 0.3 and p_br_hR < 0.001:
619 |     print(f' >>> BTFR has hR dependence (rho={rho_br_hR:.3f})')
620 |     print(f' >>> Standard BTFR is NOT exact - hR modifies it')
621 | elif abs(rho_br_hR) > 0.15 and p_br_hR < 0.05:
622 |     print(f' >>> Weak hR dependence in BTFR (rho={rho_br_hR:.3f})')
623 | else:
624 |     print(f' >>> No significant hR dependence (rho={rho_br_hR:.3f})')
625 |
626 | print(f'\n TEST 2: gc vs Sigma_bar')
627 | print(f' >>> Confirmed: R^2={r2**2:.3f} (unchanged from V-1改)')
628 |
629 | print(f'\n TEST 3: eta cancellation')
630 | if abs(sL_eS + 1.0) < 2*se_eS:
631 |     print(f' >>> eta DOES cancel Sigma_bar (slope={sL_eS:.3f} ~ -1.0)')
632 |     print(f' >>> This explains why gc vs Sigma_bar is flat!')
633 | elif abs(sL_eS) > 0.3:
634 |     print(f' >>> eta PARTIALLY cancels Sigma_bar (slope={sL_eS:.3f})')
635 | else:
636 |     print(f' >>> eta does NOT cancel Sigma_bar (slope={sL_eS:.3f})')
637 |
638 | print(f'\n TEST 4: BTFR quality')
639 | print(f' >>> vflat^4 vs M_bar_direct: slope={sL4:.3f}, R^2={r4**2:.3f}')
640 | if p_hR4 < 0.05:
641 |     print(f' >>> hR adds to BTFR (p={p_hR4:.4f}): BTFR has hR residual')
642 | else:
643 |     print(f' >>> hR does NOT add to BTFR (p={p_hR4:.4f})')
644 |
645 | print(f'\n TEST 5: c_fit (eq.8) vs gc')
646 | if n5 > 10 and np.isfinite(r_cg):
647 |     print(f' >>> slope={sL_cg:.3f}, R^2={r_cg**2:.3f}')
648 |     if abs(sL_cg - 1.0) < 2*se_cg:
649 |         print(f' >>> gc = c_fit * a0 is CONSISTENT')
650 |     else:
651 |         print(f' >>> gc = c_fit * a0 is NOT consistent (slope != 1)')
652 |
653 | print(f'\n SYNTHESIS:')
654 | print(f' The gc-Sigma_bar contradiction is resolved if:')
655 | print(f' (a) eta has Sigma_bar dependence that cancels -> Test 3')
656 | print(f' (b) BTFR itself has hR dependence -> Test 1 + Test 4')
657 | print(f' (c) Both effects combine')
658 |
659 | print(f'\n[DONE]')

```

7. [N-1 Layer 3 修正版] eta 構造の完全解明 (Test 3 バグ修正)

項目	内容
ファイル名	sparc_N1_layer3_fix.py
行数	596
検証ID	N-1 Layer 3 修正版

解析目的

Layer 3 初版の eta 計算バグ (gc一乗/二乗取り違え) を修正し、eta の依存性構造を7変数に対して完全に検証。バグ修正: $\eta^2 = gc / (a0 * GS0) \rightarrow gc^2 / (a0 * GS0)$ 。

主要テスト

- [*] 健全性チェック: $\eta \sim 0(0.1-1.0)$ になるか
- [*] eta vs Yd (5-5予測 slope=-0.44 との整合)
- [*] eta vs hR, vflat, GS0_proxy (無相関の確認)
- [*] eta vs Sigma_bar (相殺 slope=-0.5 の検定)
- [*] eta vs M_bar, c_fit
- [*] effective alpha = 0.5 + (eta vs GS0 slope)

結論

eta=0.736 (妥当)。eta ~ Yd^{-0.41}, p(-0.44)=0.72 (5-5整合)。eta vs hR: R²=0.000。eta vs vflat: R²=0.006。effective alpha=0.548。N-1 実質解決。

出力ファイル

fig_N1_layer3_fix.png, N1_layer3_fix_results.csv

スクリプト全文

```
1 | # -*- coding: utf-8 -*-
2 | """
3 | N-1 Layer 3 修正版: η2計算バグ修正 + Test 3 再実行
4 | =====
5 | バグ: eta_sq = gc / (a0 * GS0) ← gc の一乗
6 | 修正: eta_sq = gc2 / (a0 * GS0) ← gc の二乗
7 |
8 | 根拠:
9 |   gc = eta * sqrt(a0 * GS0)
10 |   gc2 = eta2 * a0 * GS0
11 |   eta2 = gc2 / (a0 * GS0)
12 |
13 | Test 1,2,4,5 はeta非依存なので前回結果を維持。
14 | Test 3 のみ再実行。
15 |
16 | 実行: uv run --with scipy --with matplotlib python sparc_N1_layer3_fix.py
17 |
18 | 著者: 坂口 忍 (坂口製麺所)
19 | 日付: 2026年4月
20 | """
21 | import os, sys, glob, warnings
22 | import numpy as np
23 | from scipy import stats
24 |
25 | import matplotlib
26 | matplotlib.use('Agg')
27 | import matplotlib.pyplot as plt
28 | from matplotlib import font_manager as _fm
29 | for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
30 |            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
31 |            r'C:\Windows\Fonts\msgothic.ttc']:
32 |     try: _fm.fontManager.addfont(_fp)
33 |     except: pass
34 | for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
35 |     try:
36 |         plt.rcParams['font.family'] = fontname
37 |         break
38 |     except: continue
39 | plt.rcParams['axes.unicode_minus'] = False
40 | warnings.filterwarnings('ignore')
41 |
42 | a0 = 1.2e-10
43 | G_SI = 6.674e-11
```

```

44 | kpc_m = 3.0857e19
45 | Msun = 1.989e30
46 | pc_m = 3.0857e16
47 |
48 | BASE = os.path.dirname(os.path.abspath(__file__))
49 | ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
50 | PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
51 | TA3 = os.path.join(BASE, 'TA3_gc_independent.csv')
52 |
53 | for p, label in [(ROTMOD, 'Rotmod_LTG'), (PHASE1, 'sparc_results.csv'),
54 |                 (TA3, 'TA3_gc_independent.csv')]:
55 |     if not os.path.exists(p):
56 |         print(f'[ERROR] {label} not found: {p}'); sys.exit(1)
57 |
58 | # =====
59 | # Loaders
60 | # =====
61 | def load_csv(path):
62 |     with open(path, 'r', encoding='utf-8-sig') as f:
63 |         header = f.readline().strip()
64 |         sep = ',' if ',' in header else None
65 |         data = {}
66 |         with open(path, 'r', encoding='utf-8-sig') as f:
67 |             cols = [c.strip() for c in f.readline().strip().split(sep)]
68 |             rows = []
69 |             for line in f:
70 |                 line = line.strip()
71 |                 if not line: continue
72 |                 rows.append([p.strip() for p in line.split(sep)])
73 |             for i, col in enumerate(cols):
74 |                 vals = []
75 |                 for row in rows:
76 |                     if i < len(row):
77 |                         try: vals.append(float(row[i]))
78 |                         except: vals.append(row[i])
79 |                     else: vals.append(np.nan)
80 |                 data[col] = vals
81 |             return data
82 |
83 | def find_name_col(data):
84 |     for c in ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY']:
85 |         if c in data: return c
86 |     for k, v in data.items():
87 |         if isinstance(v[0], str): return k
88 |     return list(data.keys())[0]
89 |
90 | def get_key(info, candidates, default=None):
91 |     for c in candidates:
92 |         if c in info:
93 |             try: return float(info[c])
94 |             except: return info[c]
95 |     return default
96 |
97 | def load_rotmod(filepath):
98 |     cols = [[] for _ in range(8)]
99 |     with open(filepath, 'r') as f:
100 |         for line in f:
101 |             line = line.strip()
102 |             if not line or line.startswith('#'): continue
103 |             parts = line.split()
104 |             if len(parts) < 6: continue
105 |             try:
106 |                 for j in range(min(len(parts), 8)):
107 |                     cols[j].append(float(parts[j]))
108 |                 for j in range(len(parts), 8):
109 |                     cols[j].append(0.0)
110 |             except ValueError: continue
111 |     return tuple(np.array(c) for c in cols)
112 |
113 | # =====
114 | # U(ε; c) functions
115 | # =====
116 | def epsilon_eq(x, c):
117 |     disc = (x + 2.0)**2 - 4.0*c
118 |     if disc < 0: return np.nan
119 |     return (-x + np.sqrt(disc)) / 2.0
120 |
121 | def U_double_prime(eps, c):
122 |     if eps >= 1.0: return np.nan
123 |     return -1.0 + c / (1.0 - eps)**2
124 |
125 | # =====
126 | # Load & process
127 | # =====
128 | print('[1] Loading data...')
129 | phase1 = load_csv(PHASE1)
130 | ta3 = load_csv(TA3)
131 | p1_nc = find_name_col(phase1)

```

```

132 | ta3_nc = find_name_col(ta3)
133 |
134 | galaxy_info = {}
135 | for i, name in enumerate(phase1[p1_nc]):
136 |     name = str(name).strip()
137 |     info = {}
138 |     for k in phase1:
139 |         if k == p1_nc: continue
140 |         try: info[k] = float(phase1[k][i])
141 |         except: info[k] = phase1[k][i]
142 |     galaxy_info[name] = info
143 |
144 | for i, name in enumerate(ta3[ta3_nc]):
145 |     name = str(name).strip()
146 |     if name in galaxy_info:
147 |         for k in ta3:
148 |             if k == ta3_nc: continue
149 |             try: galaxy_info[name][k] = float(ta3[k][i])
150 |             except: galaxy_info[name][k] = ta3[k][i]
151 |
152 | print('[2] Processing galaxies...')
153 | results = []
154 | rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
155 |
156 | for fpath in rotmod_files:
157 |     gname = os.path.splitext(os.path.basename(fpath))[0].replace('_rotmod','').strip()
158 |     info = None
159 |     for key in [gname, gname.upper(), gname.lower()]:
160 |         if key in galaxy_info:
161 |             info = galaxy_info[key]; break
162 |     if info is None: continue
163 |
164 |     ud = get_key(info, ['upsilon_d', 'Upsilon_d', 'Ud', 'ud', 'Yd'])
165 |     gc_a0 = get_key(info, ['gc_over_a0', 'gc/a0', 'gc_ratio'])
166 |     vflat = get_key(info, ['vflat', 'Vflat', 'v_flat'])
167 |     ttype = get_key(info, ['T', 't_type', 'T_type', 'ttype'])
168 |     log_mstar = get_key(info, ['log_Mstar', 'logMstar', 'log_M*'])
169 |     log_sb = get_key(info, ['log_SBdisk', 'logSBdisk', 'SBdisk'])
170 |
171 |     if ud is None or gc_a0 is None or vflat is None: continue
172 |     if np.isnan(ud) or np.isnan(gc_a0) or gc_a0<=0 or vflat<=0 or ud<=0: continue
173 |
174 |     try:
175 |         rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
176 |     except: continue
177 |     if len(rad) < 3: continue
178 |
179 |     vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
180 |     rpk_idx = np.argmax(vds)
181 |     rpk = rad[rpk_idx]
182 |     if rpk < 0.01 or rpk >= rad.max()*0.9: continue
183 |     hR_kpc = rpk / 2.15
184 |     hR_m = hR_kpc * kpc_m
185 |
186 |     vflat_ms = vflat * 1e3
187 |     gc_si = gc_a0 * a0
188 |     GS0_proxy = vflat_ms**2 / hR_m
189 |
190 |     # M_bar direct (BT)
191 |     vdisk_peak_ms = np.max(np.sqrt(max(ud, 0.01)) * np.abs(vdisk)) * 1e3
192 |     M_disk = 2.0 * vdisk_peak_ms**2 * hR_m / (0.56 * G_SI)
193 |     vgas_peak_ms = np.max(np.abs(vgas)) * 1e3
194 |     M_gas = 2.0 * vgas_peak_ms**2 * hR_m / (0.56 * G_SI) if vgas_peak_ms > 0 else 0
195 |     M_bar_direct = M_disk + M_gas
196 |     M_bar_sun = M_bar_direct / Msun
197 |
198 |     Sigma_bar = M_disk / (2 * np.pi * hR_m**2)
199 |     G_Sigma_bar = G_SI * Sigma_bar
200 |
201 |     # --- FIXED:  $\eta^2 = gc^2 / (a \times GS0\_proxy)$  ---
202 |     eta_sq = gc_si**2 / (a0 * GS0_proxy) if GS0_proxy > 0 else np.nan
203 |     log_eta_sq = np.log10(eta_sq) if np.isfinite(eta_sq) and eta_sq > 0 else np.nan
204 |
205 |     #  $\eta$  (not  $\eta^2$ )
206 |     eta_val = np.sqrt(eta_sq) if np.isfinite(eta_sq) and eta_sq > 0 else np.nan
207 |     log_eta = np.log10(eta_val) if np.isfinite(eta_val) and eta_val > 0 else np.nan
208 |
209 |     # BTFR residual
210 |     if M_bar_direct > 0:
211 |         R_BTFR = vflat_ms**4 / (G_SI * M_bar_direct)
212 |         btfr_ratio = R_BTFR / gc_si if gc_si > 0 else np.nan
213 |     else:
214 |         R_BTFR = np.nan
215 |         btfr_ratio = np.nan
216 |
217 |     # c_fit from eq(8)
218 |     c_fit = np.nan
219 |     U_pp = np.nan

```

```

220 |     if (log_mstar is not None and ttype is not None and log_sb is not None
221 |         and not np.isnan(log_mstar) and not np.isnan(ttype) and not np.isnan(log_sb)):
222 |         log_m9 = log_mstar - 9.0
223 |         log_c_fit_val = -0.894 + 0.278*log_m9 - 0.046*ttype + 0.153*log_sb
224 |         c_fit = 10**log_c_fit_val
225 |         x_rep = 0.5
226 |         if 0 < c_fit < 10:
227 |             eps_eq_val = epsilon_eq(x_rep, min(c_fit, 0.999))
228 |             if np.isfinite(eps_eq_val) and 0 < eps_eq_val < 1:
229 |                 U_pp = U_double_prime(eps_eq_val, min(c_fit, 0.999))
230 |
231 |     results.append({
232 |         'galaxy':      gname,
233 |         'gc_a0':      gc_a0,
234 |         'log_gc':     np.log10(gc_a0),
235 |         'vflat':     vflat,
236 |         'Yd':        ud,
237 |         'log_Yd':    np.log10(ud),
238 |         'hR_kpc':   hR_kpc,
239 |         'log_hR':   np.log10(hR_kpc),
240 |         'log_vflat': np.log10(vflat),
241 |         'GS0_proxy': GS0_proxy,
242 |         'log_GS0':  np.log10(GS0_proxy/a0),
243 |         'M_bar_sun': M_bar_sun,
244 |         'log_Mbar': np.log10(M_bar_sun) if M_bar_sun > 0 else np.nan,
245 |         'G_Sigma_bar': G_Sigma_bar,
246 |         'log_GSbar': np.log10(G_Sigma_bar/a0) if G_Sigma_bar > 0 else np.nan,
247 |         'eta_sq':   eta_sq,
248 |         'log_eta_sq': log_eta_sq,
249 |         'eta':     eta_val,
250 |         'log_eta': log_eta,
251 |         'btfr_ratio': btfr_ratio,
252 |         'log_btfr_r': np.log10(btfr_ratio) if np.isfinite(btfr_ratio) and btfr_ratio > 0 else np.nan,
253 |         'c_fit':     c_fit,
254 |         'log_c_fit': np.log10(c_fit) if np.isfinite(c_fit) and c_fit > 0 else np.nan,
255 |         'U_pp':     U_pp,
256 |     })
257 |
258 | N = len(results)
259 | print(f' Processed: {N} galaxies')
260 | if N < 10:
261 |     print('[ERROR] Too few. '); sys.exit(1)
262 |
263 | # =====
264 | # Arrays
265 | # =====
266 | log_gc     = np.array([r['log_gc'] for r in results])
267 | log_hR     = np.array([r['log_hR'] for r in results])
268 | log_vflat  = np.array([r['log_vflat'] for r in results])
269 | log_GS0    = np.array([r['log_GS0'] for r in results])
270 | log_Mbar   = np.array([r['log_Mbar'] for r in results])
271 | log_GSbar  = np.array([r['log_GSbar'] for r in results])
272 | log_eta_sq = np.array([r['log_eta_sq'] for r in results])
273 | log_eta    = np.array([r['log_eta'] for r in results])
274 | log_Yd     = np.array([r['log_Yd'] for r in results])
275 | log_c_fit  = np.array([r['log_c_fit'] for r in results])
276 | log_btfr_r = np.array([r['log_btfr_r'] for r in results])
277 | eta_arr    = np.array([r['eta'] for r in results])
278 | c_fit_arr  = np.array([r['c_fit'] for r in results])
279 | U_pp_arr   = np.array([r['U_pp'] for r in results])
280 |
281 | # =====
282 | # Sanity check:  $\eta$  values
283 | # =====
284 | print(f'#{n} + ' * 70)
285 | print('SANITY CHECK: eta values (FIXED)')
286 | print('=' * 70)
287 |
288 | m_eta = np.isfinite(log_eta)
289 | print(f'#{n} N = {np.sum(m_eta)}')
290 | print(f' eta distribution:')
291 | print(f'   median = {np.nanmedian(eta_arr[m_eta]):.4f}')
292 | print(f'   IQR = [{np.nanpercentile(eta_arr[m_eta], 25):.4f}, '
293 |         f' {np.nanpercentile(eta_arr[m_eta], 75):.4f}]')
294 | print(f'   range = [{np.nanmin(eta_arr[m_eta]):.4f}, {np.nanmax(eta_arr[m_eta]):.4f}]')
295 | print(f'   (5-5 reports eta0 ~ 0(1). Should be ~0.1-1.0)')
296 |
297 | print(f'#{n} eta^2 distribution:')
298 | eta_sq_arr = np.array([r['eta_sq'] for r in results])
299 | print(f'   median = {np.nanmedian(eta_sq_arr[m_eta]):.6f}')
300 | print(f'   log10(eta^2) median = {np.nanmedian(log_eta_sq[m_eta]):.4f}')
301 |
302 | # Verification: pick a typical galaxy
303 | for r in results:
304 |     if 0.2 < r['gc_a0'] < 0.3 and 80 < r['vflat'] < 120:
305 |         print(f'#{n} Verification galaxy: {r["galaxy"]}')
306 |         print(f'   gc = {r["gc_a0"]:.3f} a0 = {r["gc_a0"]*a0:.3e} m/s^2')
307 |         print(f'   vflat = {r["vflat"]:.1f} km/s')

```

```

308 |         print(f'      hR = {r["hR_kpc"]:.2f} kpc')
309 |         print(f'      GS0 = {r["GS0_proxy"]:.3e} m/s^2')
310 |         print(f'      eta^2 = gc^2/(a0*GS0) = {r["eta_sq"]:.6f}')
311 |         print(f'      eta = {r["eta"]:.4f}')
312 |         print(f'      Check: eta*sqrt(a0*GS0) = {r["eta"]*np.sqrt(a0*r["GS0_proxy"]):.3e}'
313 |               f' vs gc = {r["gc_a0"]*a0:.3e}')
314 |         break
315 |
316 | # =====
317 | # TEST 3 CORRECTED:  $\eta$  structure
318 | # =====
319 | print(f' %n' + '='*70)
320 | print('TEST 3 CORRECTED: eta residual structure')
321 | print('='*70)
322 |
323 | # --- 3a:  $\eta$  vs  $\Sigma_{\text{bar}}$  ---
324 | mask3s = np.isfinite(log_eta) & np.isfinite(log_GSbar)
325 | rho_eS, p_eS = stats.spearmanr(log_eta[mask3s], log_GSbar[mask3s])
326 | sl_eS, int_eS, r_eS, _, se_eS = stats.linregress(log_GSbar[mask3s], log_eta[mask3s])
327 | print(f' %n [3a] log(eta) vs log(G*Sigma_bar/a0):')
328 | print(f'      N = {np.sum(mask3s)}')
329 | print(f'      Spearman rho = {rho_eS:.4f}, p = {p_eS:.2e}')
330 | print(f'      slope = {sl_eS:.4f} +/- {se_eS:.4f}, R^2 = {r_eS**2:.4f}')
331 | print(f'      If eta cancels Sigma_bar: expect slope ~ -0.5')
332 | print(f'      (because gc=eta*sqrt(a0*GS0) and gc~const*Sigma_bar^0')
333 | print(f'      -> eta ~ Sigma_bar^(-0.5) to cancel)')
334 | t_cancel = abs(sl_eS - (-0.5)) / se_eS
335 | p_cancel = 2*(1-stats.t.cdf(t_cancel, df=np.sum(mask3s)-2))
336 | print(f'      p(slope=-0.5) = {p_cancel:.4f}')
337 |
338 | # --- 3b:  $\eta$  vs hR ---
339 | mask3h = np.isfinite(log_eta) & np.isfinite(log_hR)
340 | rho_eH, p_eH = stats.spearmanr(log_eta[mask3h], log_hR[mask3h])
341 | sl_eH, int_eH, r_eH, _, se_eH = stats.linregress(log_hR[mask3h], log_eta[mask3h])
342 | print(f' %n [3b] log(eta) vs log(hR):')
343 | print(f'      Spearman rho = {rho_eH:.4f}, p = {p_eH:.2e}')
344 | print(f'      slope = {sl_eH:.4f} +/- {se_eH:.4f}, R^2 = {r_eH**2:.4f}')
345 |
346 | # --- 3c:  $\eta$  vs vflat ---
347 | mask3v = np.isfinite(log_eta) & np.isfinite(log_vflat)
348 | rho_eV, p_eV = stats.spearmanr(log_eta[mask3v], log_vflat[mask3v])
349 | sl_eV, int_eV, r_eV, _, se_eV = stats.linregress(log_vflat[mask3v], log_eta[mask3v])
350 | print(f' %n [3c] log(eta) vs log(vflat):')
351 | print(f'      Spearman rho = {rho_eV:.4f}, p = {p_eV:.2e}')
352 | print(f'      slope = {sl_eV:.4f} +/- {se_eV:.4f}, R^2 = {r_eV**2:.4f}')
353 |
354 | # --- 3d:  $\eta$  vs Yd (known from 5-5: eta ~ Yd^-0.44) ---
355 | mask3y = np.isfinite(log_eta) & np.isfinite(log_Yd)
356 | rho_eY, p_eY = stats.spearmanr(log_eta[mask3y], log_Yd[mask3y])
357 | sl_eY, int_eY, r_eY, _, se_eY = stats.linregress(log_Yd[mask3y], log_eta[mask3y])
358 | print(f' %n [3d] log(eta) vs log(Yd):')
359 | print(f'      Spearman rho = {rho_eY:.4f}, p = {p_eY:.2e}')
360 | print(f'      slope = {sl_eY:.4f} +/- {se_eY:.4f}')
361 | print(f'      (5-5 predicts: slope ~ -0.44)')
362 | t_yd = abs(sl_eY - (-0.44)) / se_eY
363 | p_yd = 2*(1-stats.t.cdf(t_yd, df=np.sum(mask3y)-2))
364 | print(f'      p(slope=-0.44) = {p_yd:.4f}')
365 |
366 | # --- 3e:  $\eta$  vs M_bar direct ---
367 | mask3m = np.isfinite(log_eta) & np.isfinite(log_Mbar)
368 | rho_eM, p_eM = stats.spearmanr(log_eta[mask3m], log_Mbar[mask3m])
369 | sl_eM, int_eM, r_eM, _, se_eM = stats.linregress(log_Mbar[mask3m], log_eta[mask3m])
370 | print(f' %n [3e] log(eta) vs log(M_bar direct):')
371 | print(f'      Spearman rho = {rho_eM:.4f}, p = {p_eM:.2e}')
372 | print(f'      slope = {sl_eM:.4f} +/- {se_eM:.4f}, R^2 = {r_eM**2:.4f}')
373 |
374 | # --- 3f:  $\eta$  vs c_fit ---
375 | mask3c = np.isfinite(log_eta) & np.isfinite(log_c_fit)
376 | if np.sum(mask3c) > 10:
377 |     rho_eC, p_eC = stats.spearmanr(log_eta[mask3c], log_c_fit[mask3c])
378 |     sl_eC, int_eC, r_eC, _, se_eC = stats.linregress(log_c_fit[mask3c], log_eta[mask3c])
379 |     print(f' %n [3f] log(eta) vs log(c_fit):')
380 |     print(f'      Spearman rho = {rho_eC:.4f}, p = {p_eC:.2e}')
381 |     print(f'      slope = {sl_eC:.4f} +/- {se_eC:.4f}, R^2 = {r_eC**2:.4f}')
382 |     print(f'      (gc=c*a0 means eta=gc/sqrt(a0*GS0)=c*a0/sqrt(a0*GS0)=c*sqrt(a0/GS0)')
383 | else:
384 |     rho_eC = r_eC = sl_eC = np.nan
385 |     print(f' %n [3f] Too few c_fit values ({np.sum(mask3c)})')
386 |
387 | # --- 3g:  $\eta$  vs GS0_proxy (the tautology check) ---
388 | # Since eta = gc/sqrt(a0*GS0), and gc ~ GS0^0.5 (the geometric mean law),
389 | # eta ~ GS0^0.5 / GS0^0.5 = constant (if law is exact)
390 | # Any deviation shows eta's dependence structure
391 | mask3g = np.isfinite(log_eta) & np.isfinite(log_GS0)
392 | rho_eG, p_eG = stats.spearmanr(log_eta[mask3g], log_GS0[mask3g])
393 | sl_eG, int_eG, r_eG, _, se_eG = stats.linregress(log_GS0[mask3g], log_eta[mask3g])
394 | print(f' %n [3g] log(eta) vs log(GS0_proxy/a0):')
395 | print(f'      Spearman rho = {rho_eG:.4f}, p = {p_eG:.2e}')

```

```

396 | print(f' slope = {sl_eG:.4f} +/- {se_eG:.4f}, R^2 = {r_eG**2:.4f}')
397 | print(f' If gc=eta*sqrt(a0*GS0) exact with const eta: slope=0')
398 | print(f' Residual from 0.5 law: slope = {sl_eG:.4f} = alpha_eff - 0.5 = {sl_eG+0.5:.4f}')
399 |
400 | # =====
401 | # TEST 5 RECHECK: c_fit vs gc
402 | # =====
403 | print('\n' + '='*70)
404 | print('TEST 5 RECHECK: c_fit vs gc')
405 | print('='*70)
406 |
407 | mask5 = np.isfinite(log_c_fit) & np.isfinite(log_gc)
408 | n5 = np.sum(mask5)
409 | if n5 > 10:
410 |     sl5, int5, r5, _, se5 = stats.linregress(log_c_fit[mask5], log_gc[mask5])
411 |     rho5, p5 = stats.spearmanr(log_c_fit[mask5], log_gc[mask5])
412 |     print(f'\n N = {n5}')
413 |     print(f' log(gc/a0) vs log(c_fit):')
414 |     print(f' slope = {sl5:.4f} +/- {se5:.4f}, R^2 = {r5**2:.4f}')
415 |     print(f' Spearman rho = {rho5:.4f}, p = {p5:.2e}')
416 |     t5s = abs(sl5 - 1.0) / se5
417 |     p5s = 2*(1-stats.t.cdf(t5s, df=n5-2))
418 |     print(f' p(slope=1.0) = {p5s:.4f} (gc=c*a0 prediction)')
419 |
420 | # Does adding eta improve gc prediction from c_fit?
421 | mask5e = mask5 & np.isfinite(log_eta)
422 | if np.sum(mask5e) > 10:
423 |     X5 = np.column_stack([log_c_fit[mask5e], log_eta[mask5e], np.ones(mask5e.sum())])
424 |     y5 = log_gc[mask5e]
425 |     b5, _, _, _ = np.linalg.lstsq(X5, y5, rcond=None)
426 |     ss5r = np.sum((y5 - X5 @ b5)**2)
427 |     ss5t = np.sum((y5 - y5.mean())**2)
428 |     r2_5e = 1 - ss5r/ss5t
429 |     mse5e = ss5r / (len(y5)-3)
430 |     se5e = np.sqrt(np.diag(mse5e * np.linalg.inv(X5.T @ X5)))
431 |     print(f'\n gc = c_fit^a * eta^b:')
432 |     print(f' c_fit^{b5[0]:.3f} +/- {se5e[0]:.3f}, eta^{b5[1]:.3f} +/- {se5e[1]:.3f}')
433 |     print(f' R^2 = {r2_5e:.4f} (vs c_fit alone {r5**2:.4f})')
434 | else:
435 |     print(f' Too few c_fit values ({n5})')
436 |
437 | # =====
438 | # SYNTHESIS: What determines eta?
439 | # =====
440 | print('\n' + '='*70)
441 | print('SYNTHESIS: What determines eta?')
442 | print('='*70)
443 |
444 | print(f'\n {"Variable":<20s} | {"slope":>8s} +/- {"SE":>6s} | {"R^2":>6s} | {"rho":>6s} | {"p":>10s}')
445 | print(' ' + '='*75)
446 | for label, sl, se, r2, rho, p in [
447 |     ('*Sigma_bar', sl_eS, se_eS, r_eS**2, rho_eS, p_eS),
448 |     ('hR', sl_eH, se_eH, r_eH**2, rho_eH, p_eH),
449 |     ('vflat', sl_eV, se_eV, r_eV**2, rho_eV, p_eV),
450 |     ('Yd', sl_eY, se_eY, r_eY**2, rho_eY, p_eY),
451 |     ('M_bar direct', sl_eM, se_eM, r_eM**2, rho_eM, p_eM),
452 |     ('GS0_proxy', sl_eG, se_eG, r_eG**2, rho_eG, p_eG),
453 | ]:
454 |     if np.isfinite(sl):
455 |         print(f' {label:<20s} | {sl:>8.4f} +/- {se:>6.4f} | {r2:>6.4f} | {rho:>6.4f} | {p:>10.2e}')
456 |
457 | if np.isfinite(sl_eC):
458 |     print(f' {"c_fit":<20s} | {sl_eC:>8.4f} +/- {se_eC:>6.4f} | {r_eC**2:>6.4f} | {rho_eC:>6.4f} | {p_eC:>10.2e}')
459 |
460 | # =====
461 | # Figures
462 | # =====
463 | print('\n[3] Generating figures...')
464 | fig, axes = plt.subplots(2, 3, figsize=(17, 11))
465 |
466 | # 1: eta distribution
467 | ax = axes[0, 0]
468 | ev = eta_arr[np.isfinite(eta_arr) & (eta_arr > 0)]
469 | ax.hist(np.log10(ev), bins=30, color='steelblue', alpha=0.7, edgecolor='white')
470 | ax.axvline(np.log10(np.nanmedian(ev)), color='red', ls='--',
471 |            label=f'median={np.nanmedian(ev):.4f}')
472 | ax.set_xlabel('log10(eta)')
473 | ax.set_ylabel('Count')
474 | ax.set_title('1: eta distribution (FIXED)')
475 | ax.legend(fontsize=8)
476 |
477 | # 2: eta vs Sigma_bar
478 | ax = axes[0, 1]
479 | m = np.isfinite(log_eta) & np.isfinite(log_GSbar)
480 | ax.scatter(log_GSbar[m], log_eta[m], s=10, alpha=0.5, c='darkorange', edgecolors='none')
481 | xf = np.linspace(log_GSbar[m].min(), log_GSbar[m].max(), 100)
482 | ax.plot(xf, sl_eS*xf + int_eS, 'r-', lw=2, label=f'slope={sl_eS:.3f}')
483 | ax.plot(xf, -0.5*xf + int_eS, 'b--', lw=1.5, label='full cancel: -0.5')

```

```

484 | ax.set_xlabel('log(G*Sigma_bar / a0)')
485 | ax.set_ylabel('log(eta)')
486 | ax.set_title('2: eta vs Sigma_bar (rho={rho_eS:.3f})')
487 | ax.legend(fontsize=8)
488 | ax.grid(True, alpha=0.3)
489 |
490 | # 3: eta vs Yd
491 | ax = axes[0, 2]
492 | m = np.isfinite(log_eta) & np.isfinite(log_Yd)
493 | ax.scatter(log_Yd[m], log_eta[m], s=10, alpha=0.5, c='green', edgecolors='none')
494 | xf = np.linspace(log_Yd[m].min(), log_Yd[m].max(), 100)
495 | ax.plot(xf, sL_eY*xf + int_eY, 'r-', lw=2, label=f'slope={sL_eY:.3f}')
496 | ax.plot(xf, -0.44*xf + int_eY, 'b--', lw=1.5, label='5-5 prediction: -0.44')
497 | ax.set_xlabel('log(Yd)')
498 | ax.set_ylabel('log(eta)')
499 | ax.set_title('3: eta vs Yd (rho={rho_eY:.3f})')
500 | ax.legend(fontsize=8)
501 | ax.grid(True, alpha=0.3)
502 |
503 | # 4: eta vs GS0_proxy
504 | ax = axes[1, 0]
505 | m = np.isfinite(log_eta) & np.isfinite(log_GS0)
506 | ax.scatter(log_GS0[m], log_eta[m], s=10, alpha=0.5, c='crimson', edgecolors='none')
507 | xf = np.linspace(log_GS0[m].min(), log_GS0[m].max(), 100)
508 | ax.plot(xf, sL_eG*xf + int_eG, 'r-', lw=2, label=f'slope={sL_eG:.3f}')
509 | ax.axhline(np.nanmedian(log_eta[m]), color='grey', ls=':', label='const eta')
510 | ax.set_xlabel('log(GS0_proxy / a0)')
511 | ax.set_ylabel('log(eta)')
512 | ax.set_title('4: eta vs proxy (rho={rho_eG:.3f})')
513 | ax.legend(fontsize=8)
514 | ax.grid(True, alpha=0.3)
515 |
516 | # 5: c_fit vs gc
517 | ax = axes[1, 1]
518 | m5p = np.isfinite(log_c_fit) & np.isfinite(log_gc)
519 | if np.sum(m5p) > 5:
520 |     ax.scatter(log_c_fit[m5p], log_gc[m5p], s=10, alpha=0.5, c='purple', edgecolors='none')
521 |     xf = np.linspace(log_c_fit[m5p].min(), log_c_fit[m5p].max(), 100)
522 |     if np.isfinite(s15):
523 |         ax.plot(xf, s15*xf + int5, 'r-', lw=2, label=f'slope={s15:.3f}, R^2={r5**2:.3f}')
524 |     ax.plot(xf, 1.0*xf, 'b--', lw=1, label='gc=c*a0')
525 |     ax.set_xlabel('log(c_fit)')
526 |     ax.set_ylabel('log(gc/a0)')
527 |     ax.set_title('5: c_fit vs gc')
528 |     ax.legend(fontsize=8)
529 |     ax.grid(True, alpha=0.3)
530 |
531 | # 6: R^2 summary for eta correlations
532 | ax = axes[1, 2]
533 | labels_bar = ['Sigma_bar', 'hR', 'vflat', 'Yd', 'M_bar', 'GS0_proxy']
534 | r2s_bar = [r_eS**2, r_eH**2, r_eV**2, r_eY**2, r_eM**2, r_eG**2]
535 | if np.isfinite(r_eC):
536 |     labels_bar.append('c_fit')
537 |     r2s_bar.append(r_eC**2)
538 | cols_bar = ['darkorange', 'teal', 'crimson', 'green', 'steelblue', 'grey', 'purple'][:len(labels_bar)]
539 | ax.bar(range(len(labels_bar)), r2s_bar, color=cols_bar, alpha=0.7,
540 |        edgecolor='black', linewidth=0.5)
541 | ax.set_xticks(range(len(labels_bar)))
542 | ax.set_xticklabels(labels_bar, fontsize=8, rotation=15)
543 | ax.set_ylabel('R^2 with log(eta)')
544 | ax.set_title('6: What determines eta?')
545 | for i, v in enumerate(r2s_bar):
546 |     ax.text(i, v+0.005, f'{v:.3f}', ha='center', fontsize=8)
547 | ax.grid(True, alpha=0.3, axis='y')
548 |
549 | fig.suptitle('N-1 Layer 3 FIX: eta structure (N={N})', fontsize=14, y=1.01)
550 | fig.tight_layout()
551 | fig.savefig(os.path.join(BASE, 'fig_N1_layer3_fix.png'), dpi=150)
552 | print(' -> fig_N1_layer3_fix.png')
553 |
554 | # =====
555 | # Save CSV
556 | # =====
557 | outcsv = os.path.join(BASE, 'N1_layer3_fix_results.csv')
558 | cols_out = ['galaxy', 'gc_a0', 'log_gc', 'vflat', 'Yd', 'log_Yd', 'hR_kpc', 'log_hR',
559 |            'log_GS0', 'log_Mbar', 'log_GSbar', 'eta', 'log_eta', 'log_eta_sq',
560 |            'log_btfr_r', 'c_fit', 'log_c_fit', 'U_pp']
561 | with open(outcsv, 'w', encoding='utf-8') as f:
562 |     f.write(','.join(cols_out)+'\n')
563 |     for r in results:
564 |         f.write(','.join(str(r.get(c,'')) for c in cols_out)+'\n')
565 | print(f' -> {outcsv}')
566 |
567 | # =====
568 | # Verdict
569 | # =====
570 | print(' %n' + ' %70)
571 | print(' VERDICT (CORRECTED)')

```

```

572 | print('='*70)
573 |
574 | print(f'eta median = {np.nanmedian(eta_arr[m_eta]):.4f} (should be 0(0.1-1))')
575 |
576 | print(f'KEY: Does eta cancel Sigma_bar dependence?')
577 | print(f'log(eta) vs log(G*Sigma_bar/a0): slope = {sL_eS:.4f} +/- {se_eS:.4f}')
578 | if abs(sL_eS - (-0.5)) < 2*se_eS:
579 |     print(f'>>> YES: slope consistent with -0.5 (full cancellation)')
580 |     print(f'>>> gc ~ eta*sqrt(a0*GS0) with eta~Sigma_bar^-0.5')
581 |     print(f'>>> means gc is INDEPENDENT of Sigma_bar (as observed)')
582 |     print(f'>>> N-1 Layer 3: RESOLVED (eta absorbs Sigma_bar)')
583 | elif abs(sL_eS) > 0.2:
584 |     print(f'>>> PARTIAL: slope={sL_eS:.3f} (not -0.5 but significant)')
585 |     print(f'>>> eta partially cancels Sigma_bar')
586 | else:
587 |     print(f'>>> NO: eta does not cancel Sigma_bar')
588 |
589 | print(f'eta vs Yd: slope={sL_eY:.3f} (5-5 predicts -0.44)')
590 | print(f'p(slope=-0.44) = {p_yd:.4f}')
591 |
592 | print(f'eta vs GS0_proxy: slope={sL_eG:.3f}')
593 | print(f'(deviation from 0 = deviation of alpha from 0.5)')
594 | print(f'effective alpha = 0.5 + {sL_eG:.4f} = {0.5+sL_eG:.4f}')
595 |
596 | print(f'DONE')

```