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 ~ 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 ~ Sigma_bar^(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 ~ vflat^1.10 x 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 ~ 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)。eta ~ 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 の質量分率) から c=1-f_p を構成し、観測された gc 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 gc 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_peak/GS0 ~ 1.8)

結論

FAIL: f_p ~ 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

```
# -*- coding: utf-8 -*-
"""
V-1: 条件14 定量的機構の数値検証
====================================
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
数値計算し、観測された gc との相関を検証する。

実行: uv run --with scipy --with matplotlib python sparc_fp_verification.py

必要ファイル:
  - Rotmod_LTG/*.dat  (SPARC回転曲線)
  - phase1/sparc_results.csv  (フィット結果: vflat, Yd, rs, gc等)
  - TA3_gc_independent.csv    (gc_over_a0)

出力:
  - fp_verification_results.csv
  - fig_fp_vs_gc.png
  - fig_c_vs_sqrt_scaling.png
  - fig_fp_mechanism_summary.png
  - コンソールに統計サマリー

著者: 坂口 忍 (坂口製麺所)
日付: 2026年4月
"""
import os, sys, glob, warnings
import numpy as np
from scipy import stats

# --- matplotlib setup (IPAGothic for Japanese labels) ---
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import font_manager as _fm
for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
            r'C:¥Windows¥Fonts¥msgothic.ttc']:
    try: _fm.fontManager.addfont(_fp)
    except: pass
# Try IPAGothic first, fall back to MS Gothic on Windows
for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
    try:
        plt.rcParams['font.family'] = fontname
        break
    except:
        continue
plt.rcParams['axes.unicode_minus'] = False

warnings.filterwarnings('ignore')

# ============================================================
# 0. Constants
```

```
# ============================================================
a0 = 1.2e-10       # m/s^2  MOND acceleration
G_SI = 6.674e-11   # m^3 kg^-1 s^-2
kpc_m = 3.0857e19  # 1 kpc in meters
Msun = 1.989e30    # kg

# ============================================================
# 1. Paths -- adjust to your local layout
# ============================================================
BASE = os.path.dirname(os.path.abspath(__file__))
ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
TA3    = os.path.join(BASE, 'TA3_gc_independent.csv')

for p, label in [(ROTMOD, 'Rotmod_LTG'), (PHASE1, 'sparc_results.csv'), (TA3, 'TA3_gc_independent.csv')]:
    if not os.path.exists(p):
        print(f'[ERROR] {label} not found: {p}')
        print(f'  -> BASE = {BASE}')
        sys.exit(1)

# ============================================================
# 2. Load pipeline results
# ============================================================
def load_csv_flexible(path):
    """Load CSV handling both comma and whitespace delimiters."""
    with open(path, 'r', encoding='utf-8-sig') as f:
        header = f.readline().strip()
    sep = ',' if ',' in header else None
    data = {}
    with open(path, 'r', encoding='utf-8-sig') as f:
        cols = [c.strip() for c in f.readline().strip().split(sep)]
        rows = []
        for line in f:
            line = line.strip()
            if not line: continue
            parts = line.split(sep)
            rows.append([p.strip() for p in parts])
    for i, col in enumerate(cols):
        vals = []
        for row in rows:
            if i < len(row):
                try: vals.append(float(row[i]))
                except: vals.append(row[i])
            else:
                vals.append(np.nan)
        data[col] = vals
    return data

print('[1] Loading pipeline results...')
phase1 = load_csv_flexible(PHASE1)
ta3    = load_csv_flexible(TA3)

# Build lookup: galaxy_name -> {vflat, upsilon_d, gc_over_a0, ...}
# Identify name column
def find_name_col(data):
    for candidate in ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY']:
        if candidate in data:
            return candidate
    # First column with string values
    for k, v in data.items():
        if isinstance(v[0], str):
            return k
    return list(data.keys())[0]

p1_name_col = find_name_col(phase1)
ta3_name_col = find_name_col(ta3)

galaxy_info = {}
for i, name in enumerate(phase1[p1_name_col]):
    name = str(name).strip()
    info = {}
    for k in phase1:
        if k == p1_name_col: continue
        try: info[k] = float(phase1[k][i])
        except: info[k] = phase1[k][i]
    galaxy_info[name] = info

# Merge TA3 gc_over_a0
for i, name in enumerate(ta3[ta3_name_col]):
    name = str(name).strip()
    if name in galaxy_info:
        for k in ta3:
            if k == ta3_name_col: continue
            try: galaxy_info[name][k] = float(ta3[k][i])
            except: galaxy_info[name][k] = ta3[k][i]

print(f'  phase1: {len(phase1[p1_name_col])} galaxies')
print(f'  TA3:    {len(ta3[ta3_name_col])} galaxies')
```

```
|
| # ============================================================
| # 3. Core computation: f_p from g_N(r) profile
| # ============================================================
| def compute_fp(radii_kpc, vgas, vdisk, vbul, upsilon_d, upsilon_b=0.7):
|     """
|     Compute f_p = mass fraction where g_N(r) < a0.
|
|     For an exponential disk, Sigma(r) ~ exp(-r/hR). We approximate
|     the local mass weight as proportional to |V_bar(r)|^2 * r
|     (since g_N ~ V_bar^2/r, and dM ~ Sigma 2pi r dr ~ g_N r^2 dr / G).
|
|     Actually, the simplest mass weight for a thin disk is:
|       dM/dr = 2 pi r Sigma(r)
|     We approximate Sigma(r) from the disk velocity:
|       V_disk^2(r) = 4 pi G Sigma_0 h_R y^2 [I0 K0 - I1 K1]
|     Since we don't have Sigma(r) directly, we use:
|       g_N(r) = V_bar^2(r) / r   [centripetal acceleration from baryons]
|     and weight by the mass enclosed in each radial annulus.
|
|     For simplicity, we use mass weight ~ |V_disk(r)|^2 as proxy for
|     Sigma(r) (valid for exponential disk).
|
|     Parameters
|     ----------
|     radii_kpc : array, radii in kpc
|     vgas, vdisk, vbul : arrays, velocity components in km/s
|     upsilon_d : float, disk mass-to-light ratio
|     upsilon_b : float, bulge mass-to-light ratio (default 0.7)
|
|     Returns
|     -------
|     f_p : float, plastic mass fraction
|     g_N_array : array, g_N(r) in m/s^2
|     """
|     r_m = radii_kpc * kpc_m  # convert to meters
|
|     # Baryonic velocity: V_bar^2 = Vgas^2 + Yd * Vdisk^2 + Yb * Vbul^2
|     # Note: SPARC files give Vdisk for Yd=1, so scale by sqrt(Yd)
|     vbar2 = (vgas**2
|              + upsilon_d * np.sign(vdisk) * vdisk**2
|              + upsilon_b * np.sign(vbul) * vbul**2)
|     # Handle sign convention: V can be negative (counter-rotating)
|     vbar2 = np.abs(vbar2)
|
|     # g_N(r) = V_bar^2 / r
|     g_N = np.zeros_like(r_m)
|     mask = r_m > 0
|     g_N[mask] = (vbar2[mask] * 1e6) / r_m[mask]  # km/s -> m/s: *1e3, squared: *1e6
|
|     # Mass weight proxy: for exponential disk, dM/dr ~ Sigma(r) * r
|     # Sigma(r) is proportional to the disk contribution:
|     # We use |Vdisk(r)|^2 as proxy for Sigma(r) * r (since Vdisk^2 ~ 4piG Sigma hR * Bessel)
|     # A simpler and more robust proxy: weight = max(|Vdisk|^2, |Vgas|^2) * delta_r
|     weight = np.abs(vdisk**2) * upsilon_d + np.abs(vgas**2)
|     weight = np.maximum(weight, 1e-6)
|
|     # Compute f_p = sum(weight where g_N < a0) / sum(weight)
|     plastic_mask = g_N < a0
|     total_weight = np.sum(weight)
|     if total_weight <= 0:
|         return np.nan, g_N
|
|     f_p = np.sum(weight[plastic_mask]) / total_weight
|     return f_p, g_N
|
|
| def load_rotmod(filepath):
|     """Load SPARC rotation curve file."""
|     rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = [], [], [], [], [], [], [], []
|     with open(filepath, 'r') as f:
|         for line in f:
|             line = line.strip()
|             if not line or line.startswith('#'): continue
|             parts = line.split()
|             if len(parts) < 6: continue
|             try:
|                 rad.append(float(parts[0]))
|                 vobs.append(float(parts[1]))
|                 errv.append(float(parts[2]))
|                 vgas.append(float(parts[3]))
|                 vdisk.append(float(parts[4]))
|                 vbul.append(float(parts[5]))
|                 if len(parts) > 6: sbdisk.append(float(parts[6]))
|                 else: sbdisk.append(0.0)
|                 if len(parts) > 7: sbbul.append(float(parts[7]))
|                 else: sbbul.append(0.0)
|             except ValueError:
```

```
                continue
    return (np.array(rad), np.array(vobs), np.array(errv),
            np.array(vgas), np.array(vdisk), np.array(vbul),
            np.array(sbdisk), np.array(sbbul))


# ============================================================
# 4. Process all galaxies
# ============================================================
print('[2] Processing SPARC galaxies...')

results = []
rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
print(f'  Found {len(rotmod_files)} rotmod files')

# Detect key column names in phase1
def get_key(info, candidates, default=None):
    for c in candidates:
        if c in info:
            return info[c]
    return default

for fpath in rotmod_files:
    gname = os.path.splitext(os.path.basename(fpath))[0]
    gname_clean = gname.replace('_rotmod', '').strip()

    # Find matching pipeline entry
    info = None
    for key in [gname_clean, gname, gname_clean.upper(), gname_clean.lower()]:
        if key in galaxy_info:
            info = galaxy_info[key]
            break
    if info is None:
        continue

    # Get Yd (optimal) and gc
    ud = get_key(info, ['upsilon_d', 'Upsilon_d', 'Ud', 'ud', 'Yd'], None)
    gc_a0 = get_key(info, ['gc_over_a0', 'gc/a0', 'gc_ratio'], None)
    vflat_val = get_key(info, ['vflat', 'Vflat', 'v_flat'], None)

    if ud is None or gc_a0 is None or vflat_val is None:
        continue

    try:
        ud = float(ud)
        gc_a0 = float(gc_a0)
        vflat_val = float(vflat_val)
    except (ValueError, TypeError):
        continue

    if np.isnan(ud) or np.isnan(gc_a0) or gc_a0 <= 0:
        continue

    # Load rotation curve
    try:
        rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
    except:
        continue

    if len(rad) < 3:
        continue

    # Compute f_p
    f_p, g_N = compute_fp(rad, vgas, vdisk, vbul, ud)
    if np.isnan(f_p):
        continue

    c_membrane = 1.0 - f_p

    # Compute G*Sigma0 proxy = vflat^2 / hR
    # Estimate hR from rpk / 2.15 (pipeline convention)
    vds = np.sqrt(np.maximum(ud, 0.01)) * np.abs(vdisk)
    rpk_idx = np.argmax(vds)
    rpk = rad[rpk_idx]
    if rpk < 0.01 or rpk >= rad.max() * 0.9:
        # Edge filter (same as pipeline)
        continue
    hR = rpk / 2.15  # kpc
    hR_m = hR * kpc_m
    vflat_ms = vflat_val * 1e3
    GS0 = vflat_ms**2 / hR_m  # m/s^2

    # Theoretical prediction: c ~ sqrt(G*Sigma0 / a0)
    sqrt_ratio = np.sqrt(GS0 / a0)

    results.append({
        'galaxy': gname_clean,
        'f_p': f_p,
```

```
        'c_membrane': c_membrane,
        'gc_over_a0': gc_a0,
        'log_gc_a0': np.log10(gc_a0),
        'vflat': vflat_val,
        'Yd': ud,
        'hR_kpc': hR,
        'GS0': GS0,
        'log_GS0_a0': np.log10(GS0 / a0),
        'sqrt_GS0_a0': sqrt_ratio,
        'n_points': len(rad),
        'g_N_max': np.max(g_N),
        'g_N_min': np.min(g_N[g_N > 0]) if np.any(g_N > 0) else 0,
    })

N = len(results)
print(f'  Successfully processed: {N} galaxies')

if N < 10:
    print('[ERROR] Too few galaxies. Check file paths and column names.')
    print(f'  Sample galaxy_info keys: {list(galaxy_info.keys())[:5]}')
    if galaxy_info:
        sample = list(galaxy_info.values())[0]
        print(f'  Sample info keys: {list(sample.keys())}')
    sys.exit(1)

# ============================================================
# 5. Statistical analysis
# ============================================================
print('\n' + '='*60)
print('V-1: f_p vs gc correlation analysis')
print('='*60)

fp_arr    = np.array([r['f_p'] for r in results])
c_arr     = np.array([r['c_membrane'] for r in results])
gc_arr    = np.array([r['gc_over_a0'] for r in results])
log_gc    = np.array([r['log_gc_a0'] for r in results])
sqrt_arr  = np.array([r['sqrt_GS0_a0'] for r in results])
log_GS0   = np.array([r['log_GS0_a0'] for r in results])

# --- Test 1: f_p vs log(gc/a0) ---
rho1, p1 = stats.spearmanr(fp_arr, log_gc)
print(f'\n[Test 1] f_p vs log(gc/a0):')
print(f'  Spearman rho = {rho1:.3f}, p = {p1:.2e}')
print(f'  Expected: strong negative correlation (rho < -0.6)')

# --- Test 2: c = 1-f_p vs gc/a0 ---
rho2, p2 = stats.spearmanr(c_arr, gc_arr)
print(f'\n[Test 2] c = 1-f_p vs gc/a0:')
print(f'  Spearman rho = {rho2:.3f}, p = {p2:.2e}')

# --- Test 3: c vs sqrt(G*Sigma0/a0) -- the key prediction ---
rho3, p3 = stats.spearmanr(c_arr, sqrt_arr)
print(f'\n[Test 3] c = 1-f_p vs sqrt(G*Sigma0/a0):')
print(f'  Spearman rho = {rho3:.3f}, p = {p3:.2e}')
print(f'  This is the KEY test: c should scale as sqrt(G*Sigma0/a0)')

# --- Test 4: log(c) vs 0.5*log(G*Sigma0/a0) -- slope test ---
valid = (c_arr > 0) & (sqrt_arr > 0)
log_c = np.log10(c_arr[valid])
log_sqrt = np.log10(sqrt_arr[valid])
log_GS0_v = log_GS0[valid]

slope, intercept, r_val, p_slope, se_slope = stats.linregress(log_GS0_v, log_c)
print(f'\n[Test 4] log(c) vs log(G*Sigma0/a0) regression:')
print(f'  slope = {slope:.3f} +/- {se_slope:.3f}')
print(f'  Expected slope = 0.5 (sqrt scaling)')
print(f'  p(slope=0.5) = {2*(1-stats.t.cdf(abs(slope-0.5)/se_slope, df=sum(valid)-2)):.3f}')
print(f'  R^2 = {r_val**2:.3f}')
print(f'  intercept = {intercept:.3f} (= log(eta) effectively)')

# --- Derived eta ---
eta_eff = 10**intercept
print(f'\n[Derived] eta_eff = 10^{intercept:.3f} = {eta_eff:.3f}')

# --- Summary statistics ---
print(f'\n--- Summary ---')
print(f'N = {N}')
print(f'f_p: median = {np.median(fp_arr):.3f}, '
      f'IQR = [{np.percentile(fp_arr,25):.3f}, {np.percentile(fp_arr,75):.3f}]')
print(f'c = 1-f_p: median = {np.median(c_arr):.3f}, '
      f'IQR = [{np.percentile(c_arr,25):.3f}, {np.percentile(c_arr,75):.3f}]')
print(f'gc/a0: median = {np.median(gc_arr):.3f}')

# Binney & Tremaine check: g_peak vs 1.8*G*Sigma0
g_N_max_arr = np.array([r['g_N_max'] for r in results])
GS0_arr = np.array([r['GS0'] for r in results])
ratio_peak = g_N_max_arr / GS0_arr
print(f'\n[BT check] g_N_max / (G*Sigma0): median = {np.median(ratio_peak):.2f} (expect ~1.8)')
```

```
|
| # ============================================================
| # 6. Figures
| # ============================================================
| print('¥n[3] Generating figures...')
|
| # --- Fig 1: f_p vs log(gc/a0) ---
| fig, ax = plt.subplots(1, 1, figsize=(7, 5))
| ax.scatter(fp_arr, log_gc, s=12, alpha=0.6, c=log_GS0, cmap='viridis', edgecolors='none')
| ax.set_xlabel('f_p  (g_N < a0 mass fraction)', fontsize=11)
| ax.set_ylabel('log(gc / a0)', fontsize=11)
| ax.set_title(f'V-1: f_p vs gc  (N={N}, rho={rho1:.3f}, p={p1:.1e})', fontsize=12)
| cb = plt.colorbar(ax.collections[0], ax=ax, label='log(G*Sigma0/a0)')
| # Add trend line
| z = np.polyfit(fp_arr, log_gc, 1)
| xline = np.linspace(0, 1, 100)
| ax.plot(xline, np.polyval(z, xline), 'r--', lw=1.5, label=f'linear fit: slope={z[0]:.2f}')
| ax.legend(fontsize=9)
| ax.grid(True, alpha=0.3)
| fig.tight_layout()
| fig.savefig(os.path.join(BASE, 'fig_fp_vs_gc.png'), dpi=150)
| print('  -> fig_fp_vs_gc.png')
|
| # --- Fig 2: c = 1-f_p vs sqrt(G*Sigma0/a0) with slope test ---
| fig, axes = plt.subplots(1, 2, figsize=(13, 5))
|
| ax = axes[0]
| ax.scatter(sqrt_arr, c_arr, s=12, alpha=0.6, c='steelblue', edgecolors='none')
| ax.set_xlabel('sqrt(G*Sigma0 / a0)', fontsize=11)
| ax.set_ylabel('c = 1 - f_p', fontsize=11)
| ax.set_title(f'c vs sqrt scaling  (rho={rho3:.3f})', fontsize=12)
| # 1:1 reference (with eta normalization)
| xref = np.linspace(0, np.max(sqrt_arr)*1.1, 100)
| ax.plot(xref, eta_eff * xref, 'r-', lw=1.5, label=f'c = {eta_eff:.2f} * sqrt(GS0/a0)')
| ax.legend(fontsize=9)
| ax.grid(True, alpha=0.3)
|
| ax = axes[1]
| ax.scatter(log_GS0_v, log_c, s=12, alpha=0.6, c='darkorange', edgecolors='none')
| ax.set_xlabel('log(G*Sigma0 / a0)', fontsize=11)
| ax.set_ylabel('log(c)', fontsize=11)
| ax.set_title(f'Slope test: {slope:.3f} +/- {se_slope:.3f} (expect 0.5)', fontsize=12)
| xfit = np.linspace(log_GS0_v.min(), log_GS0_v.max(), 100)
| ax.plot(xfit, slope*xfit + intercept, 'r-', lw=1.5, label=f'fit: slope={slope:.3f}')
| ax.plot(xfit, 0.5*xfit + intercept, 'b--', lw=1.0, label='theory: slope=0.5')
| ax.legend(fontsize=9)
| ax.grid(True, alpha=0.3)
|
| fig.tight_layout()
| fig.savefig(os.path.join(BASE, 'fig_c_vs_sqrt_scaling.png'), dpi=150)
| print('  -> fig_c_vs_sqrt_scaling.png')
|
| # --- Fig 3: 4-panel summary ---
| fig, axes = plt.subplots(2, 2, figsize=(12, 10))
|
| # Panel A: f_p distribution
| ax = axes[0, 0]
| ax.hist(fp_arr, bins=30, color='steelblue', alpha=0.7, edgecolor='white')
| ax.axvline(np.median(fp_arr), color='red', ls='--', label=f'median={np.median(fp_arr):.2f}')
| ax.set_xlabel('f_p', fontsize=11)
| ax.set_ylabel('Count', fontsize=11)
| ax.set_title('A: f_p distribution', fontsize=12)
| ax.legend()
|
| # Panel B: c vs gc/a0
| ax = axes[0, 1]
| ax.scatter(c_arr, gc_arr, s=12, alpha=0.6, c='steelblue', edgecolors='none')
| ax.set_xlabel('c = 1 - f_p', fontsize=11)
| ax.set_ylabel('gc / a0', fontsize=11)
| ax.set_title(f'B: c vs gc/a0  (rho={rho2:.3f})', fontsize=12)
| # Plot c*a0 line (gc = c*a0 means gc/a0 = c)
| xline = np.linspace(0, 1, 100)
| ax.plot(xline, xline, 'r--', lw=1, label='gc/a0 = c (exact match)')
| ax.legend(fontsize=9)
| ax.grid(True, alpha=0.3)
|
| # Panel C: g_peak / G*Sigma0 distribution (BT check)
| ax = axes[1, 0]
| ax.hist(ratio_peak[np.isfinite(ratio_peak)], bins=30, color='darkorange', alpha=0.7, edgecolor='white')
| ax.axvline(1.8, color='red', ls='--', lw=2, label='BT prediction: 1.8')
| ax.axvline(np.median(ratio_peak), color='blue', ls='--', label=f'median={np.median(ratio_peak):.2f}')
| ax.set_xlabel('g_N_max / (G*Sigma0)', fontsize=11)
| ax.set_ylabel('Count', fontsize=11)
| ax.set_title('C: Binney-Tremaine check', fontsize=12)
| ax.legend(fontsize=9)
|
| # Panel D: residual from sqrt law
| ax = axes[1, 1]
```

```
if np.sum(valid) > 5:
    residual = log_c - (0.5 * log_GS0_v + intercept)
    ax.hist(residual, bins=25, color='green', alpha=0.7, edgecolor='white')
    ax.axvline(0, color='red', ls='--')
    ax.set_xlabel('log(c) - [0.5*log(GS0/a0) + const]', fontsize=11)
    ax.set_ylabel('Count', fontsize=11)
    ax.set_title(f'D: Residual from sqrt law (sigma={np.std(residual):.3f} dex)', fontsize=12)

fig.suptitle(f'V-1: Condition-14 Mechanism Verification (N={N})', fontsize=14, y=1.01)
fig.tight_layout()
fig.savefig(os.path.join(BASE, 'fig_fp_mechanism_summary.png'), dpi=150)
print('  -> fig_fp_mechanism_summary.png')

# ============================================================
# 7. Save results
# ============================================================
outcsv = os.path.join(BASE, 'fp_verification_results.csv')
with open(outcsv, 'w', encoding='utf-8') as f:
    cols = ['galaxy', 'f_p', 'c_membrane', 'gc_over_a0', 'log_gc_a0',
            'vflat', 'Yd', 'hR_kpc', 'GS0', 'log_GS0_a0', 'sqrt_GS0_a0',
            'g_N_max', 'g_N_min', 'n_points']
    f.write(','.join(cols) + '¥n')
    for r in results:
        f.write(','.join(str(r[c]) for c in cols) + '¥n')
print(f'¥n[4] Results saved: {outcsv}')

# ============================================================
# 8. Final verdict
# ============================================================
print('¥n' + '='*60)
print('VERDICT')
print('='*60)

verdict_fp_gc = 'PASS' if (rho1 < -0.5 and p1 < 0.001) else 'FAIL'
verdict_slope = 'PASS' if abs(slope - 0.5) < 2*se_slope else 'MARGINAL' if abs(slope-0.5) < 3*se_slope else 'FAIL'
verdict_bt = 'PASS' if 1.0 < np.median(ratio_peak) < 3.0 else 'CHECK'

print(f'  [V-1a] f_p vs gc correlation:      {verdict_fp_gc}  (rho={rho1:.3f}, p={p1:.1e})')
print(f'  [V-1b] sqrt scaling slope:          {verdict_slope}  (slope={slope:.3f}+/-{se_slope:.3f}, expect 0.5)')
print(f'  [V-1c] BT coefficient check:        {verdict_bt}  (median ratio={np.median(ratio_peak):.2f}, expect ~1.8)')
print(f'  [V-2]  eta effective:               {eta_eff:.3f}')
print()

if verdict_fp_gc == 'PASS' and verdict_slope in ('PASS', 'MARGINAL'):
    print('  >>> Condition-14 mechanism SUPPORTED: f_p determines gc via sqrt scaling')
    print(f'  >>> c = 1-f_p ~ {eta_eff:.2f} * sqrt(G*Sigma0/a0)')
else:
    print('  >>> Results require further investigation')
    print(f'  >>> f_p-gc: {verdict_fp_gc}, slope: {verdict_slope}')

print('¥n[DONE]')
```

2. [V-1改] gc vs 真のバリオン面密度 Sigma_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 ~ Sigma_bar^(1/3) を、BT式で直接計算した Sigma_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 vflat^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 Sigma_bar に適合するか

結論

FAIL: Sigma_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

```
# -*- coding: utf-8 -*-
"""
V-1改: 条件14 機構再検討 -- gc ∝ Sigma_bar^(1/3) の検証
======================================================================
幾何平均法則 gc = eta*sqrt(a0*GS0_proxy) と BTFR v_flat^4 = gc*G*M_bar
を組み合わせると、真のバリオン面密度 Sigma_bar との関係は:

    gc = eta' * a0^(2/3) * (G*Sigma_bar)^(1/3)

すなわち log(gc/a0) vs log(G*Sigma_bar/a0) のスロープ = 1/3 が予測される。

本スクリプトは SPARC 175銀河でこれを検証する。

実行: uv run --with scipy --with matplotlib python sparc_fp_v1kai.py

必要ファイル:
  - Rotmod_LTG/*.dat
  - phase1/sparc_results.csv
  - TA3_gc_independent.csv

出力:
  - fig_v1kai_slope_test.png       (主要結果: 1/3スロープ検定)
  - fig_v1kai_4panel.png           (4パネルサマリー)
  - v1kai_results.csv
  - コンソールに統計サマリー

著者: 坂口 忍（坂口製麺所）
日付: 2026年4月
"""
import os, sys, glob, warnings
import numpy as np
from scipy import stats

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import font_manager as _fm
for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
            r'C:¥Windows¥Fonts¥msgothic.ttc']:
    try: _fm.fontManager.addfont(_fp)
    except: pass
for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
    try:
        plt.rcParams['font.family'] = fontname
        break
    except:
        continue
```

```
plt.rcParams['axes.unicode_minus'] = False
warnings.filterwarnings('ignore')

# ============================================================
# 0. Constants
# ============================================================
a0      = 1.2e-10       # m/s^2
G_SI    = 6.674e-11     # m^3 kg^-1 s^-2
kpc_m   = 3.0857e19     # m
Msun    = 1.989e30      # kg
pc_m    = 3.0857e16     # m

# Milgrom critical surface density
Sigma_crit = a0 / G_SI  # kg/m^2
Sigma_crit_Msun_pc2 = Sigma_crit / Msun * pc_m**2
print(f'Milgrom critical surface density: {Sigma_crit_Msun_pc2:.0f} M_sun/pc^2')

# ============================================================
# 1. Paths
# ============================================================
BASE   = os.path.dirname(os.path.abspath(__file__))
ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
TA3    = os.path.join(BASE, 'TA3_gc_independent.csv')

for p, label in [(ROTMOD, 'Rotmod_LTG'), (PHASE1, 'sparc_results.csv'),
                 (TA3, 'TA3_gc_independent.csv')]:
    if not os.path.exists(p):
        print(f'[ERROR] {label} not found: {p}')
        sys.exit(1)

# ============================================================
# 2. Load CSV helper
# ============================================================
def load_csv(path):
    with open(path, 'r', encoding='utf-8-sig') as f:
        header = f.readline().strip()
    sep = ',' if ',' in header else None
    data = {}
    with open(path, 'r', encoding='utf-8-sig') as f:
        cols = [c.strip() for c in f.readline().strip().split(sep)]
        rows = []
        for line in f:
            line = line.strip()
            if not line:
                continue
            rows.append([p.strip() for p in line.split(sep)])
    for i, col in enumerate(cols):
        vals = []
        for row in rows:
            if i < len(row):
                try:    vals.append(float(row[i]))
                except: vals.append(row[i])
            else:
                vals.append(np.nan)
        data[col] = vals
    return data

def find_name_col(data):
    for c in ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY']:
        if c in data:
            return c
    for k, v in data.items():
        if isinstance(v[0], str):
            return k
    return list(data.keys())[0]

def get_key(info, candidates, default=None):
    for c in candidates:
        if c in info:
            try:    return float(info[c])
            except: return info[c]
    return default

# ============================================================
# 3. Load pipeline data
# ============================================================
print('[1] Loading pipeline data...')
phase1 = load_csv(PHASE1)
ta3    = load_csv(TA3)

p1_nc = find_name_col(phase1)
ta3_nc = find_name_col(ta3)

galaxy_info = {}
for i, name in enumerate(phase1[p1_nc]):
    name = str(name).strip()
    info = {}
```

```
    for k in phase1:
        if k == p1_nc: continue
        try:    info[k] = float(phase1[k][i])
        except: info[k] = phase1[k][i]
    galaxy_info[name] = info

for i, name in enumerate(ta3[ta3_nc]):
    name = str(name).strip()
    if name in galaxy_info:
        for k in ta3:
            if k == ta3_nc: continue
            try:    galaxy_info[name][k] = float(ta3[k][i])
            except: galaxy_info[name][k] = ta3[k][i]

print(f'  phase1: {len(phase1[p1_nc])} galaxies')
print(f'  TA3:    {len(ta3[ta3_nc])} galaxies')

# ============================================================
# 4. Load rotmod and compute Sigma_bar
# ============================================================
def load_rotmod(filepath):
    cols = [[] for _ in range(8)]
    with open(filepath, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith('#'):
                continue
            parts = line.split()
            if len(parts) < 6:
                continue
            try:
                for j in range(min(len(parts), 8)):
                    cols[j].append(float(parts[j]))
                for j in range(len(parts), 8):
                    cols[j].append(0.0)
            except ValueError:
                continue
    return tuple(np.array(c) for c in cols)


print('[2] Processing galaxies...')
results = []
rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
print(f'  Found {len(rotmod_files)} rotmod files')

for fpath in rotmod_files:
    gname = os.path.splitext(os.path.basename(fpath))[0]
    gname_clean = gname.replace('_rotmod', '').strip()

    info = None
    for key in [gname_clean, gname, gname_clean.upper(), gname_clean.lower()]:
        if key in galaxy_info:
            info = galaxy_info[key]
            break
    if info is None:
        continue

    # Required quantities
    ud      = get_key(info, ['upsilon_d', 'Upsilon_d', 'Ud', 'ud', 'Yd'])
    gc_a0   = get_key(info, ['gc_over_a0', 'gc/a0', 'gc_ratio'])
    vflat   = get_key(info, ['vflat', 'Vflat', 'v_flat'])

    if ud is None or gc_a0 is None or vflat is None:
        continue
    if np.isnan(ud) or np.isnan(gc_a0) or gc_a0 <= 0 or vflat <= 0:
        continue

    # Load rotation curve
    try:
        rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
    except:
        continue
    if len(rad) < 3:
        continue

    # --- Compute h_R from r_pk (pipeline convention) ---
    vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
    rpk_idx = np.argmax(vds)
    rpk = rad[rpk_idx]
    if rpk < 0.01 or rpk >= rad.max() * 0.9:
        continue
    hR_kpc = rpk / 2.15

    # --- Method A: Sigma_bar from SBdisk (if available) ---
    # SBdisk in SPARC is face-on surface brightness in L_sun/pc^2 (3.6um)
    # Sigma_bar = Yd * SBdisk_central
    # SBdisk in rotmod is mag/arcsec^2 at each radius.
    # Central SB = extrapolation or use max(SBdisk)
```

```
    # Actually: SBdisk column in rotmod is in L_sun/pc^2
    # Use the disk luminosity profile to get total L_disk
    #
    # --- Method B: Sigma_bar from Vdisk profile (more robust) ---
    # For exponential disk: V_disk_peak^2 ~ pi*G*Sigma0_bar * hR * f(2.2)
    # f(2.2) = 2.2^2 * [I0*K0 - I1*K1] evaluated at y=1.1
    # This gives Sigma0_bar from V_disk_peak directly.
    # But simpler: M_disk = integral of disk mass
    # M_disk ~ Yd * L_disk
    # Use: V_disk^2(r) relates to Sigma(r) directly
    #
    # --- Method C (cleanest): Use vflat + hR to get GS0_proxy,
    #     then compute Sigma_bar by removing gc contamination ---
    # GS0_proxy = vflat^2 / hR  [this is what the geometric mean law uses]
    # From the derivation: GS0_proxy = sqrt(2pi * gc * G * Sigma_bar)
    # So: G*Sigma_bar = GS0_proxy^2 / (2pi * gc)
    #                 = (vflat^2/hR)^2 / (2pi * gc_a0 * a0)
    #
    hR_m = hR_kpc * kpc_m
    vflat_ms = vflat * 1e3  # km/s -> m/s
    GS0_proxy = vflat_ms**2 / hR_m  # m/s^2
    #
    gc = gc_a0 * a0  # m/s^2
    #
    # Method C: derived from BTFR decontamination
    G_Sigma_bar_C = GS0_proxy**2 / (2 * np.pi * gc)  # m/s^2
    #
    # --- Method D (fully independent): Sigma_bar from V_disk peak ---
    # V_disk_peak = max(sqrt(Yd) * |Vdisk|)
    # For exponential disk: V_disk_peak^2 = 0.56 * pi * G * Sigma0_bar * hR
    #   (Binney & Tremaine eq 2.165, evaluated at r=2.2hR)
    # => G * Sigma0_bar = V_disk_peak^2 / (0.56 * pi * hR)
    vdisk_peak = np.max(np.sqrt(max(ud, 0.01)) * np.abs(vdisk))
    vdisk_peak_ms = vdisk_peak * 1e3
    G_Sigma0_bar_D = vdisk_peak_ms**2 / (0.56 * np.pi * hR_m)  # m/s^2
    #
    # --- Method E: total Sigma_bar = disk + gas ---
    # Gas contribution: V_gas_peak
    vgas_peak_ms = np.max(np.abs(vgas)) * 1e3
    # Approximate total baryonic: add gas disk
    # G_Sigma_bar_total ~ G_Sigma0_bar_D + G_Sigma_gas
    # For gas: same exponential approximation
    G_Sigma_gas = vgas_peak_ms**2 / (0.56 * np.pi * hR_m) if vgas_peak_ms > 0 else 0
    G_Sigma_bar_E = G_Sigma0_bar_D + G_Sigma_gas
    #
    # Store all methods
    results.append({
        'galaxy':         gname_clean,
        'gc_over_a0':     gc_a0,
        'log_gc_a0':      np.log10(gc_a0),
        'vflat':          vflat,
        'Yd':             ud,
        'hR_kpc':         hR_kpc,
        'GS0_proxy':      GS0_proxy,
        'log_GS0_proxy':  np.log10(GS0_proxy / a0),
        # Method C: BTFR decontaminated
        'G_Sbar_C':       G_Sigma_bar_C,
        'log_GSbar_C':    np.log10(G_Sigma_bar_C / a0) if G_Sigma_bar_C > 0 else np.nan,
        # Method D: V_disk peak (BT formula)
        'G_Sbar_D':       G_Sigma0_bar_D,
        'log_GSbar_D':    np.log10(G_Sigma0_bar_D / a0) if G_Sigma0_bar_D > 0 else np.nan,
        # Method E: disk + gas
        'G_Sbar_E':       G_Sigma_bar_E,
        'log_GSbar_E':    np.log10(G_Sigma_bar_E / a0) if G_Sigma_bar_E > 0 else np.nan,
        # For diagnostics
        'vdisk_peak':     vdisk_peak,
        'vgas_peak':      np.max(np.abs(vgas)),
    })

N = len(results)
print(f'  Processed: {N} galaxies')

if N < 10:
    print('[ERROR] Too few galaxies.')
    sys.exit(1)

# ============================================================
# 5. Statistical analysis
# ============================================================
log_gc = np.array([r['log_gc_a0'] for r in results])

# --- Method D: V_disk peak (fully independent of gc) ---
log_GSbar_D = np.array([r['log_GSbar_D'] for r in results])
valid_D = np.isfinite(log_GSbar_D) & np.isfinite(log_gc)

# --- Method E: disk + gas ---
log_GSbar_E = np.array([r['log_GSbar_E'] for r in results])
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 = vflat^2/hR', 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 |
```

```
for ax, logx, mask, reg, title, expected in [
    (axes[0], log_GS0, valid_0, r0,
     'Proxy: vflat^2/hR (baseline)', 0.5),
    (axes[1], log_GSbar_D, valid_D, rD,
     'V_disk peak (BT, independent)', 1/3),
    (axes[2], log_GSbar_E, valid_E, rE,
     'Disk+Gas (BT, independent)', 1/3),
]:
    if reg is None:
        continue
    xv, yv = logx[mask], log_gc[mask]
    ax.scatter(xv, yv, s=10, alpha=0.5, c='steelblue', edgecolors='none')
    xfit = np.linspace(xv.min(), xv.max(), 100)
    ax.plot(xfit, reg['slope']*xfit + reg['intercept'], 'r-', lw=2,
            label=f'fit: slope={reg["slope"]:.3f}+/-{reg["se"]:.3f}')
    ax.plot(xfit, expected*xfit + reg['intercept'], 'b--', lw=1.5,
            label=f'theory: slope={expected:.3f}')
    ax.set_xlabel('log(G*Sigma / a0)', fontsize=10)
    ax.set_ylabel('log(gc / a0)', fontsize=10)
    ax.set_title(f'{title}¥np(slope={expected})={reg["p_slope"]:.3f} [{reg["verdict"]}]',
                 fontsize=10)
    ax.legend(fontsize=8, loc='upper left')
    ax.grid(True, alpha=0.3)

fig.suptitle(f'V-1 revised: gc vs Sigma_bar scaling (N={N})', fontsize=13)
fig.tight_layout()
fig.savefig(os.path.join(BASE, 'fig_v1kai_slope_test.png'), dpi=150)
print('  -> fig_v1kai_slope_test.png')

# --- Fig 2: 4-panel summary ---
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# A: Sigma_bar distribution
ax = axes[0, 0]
vals = log_GSbar_D[valid_D]
ax.hist(vals, bins=30, color='steelblue', alpha=0.7, edgecolor='white')
ax.axvline(0, color='red', ls='--', lw=2, label='G*Sigma_bar = a0')
ax.set_xlabel('log(G*Sigma_bar / a0)', fontsize=10)
ax.set_ylabel('Count', fontsize=10)
ax.set_title('A: True baryon surface density', fontsize=11)
ax.legend(fontsize=9)

# B: slope comparison across methods
ax = axes[0, 1]
methods = []
slopes  = []
errors  = []
colors_bar = []
for label, reg, col in [
    ('Proxy¥n(vflat^2/hR)', r0, 'steelblue'),
    ('Vdisk peak¥n(BT)', rD, 'darkorange'),
    ('Disk+Gas¥n(BT)', rE, 'green'),
    ('BTFR decontam¥n(circular)', rC, 'grey'),
]:
    if reg is not None:
        methods.append(label)
        slopes.append(reg['slope'])
        errors.append(reg['se'])
        colors_bar.append(col)

x_pos = np.arange(len(methods))
ax.bar(x_pos, slopes, yerr=errors, color=colors_bar, alpha=0.7,
       capsize=5, edgecolor='black', linewidth=0.5)
ax.axhline(0.5, color='blue', ls='--', lw=1.5, label='0.5 (proxy prediction)')
ax.axhline(1/3, color='red', ls='--', lw=1.5, label='1/3 (Sigma_bar prediction)')
ax.set_xticks(x_pos)
ax.set_xticklabels(methods, fontsize=8)
ax.set_ylabel('Slope', fontsize=10)
ax.set_title('B: Slope comparison', fontsize=11)
ax.legend(fontsize=8)
ax.grid(True, alpha=0.3, axis='y')

# C: Residuals from 1/3 law (Method D)
ax = axes[1, 0]
if rD is not None:
    xv = log_GSbar_D[valid_D]
    yv = log_gc[valid_D]
    resid = yv - (1/3 * xv + rD['intercept'])
    ax.hist(resid, bins=25, color='darkorange', alpha=0.7, edgecolor='white')
    ax.axvline(0, color='red', ls='--')
    ax.set_xlabel('Residual from 1/3 law [dex]', fontsize=10)
    ax.set_ylabel('Count', fontsize=10)
    ax.set_title(f'C: Residual (sigma={np.std(resid):.3f} dex)', fontsize=11)

# D: Residuals from 0.5 law (proxy, for comparison)
ax = axes[1, 1]
if r0 is not None:
    xv = log_GS0[valid_0]
```

```
    yv = log_gc[valid_0]
    resid0 = yv - (0.5 * xv + r0['intercept'])
    ax.hist(resid0, bins=25, color='steelblue', alpha=0.7, edgecolor='white')
    ax.axvline(0, color='red', ls='--')
    ax.set_xlabel('Residual from 0.5 law (proxy) [dex]', fontsize=10)
    ax.set_ylabel('Count', fontsize=10)
    ax.set_title(f'D: Proxy residual (sigma={np.std(resid0):.3f} dex)', fontsize=11)

fig.suptitle(f'V-1 revised summary (N={N})', fontsize=13, y=1.01)
fig.tight_layout()
fig.savefig(os.path.join(BASE, 'fig_v1kai_4panel.png'), dpi=150)
print('  -> fig_v1kai_4panel.png')

# ============================================================
# 7. Save CSV
# ============================================================
outcsv = os.path.join(BASE, 'v1kai_results.csv')
cols_out = ['galaxy', 'gc_over_a0', 'log_gc_a0', 'vflat', 'Yd', 'hR_kpc',
            'GS0_proxy', 'log_GS0_proxy',
            'G_Sbar_D', 'log_GSbar_D', 'G_Sbar_E', 'log_GSbar_E',
            'G_Sbar_C', 'log_GSbar_C',
            'vdisk_peak', 'vgas_peak']
with open(outcsv, 'w', encoding='utf-8') as f:
    f.write(','.join(cols_out) + '¥n')
    for r in results:
        f.write(','.join(str(r.get(c, '')) for c in cols_out) + '¥n')
print(f'¥n[4] Saved: {outcsv}')

# ============================================================
# 8. Final Summary
# ============================================================
print('¥n' + '='*70)
print('FINAL SUMMARY')
print('='*70)

print('¥n  Method                      | slope   +/- SE    | expected | p(match) | R^2    | verdict')
print('  ' + '-'*95)
for label, reg, expected in [
    ('Proxy vflat^2/hR (baseline)', r0, 0.5),
    ('Vdisk peak BT (independent)', rD, 1/3),
    ('Disk+Gas BT (independent)',    rE, 1/3),
    ('BTFR decontam (circular)',     rC, 1/3),
    ('Vdisk peak BT vs 0.5',        rD5, 0.5),
    ('Disk+Gas BT vs 0.5',          rE5, 0.5),
]:
    if reg is None:
        continue
    print(f'  {label:30s} | {reg["slope"]:.4f} +/- {reg["se"]:.4f} | '
          f'{expected:.4f}   | {reg["p_slope"]:.4f}   | '
          f'{reg["r2"]:.4f} | {reg["verdict"]}')

print('¥n  Key question: Is the slope with TRUE Sigma_bar closer to 1/3 or 0.5?')
if rD is not None and rD5 is not None:
    if rD['p_slope'] > rD5['p_slope']:
        print('  >>> 1/3 is BETTER fit for true Sigma_bar')
    elif rD5['p_slope'] > rD['p_slope']:
        print('  >>> 0.5 is BETTER fit for true Sigma_bar')
    else:
        print('  >>> Inconclusive')

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 = vflat^2/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-1c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), R^2=0.529。p(vflat=1.0)=0.24, p(hR=-0.5)=0.49。M_bar直接 R^2=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

```
# -*- coding: utf-8 -*-
"""
V-1b: gc vs M* 直接回帰 -- gcの駆動力はバリオン総質量か？
================================================================
V-1c改で Sigma_bar との相関が崩壊（R^2=0.016）したのに対し、
T-5では log M* が gc と最強相関（r=+0.80）を示す。

本スクリプトは gc vs M* の直接回帰を行い:
  (1) gc ∝ M*^beta のbetaを推定
  (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
  (3) hR の寄与を分離（M* と hR の独立な効果）

実行: uv run --with scipy --with matplotlib python sparc_gc_vs_Mstar.py

必要ファイル:
  - Rotmod_LTG/*.dat
  - phase1/sparc_results.csv
  - TA3_gc_independent.csv

著者: 坂口 忍（坂口製麺所）
日付: 2026年4月
"""
import os, sys, glob, warnings
import numpy as np
from scipy import stats

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import font_manager as _fm
for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
            r'C:¥Windows¥Fonts¥msgothic.ttc']:
    try: _fm.fontManager.addfont(_fp)
    except: pass
for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
    try:
        plt.rcParams['font.family'] = fontname
        break
    except: continue
plt.rcParams['axes.unicode_minus'] = False
```

```
warnings.filterwarnings('ignore')

# ============================================================
# Constants
# ============================================================
a0     = 1.2e-10
G_SI   = 6.674e-11
kpc_m  = 3.0857e19
Msun   = 1.989e30
pc_m   = 3.0857e16

# ============================================================
# Paths
# ============================================================
BASE   = os.path.dirname(os.path.abspath(__file__))
ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
TA3    = os.path.join(BASE, 'TA3_gc_independent.csv')

for p, label in [(ROTMOD,'Rotmod_LTG'),(PHASE1,'sparc_results.csv'),(TA3,'TA3_gc_independent.csv')]:
    if not os.path.exists(p):
        print(f'[ERROR] {label} not found: {p}'); sys.exit(1)

# ============================================================
# CSV loader
# ============================================================
def load_csv(path):
    with open(path, 'r', encoding='utf-8-sig') as f:
        header = f.readline().strip()
    sep = ',' if ',' in header else None
    data = {}
    with open(path, 'r', encoding='utf-8-sig') as f:
        cols = [c.strip() for c in f.readline().strip().split(sep)]
        rows = []
        for line in f:
            line = line.strip()
            if not line: continue
            rows.append([p.strip() for p in line.split(sep)])
    for i, col in enumerate(cols):
        vals = []
        for row in rows:
            if i < len(row):
                try:    vals.append(float(row[i]))
                except: vals.append(row[i])
            else: vals.append(np.nan)
        data[col] = vals
    return data

def find_name_col(data):
    for c in ['galaxy','Galaxy','name','Name','GALAXY']:
        if c in data: return c
    for k,v in data.items():
        if isinstance(v[0], str): return k
    return list(data.keys())[0]

def get_key(info, candidates, default=None):
    for c in candidates:
        if c in info:
            try: return float(info[c])
            except: return info[c]
    return default

# ============================================================
# Load data
# ============================================================
print('[1] Loading data...')
phase1 = load_csv(PHASE1)
ta3    = load_csv(TA3)
p1_nc  = find_name_col(phase1)
ta3_nc = find_name_col(ta3)

galaxy_info = {}
for i, name in enumerate(phase1[p1_nc]):
    name = str(name).strip()
    info = {}
    for k in phase1:
        if k == p1_nc: continue
        try:    info[k] = float(phase1[k][i])
        except: info[k] = phase1[k][i]
    galaxy_info[name] = info

for i, name in enumerate(ta3[ta3_nc]):
    name = str(name).strip()
    if name in galaxy_info:
        for k in ta3:
            if k == ta3_nc: continue
            try:    galaxy_info[name][k] = float(ta3[k][i])
            except: galaxy_info[name][k] = ta3[k][i]
```

```
|
| # ============================================================
| # Load rotmod and compute quantities
| # ============================================================
| def load_rotmod(filepath):
|     cols = [[] for _ in range(8)]
|     with open(filepath, 'r') as f:
|         for line in f:
|             line = line.strip()
|             if not line or line.startswith('#'): continue
|             parts = line.split()
|             if len(parts) < 6: continue
|             try:
|                 for j in range(min(len(parts),8)):
|                     cols[j].append(float(parts[j]))
|                 for j in range(len(parts),8):
|                     cols[j].append(0.0)
|             except ValueError: continue
|     return tuple(np.array(c) for c in cols)
|
| print('[2] Processing galaxies...')
| results = []
| rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
|
| for fpath in rotmod_files:
|     gname = os.path.splitext(os.path.basename(fpath))[0].replace('_rotmod','').strip()
|     info = None
|     for key in [gname, gname.upper(), gname.lower()]:
|         if key in galaxy_info:
|             info = galaxy_info[key]; break
|     if info is None: continue
|
|     ud    = get_key(info, ['upsilon_d','Upsilon_d','Ud','ud','Yd'])
|     gc_a0 = get_key(info, ['gc_over_a0','gc/a0','gc_ratio'])
|     vflat = get_key(info, ['vflat','Vflat','v_flat'])
|
|     if ud is None or gc_a0 is None or vflat is None: continue
|     if np.isnan(ud) or np.isnan(gc_a0) or gc_a0<=0 or vflat<=0: continue
|
|     try:
|         rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
|     except: continue
|     if len(rad) < 3: continue
|
|     # hR from rpk
|     vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
|     rpk_idx = np.argmax(vds)
|     rpk = rad[rpk_idx]
|     if rpk < 0.01 or rpk >= rad.max()*0.9: continue
|     hR_kpc = rpk / 2.15
|     hR_m   = hR_kpc * kpc_m
|
|     vflat_ms = vflat * 1e3
|     GS0_proxy = vflat_ms**2 / hR_m
|
|     # --- M_star estimation ---
|     # Method 1: from BTFR -- M_bar = vflat^4 / (gc * G)
|     gc_si = gc_a0 * a0
|     M_bar_btfr = vflat_ms**4 / (gc_si * G_SI)  # kg
|     M_bar_btfr_sun = M_bar_btfr / Msun
|
|     # Method 2: from disk velocity profile (gc-independent)
|     # M_disk = Yd * L_disk
|     # For exponential disk: V_disk_peak^2 = 0.56*pi*G*Sigma0*hR
|     # => Sigma0 = V_disk_peak^2 / (0.56*pi*G*hR)
|     # => M_disk = 2*pi*Sigma0*hR^2 = 2*V_disk_peak^2*hR / (0.56*G)
|     vdisk_peak_ms = np.max(np.sqrt(max(ud,0.01)) * np.abs(vdisk)) * 1e3
|     M_disk_direct = 2.0 * vdisk_peak_ms**2 * hR_m / (0.56 * G_SI)
|     M_disk_direct_sun = M_disk_direct / Msun
|
|     # Gas mass (same approach)
|     vgas_peak_ms = np.max(np.abs(vgas)) * 1e3
|     M_gas_direct = 2.0 * vgas_peak_ms**2 * hR_m / (0.56 * G_SI) if vgas_peak_ms > 0 else 0
|     M_gas_direct_sun = M_gas_direct / Msun
|
|     M_bar_direct_sun = M_disk_direct_sun + M_gas_direct_sun
|
|     # Sigma_bar (gc-independent)
|     Sigma_bar = M_disk_direct / (2 * np.pi * hR_m**2)  # kg/m^2
|     G_Sigma_bar = G_SI * Sigma_bar  # m/s^2
|
|     results.append({
|         'galaxy':           gname,
|         'gc_over_a0':       gc_a0,
|         'log_gc':           np.log10(gc_a0),
|         'vflat':            vflat,
|         'Yd':               ud,
|         'hR_kpc':           hR_kpc,
```

```
        'GS0_proxy':        GS0_proxy,
        'log_GS0':          np.log10(GS0_proxy / a0),
        # BTFR mass (circular)
        'log_Mbar_btfr':    np.log10(M_bar_btfr_sun) if M_bar_btfr_sun>0 else np.nan,
        # Direct mass (independent)
        'M_disk_sun':       M_disk_direct_sun,
        'M_gas_sun':        M_gas_direct_sun,
        'M_bar_direct_sun': M_bar_direct_sun,
        'log_Mbar_direct':  np.log10(M_bar_direct_sun) if M_bar_direct_sun>0 else np.nan,
        'log_Mdisk':        np.log10(M_disk_direct_sun) if M_disk_direct_sun>0 else np.nan,
        # Surface density (independent)
        'G_Sigma_bar':      G_Sigma_bar,
        'log_GSbar':        np.log10(G_Sigma_bar/a0) if G_Sigma_bar>0 else np.nan,
        # Individual components for decomposition
        'log_vflat':        np.log10(vflat),
        'log_hR':           np.log10(hR_kpc),
        'vdisk_peak':       vdisk_peak_ms/1e3,
    })

N = len(results)
print(f'  Processed: {N} galaxies')
if N < 10:
    print('[ERROR] Too few galaxies.'); sys.exit(1)

# ============================================================
# Arrays
# ============================================================
log_gc    = np.array([r['log_gc'] for r in results])
log_GS0   = np.array([r['log_GS0'] for r in results])
log_Mbar  = np.array([r['log_Mbar_direct'] for r in results])
log_Mdisk = np.array([r['log_Mdisk'] for r in results])
log_GSbar = np.array([r['log_GSbar'] for r in results])
log_vflat = np.array([r['log_vflat'] for r in results])
log_hR    = np.array([r['log_hR'] for r in results])
log_Mbtfr = np.array([r['log_Mbar_btfr'] for r in results])

# ============================================================
# Regression helper
# ============================================================
def regtest(x, y, label, expected_slopes=None):
    mask = np.isfinite(x) & np.isfinite(y)
    xv, yv = x[mask], y[mask]
    n = len(xv)
    if n < 10:
        print(f'  [{label}] N={n} too few'); return None

    slope, intercept, r_val, p_val, se = stats.linregress(xv, yv)
    rho_s, p_s = stats.spearmanr(xv, yv)

    print(f'¥n  [{label}]  N={n}')
    print(f'    slope    = {slope:.4f} +/- {se:.4f}')
    print(f'    R^2      = {r_val**2:.4f}')
    print(f'    Spearman = {rho_s:.4f}  (p={p_s:.2e})')

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

    return {'slope':slope, 'se':se, 'intercept':intercept,
            'r2':r_val**2, 'rho':rho_s, 'p_s':p_s, 'n':n,
            'xv':xv, 'yv':yv}

# ============================================================
# Tests
# ============================================================
print('¥n' + '='*70)
print('V-1b: gc vs M* direct regression')
print('='*70)

print('¥n--- A: Baseline proxy (vflat^2/hR) ---')
rA = regtest(log_GS0, log_gc, 'proxy vflat^2/hR', [0.5])

print('¥n--- B: gc vs M_bar_direct (gc-independent, BT) ---')
rB = regtest(log_Mbar, log_gc, 'M_bar direct (BT)', [1/3, 1/4, 0.2, 0.125])

print('¥n--- C: gc vs M_disk only (gc-independent) ---')
rC = regtest(log_Mdisk, log_gc, 'M_disk only', [1/3, 1/4, 0.2, 0.125])

print('¥n--- D: gc vs Sigma_bar (gc-independent, V-1kai recheck) ---')
rD = regtest(log_GSbar, log_gc, 'G*Sigma_bar (BT)', [1/3, 0.5])

print('¥n--- E: gc vs vflat only ---')
rE = regtest(log_vflat, log_gc, 'vflat only', [1.0, 2.0])

print('¥n--- F: gc vs hR only ---')
```

```
rF = regtest(log_hR, log_gc, 'hR only', [-0.5, -1.0])

print('¥n--- G: gc vs M_bar BTFR (circular, diagnostic) ---')
rG = regtest(log_Mbtfr, log_gc, 'M_bar BTFR (circular)', [1/3, 1/4])

# ============================================================
# Multivariate: gc = a * log(vflat) + b * log(hR) + c
# ============================================================
print('¥n--- H: Multivariate log(gc) = a*log(vflat) + b*log(hR) + c ---')
mask = np.isfinite(log_gc) & np.isfinite(log_vflat) & np.isfinite(log_hR)
X = np.column_stack([log_vflat[mask], log_hR[mask], np.ones(mask.sum())])
y = log_gc[mask]
# OLS
beta, residuals, rank, sv = np.linalg.lstsq(X, y, rcond=None)
y_pred = X @ beta
ss_res = np.sum((y - y_pred)**2)
ss_tot = np.sum((y - y.mean())**2)
r2_multi = 1 - ss_res/ss_tot
n_multi = len(y)
# Standard errors
if len(residuals) > 0:
    mse = residuals[0] / (n_multi - 3)
else:
    mse = ss_res / (n_multi - 3)
se_beta = np.sqrt(np.diag(mse * np.linalg.inv(X.T @ X)))

print(f'  N = {n_multi}')
print(f'  log(gc) = {beta[0]:.4f}*log(vflat) + {beta[1]:.4f}*log(hR) + {beta[2]:.4f}')
print(f'  SE:       {se_beta[0]:.4f}              {se_beta[1]:.4f}              {se_beta[2]:.4f}')
print(f'  R^2 = {r2_multi:.4f}')
print(f'  vflat exponent: {beta[0]:.3f} +/- {se_beta[0]:.3f}')
print(f'  hR exponent:    {beta[1]:.3f} +/- {se_beta[1]:.3f}')

# Test: is it consistent with proxy (vflat^2/hR)^0.5 = vflat^1 * hR^(-0.5)?
print(f'¥n  Proxy decomposition test:')
print(f'    (vflat^2/hR)^0.5 predicts: vflat^1.0, hR^-0.5')
print(f'    Observed:                  vflat^{beta[0]:.3f}, hR^{beta[1]:.3f}')
t_vf = abs(beta[0] - 1.0) / se_beta[0]
p_vf = 2*(1-stats.t.cdf(t_vf, df=n_multi-3))
t_hr = abs(beta[1] - (-0.5)) / se_beta[1]
p_hr = 2*(1-stats.t.cdf(t_hr, df=n_multi-3))
print(f'    p(vflat=1.0) = {p_vf:.4f}')
print(f'    p(hR=-0.5)   = {p_hr:.4f}')

# ============================================================
# Partial correlation: gc vs hR controlling for vflat
# ============================================================
print('¥n--- I: Partial correlations ---')
# gc vs hR | vflat
mask_p = np.isfinite(log_gc) & np.isfinite(log_vflat) & np.isfinite(log_hR)
gc_p = log_gc[mask_p]
vf_p = log_vflat[mask_p]
hr_p = log_hR[mask_p]

# Residualize gc and hR against vflat
_, _, r_gc_vf, _, _ = stats.linregress(vf_p, gc_p)
res_gc = gc_p - (stats.linregress(vf_p, gc_p)[0]*vf_p + stats.linregress(vf_p, gc_p)[1])
res_hr = hr_p - (stats.linregress(vf_p, hr_p)[0]*vf_p + stats.linregress(vf_p, hr_p)[1])
rho_partial, p_partial = stats.spearmanr(res_gc, res_hr)
print(f'  gc vs hR | vflat: rho_partial = {rho_partial:.4f}, p = {p_partial:.4e}')
print(f'  (Does hR contribute BEYOND vflat?)')

# gc vs vflat | hR
res_gc2 = gc_p - (stats.linregress(hr_p, gc_p)[0]*hr_p + stats.linregress(hr_p, gc_p)[1])
res_vf2 = vf_p - (stats.linregress(hr_p, vf_p)[0]*hr_p + stats.linregress(hr_p, vf_p)[1])
rho_partial2, p_partial2 = stats.spearmanr(res_gc2, res_vf2)
print(f'  gc vs vflat | hR: rho_partial = {rho_partial2:.4f}, p = {p_partial2:.4e}')
print(f'  (Does vflat contribute BEYOND hR?)')

# ============================================================
# Figures
# ============================================================
print('¥n[3] Generating figures...')

fig, axes = plt.subplots(2, 3, figsize=(17, 11))

# A: proxy baseline
ax = axes[0,0]
if rA:
    ax.scatter(rA['xv'], rA['yv'], s=10, alpha=0.5, c='steelblue', edgecolors='none')
    xf = np.linspace(rA['xv'].min(), rA['xv'].max(), 100)
    ax.plot(xf, rA['slope']*xf+rA['intercept'], 'r-', lw=2,
            label=f'slope={rA["slope"]:.3f}, R^2={rA["r2"]:.3f}')
    ax.plot(xf, 0.5*xf+rA['intercept'], 'b--', lw=1, label='theory: 0.5')
    ax.set_xlabel('log(vflat^2/hR / a0)')
    ax.set_ylabel('log(gc/a0)')
    ax.set_title('A: Proxy (baseline)')
    ax.legend(fontsize=8)
```

```
    ax.grid(True, alpha=0.3)

# B: M_bar direct
ax = axes[0,1]
if rB:
    ax.scatter(rB['xv'], rB['yv'], s=10, alpha=0.5, c='darkorange', edgecolors='none')
    xf = np.linspace(rB['xv'].min(), rB['xv'].max(), 100)
    ax.plot(xf, rB['slope']*xf+rB['intercept'], 'r-', lw=2,
            label=f'slope={rB["slope"]:.3f}, R^2={rB["r2"]:.3f}')
    for es, col, ls in [(1/3,'blue','--'),(1/4,'green','--'),(0.125,'purple',':')]:
        ax.plot(xf, es*xf+rB['intercept'], color=col, ls=ls, lw=1,
                label=f'{es:.3f}')
    ax.set_xlabel('log(M_bar / M_sun)  [direct, gc-free]')
    ax.set_ylabel('log(gc/a0)')
    ax.set_title('B: M_bar direct (BT)')
    ax.legend(fontsize=7)
    ax.grid(True, alpha=0.3)

# C: Sigma_bar
ax = axes[0,2]
if rD:
    ax.scatter(rD['xv'], rD['yv'], s=10, alpha=0.5, c='green', edgecolors='none')
    xf = np.linspace(rD['xv'].min(), rD['xv'].max(), 100)
    ax.plot(xf, rD['slope']*xf+rD['intercept'], 'r-', lw=2,
            label=f'slope={rD["slope"]:.3f}, R^2={rD["r2"]:.3f}')
    ax.set_xlabel('log(G*Sigma_bar / a0)  [direct]')
    ax.set_ylabel('log(gc/a0)')
    ax.set_title('C: Sigma_bar (recheck)')
    ax.legend(fontsize=8)
    ax.grid(True, alpha=0.3)

# D: vflat only
ax = axes[1,0]
if rE:
    ax.scatter(rE['xv'], rE['yv'], s=10, alpha=0.5, c='crimson', edgecolors='none')
    xf = np.linspace(rE['xv'].min(), rE['xv'].max(), 100)
    ax.plot(xf, rE['slope']*xf+rE['intercept'], 'r-', lw=2,
            label=f'slope={rE["slope"]:.3f}, R^2={rE["r2"]:.3f}')
    ax.set_xlabel('log(vflat / km/s)')
    ax.set_ylabel('log(gc/a0)')
    ax.set_title('D: vflat only')
    ax.legend(fontsize=8)
    ax.grid(True, alpha=0.3)

# E: hR only
ax = axes[1,1]
if rF:
    ax.scatter(rF['xv'], rF['yv'], s=10, alpha=0.5, c='teal', edgecolors='none')
    xf = np.linspace(rF['xv'].min(), rF['xv'].max(), 100)
    ax.plot(xf, rF['slope']*xf+rF['intercept'], 'r-', lw=2,
            label=f'slope={rF["slope"]:.3f}, R^2={rF["r2"]:.3f}')
    ax.set_xlabel('log(hR / kpc)')
    ax.set_ylabel('log(gc/a0)')
    ax.set_title('E: hR only')
    ax.legend(fontsize=8)
    ax.grid(True, alpha=0.3)

# F: R^2 comparison bar chart
ax = axes[1,2]
labels_bar = []
r2_bar = []
cols_bar = []
for label, reg, col in [
    ('Proxy¥nvflat^2/hR', rA, 'steelblue'),
    ('M_bar¥n(direct)', rB, 'darkorange'),
    ('Sigma_bar¥n(direct)', rD, 'green'),
    ('vflat¥nonly', rE, 'crimson'),
    ('hR¥nonly', rF, 'teal'),
    ('M_bar¥n(BTFR circ)', rG, 'grey'),
]:
    if reg:
        labels_bar.append(label)
        r2_bar.append(reg['r2'])
        cols_bar.append(col)

# Add multivariate
labels_bar.append('vflat+hR¥nmultivar')
r2_bar.append(r2_multi)
cols_bar.append('purple')

xpos = np.arange(len(labels_bar))
ax.bar(xpos, r2_bar, color=cols_bar, alpha=0.7, edgecolor='black', linewidth=0.5)
ax.set_xticks(xpos)
ax.set_xticklabels(labels_bar, fontsize=8)
ax.set_ylabel('R^2', fontsize=11)
ax.set_title('F: Predictive power comparison', fontsize=11)
ax.grid(True, alpha=0.3, axis='y')
for i, v in enumerate(r2_bar):
```

```
    ax.text(i, v+0.01, f'{v:.3f}', ha='center', fontsize=8)

fig.suptitle(f'V-1b: What drives gc? (N={N})', fontsize=14, y=1.01)
fig.tight_layout()
fig.savefig(os.path.join(BASE, 'fig_gc_vs_Mstar.png'), dpi=150)
print('  -> fig_gc_vs_Mstar.png')

# ============================================================
# Save CSV
# ============================================================
outcsv = os.path.join(BASE, 'gc_vs_Mstar_results.csv')
cols_out = ['galaxy','gc_over_a0','log_gc','vflat','Yd','hR_kpc',
            'log_GS0','log_Mbar_direct','log_Mdisk','log_GSbar',
            'log_vflat','log_hR','log_Mbar_btfr','vdisk_peak']
with open(outcsv, 'w', encoding='utf-8') as f:
    f.write(','.join(cols_out)+'¥n')
    for r in results:
        f.write(','.join(str(r.get(c,'')) for c in cols_out)+'¥n')
print(f'¥n[4] Saved: {outcsv}')

# ============================================================
# Final Summary
# ============================================================
print('¥n'+'='*70)
print('FINAL SUMMARY')
print('='*70)

print(f'¥n  {"Predictor":<25s} | {"slope":>8s} +/- {"SE":>6s} | {"R^2":>6s} | {"Spearman":>8s}')
print('  '+'-'*70)
for label, reg in [
    ('Proxy vflat^2/hR', rA),
    ('M_bar direct (BT)', rB),
    ('M_disk only', rC),
    ('G*Sigma_bar (BT)', rD),
    ('vflat only', rE),
    ('hR only', rF),
    ('M_bar BTFR (circ)', rG),
]:
    if reg:
        print(f'  {label:<25s} | {reg["slope"]:>8.4f} +/- {reg["se"]:>6.4f} | '
              f'{reg["r2"]:>6.4f} | {reg["rho"]:>8.4f}')
print(f'  {"vflat+hR multivar":<25s} | {"--":>8s}     {"--":>6s} | {r2_multi:>6.4f} | {"--":>8s}')

print(f'¥n  Multivariate decomposition:')
print(f'    gc ~ vflat^{beta[0]:.3f} * hR^{beta[1]:.3f}')
print(f'    Proxy predicts: vflat^1.0 * hR^-0.5')
print(f'    p(vflat=1.0) = {p_vf:.4f}')
print(f'    p(hR=-0.5)   = {p_hr:.4f}')

print(f'¥n  Partial correlations:')
print(f'    gc vs hR | vflat:  rho={rho_partial:.4f}, p={p_partial:.4e}')
print(f'    gc vs vflat | hR:  rho={rho_partial2:.4f}, p={p_partial2:.4e}')

print('¥n  KEY QUESTION: Is gc driven by M* (mass) or proxy (vflat^2/hR)?')
if rB and rA:
    if rB['r2'] > rA['r2'] * 0.8:
        print('  >>> M* has comparable predictive power -> mass is the driver')
    elif rB['r2'] < 0.1:
        print('  >>> M* (direct) has NO predictive power -> proxy encodes something else')
        print('  >>> gc depends on vflat (which includes DM/membrane effect)')
    else:
        print('  >>> Intermediate case -> further investigation needed')

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_outer = g_N(r_outer)/gc
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

```
# -*- coding: utf-8 -*-
"""
N-1 第二層検証: MOND遷移域補正と hR 偏相関の起源
======================================================================
式(5)完全形 g_obs = (g_N + sqrt(g_N^2 + 4*gc*g_N)) / 2 において、
MOND極限 g_obs ~ sqrt(gc*g_N) は g_N << gc のとき成立する。

仮説: hR の偏相関（rho=-0.356）は、コンパクト銀河（小hR）で
g_N(r_outer)/gc が大きく MOND極限からのずれが大きいことに起因する。

検証:
  (1) 各銀河で x_outer = g_N(r_outer)/gc を計算
  (2) x_outer と hR の相関を確認
  (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
  (4) 補正後の残差構造を確認

実行: uv run --with scipy --with matplotlib python sparc_N1_layer2.py

必要ファイル:
  - Rotmod_LTG/*.dat
  - phase1/sparc_results.csv
  - TA3_gc_independent.csv

著者: 坂口 忍（坂口製麺所）
日付: 2026年4月
"""
import os, sys, glob, warnings
import numpy as np
from scipy import stats

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import font_manager as _fm
for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
            r'C:¥Windows¥Fonts¥msgothic.ttc']:
    try: _fm.fontManager.addfont(_fp)
    except: pass
for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
    try:
        plt.rcParams['font.family'] = fontname
        break
    except: continue
plt.rcParams['axes.unicode_minus'] = False
warnings.filterwarnings('ignore')

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

```
# Constants
# ============================================================
a0     = 1.2e-10
G_SI   = 6.674e-11
kpc_m  = 3.0857e19

# ============================================================
# Paths
# ============================================================
BASE   = os.path.dirname(os.path.abspath(__file__))
ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
TA3    = os.path.join(BASE, 'TA3_gc_independent.csv')

for p, label in [(ROTMOD,'Rotmod_LTG'),(PHASE1,'sparc_results.csv'),
                 (TA3,'TA3_gc_independent.csv')]:
    if not os.path.exists(p):
        print(f'[ERROR] {label} not found: {p}'); sys.exit(1)

# ============================================================
# CSV / rotmod loaders (same as previous scripts)
# ============================================================
def load_csv(path):
    with open(path, 'r', encoding='utf-8-sig') as f:
        header = f.readline().strip()
    sep = ',' if ',' in header else None
    data = {}
    with open(path, 'r', encoding='utf-8-sig') as f:
        cols = [c.strip() for c in f.readline().strip().split(sep)]
        rows = []
        for line in f:
            line = line.strip()
            if not line: continue
            rows.append([p.strip() for p in line.split(sep)])
    for i, col in enumerate(cols):
        vals = []
        for row in rows:
            if i < len(row):
                try:    vals.append(float(row[i]))
                except: vals.append(row[i])
            else: vals.append(np.nan)
        data[col] = vals
    return data

def find_name_col(data):
    for c in ['galaxy','Galaxy','name','Name','GALAXY']:
        if c in data: return c
    for k,v in data.items():
        if isinstance(v[0], str): return k
    return list(data.keys())[0]

def get_key(info, candidates, default=None):
    for c in candidates:
        if c in info:
            try: return float(info[c])
            except: return info[c]
    return default

def load_rotmod(filepath):
    cols = [[] for _ in range(8)]
    with open(filepath, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith('#'): continue
            parts = line.split()
            if len(parts) < 6: continue
            try:
                for j in range(min(len(parts),8)):
                    cols[j].append(float(parts[j]))
                for j in range(len(parts),8):
                    cols[j].append(0.0)
            except ValueError: continue
    return tuple(np.array(c) for c in cols)

# ============================================================
# Load data
# ============================================================
print('[1] Loading data...')
phase1 = load_csv(PHASE1)
ta3    = load_csv(TA3)
p1_nc  = find_name_col(phase1)
ta3_nc = find_name_col(ta3)

galaxy_info = {}
for i, name in enumerate(phase1[p1_nc]):
    name = str(name).strip()
    info = {}
    for k in phase1:
```

```
        if k == p1_nc: continue
        try:    info[k] = float(phase1[k][i])
        except: info[k] = phase1[k][i]
    galaxy_info[name] = info

for i, name in enumerate(ta3[ta3_nc]):
    name = str(name).strip()
    if name in galaxy_info:
        for k in ta3:
            if k == ta3_nc: continue
            try:    galaxy_info[name][k] = float(ta3[k][i])
            except: galaxy_info[name][k] = ta3[k][i]

# ============================================================
# Process galaxies
# ============================================================
print('[2] Processing galaxies...')
results = []
rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))

for fpath in rotmod_files:
    gname = os.path.splitext(os.path.basename(fpath))[0].replace('_rotmod','').strip()
    info = None
    for key in [gname, gname.upper(), gname.lower()]:
        if key in galaxy_info:
            info = galaxy_info[key]; break
    if info is None: continue

    ud    = get_key(info, ['upsilon_d','Upsilon_d','Ud','ud','Yd'])
    gc_a0 = get_key(info, ['gc_over_a0','gc/a0','gc_ratio'])
    vflat = get_key(info, ['vflat','Vflat','v_flat'])

    if ud is None or gc_a0 is None or vflat is None: continue
    if np.isnan(ud) or np.isnan(gc_a0) or gc_a0<=0 or vflat<=0: continue

    try:
        rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
    except: continue
    if len(rad) < 3: continue

    # hR from rpk
    vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
    rpk_idx = np.argmax(vds)
    rpk = rad[rpk_idx]
    if rpk < 0.01 or rpk >= rad.max()*0.9: continue
    hR_kpc = rpk / 2.15
    hR_m   = hR_kpc * kpc_m

    # g_N(r) from baryonic components
    r_m = rad * kpc_m
    vbar2 = (vgas**2 + ud * np.sign(vdisk)*vdisk**2
             + 0.7 * np.sign(vbul)*vbul**2)
    vbar2 = np.abs(vbar2)
    g_N = np.zeros_like(r_m)
    mask_r = r_m > 0
    g_N[mask_r] = vbar2[mask_r] * 1e6 / r_m[mask_r]

    gc_si = gc_a0 * a0

    # --- Key quantity: x_outer = g_N(r_outer) / gc ---
    # Use outer 3 points average for stability
    n_outer = min(3, len(rad))
    g_N_outer = np.mean(g_N[-n_outer:])
    x_outer = g_N_outer / gc_si if gc_si > 0 else np.nan

    # --- g_N at various radii ---
    # At r = 2.2 hR (peak region)
    r_peak_target = 2.2 * hR_kpc
    idx_peak = np.argmin(np.abs(rad - r_peak_target))
    g_N_peak = g_N[idx_peak]
    x_peak = g_N_peak / gc_si if gc_si > 0 else np.nan

    # --- MOND transition correction ---
    # Full formula: g_obs = (g_N + sqrt(g_N^2 + 4*gc*g_N)) / 2
    # Deep MOND: g_obs_MOND ~ sqrt(gc * g_N)
    # Ratio: g_obs_full / g_obs_MOND = correction factor
    # At r_outer:
    if g_N_outer > 0 and gc_si > 0:
        g_obs_full = (g_N_outer + np.sqrt(g_N_outer**2 + 4*gc_si*g_N_outer)) / 2
        g_obs_mond = np.sqrt(gc_si * g_N_outer)
        correction = g_obs_full / g_obs_mond  # >1 always, larger for larger x
    else:
        correction = np.nan

    # --- vflat from full formula vs MOND limit ---
    # vflat_obs^2 / r_outer = g_obs_full
    # vflat_mond^2 / r_outer = g_obs_mond = sqrt(gc * g_N)
    # So vflat_obs / vflat_mond = sqrt(correction)
```

```
    # And (vflat_obs^2/hR) / (vflat_mond^2/hR) = correction
    # The proxy GS0 = vflat^2/hR includes this correction factor

    vflat_ms = vflat * 1e3
    GS0_proxy = vflat_ms**2 / hR_m

    # --- Corrected proxy: remove transition effect ---
    # GS0_corrected = GS0_proxy / correction^2
    # (because vflat^2 ~ g_obs * r, and correction = g_obs_full/g_obs_mond)
    # Actually: vflat^2/r = g_obs at r_outer
    # proxy = vflat^2/hR = g_obs(r_outer) * r_outer/hR
    # MOND proxy = sqrt(gc*g_N(r_outer)) * r_outer/hR
    # So proxy/MOND_proxy = correction
    # To get "MOND-corrected proxy": divide by correction

    if np.isfinite(correction) and correction > 0:
        GS0_corrected = GS0_proxy / correction
    else:
        GS0_corrected = np.nan

    # Also compute a per-point mean correction
    corrections_all = np.ones_like(g_N)
    for i_r in range(len(g_N)):
        if g_N[i_r] > 0 and gc_si > 0:
            gf = (g_N[i_r] + np.sqrt(g_N[i_r]**2 + 4*gc_si*g_N[i_r])) / 2
            gm = np.sqrt(gc_si * g_N[i_r])
            corrections_all[i_r] = gf / gm
    mean_correction = np.mean(corrections_all[g_N > 0])

    results.append({
        'galaxy':         gname,
        'gc_over_a0':     gc_a0,
        'log_gc':         np.log10(gc_a0),
        'vflat':          vflat,
        'Yd':             ud,
        'hR_kpc':         hR_kpc,
        'log_hR':         np.log10(hR_kpc),
        'log_vflat':      np.log10(vflat),
        'GS0_proxy':      GS0_proxy,
        'log_GS0':        np.log10(GS0_proxy/a0),
        'x_outer':        x_outer,
        'log_x_outer':    np.log10(x_outer) if x_outer > 0 else np.nan,
        'x_peak':         x_peak,
        'g_N_outer':      g_N_outer,
        'g_N_peak':       g_N_peak,
        'correction':     correction,
        'log_correction': np.log10(correction) if np.isfinite(correction) and correction>0 else np.nan,
        'mean_correction': mean_correction,
        'GS0_corrected':  GS0_corrected,
        'log_GS0_corr':   np.log10(GS0_corrected/a0) if np.isfinite(GS0_corrected) and GS0_corrected>0 else np.nan,
        'r_outer_kpc':    rad[-1],
        'r_outer_over_hR': rad[-1] / hR_kpc if hR_kpc > 0 else np.nan,
    })

N = len(results)
print(f'  Processed: {N} galaxies')
if N < 10:
    print('[ERROR] Too few galaxies.'); sys.exit(1)

# ============================================================
# Arrays
# ============================================================
log_gc      = np.array([r['log_gc'] for r in results])
log_hR      = np.array([r['log_hR'] for r in results])
log_vflat   = np.array([r['log_vflat'] for r in results])
log_GS0     = np.array([r['log_GS0'] for r in results])
x_outer     = np.array([r['x_outer'] for r in results])
log_x_outer = np.array([r['log_x_outer'] for r in results])
correction  = np.array([r['correction'] for r in results])
log_corr    = np.array([r['log_correction'] for r in results])
log_GS0_c   = np.array([r['log_GS0_corr'] for r in results])
r_over_hR   = np.array([r['r_outer_over_hR'] for r in results])

# ============================================================
# Analysis
# ============================================================
print('\n' + '='*70)
print('N-1 Layer 2: MOND transition correction analysis')
print('='*70)

# --- A: x_outer distribution ---
valid = np.isfinite(x_outer) & (x_outer > 0)
print(f'\n[A] x_outer = g_N(r_outer)/gc distribution (N={np.sum(valid)})')
print(f'    median = {np.nanmedian(x_outer[valid]):.4f}')
print(f'    IQR = [{np.nanpercentile(x_outer[valid],25):.4f}, '
      f'{np.nanpercentile(x_outer[valid],75):.4f}]')
print(f'    range = [{np.nanmin(x_outer[valid]):.4f}, {np.nanmax(x_outer[valid]):.4f}]')
print(f'    fraction x_outer > 0.1 (significant transition): '
```

```
      f'{np.sum(x_outer[valid]>0.1)/np.sum(valid):.1%}')
print(f'    fraction x_outer > 0.5 (strong transition): '
      f'{np.sum(x_outer[valid]>0.5)/np.sum(valid):.1%}')

# --- B: x_outer vs hR correlation ---
mask_b = np.isfinite(log_x_outer) & np.isfinite(log_hR)
rho_xh, p_xh = stats.spearmanr(log_x_outer[mask_b], log_hR[mask_b])
print(f'¥n[B] x_outer vs hR correlation')
print(f'    Spearman rho(log x_outer, log hR) = {rho_xh:.4f}, p = {p_xh:.2e}')
print(f'    Expected: negative (compact galaxies have larger x_outer)')

slope_xh, int_xh, r_xh, _, se_xh = stats.linregress(
    log_hR[mask_b], log_x_outer[mask_b])
print(f'    Linear: log(x_outer) = {slope_xh:.3f}*log(hR) + {int_xh:.3f}, '
      f'R^2={r_xh**2:.3f}')

# --- C: correction factor distribution ---
valid_c = np.isfinite(correction)
print(f'¥n[C] Correction factor g_obs_full / g_obs_MOND (N={np.sum(valid_c)})')
print(f'    median = {np.nanmedian(correction[valid_c]):.4f}')
print(f'    IQR = [{np.nanpercentile(correction[valid_c],25):.4f}, '
      f'{np.nanpercentile(correction[valid_c],75):.4f}]')
print(f'    range = [{np.nanmin(correction[valid_c]):.4f}, '
      f'{np.nanmax(correction[valid_c]):.4f}]')

# --- D: Does correction correlate with hR? ---
mask_d = np.isfinite(log_corr) & np.isfinite(log_hR)
rho_ch, p_ch = stats.spearmanr(log_corr[mask_d], log_hR[mask_d])
print(f'¥n[D] Correction vs hR')
print(f'    Spearman rho(log correction, log hR) = {rho_ch:.4f}, p = {p_ch:.2e}')

# --- E: Original proxy regression (baseline) ---
mask_e = np.isfinite(log_GS0) & np.isfinite(log_gc)
slope_0, int_0, r_0, _, se_0 = stats.linregress(log_GS0[mask_e], log_gc[mask_e])
print(f'¥n[E] Baseline: log(gc) vs log(proxy/a0)')
print(f'    slope = {slope_0:.4f} +/- {se_0:.4f}, R^2 = {r_0**2:.4f}')

# --- F: Corrected proxy regression ---
mask_f = np.isfinite(log_GS0_c) & np.isfinite(log_gc)
if np.sum(mask_f) > 10:
    slope_c, int_c, r_c, _, se_c = stats.linregress(log_GS0_c[mask_f], log_gc[mask_f])
    print(f'¥n[F] Corrected proxy: log(gc) vs log(proxy_corrected/a0)')
    print(f'    slope = {slope_c:.4f} +/- {se_c:.4f}, R^2 = {r_c**2:.4f}')
    print(f'    (Compare with baseline R^2 = {r_0**2:.4f})')
else:
    slope_c, r_c, se_c = np.nan, np.nan, np.nan
    print(f'¥n[F] Too few valid corrected points ({np.sum(mask_f)})')

# --- G: Multivariate with correction ---
# log(gc) = a*log(vflat) + b*log(hR) + c  -- original
# log(gc) = a*log(vflat) + b*log(hR) + d*log(correction) + c  -- with correction
mask_g = (np.isfinite(log_gc) & np.isfinite(log_vflat) &
          np.isfinite(log_hR) & np.isfinite(log_corr))

print(f'¥n[G] Multivariate analysis (N={np.sum(mask_g)})')

# G1: without correction (reproduce V-1b)
X1 = np.column_stack([log_vflat[mask_g], log_hR[mask_g], np.ones(mask_g.sum())])
y = log_gc[mask_g]
beta1, res1, _, _ = np.linalg.lstsq(X1, y, rcond=None)
ss_res1 = np.sum((y - X1 @ beta1)**2)
ss_tot = np.sum((y - y.mean())**2)
r2_1 = 1 - ss_res1/ss_tot
mse1 = ss_res1 / (len(y)-3)
se1 = np.sqrt(np.diag(mse1 * np.linalg.inv(X1.T @ X1)))

print(f'  G1: log(gc) = {beta1[0]:.4f}*log(vflat) + {beta1[1]:.4f}*log(hR) + {beta1[2]:.4f}')
print(f'      SE:        {se1[0]:.4f}                 {se1[1]:.4f}')
print(f'      R^2 = {r2_1:.4f}')

# G2: with correction
X2 = np.column_stack([log_vflat[mask_g], log_hR[mask_g], log_corr[mask_g],
                      np.ones(mask_g.sum())])
beta2, res2, _, _ = np.linalg.lstsq(X2, y, rcond=None)
ss_res2 = np.sum((y - X2 @ beta2)**2)
r2_2 = 1 - ss_res2/ss_tot
mse2 = ss_res2 / (len(y)-4)
se2 = np.sqrt(np.diag(mse2 * np.linalg.inv(X2.T @ X2)))

print(f'¥n  G2: log(gc) = {beta2[0]:.4f}*log(vflat) + {beta2[1]:.4f}*log(hR) '
      f'+ {beta2[2]:.4f}*log(corr) + {beta2[3]:.4f}')
print(f'      SE:        {se2[0]:.4f}                 {se2[1]:.4f}                '
      f'{se2[2]:.4f}')
print(f'      R^2 = {r2_2:.4f}  (vs {r2_1:.4f} without correction)')

# Is correction term significant?
t_corr = abs(beta2[2]) / se2[2]
p_corr = 2*(1-stats.t.cdf(t_corr, df=len(y)-4))
```

```
print(f'      p(correction coeff = 0) = {p_corr:.4f}')

# Did hR coefficient change?
print(f'¥n  hR coefficient comparison:')
print(f'    Without correction: {beta1[1]:.4f} +/- {se1[1]:.4f}')
print(f'    With correction:    {beta2[1]:.4f} +/- {se2[1]:.4f}')
delta_hR = abs(beta2[1]) - abs(beta1[1])
print(f'    |hR coeff| change:  {delta_hR:+.4f}')
if abs(beta2[1]) < abs(beta1[1]) * 0.5:
    print(f'    >>> hR effect ABSORBED by correction (>50% reduction)')
elif abs(beta2[1]) < abs(beta1[1]) * 0.8:
    print(f'    >>> hR effect PARTIALLY absorbed ({(1-abs(beta2[1])/abs(beta1[1]))*100:.0f}% reduction)')
else:
    print(f'    >>> hR effect NOT absorbed by correction')

# --- H: Partial correlations with correction controlled ---
print(f'¥n[H] Partial correlations controlling for correction')

# gc vs hR | vflat, correction
X_ctrl = np.column_stack([log_vflat[mask_g], log_corr[mask_g]])
y_gc = log_gc[mask_g]
y_hR = log_hR[mask_g]

# Residualize gc and hR against (vflat, correction)
beta_gc = np.linalg.lstsq(np.column_stack([X_ctrl, np.ones(len(X_ctrl))]),
                          y_gc, rcond=None)[0]
res_gc = y_gc - np.column_stack([X_ctrl, np.ones(len(X_ctrl))]) @ beta_gc

beta_hr = np.linalg.lstsq(np.column_stack([X_ctrl, np.ones(len(X_ctrl))]),
                          y_hR, rcond=None)[0]
res_hR = y_hR - np.column_stack([X_ctrl, np.ones(len(X_ctrl))]) @ beta_hr

rho_partial_new, p_partial_new = stats.spearmanr(res_gc, res_hR)
print(f'  gc vs hR | (vflat, correction): rho = {rho_partial_new:.4f}, p = {p_partial_new:.4e}')

# Original partial for comparison
X_ctrl0 = log_vflat[mask_g].reshape(-1,1)
beta_gc0 = np.linalg.lstsq(np.column_stack([X_ctrl0, np.ones(len(X_ctrl0))]),
                           y_gc, rcond=None)[0]
res_gc0 = y_gc - np.column_stack([X_ctrl0, np.ones(len(X_ctrl0))]) @ beta_gc0
beta_hr0 = np.linalg.lstsq(np.column_stack([X_ctrl0, np.ones(len(X_ctrl0))]),
                           y_hR, rcond=None)[0]
res_hR0 = y_hR - np.column_stack([X_ctrl0, np.ones(len(X_ctrl0))]) @ beta_hr0
rho_partial_orig, p_partial_orig = stats.spearmanr(res_gc0, res_hR0)

print(f'  gc vs hR | vflat only (original):    rho = {rho_partial_orig:.4f}, p = {p_partial_orig:.4e}')
print(f'  Reduction: {rho_partial_orig:.4f} -> {rho_partial_new:.4f} '
      f'({(1-abs(rho_partial_new)/abs(rho_partial_orig))*100:.1f}%)')

# ============================================================
# Figures
# ============================================================
print('¥n[3] Generating figures...')

fig, axes = plt.subplots(2, 3, figsize=(17, 11))

# A: x_outer distribution
ax = axes[0, 0]
xo_valid = x_outer[np.isfinite(x_outer) & (x_outer > 0)]
ax.hist(np.log10(xo_valid), bins=30, color='steelblue', alpha=0.7, edgecolor='white')
ax.axvline(np.log10(0.1), color='orange', ls='--', lw=2, label='x=0.1 (10% transition)')
ax.axvline(np.log10(0.5), color='red', ls='--', lw=2, label='x=0.5 (strong transition)')
ax.set_xlabel('log(x_outer) = log(g_N(r_outer)/gc)')
ax.set_ylabel('Count')
ax.set_title('A: MOND departure parameter')
ax.legend(fontsize=8)

# B: x_outer vs hR
ax = axes[0, 1]
mb = np.isfinite(log_x_outer) & np.isfinite(log_hR)
ax.scatter(log_hR[mb], log_x_outer[mb], s=10, alpha=0.5, c='steelblue', edgecolors='none')
xf = np.linspace(log_hR[mb].min(), log_hR[mb].max(), 100)
ax.plot(xf, slope_xh*xf + int_xh, 'r-', lw=2,
        label=f'rho={rho_xh:.3f}, p={p_xh:.1e}')
ax.set_xlabel('log(hR / kpc)')
ax.set_ylabel('log(x_outer)')
ax.set_title('B: x_outer vs hR')
ax.legend(fontsize=8)
ax.grid(True, alpha=0.3)

# C: correction factor vs hR
ax = axes[0, 2]
md = np.isfinite(log_corr) & np.isfinite(log_hR)
ax.scatter(log_hR[md], log_corr[md], s=10, alpha=0.5, c='darkorange', edgecolors='none')
ax.set_xlabel('log(hR / kpc)')
ax.set_ylabel('log(correction factor)')
ax.set_title(f'C: Correction vs hR (rho={rho_ch:.3f})')
ax.grid(True, alpha=0.3)
```

```
|
| # D: Original vs corrected proxy
| ax = axes[1, 0]
| mf = np.isfinite(log_GS0) & np.isfinite(log_gc)
| ax.scatter(log_GS0[mf], log_gc[mf], s=10, alpha=0.3, c='steelblue',
|            edgecolors='none', label=f'Original R^2={r_0**2:.3f}')
| mfc = np.isfinite(log_GS0_c) & np.isfinite(log_gc)
| if np.sum(mfc) > 5:
|     ax.scatter(log_GS0_c[mfc], log_gc[mfc], s=10, alpha=0.3, c='red',
|                edgecolors='none', label=f'Corrected R^2={r_c**2:.3f}')
| ax.set_xlabel('log(proxy / a0)')
| ax.set_ylabel('log(gc / a0)')
| ax.set_title('D: Original vs corrected proxy')
| ax.legend(fontsize=8)
| ax.grid(True, alpha=0.3)
|
| # E: hR partial correlation before/after
| ax = axes[1, 1]
| ax.scatter(res_hR0, res_gc0, s=10, alpha=0.3, c='steelblue', edgecolors='none',
|            label=f'Original: rho={rho_partial_orig:.3f}')
| ax.scatter(res_hR, res_gc, s=10, alpha=0.3, c='red', edgecolors='none',
|            label=f'+ correction: rho={rho_partial_new:.3f}')
| ax.set_xlabel('hR residual (after controlling)')
| ax.set_ylabel('gc residual (after controlling)')
| ax.set_title('E: hR partial correlation')
| ax.legend(fontsize=8)
| ax.grid(True, alpha=0.3)
|
| # F: R^2 comparison
| ax = axes[1, 2]
| labels = ['vflat+hR¥n(original)', 'vflat+hR¥n+correction', 'proxy¥noriginal',
|           'proxy¥ncorrected']
| r2s = [r2_1, r2_2, r_0**2, r_c**2 if np.isfinite(r_c) else 0]
| cols = ['steelblue', 'darkorange', 'green', 'red']
| ax.bar(range(len(labels)), r2s, color=cols, alpha=0.7, edgecolor='black', linewidth=0.5)
| ax.set_xticks(range(len(labels)))
| ax.set_xticklabels(labels, fontsize=8)
| ax.set_ylabel('R^2')
| ax.set_title('F: Model comparison')
| for i, v in enumerate(r2s):
|     ax.text(i, v+0.01, f'{v:.3f}', ha='center', fontsize=9)
| ax.grid(True, alpha=0.3, axis='y')
|
| fig.suptitle(f'N-1 Layer 2: MOND transition correction (N={N})', fontsize=14, y=1.01)
| fig.tight_layout()
| fig.savefig(os.path.join(BASE, 'fig_N1_layer2.png'), dpi=150)
| print('  -> fig_N1_layer2.png')
|
| # ============================================================
| # Save CSV
| # ============================================================
| outcsv = os.path.join(BASE, 'N1_layer2_results.csv')
| cols_out = ['galaxy','gc_over_a0','log_gc','vflat','hR_kpc','log_hR','log_vflat',
|             'x_outer','log_x_outer','correction','log_correction',
|             'GS0_proxy','log_GS0','GS0_corrected','log_GS0_corr',
|             'r_outer_kpc','r_outer_over_hR']
| with open(outcsv, 'w', encoding='utf-8') as f:
|     f.write(','.join(cols_out)+'¥n')
|     for r in results:
|         f.write(','.join(str(r.get(c,'')) for c in cols_out)+'¥n')
| print(f'  -> {outcsv}')
|
| # ============================================================
| # Verdict
| # ============================================================
| print('¥n' + '='*70)
| print('VERDICT')
| print('='*70)
|
| print(f'¥n  x_outer median: {np.nanmedian(x_outer[valid]):.4f}')
| if np.nanmedian(x_outer[valid]) < 0.05:
|     print('  >>> Most galaxies are deep in MOND regime (x_outer << 1)')
|     print('  >>> Transition correction is negligible for most galaxies')
|     print('  >>> hR partial correlation is NOT explained by transition effect')
| elif np.nanmedian(x_outer[valid]) < 0.3:
|     print('  >>> Moderate MOND departure for typical galaxies')
|     print('  >>> Transition correction may partially explain hR effect')
| else:
|     print('  >>> Significant MOND departure - transition correction is important')
|
| print(f'¥n  hR partial correlation:')
| print(f'    Original (gc vs hR | vflat):              rho = {rho_partial_orig:.4f}')
| print(f'    After correction (gc vs hR | vflat,corr): rho = {rho_partial_new:.4f}')
| reduction = (1-abs(rho_partial_new)/abs(rho_partial_orig))*100
| print(f'    Reduction: {reduction:.1f}%')
|
| if reduction > 70:
|     print('  >>> hR effect EXPLAINED by MOND transition correction')
```

```
    print('  >>> N-1 Layer 2: RESOLVED')
elif reduction > 30:
    print('  >>> hR effect PARTIALLY explained by transition correction')
    print('  >>> N-1 Layer 2: PARTIAL')
else:
    print('  >>> hR effect NOT explained by transition correction')
    print('  >>> N-1 Layer 2: hR contribution is INDEPENDENT of MOND transition')

print(f'¥n  Correction coefficient in multivariate:')
print(f'    coeff = {beta2[2]:.4f} +/- {se2[2]:.4f}, p = {p_corr:.4f}')
if p_corr < 0.05:
    print('  >>> Correction is statistically significant in multivariate model')
else:
    print('  >>> Correction is NOT significant in multivariate model')

print('¥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 -> 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

```
# -*- coding: utf-8 -*-
"""
N-1 Layer 2b: 循環性チェック + Yd-hR 隠れた相関
================================================================
検証1: Yd vs hR の相関 -- hR偏相関がYd変動の代理か
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

実行: uv run --with scipy --with matplotlib python sparc_N1_layer2b.py

必要ファイル:
  - Rotmod_LTG/*.dat
  - phase1/sparc_results.csv
  - TA3_gc_independent.csv

著者: 坂口 忍 (坂口製麺所)
日付: 2026年4月
"""
import os, sys, glob, warnings
import numpy as np
from scipy import stats

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import font_manager as _fm
for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
            r'C:¥Windows¥Fonts¥msgothic.ttc']:
    try: _fm.fontManager.addfont(_fp)
    except: pass
for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
    try:
        plt.rcParams['font.family'] = fontname
        break
    except: continue
plt.rcParams['axes.unicode_minus'] = False
warnings.filterwarnings('ignore')

a0     = 1.2e-10
G_SI   = 6.674e-11
kpc_m  = 3.0857e19

BASE   = os.path.dirname(os.path.abspath(__file__))
ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
```

```
TA3    = os.path.join(BASE, 'TA3_gc_independent.csv')

for p, label in [(ROTMOD,'Rotmod_LTG'),(PHASE1,'sparc_results.csv'),
                 (TA3,'TA3_gc_independent.csv')]:
    if not os.path.exists(p):
        print(f'[ERROR] {label} not found: {p}'); sys.exit(1)

# ============================================================
# Loaders (same as previous)
# ============================================================
def load_csv(path):
    with open(path, 'r', encoding='utf-8-sig') as f:
        header = f.readline().strip()
    sep = ',' if ',' in header else None
    data = {}
    with open(path, 'r', encoding='utf-8-sig') as f:
        cols = [c.strip() for c in f.readline().strip().split(sep)]
        rows = []
        for line in f:
            line = line.strip()
            if not line: continue
            rows.append([p.strip() for p in line.split(sep)])
    for i, col in enumerate(cols):
        vals = []
        for row in rows:
            if i < len(row):
                try:    vals.append(float(row[i]))
                except: vals.append(row[i])
            else: vals.append(np.nan)
        data[col] = vals
    return data

def find_name_col(data):
    for c in ['galaxy','Galaxy','name','Name','GALAXY']:
        if c in data: return c
    for k,v in data.items():
        if isinstance(v[0], str): return k
    return list(data.keys())[0]

def get_key(info, candidates, default=None):
    for c in candidates:
        if c in info:
            try: return float(info[c])
            except: return info[c]
    return default

def load_rotmod(filepath):
    cols = [[] for _ in range(8)]
    with open(filepath, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith('#'): continue
            parts = line.split()
            if len(parts) < 6: continue
            try:
                for j in range(min(len(parts),8)):
                    cols[j].append(float(parts[j]))
                for j in range(len(parts),8):
                    cols[j].append(0.0)
            except ValueError: continue
    return tuple(np.array(c) for c in cols)

# ============================================================
# Load data
# ============================================================
print('[1] Loading data...')
phase1 = load_csv(PHASE1)
ta3    = load_csv(TA3)
p1_nc  = find_name_col(phase1)
ta3_nc = find_name_col(ta3)

galaxy_info = {}
for i, name in enumerate(phase1[p1_nc]):
    name = str(name).strip()
    info = {}
    for k in phase1:
        if k == p1_nc: continue
        try:    info[k] = float(phase1[k][i])
        except: info[k] = phase1[k][i]
    galaxy_info[name] = info

for i, name in enumerate(ta3[ta3_nc]):
    name = str(name).strip()
    if name in galaxy_info:
        for k in ta3:
            if k == ta3_nc: continue
            try:    galaxy_info[name][k] = float(ta3[k][i])
            except: galaxy_info[name][k] = ta3[k][i]
```

```
|
| # ============================================================
| # Process galaxies
| # ============================================================
| print('[2] Processing galaxies...')
| results = []
| rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
|
| for fpath in rotmod_files:
|     gname = os.path.splitext(os.path.basename(fpath))[0].replace('_rotmod','').strip()
|     info = None
|     for key in [gname, gname.upper(), gname.lower()]:
|         if key in galaxy_info:
|             info = galaxy_info[key]; break
|     if info is None: continue
|
|     ud    = get_key(info, ['upsilon_d','Upsilon_d','Ud','ud','Yd'])
|     gc_a0 = get_key(info, ['gc_over_a0','gc/a0','gc_ratio'])
|     vflat = get_key(info, ['vflat','Vflat','v_flat'])
|
|     if ud is None or gc_a0 is None or vflat is None: continue
|     if np.isnan(ud) or np.isnan(gc_a0) or gc_a0<=0 or vflat<=0 or ud<=0: continue
|
|     try:
|         rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
|     except: continue
|     if len(rad) < 3: continue
|
|     vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
|     rpk_idx = np.argmax(vds)
|     rpk = rad[rpk_idx]
|     if rpk < 0.01 or rpk >= rad.max()*0.9: continue
|     hR_kpc = rpk / 2.15
|     hR_m   = hR_kpc * kpc_m
|
|     # g_N at outer points
|     r_m = rad * kpc_m
|     vbar2 = np.abs(vgas**2 + ud * np.sign(vdisk)*vdisk**2
|                    + 0.7 * np.sign(vbul)*vbul**2)
|     g_N = np.zeros_like(r_m)
|     mask_r = r_m > 0
|     g_N[mask_r] = vbar2[mask_r] * 1e6 / r_m[mask_r]
|
|     gc_si = gc_a0 * a0
|     vflat_ms = vflat * 1e3
|     GS0_proxy = vflat_ms**2 / hR_m
|
|     # outer g_N (average of last 3 points)
|     n_out = min(3, len(rad))
|     g_N_outer = np.mean(g_N[-n_out:])
|
|     # --- Correction WITH gc (original, potentially circular) ---
|     if g_N_outer > 0 and gc_si > 0:
|         gf = (g_N_outer + np.sqrt(g_N_outer**2 + 4*gc_si*g_N_outer)) / 2
|         gm = np.sqrt(gc_si * g_N_outer)
|         corr_gc = gf / gm
|     else:
|         corr_gc = np.nan
|
|     # --- Correction WITHOUT gc (gc -> a0, breaks circularity) ---
|     if g_N_outer > 0:
|         gf_a0 = (g_N_outer + np.sqrt(g_N_outer**2 + 4*a0*g_N_outer)) / 2
|         gm_a0 = np.sqrt(a0 * g_N_outer)
|         corr_a0 = gf_a0 / gm_a0
|     else:
|         corr_a0 = np.nan
|
|     # --- Per-point mean corrections (both versions) ---
|     corrs_gc_all = []
|     corrs_a0_all = []
|     for i_r in range(len(g_N)):
|         if g_N[i_r] > 0:
|             gf1 = (g_N[i_r] + np.sqrt(g_N[i_r]**2 + 4*gc_si*g_N[i_r])) / 2
|             gm1 = np.sqrt(gc_si * g_N[i_r])
|             corrs_gc_all.append(gf1/gm1)
|
|             gf2 = (g_N[i_r] + np.sqrt(g_N[i_r]**2 + 4*a0*g_N[i_r])) / 2
|             gm2 = np.sqrt(a0 * g_N[i_r])
|             corrs_a0_all.append(gf2/gm2)
|
|     mean_corr_gc = np.mean(corrs_gc_all) if corrs_gc_all else np.nan
|     mean_corr_a0 = np.mean(corrs_a0_all) if corrs_a0_all else np.nan
|
|     results.append({
|         'galaxy':     gname,
|         'gc_a0':      gc_a0,
|         'log_gc':     np.log10(gc_a0),
|         'vflat':      vflat,
```

```
        'Yd':         ud,
        'log_Yd':     np.log10(ud),
        'hR_kpc':     hR_kpc,
        'log_hR':     np.log10(hR_kpc),
        'log_vflat':  np.log10(vflat),
        'GS0_proxy':  GS0_proxy,
        'log_GS0':    np.log10(GS0_proxy/a0),
        'g_N_outer':  g_N_outer,
        'corr_gc':    corr_gc,
        'log_corr_gc': np.log10(corr_gc) if np.isfinite(corr_gc) and corr_gc>0 else np.nan,
        'corr_a0':    corr_a0,
        'log_corr_a0': np.log10(corr_a0) if np.isfinite(corr_a0) and corr_a0>0 else np.nan,
        'mean_corr_gc': mean_corr_gc,
        'mean_corr_a0': mean_corr_a0,
    })

N = len(results)
print(f'  Processed: {N} galaxies')
if N < 10:
    print('[ERROR] Too few.'); sys.exit(1)

# ============================================================
# Arrays
# ============================================================
log_gc      = np.array([r['log_gc'] for r in results])
log_hR      = np.array([r['log_hR'] for r in results])
log_vflat   = np.array([r['log_vflat'] for r in results])
log_GS0     = np.array([r['log_GS0'] for r in results])
log_Yd      = np.array([r['log_Yd'] for r in results])
Yd_arr      = np.array([r['Yd'] for r in results])
log_corr_gc = np.array([r['log_corr_gc'] for r in results])
log_corr_a0 = np.array([r['log_corr_a0'] for r in results])

# ============================================================
# PART 1: Yd vs hR correlation
# ============================================================
print('\n' + '='*70)
print('PART 1: Yd vs hR hidden correlation')
print('='*70)

mask1 = np.isfinite(log_Yd) & np.isfinite(log_hR)
rho_Yd_hR, p_Yd_hR = stats.spearmanr(log_Yd[mask1], log_hR[mask1])
sl_yh, int_yh, r_yh, p_yh, se_yh = stats.linregress(log_hR[mask1], log_Yd[mask1])
print(f'\n  [1a] Yd vs hR:')
print(f'    Spearman rho = {rho_Yd_hR:.4f}, p = {p_Yd_hR:.2e}')
print(f'    Linear: log(Yd) = {sl_yh:.3f}*log(hR) + {int_yh:.3f}, R^2={r_yh**2:.3f}')

# Yd vs vflat
mask1v = np.isfinite(log_Yd) & np.isfinite(log_vflat)
rho_Yd_vf, p_Yd_vf = stats.spearmanr(log_Yd[mask1v], log_vflat[mask1v])
print(f'\n  [1b] Yd vs vflat:')
print(f'    Spearman rho = {rho_Yd_vf:.4f}, p = {p_Yd_vf:.2e}')

# Yd vs gc
mask1g = np.isfinite(log_Yd) & np.isfinite(log_gc)
rho_Yd_gc, p_Yd_gc = stats.spearmanr(log_Yd[mask1g], log_gc[mask1g])
print(f'\n  [1c] Yd vs gc:')
print(f'    Spearman rho = {rho_Yd_gc:.4f}, p = {p_Yd_gc:.2e}')

# Partial: gc vs hR | (vflat, Yd)
mask_p1 = np.isfinite(log_gc) & np.isfinite(log_hR) & np.isfinite(log_vflat) & np.isfinite(log_Yd)
X_ctrl = np.column_stack([log_vflat[mask_p1], log_Yd[mask_p1], np.ones(mask_p1.sum())])
y_gc = log_gc[mask_p1]
y_hR = log_hR[mask_p1]

b_gc = np.linalg.lstsq(X_ctrl, y_gc, rcond=None)[0]
res_gc = y_gc - X_ctrl @ b_gc
b_hR = np.linalg.lstsq(X_ctrl, y_hR, rcond=None)[0]
res_hR = y_hR - X_ctrl @ b_hR
rho_hR_Yd, p_hR_Yd = stats.spearmanr(res_gc, res_hR)

# Original partial (vflat only)
X_ctrl0 = np.column_stack([log_vflat[mask_p1], np.ones(mask_p1.sum())])
b_gc0 = np.linalg.lstsq(X_ctrl0, y_gc, rcond=None)[0]
res_gc0 = y_gc - X_ctrl0 @ b_gc0
b_hR0 = np.linalg.lstsq(X_ctrl0, y_hR, rcond=None)[0]
res_hR0 = y_hR - X_ctrl0 @ b_hR0
rho_hR_orig, p_hR_orig = stats.spearmanr(res_gc0, res_hR0)

print(f'\n  [1d] hR partial correlations:')
print(f'    gc vs hR | vflat:          rho = {rho_hR_orig:.4f}, p = {p_hR_orig:.4e}')
print(f'    gc vs hR | (vflat, Yd):    rho = {rho_hR_Yd:.4f}, p = {p_hR_Yd:.4e}')
red_yd = (1 - abs(rho_hR_Yd)/abs(rho_hR_orig))*100 if abs(rho_hR_orig)>0 else 0
print(f'    Reduction by adding Yd:    {red_yd:.1f}%')

# Multivariate: gc = a*vflat + b*hR + c*Yd + d
X_m3 = np.column_stack([log_vflat[mask_p1], log_hR[mask_p1], log_Yd[mask_p1],
                        np.ones(mask_p1.sum())])
```

```
b3, _, _, _ = np.linalg.lstsq(X_m3, y_gc, rcond=None)
ss3 = np.sum((y_gc - X_m3 @ b3)**2)
ss_tot = np.sum((y_gc - y_gc.mean())**2)
r2_3 = 1 - ss3/ss_tot
mse3 = ss3 / (len(y_gc)-4)
se3 = np.sqrt(np.diag(mse3 * np.linalg.inv(X_m3.T @ X_m3)))

print(f'¥n  [1e] Multivariate: vflat + hR + Yd')
print(f'    log(gc) = {b3[0]:.4f}*log(vflat) + {b3[1]:.4f}*log(hR) '
      f'+ {b3[2]:.4f}*log(Yd) + {b3[3]:.4f}')
print(f'    SE:        {se3[0]:.4f}                 {se3[1]:.4f}                '
      f'{se3[2]:.4f}')
print(f'    R^2 = {r2_3:.4f}')
t_yd = abs(b3[2]) / se3[2]
p_yd_coeff = 2*(1-stats.t.cdf(t_yd, df=len(y_gc)-4))
print(f'    p(Yd coeff = 0) = {p_yd_coeff:.4f}')

# Compare with vflat+hR only (same mask)
X_m2 = np.column_stack([log_vflat[mask_p1], log_hR[mask_p1], np.ones(mask_p1.sum())])
b2, _, _, _ = np.linalg.lstsq(X_m2, y_gc, rcond=None)
ss2 = np.sum((y_gc - X_m2 @ b2)**2)
r2_2 = 1 - ss2/ss_tot
print(f'    Compare: vflat+hR only R^2 = {r2_2:.4f}')
print(f'    Delta R^2 from Yd: +{r2_3 - r2_2:.4f}')

# ============================================================
# PART 2: gc-free correction (circularity removal)
# ============================================================
print('¥n' + '='*70)
print('PART 2: gc-free correction (gc -> a0)')
print('='*70)

# Compare distributions
valid_c = np.isfinite(log_corr_gc) & np.isfinite(log_corr_a0)
print(f'¥n  [2a] Correction distributions (N={np.sum(valid_c)}):')
print(f'    With gc:  median = {np.nanmedian(np.array([r["corr_gc"] for r in results])[valid_c]):.4f}')
print(f'    With a0:  median = {np.nanmedian(np.array([r["corr_a0"] for r in results])[valid_c]):.4f}')

# Correlation between two corrections
rho_cc, p_cc = stats.spearmanr(log_corr_gc[valid_c], log_corr_a0[valid_c])
print(f'¥n  [2b] corr_gc vs corr_a0: rho = {rho_cc:.4f}, p = {p_cc:.2e}')

# corr_a0 vs gc (should be LESS correlated than corr_gc vs gc if circularity matters)
mask_cg = np.isfinite(log_corr_gc) & np.isfinite(log_gc)
rho_cg_gc, p_cg_gc = stats.spearmanr(log_corr_gc[mask_cg], log_gc[mask_cg])
mask_ca = np.isfinite(log_corr_a0) & np.isfinite(log_gc)
rho_ca_gc, p_ca_gc = stats.spearmanr(log_corr_a0[mask_ca], log_gc[mask_ca])
print(f'¥n  [2c] Circularity check:')
print(f'    corr_gc vs gc: rho = {rho_cg_gc:.4f}, p = {p_cg_gc:.2e}  (circular)')
print(f'    corr_a0 vs gc: rho = {rho_ca_gc:.4f}, p = {p_ca_gc:.2e}  (gc-free)')

# Multivariate with gc-free correction
mask_2m = (np.isfinite(log_gc) & np.isfinite(log_vflat) &
           np.isfinite(log_hR) & np.isfinite(log_corr_a0))

print(f'¥n  [2d] Multivariate with gc-free correction (N={np.sum(mask_2m)})')

y2 = log_gc[mask_2m]
ss_tot2 = np.sum((y2 - y2.mean())**2)

# Without correction
X_nc = np.column_stack([log_vflat[mask_2m], log_hR[mask_2m], np.ones(mask_2m.sum())])
b_nc, _, _, _ = np.linalg.lstsq(X_nc, y2, rcond=None)
r2_nc = 1 - np.sum((y2 - X_nc @ b_nc)**2)/ss_tot2
mse_nc = np.sum((y2 - X_nc @ b_nc)**2) / (len(y2)-3)
se_nc = np.sqrt(np.diag(mse_nc * np.linalg.inv(X_nc.T @ X_nc)))

print(f'    G1: vflat+hR: R^2 = {r2_nc:.4f}')
print(f'        vflat^{b_nc[0]:.3f}+/-{se_nc[0]:.3f}, hR^{b_nc[1]:.3f}+/-{se_nc[1]:.3f}')

# With corr_gc (circular)
X_cg = np.column_stack([log_vflat[mask_2m], log_hR[mask_2m],
                        log_corr_gc[mask_2m], np.ones(mask_2m.sum())])
b_cg, _, _, _ = np.linalg.lstsq(X_cg, y2, rcond=None)
r2_cg = 1 - np.sum((y2 - X_cg @ b_cg)**2)/ss_tot2
mse_cg = np.sum((y2 - X_cg @ b_cg)**2) / (len(y2)-4)
se_cg = np.sqrt(np.diag(mse_cg * np.linalg.inv(X_cg.T @ X_cg)))
t_cg = abs(b_cg[2]) / se_cg[2]
p_cg = 2*(1-stats.t.cdf(t_cg, df=len(y2)-4))

print(f'¥n    G2: vflat+hR+corr_gc (circular): R^2 = {r2_cg:.4f}')
print(f'        vflat^{b_cg[0]:.3f}, hR^{b_cg[1]:.3f}, corr_gc^{b_cg[2]:.3f}+/-{se_cg[2]:.3f}')
print(f'        p(corr_gc=0) = {p_cg:.4f}')

# With corr_a0 (gc-free)
X_ca = np.column_stack([log_vflat[mask_2m], log_hR[mask_2m],
                        log_corr_a0[mask_2m], np.ones(mask_2m.sum())])
b_ca, _, _, _ = np.linalg.lstsq(X_ca, y2, rcond=None)
```

```
r2_ca = 1 - np.sum((y2 - X_ca @ b_ca)**2)/ss_tot2
mse_ca = np.sum((y2 - X_ca @ b_ca)**2) / (len(y2)-4)
se_ca = np.sqrt(np.diag(mse_ca * np.linalg.inv(X_ca.T @ X_ca)))
t_ca = abs(b_ca[2]) / se_ca[2]
p_ca = 2*(1-stats.t.cdf(t_ca, df=len(y2)-4))

print(f'¥n    G3: vflat+hR+corr_a0 (gc-free): R^2 = {r2_ca:.4f}')
print(f'        vflat^{b_ca[0]:.3f}+/-{se_ca[0]:.3f}, hR^{b_ca[1]:.3f}+/-{se_ca[1]:.3f}, '
      f'corr_a0^{b_ca[2]:.3f}+/-{se_ca[2]:.3f}')
print(f'        p(corr_a0=0) = {p_ca:.4f}')

# hR partial with gc-free correction controlled
X_ctrl2 = np.column_stack([log_vflat[mask_2m], log_corr_a0[mask_2m],
                           np.ones(mask_2m.sum())])
b_gc2 = np.linalg.lstsq(X_ctrl2, y2, rcond=None)[0]
res_gc2 = y2 - X_ctrl2 @ b_gc2
y_hR2 = log_hR[mask_2m]
b_hR2 = np.linalg.lstsq(X_ctrl2, y_hR2, rcond=None)[0]
res_hR2 = y_hR2 - X_ctrl2 @ b_hR2
rho_hR_a0, p_hR_a0 = stats.spearmanr(res_gc2, res_hR2)

# Original partial (same mask)
X_ctrl0b = np.column_stack([log_vflat[mask_2m], np.ones(mask_2m.sum())])
b_gc0b = np.linalg.lstsq(X_ctrl0b, y2, rcond=None)[0]
res_gc0b = y2 - X_ctrl0b @ b_gc0b
b_hR0b = np.linalg.lstsq(X_ctrl0b, y_hR2, rcond=None)[0]
res_hR0b = y_hR2 - X_ctrl0b @ b_hR0b
rho_hR_0b, p_hR_0b = stats.spearmanr(res_gc0b, res_hR0b)

print(f'¥n  [2e] hR partial correlations:')
print(f'    gc vs hR | vflat:               rho = {rho_hR_0b:.4f}')
print(f'    gc vs hR | (vflat, corr_gc):    rho = (from Layer2: -0.227)')
print(f'    gc vs hR | (vflat, corr_a0):    rho = {rho_hR_a0:.4f}, p = {p_hR_a0:.4e}')
red_a0 = (1 - abs(rho_hR_a0)/abs(rho_hR_0b))*100 if abs(rho_hR_0b)>0 else 0
print(f'    Reduction (gc-free correction):  {red_a0:.1f}%')

# ============================================================
# PART 3: Full 4-variable model
# ============================================================
print('¥n' + '='*70)
print('PART 3: Full model comparison')
print('='*70)

mask_all = (np.isfinite(log_gc) & np.isfinite(log_vflat) & np.isfinite(log_hR)
            & np.isfinite(log_Yd) & np.isfinite(log_corr_a0))

y_all = log_gc[mask_all]
ss_all = np.sum((y_all - y_all.mean())**2)

models = {}

# M1: vflat + hR
X_1 = np.column_stack([log_vflat[mask_all], log_hR[mask_all], np.ones(mask_all.sum())])
b_1 = np.linalg.lstsq(X_1, y_all, rcond=None)[0]
r2_m1 = 1 - np.sum((y_all - X_1 @ b_1)**2)/ss_all
models['vflat+hR'] = {'r2': r2_m1, 'k': 2, 'beta': b_1}

# M2: vflat + hR + Yd
X_2 = np.column_stack([log_vflat[mask_all], log_hR[mask_all], log_Yd[mask_all],
                       np.ones(mask_all.sum())])
b_2 = np.linalg.lstsq(X_2, y_all, rcond=None)[0]
r2_m2 = 1 - np.sum((y_all - X_2 @ b_2)**2)/ss_all
mse_m2 = np.sum((y_all - X_2 @ b_2)**2) / (len(y_all)-4)
se_m2 = np.sqrt(np.diag(mse_m2 * np.linalg.inv(X_2.T @ X_2)))
models['vflat+hR+Yd'] = {'r2': r2_m2, 'k': 3, 'beta': b_2, 'se': se_m2}

# M3: vflat + hR + corr_a0
X_3 = np.column_stack([log_vflat[mask_all], log_hR[mask_all], log_corr_a0[mask_all],
                       np.ones(mask_all.sum())])
b_3 = np.linalg.lstsq(X_3, y_all, rcond=None)[0]
r2_m3 = 1 - np.sum((y_all - X_3 @ b_3)**2)/ss_all
mse_m3 = np.sum((y_all - X_3 @ b_3)**2) / (len(y_all)-4)
se_m3 = np.sqrt(np.diag(mse_m3 * np.linalg.inv(X_3.T @ X_3)))
models['vflat+hR+corr_a0'] = {'r2': r2_m3, 'k': 3, 'beta': b_3, 'se': se_m3}

# M4: vflat + hR + Yd + corr_a0
X_4 = np.column_stack([log_vflat[mask_all], log_hR[mask_all], log_Yd[mask_all],
                       log_corr_a0[mask_all], np.ones(mask_all.sum())])
b_4 = np.linalg.lstsq(X_4, y_all, rcond=None)[0]
r2_m4 = 1 - np.sum((y_all - X_4 @ b_4)**2)/ss_all
mse_m4 = np.sum((y_all - X_4 @ b_4)**2) / (len(y_all)-5)
se_m4 = np.sqrt(np.diag(mse_m4 * np.linalg.inv(X_4.T @ X_4)))
models['vflat+hR+Yd+corr_a0'] = {'r2': r2_m4, 'k': 4, 'beta': b_4, 'se': se_m4}

n_all = mask_all.sum()
print(f'¥n  N = {n_all}')
print(f'¥n  {"Model":<28s} | {"R^2":>6s} | {"AIC":>8s} | k')
print('  ' + '-'*55)
```

```
for name, m in models.items():
    k = m['k']
    r2 = m['r2']
    ss_r = (1-r2) * ss_all
    aic = n_all * np.log(ss_r/n_all) + 2*(k+1)
    m['aic'] = aic
    print(f'  {name:<28s} | {r2:>6.4f} | {aic:>8.1f} | {k}')

aic_base = models['vflat+hR']['aic']
print(f'¥n  dAIC vs baseline (vflat+hR):')
for name, m in models.items():
    print(f'    {name:<28s}: dAIC = {m["aic"]-aic_base:>+7.1f}')

print(f'¥n  Full model (M4) coefficients:')
print(f'    vflat:   {b_4[0]:.4f} +/- {se_m4[0]:.4f}')
print(f'    hR:      {b_4[1]:.4f} +/- {se_m4[1]:.4f}')
print(f'    Yd:      {b_4[2]:.4f} +/- {se_m4[2]:.4f}')
print(f'    corr_a0: {b_4[3]:.4f} +/- {se_m4[3]:.4f}')

# Final partial: hR after everything
X_fin = np.column_stack([log_vflat[mask_all], log_Yd[mask_all],
                         log_corr_a0[mask_all], np.ones(mask_all.sum())])
b_fin_gc = np.linalg.lstsq(X_fin, y_all, rcond=None)[0]
res_fin_gc = y_all - X_fin @ b_fin_gc
y_hR_all = log_hR[mask_all]
b_fin_hR = np.linalg.lstsq(X_fin, y_hR_all, rcond=None)[0]
res_fin_hR = y_hR_all - X_fin @ b_fin_hR
rho_fin, p_fin = stats.spearmanr(res_fin_gc, res_fin_hR)

print(f'¥n  [Final] gc vs hR | (vflat, Yd, corr_a0):')
print(f'    rho = {rho_fin:.4f}, p = {p_fin:.4e}')
print(f'    Original (| vflat only): rho = {rho_hR_orig:.4f}')
total_red = (1 - abs(rho_fin)/abs(rho_hR_orig))*100 if abs(rho_hR_orig)>0 else 0
print(f'    Total reduction: {total_red:.1f}%')

# ============================================================
# Figures
# ============================================================
print('¥n[3] Generating figures...')
fig, axes = plt.subplots(2, 3, figsize=(17, 11))

# 1: Yd vs hR
ax = axes[0, 0]
m = np.isfinite(log_Yd) & np.isfinite(log_hR)
ax.scatter(log_hR[m], log_Yd[m], s=12, alpha=0.5, c='steelblue', edgecolors='none')
xf = np.linspace(log_hR[m].min(), log_hR[m].max(), 100)
ax.plot(xf, sl_yh*xf + int_yh, 'r-', lw=2, label=f'rho={rho_Yd_hR:.3f}, p={p_Yd_hR:.1e}')
ax.set_xlabel('log(hR / kpc)')
ax.set_ylabel('log(Yd)')
ax.set_title('1: Yd vs hR correlation')
ax.legend(fontsize=8)
ax.grid(True, alpha=0.3)

# 2: corr_gc vs corr_a0
ax = axes[0, 1]
vc = np.isfinite(log_corr_gc) & np.isfinite(log_corr_a0)
ax.scatter(log_corr_a0[vc], log_corr_gc[vc], s=12, alpha=0.5, c='darkorange', edgecolors='none')
lim = [min(log_corr_a0[vc].min(), log_corr_gc[vc].min()),
       max(log_corr_a0[vc].max(), log_corr_gc[vc].max())]
ax.plot(lim, lim, 'k--', lw=1, label='1:1')
ax.set_xlabel('log(corr_a0) [gc-free]')
ax.set_ylabel('log(corr_gc) [circular]')
ax.set_title(f'2: Two corrections (rho={rho_cc:.3f})')
ax.legend(fontsize=8)
ax.grid(True, alpha=0.3)

# 3: corr_a0 vs gc (should be weaker)
ax = axes[0, 2]
ax.scatter(log_corr_a0[mask_ca], log_gc[mask_ca], s=12, alpha=0.5, c='green', edgecolors='none')
ax.set_xlabel('log(corr_a0) [gc-free]')
ax.set_ylabel('log(gc/a0)')
ax.set_title(f'3: corr_a0 vs gc (rho={rho_ca_gc:.3f})')
ax.grid(True, alpha=0.3)

# 4: R^2 comparison
ax = axes[1, 0]
names = list(models.keys())
r2s = [models[n]['r2'] for n in names]
cols = ['steelblue', 'darkorange', 'green', 'red']
ax.bar(range(len(names)), r2s, color=cols[:len(names)], alpha=0.7,
       edgecolor='black', linewidth=0.5)
ax.set_xticks(range(len(names)))
ax.set_xticklabels([n.replace('+','+¥n') for n in names], fontsize=7)
ax.set_ylabel('R^2')
ax.set_title('4: Model comparison')
for i,v in enumerate(r2s):
    ax.text(i, v+0.01, f'{v:.3f}', ha='center', fontsize=8)
ax.grid(True, alpha=0.3, axis='y')
```

```
|
| # 5: hR partial correlation progression
| ax = axes[1, 1]
| stages = ['| vflat', '| vflat,Yd', '| vflat,corr_a0', '| vflat,Yd,corr_a0']
| rhos = [rho_hR_orig, rho_hR_Yd, rho_hR_a0, rho_fin]
| barcolors = ['steelblue', 'darkorange', 'green', 'red']
| ax.bar(range(len(stages)), [abs(r) for r in rhos], color=barcolors, alpha=0.7,
|        edgecolor='black', linewidth=0.5)
| ax.set_xticks(range(len(stages)))
| ax.set_xticklabels(stages, fontsize=7, rotation=15)
| ax.set_ylabel('|rho| (hR partial)')
| ax.set_title('5: hR partial reduction')
| for i,v in enumerate(rhos):
|     ax.text(i, abs(v)+0.01, f'{v:.3f}', ha='center', fontsize=8)
| ax.axhline(0.1, color='grey', ls=':', label='negligible threshold')
| ax.legend(fontsize=7)
| ax.grid(True, alpha=0.3, axis='y')
|
| # 6: Residual from full model
| ax = axes[1, 2]
| resid_full = y_all - X_4 @ b_4
| ax.hist(resid_full, bins=25, color='purple', alpha=0.7, edgecolor='white')
| ax.axvline(0, color='red', ls='--')
| ax.set_xlabel('Residual [dex]')
| ax.set_ylabel('Count')
| ax.set_title(f'6: Full model residual (sigma={np.std(resid_full):.3f})')
|
| fig.suptitle(f'N-1 Layer 2b: Circularity + Yd check (N={N})', fontsize=14, y=1.01)
| fig.tight_layout()
| fig.savefig(os.path.join(BASE, 'fig_N1_layer2b.png'), dpi=150)
| print('  -> fig_N1_layer2b.png')
|
| # ============================================================
| # Save CSV
| # ============================================================
| outcsv = os.path.join(BASE, 'N1_layer2b_results.csv')
| cols_out = ['galaxy','gc_a0','log_gc','vflat','Yd','log_Yd','hR_kpc','log_hR',
|             'log_vflat','log_GS0','corr_gc','log_corr_gc','corr_a0','log_corr_a0']
| with open(outcsv, 'w', encoding='utf-8') as f:
|     f.write(','.join(cols_out)+'¥n')
|     for r in results:
|         f.write(','.join(str(r.get(c,'')) for c in cols_out)+'¥n')
| print(f'  -> {outcsv}')
|
| # ============================================================
| # Final Verdict
| # ============================================================
| print('¥n' + '='*70)
| print('FINAL VERDICT')
| print('='*70)
|
| print(f'¥n  PART 1: Yd-hR correlation')
| if abs(rho_Yd_hR) > 0.3 and p_Yd_hR < 0.01:
|     print(f'    >>> Yd and hR ARE correlated (rho={rho_Yd_hR:.3f})')
|     print(f'    >>> hR partial may partly reflect Yd variation')
| else:
|     print(f'    >>> Yd and hR are NOT strongly correlated (rho={rho_Yd_hR:.3f})')
|     print(f'    >>> hR effect is INDEPENDENT of Yd')
|
| print(f'¥n  PART 2: Circularity')
| print(f'    corr_gc vs gc: rho = {rho_cg_gc:.3f} (circular)')
| print(f'    corr_a0 vs gc: rho = {rho_ca_gc:.3f} (gc-free)')
| if abs(rho_ca_gc) < abs(rho_cg_gc) * 0.5:
|     print(f'    >>> Layer 2 R^2 boost was LARGELY circular')
| elif abs(rho_ca_gc) < abs(rho_cg_gc) * 0.8:
|     print(f'    >>> Layer 2 R^2 boost was PARTIALLY circular')
| else:
|     print(f'    >>> Layer 2 R^2 boost was GENUINE (not circular)')
|
| print(f'¥n  PART 3: hR partial after all controls')
| print(f'    Original:  rho = {rho_hR_orig:.4f}')
| print(f'    Final:     rho = {rho_fin:.4f}')
| print(f'    Reduction: {total_red:.1f}%')
| if abs(rho_fin) < 0.1:
|     print(f'    >>> hR effect FULLY EXPLAINED by Yd + transition')
| elif abs(rho_fin) < 0.2:
|     print(f'    >>> hR effect MOSTLY explained ({total_red:.0f}%)')
| else:
|     print(f'    >>> hR effect PARTIALLY explained ({total_red:.0f}%), residual remains')
|
| 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 ~ Sigma_bar が予測されるのにgc vs Sigma_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 = vflat^4/(G*M_bar) vs hR
- [*] Test 2: gc vs Sigma_bar (V-1改の再確認)
- [*] Test 4: vflat^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

```
# -*- coding: utf-8 -*-
"""
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
======================================================================
式(5') + 式(5) の連立から gc = η□ × 2π × a□² × G × Σ_bar が導出される。
しかし V-1改で gc vs Σ_bar は R²=0.016。この矛盾の源を特定する。

テスト1: BTFR残差 = vflat□/(G×M_bar_direct) vs hR
  → BTFRにhR依存性があるか

テスト2: gc vs M_bar/hR² (= Σ_bar) の直接回帰
  → 代数的予測 gc ∝ Σ_bar の再確認

テスト3: η_eff = gc / (η□² × a□ × vflat²/hR) の残差構造
  → ηがΣ_bar依存性を持ち相殺しているか

テスト4: BTFR成立度 -- vflat□ vs gc×G×M_bar の直接確認
  → BTFR自体がどの程度成立しているか

テスト5: U(ε;c) の曲率 U''(ε_eq;c) とgcの関係
  → c を経験式(8)から計算し、U'' とgcの相関を確認

実行: uv run --with scipy --with matplotlib python sparc_N1_layer3.py

著者: 坂口 忍 (坂口製麺所)
日付: 2026年4月
"""
import os, sys, glob, warnings
import numpy as np
from scipy import stats

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import font_manager as _fm
for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
            r'C:¥Windows¥Fonts¥msgothic.ttc']:
    try: _fm.fontManager.addfont(_fp)
    except: pass
for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
    try:
        plt.rcParams['font.family'] = fontname
        break
    except: continue
```

```
plt.rcParams['axes.unicode_minus'] = False
warnings.filterwarnings('ignore')

a0     = 1.2e-10
G_SI   = 6.674e-11
kpc_m  = 3.0857e19
Msun   = 1.989e30
pc_m   = 3.0857e16

BASE   = os.path.dirname(os.path.abspath(__file__))
ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
TA3    = os.path.join(BASE, 'TA3_gc_independent.csv')

for p, label in [(ROTMOD,'Rotmod_LTG'),(PHASE1,'sparc_results.csv'),
                 (TA3,'TA3_gc_independent.csv')]:
    if not os.path.exists(p):
        print(f'[ERROR] {label} not found: {p}'); sys.exit(1)

# ============================================================
# Loaders
# ============================================================
def load_csv(path):
    with open(path, 'r', encoding='utf-8-sig') as f:
        header = f.readline().strip()
    sep = ',' if ',' in header else None
    data = {}
    with open(path, 'r', encoding='utf-8-sig') as f:
        cols = [c.strip() for c in f.readline().strip().split(sep)]
        rows = []
        for line in f:
            line = line.strip()
            if not line: continue
            rows.append([p.strip() for p in line.split(sep)])
    for i, col in enumerate(cols):
        vals = []
        for row in rows:
            if i < len(row):
                try:    vals.append(float(row[i]))
                except: vals.append(row[i])
            else: vals.append(np.nan)
        data[col] = vals
    return data

def find_name_col(data):
    for c in ['galaxy','Galaxy','name','Name','GALAXY']:
        if c in data: return c
    for k,v in data.items():
        if isinstance(v[0], str): return k
    return list(data.keys())[0]

def get_key(info, candidates, default=None):
    for c in candidates:
        if c in info:
            try: return float(info[c])
            except: return info[c]
    return default

def load_rotmod(filepath):
    cols = [[] for _ in range(8)]
    with open(filepath, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith('#'): continue
            parts = line.split()
            if len(parts) < 6: continue
            try:
                for j in range(min(len(parts),8)):
                    cols[j].append(float(parts[j]))
                for j in range(len(parts),8):
                    cols[j].append(0.0)
            except ValueError: continue
    return tuple(np.array(c) for c in cols)

# ============================================================
# U(ε;c) functions
# ============================================================
def U(eps, c):
    """U(ε;c) = -ε - ε²/2 - c*ln(1-ε)"""
    if eps >= 1.0 or eps < 0:
        return np.nan
    return -eps - eps**2/2.0 - c * np.log(1.0 - eps)

def U_prime(eps, c):
    """U'(ε;c) = -1 - ε + c/(1-ε)"""
    if eps >= 1.0:
        return np.nan
    return -1.0 - eps + c/(1.0 - eps)
```

```
|
| def U_double_prime(eps, c):
|     """U''(ε;c) = -1 + c/(1-ε)²"""
|     if eps >= 1.0:
|         return np.nan
|     return -1.0 + c / (1.0 - eps)**2
|
| def epsilon_eq(x, c):
|     """ε_eq(x,c) = (-x + sqrt((x+2)² - 4c)) / 2"""
|     disc = (x + 2.0)**2 - 4.0*c
|     if disc < 0:
|         return np.nan
|     return (-x + np.sqrt(disc)) / 2.0
|
| # ============================================================
| # Load data
| # ============================================================
| print('[1] Loading data...')
| phase1 = load_csv(PHASE1)
| ta3    = load_csv(TA3)
| p1_nc  = find_name_col(phase1)
| ta3_nc = find_name_col(ta3)
|
| galaxy_info = {}
| for i, name in enumerate(phase1[p1_nc]):
|     name = str(name).strip()
|     info = {}
|     for k in phase1:
|         if k == p1_nc: continue
|         try:    info[k] = float(phase1[k][i])
|         except: info[k] = phase1[k][i]
|     galaxy_info[name] = info
|
| for i, name in enumerate(ta3[ta3_nc]):
|     name = str(name).strip()
|     if name in galaxy_info:
|         for k in ta3:
|             if k == ta3_nc: continue
|             try:    galaxy_info[name][k] = float(ta3[k][i])
|             except: galaxy_info[name][k] = ta3[k][i]
|
| # ============================================================
| # Process
| # ============================================================
| print('[2] Processing galaxies...')
| results = []
| rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))
|
| for fpath in rotmod_files:
|     gname = os.path.splitext(os.path.basename(fpath))[0].replace('_rotmod','').strip()
|     info = None
|     for key in [gname, gname.upper(), gname.lower()]:
|         if key in galaxy_info:
|             info = galaxy_info[key]; break
|     if info is None: continue
|
|     ud    = get_key(info, ['upsilon_d','Upsilon_d','Ud','ud','Yd'])
|     gc_a0 = get_key(info, ['gc_over_a0','gc/a0','gc_ratio'])
|     vflat = get_key(info, ['vflat','Vflat','v_flat'])
|     # T-type for c_fit calculation
|     ttype = get_key(info, ['T','t_type','T_type','ttype'])
|     # Stellar mass
|     log_mstar = get_key(info, ['log_Mstar','logMstar','log_M*'])
|     # Surface brightness
|     log_sb = get_key(info, ['log_SBdisk','logSBdisk','SBdisk'])
|
|     if ud is None or gc_a0 is None or vflat is None: continue
|     if np.isnan(ud) or np.isnan(gc_a0) or gc_a0<=0 or vflat<=0 or ud<=0: continue
|
|     try:
|         rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
|     except: continue
|     if len(rad) < 3: continue
|
|     vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
|     rpk_idx = np.argmax(vds)
|     rpk = rad[rpk_idx]
|     if rpk < 0.01 or rpk >= rad.max()*0.9: continue
|     hR_kpc = rpk / 2.15
|     hR_m   = hR_kpc * kpc_m
|
|     vflat_ms = vflat * 1e3
|     gc_si = gc_a0 * a0
|     GS0_proxy = vflat_ms**2 / hR_m
|
|     # M_bar direct (BT formula, gc-independent)
|     vdisk_peak_ms = np.max(np.sqrt(max(ud, 0.01)) * np.abs(vdisk)) * 1e3
|     M_disk = 2.0 * vdisk_peak_ms**2 * hR_m / (0.56 * G_SI)
```

```
    vgas_peak_ms = np.max(np.abs(vgas)) * 1e3
    M_gas = 2.0 * vgas_peak_ms**2 * hR_m / (0.56 * G_SI) if vgas_peak_ms > 0 else 0
    M_bar_direct = M_disk + M_gas
    M_bar_sun = M_bar_direct / Msun

    # Sigma_bar direct
    Sigma_bar = M_disk / (2 * np.pi * hR_m**2)  # kg/m^2
    G_Sigma_bar = G_SI * Sigma_bar

    # --- Test 1: BTFR residual ---
    # BTFR: vflat^4 = gc * G * M_bar
    # Residual: R_BTFR = vflat^4 / (G * M_bar_direct)
    # If BTFR is exact, R_BTFR = gc
    if M_bar_direct > 0:
        R_BTFR = vflat_ms**4 / (G_SI * M_bar_direct)
        log_R_BTFR = np.log10(R_BTFR / a0)
    else:
        R_BTFR = np.nan
        log_R_BTFR = np.nan

    # BTFR ratio: should be gc if BTFR exact
    btfr_ratio = R_BTFR / gc_si if gc_si > 0 and np.isfinite(R_BTFR) else np.nan

    # --- Test 3: η residual ---
    # From proxy: gc = η² × a□ × vflat²/hR
    # So η² = gc / (a□ × vflat²/hR) = gc / (a□ × GS0_proxy)
    eta_sq = gc_si / (a0 * GS0_proxy) if GS0_proxy > 0 else np.nan
    log_eta_sq = np.log10(eta_sq) if np.isfinite(eta_sq) and eta_sq > 0 else np.nan

    # --- Test 5: c_fit from empirical formula (8) ---
    # log c_fit = -0.894 + 0.278*log(M*/10^9 Msun) - 0.046*T + 0.153*log(SBdisk)
    c_fit = np.nan
    eps_eq_val = np.nan
    U_pp = np.nan

    if (log_mstar is not None and ttype is not None and log_sb is not None
        and not np.isnan(log_mstar) and not np.isnan(ttype) and not np.isnan(log_sb)):
        # log_mstar in SPARC is log10(M*/Msun), need log10(M*/10^9 Msun) = log_mstar - 9
        log_m9 = log_mstar - 9.0
        log_c_fit = -0.894 + 0.278*log_m9 - 0.046*ttype + 0.153*log_sb
        c_fit = 10**log_c_fit

        # Compute ε_eq at x = g_N/gc ≈ 0.5 (typical transition)
        # Use x=0.5 as representative
        x_rep = 0.5
        if 0 < c_fit < 10:
            eps_eq_val = epsilon_eq(x_rep, min(c_fit, 0.999))
            if np.isfinite(eps_eq_val) and 0 < eps_eq_val < 1:
                U_pp = U_double_prime(eps_eq_val, min(c_fit, 0.999))

    results.append({
        'galaxy':       gname,
        'gc_a0':        gc_a0,
        'log_gc':       np.log10(gc_a0),
        'vflat':        vflat,
        'Yd':           ud,
        'log_Yd':       np.log10(ud),
        'hR_kpc':       hR_kpc,
        'log_hR':       np.log10(hR_kpc),
        'log_vflat':    np.log10(vflat),
        'GS0_proxy':    GS0_proxy,
        'log_GS0':      np.log10(GS0_proxy/a0),
        'M_bar_sun':    M_bar_sun,
        'log_Mbar':     np.log10(M_bar_sun) if M_bar_sun > 0 else np.nan,
        'G_Sigma_bar':  G_Sigma_bar,
        'log_GSbar':    np.log10(G_Sigma_bar/a0) if G_Sigma_bar > 0 else np.nan,
        'R_BTFR':       R_BTFR,
        'log_R_BTFR':   log_R_BTFR,
        'btfr_ratio':   btfr_ratio,
        'log_btfr_ratio': np.log10(btfr_ratio) if np.isfinite(btfr_ratio) and btfr_ratio > 0 else np.nan,
        'eta_sq':       eta_sq,
        'log_eta_sq':   log_eta_sq,
        'ttype':        ttype if ttype is not None else np.nan,
        'log_mstar':    log_mstar if log_mstar is not None else np.nan,
        'log_sb':       log_sb if log_sb is not None else np.nan,
        'c_fit':        c_fit,
        'log_c_fit':    np.log10(c_fit) if np.isfinite(c_fit) and c_fit > 0 else np.nan,
        'eps_eq':       eps_eq_val,
        'U_pp':         U_pp,
    })

N = len(results)
print(f'  Processed: {N} galaxies')
if N < 10:
    print('[ERROR] Too few.'); sys.exit(1)

# ============================================================
# Arrays
```

```
# ============================================================
log_gc       = np.array([r['log_gc'] for r in results])
log_hR       = np.array([r['log_hR'] for r in results])
log_vflat    = np.array([r['log_vflat'] for r in results])
log_GS0      = np.array([r['log_GS0'] for r in results])
log_Mbar     = np.array([r['log_Mbar'] for r in results])
log_GSbar    = np.array([r['log_GSbar'] for r in results])
log_R_BTFR   = np.array([r['log_R_BTFR'] for r in results])
log_btfr_r   = np.array([r['log_btfr_ratio'] for r in results])
log_eta_sq   = np.array([r['log_eta_sq'] for r in results])
log_Yd       = np.array([r['log_Yd'] for r in results])
log_c_fit    = np.array([r['log_c_fit'] for r in results])
U_pp_arr     = np.array([r['U_pp'] for r in results])
c_fit_arr    = np.array([r['c_fit'] for r in results])

# ============================================================
# Test 1: BTFR residual vs hR
# ============================================================
print('¥n' + '='*70)
print('TEST 1: BTFR residual vs hR')
print('='*70)

mask1 = np.isfinite(log_btfr_r) & np.isfinite(log_hR)
print(f'¥n  N = {np.sum(mask1)}')

# BTFR ratio distribution
br = np.array([r['btfr_ratio'] for r in results])
br_valid = br[np.isfinite(br) & (br > 0)]
print(f'  BTFR ratio = vflat^4/(G*M_bar_direct*gc):')
print(f'    median = {np.nanmedian(br_valid):.2f}')
print(f'    IQR = [{np.nanpercentile(br_valid,25):.2f}, {np.nanpercentile(br_valid,75):.2f}]')
print(f'    (should be ~1.0 if BTFR exact with M_bar_direct)')

# BTFR residual vs hR
rho_br_hR, p_br_hR = stats.spearmanr(log_btfr_r[mask1], log_hR[mask1])
sl_br, int_br, r_br, _, se_br = stats.linregress(log_hR[mask1], log_btfr_r[mask1])
print(f'¥n  log(BTFR_ratio) vs log(hR):')
print(f'    Spearman rho = {rho_br_hR:.4f}, p = {p_br_hR:.2e}')
print(f'    slope = {sl_br:.3f} +/- {se_br:.3f}, R^2 = {r_br**2:.3f}')

# Also check: does R_BTFR predict gc?
mask1b = np.isfinite(log_R_BTFR) & np.isfinite(log_gc)
sl_rb, int_rb, r_rb, _, se_rb = stats.linregress(log_R_BTFR[mask1b], log_gc[mask1b])
rho_rb, p_rb = stats.spearmanr(log_R_BTFR[mask1b], log_gc[mask1b])
print(f'¥n  log(R_BTFR/a0) vs log(gc/a0):')
print(f'    slope = {sl_rb:.3f} +/- {se_rb:.3f}, R^2 = {r_rb**2:.3f}')
print(f'    Spearman rho = {rho_rb:.4f}')
print(f'    (should be slope=1, R^2=1 if BTFR exact)')

# ============================================================
# Test 2: gc vs Σ_bar (re-confirm V-1改)
# ============================================================
print('¥n' + '='*70)
print('TEST 2: gc vs Sigma_bar (re-confirmation)')
print('='*70)

mask2 = np.isfinite(log_GSbar) & np.isfinite(log_gc)
sl2, int2, r2, _, se2 = stats.linregress(log_GSbar[mask2], log_gc[mask2])
rho2, p2 = stats.spearmanr(log_GSbar[mask2], log_gc[mask2])
print(f'¥n  N = {np.sum(mask2)}')
print(f'  slope = {sl2:.4f} +/- {se2:.4f}, R^2 = {r2**2:.4f}')
print(f'  Spearman rho = {rho2:.4f}, p = {p2:.2e}')
print(f'  (V-1改 reported: slope=0.080, R^2=0.016)')

# ============================================================
# Test 3: η residual structure
# ============================================================
print('¥n' + '='*70)
print('TEST 3: eta residual structure')
print('='*70)

mask3 = np.isfinite(log_eta_sq)
print(f'¥n  N = {np.sum(mask3)}')
print(f'  log(eta^2) distribution:')
print(f'    median = {np.nanmedian(log_eta_sq[mask3]):.4f}')
print(f'    std = {np.nanstd(log_eta_sq[mask3]):.4f} dex')
print(f'    eta = 10^(median/2) = {10**(np.nanmedian(log_eta_sq[mask3])/2):.4f}')

# Does η correlate with Σ_bar? If so, η cancels the Σ_bar dependence
mask3s = np.isfinite(log_eta_sq) & np.isfinite(log_GSbar)
rho_eta_S, p_eta_S = stats.spearmanr(log_eta_sq[mask3s], log_GSbar[mask3s])
sl_eS, int_eS, r_eS, _, se_eS = stats.linregress(log_GSbar[mask3s], log_eta_sq[mask3s])
print(f'¥n  log(eta^2) vs log(G*Sigma_bar/a0):')
print(f'    Spearman rho = {rho_eta_S:.4f}, p = {p_eta_S:.2e}')
print(f'    slope = {sl_eS:.3f} +/- {se_eS:.3f}, R^2 = {r_eS**2:.3f}')
print(f'    If eta cancels Sigma_bar: expect slope ~ -1.0')

# η vs hR
```

```
mask3h = np.isfinite(log_eta_sq) & np.isfinite(log_hR)
rho_eta_hR, p_eta_hR = stats.spearmanr(log_eta_sq[mask3h], log_hR[mask3h])
print(f'¥n  log(eta^2) vs log(hR):')
print(f'    Spearman rho = {rho_eta_hR:.4f}, p = {p_eta_hR:.2e}')

# η vs Yd (known from 5-5: eta ~ Yd^-0.44)
mask3y = np.isfinite(log_eta_sq) & np.isfinite(log_Yd)
rho_eta_Yd, p_eta_Yd = stats.spearmanr(log_eta_sq[mask3y], log_Yd[mask3y])
sl_eY, int_eY, r_eY, _, se_eY = stats.linregress(log_Yd[mask3y], log_eta_sq[mask3y])
print(f'¥n  log(eta^2) vs log(Yd):')
print(f'    Spearman rho = {rho_eta_Yd:.4f}, p = {p_eta_Yd:.2e}')
print(f'    slope = {sl_eY:.3f} +/- {se_eY:.3f}')
print(f'    (5-5 reports eta ~ Yd^-0.44, so expect slope ~ -0.88 for eta^2)')

# ============================================================
# Test 4: BTFR quality check
# ============================================================
print('¥n' + '='*70)
print('TEST 4: BTFR direct check')
print('='*70)

# vflat^4 vs M_bar_direct (gc-free)
log_vf4 = 4 * log_vflat  # log(vflat^4)
mask4 = np.isfinite(log_vf4) & np.isfinite(log_Mbar)
sl4, int4, r4, _, se4 = stats.linregress(log_Mbar[mask4], log_vf4[mask4])
rho4, p4 = stats.spearmanr(log_Mbar[mask4], log_vf4[mask4])
print(f'¥n  N = {np.sum(mask4)}')
print(f'  log(vflat^4) vs log(M_bar_direct):')
print(f'    slope = {sl4:.3f} +/- {se4:.3f} (expect 1.0 for BTFR)')
print(f'    R^2 = {r4**2:.3f}')
print(f'    Spearman rho = {rho4:.4f}')

# p(slope=1)
t4 = abs(sl4 - 1.0) / se4
p4s = 2*(1-stats.t.cdf(t4, df=np.sum(mask4)-2))
print(f'    p(slope=1.0) = {p4s:.4f}')

# BTFR with hR correction: vflat^4 vs M_bar * hR^b
# Multivariate: log(vflat^4) = a*log(M_bar) + b*log(hR) + c
mask4m = np.isfinite(log_vf4) & np.isfinite(log_Mbar) & np.isfinite(log_hR)
X4 = np.column_stack([log_Mbar[mask4m], log_hR[mask4m], np.ones(mask4m.sum())])
y4 = log_vf4[mask4m]
b4, _, _, _ = np.linalg.lstsq(X4, y4, rcond=None)
ss4 = np.sum((y4 - X4 @ b4)**2)
ss4t = np.sum((y4 - y4.mean())**2)
r2_4m = 1 - ss4/ss4t
mse4m = ss4 / (len(y4)-3)
se4m = np.sqrt(np.diag(mse4m * np.linalg.inv(X4.T @ X4)))

print(f'¥n  Multivariate BTFR: log(vflat^4) = {b4[0]:.3f}*log(M_bar) + {b4[1]:.3f}*log(hR)')
print(f'    SE: {se4m[0]:.3f}, {se4m[1]:.3f}')
print(f'    R^2 = {r2_4m:.3f} (vs univariate {r4**2:.3f})')
t_hR4 = abs(b4[1]) / se4m[1]
p_hR4 = 2*(1-stats.t.cdf(t_hR4, df=len(y4)-3))
print(f'    p(hR coeff = 0) = {p_hR4:.4f}')
print(f'    -> hR adds to BTFR? {"YES" if p_hR4 < 0.05 else "NO"}')

# ============================================================
# Test 5: U''(ε_eq; c_fit) vs gc
# ============================================================
print('¥n' + '='*70)
print('TEST 5: U¥'¥'(epsilon_eq; c_fit) vs gc')
print('='*70)

mask5 = np.isfinite(log_c_fit) & np.isfinite(log_gc) & np.isfinite(U_pp_arr)
n5 = np.sum(mask5)
print(f'¥n  N with valid c_fit and U¥'¥': {n5}')

if n5 > 10:
    # c_fit vs gc
    rho_cg, p_cg = stats.spearmanr(log_c_fit[mask5], log_gc[mask5])
    sl_cg, int_cg, r_cg, _, se_cg = stats.linregress(log_c_fit[mask5], log_gc[mask5])
    print(f'¥n  log(c_fit) vs log(gc/a0):')
    print(f'    Spearman rho = {rho_cg:.4f}, p = {p_cg:.2e}')
    print(f'    slope = {sl_cg:.3f} +/- {se_cg:.3f}, R^2 = {r_cg**2:.3f}')
    print(f'    (gc = c*a0 predicts slope=1.0)')
    t5 = abs(sl_cg - 1.0) / se_cg
    p5s = 2*(1-stats.t.cdf(t5, df=n5-2))
    print(f'    p(slope=1.0) = {p5s:.4f}')

    # U'' vs gc
    log_Upp = np.log10(np.abs(U_pp_arr[mask5]))
    sign_Upp = np.sign(U_pp_arr[mask5])
    rho_ug, p_ug = stats.spearmanr(log_Upp, log_gc[mask5])
    print(f'¥n  log|U¥'¥'| vs log(gc/a0):')
    print(f'    Spearman rho = {rho_ug:.4f}, p = {p_ug:.2e}')
    print(f'    U¥'¥' > 0 fraction: {np.sum(sign_Upp > 0)/len(sign_Upp):.1%}')
    print(f'    (U¥'¥' > 0 means stable equilibrium)')
```

```
|
|     # c_fit distribution
|     cf_valid = c_fit_arr[np.isfinite(c_fit_arr) & (c_fit_arr > 0)]
|     print(f'¥n  c_fit distribution:')
|     print(f'    median = {np.nanmedian(cf_valid):.3f}')
|     print(f'    IQR = [{np.nanpercentile(cf_valid,25):.3f}, {np.nanpercentile(cf_valid,75):.3f}]')
|     print(f'    fraction c_fit > 1: {np.sum(cf_valid > 1)/len(cf_valid):.1%}')
| else:
|     print('  Too few galaxies with c_fit data. Skipping.')
|     rho_cg = r_cg = np.nan
|
| # ============================================================
| # Figures
| # ============================================================
| print('¥n[3] Generating figures...')
| fig, axes = plt.subplots(2, 3, figsize=(17, 11))
|
| # 1: BTFR ratio distribution
| ax = axes[0, 0]
| br_log = log_btfr_r[np.isfinite(log_btfr_r)]
| ax.hist(br_log, bins=30, color='steelblue', alpha=0.7, edgecolor='white')
| ax.axvline(0, color='red', ls='--', lw=2, label='BTFR exact (ratio=1)')
| ax.set_xlabel('log(vflat^4 / (G*M_bar*gc))')
| ax.set_ylabel('Count')
| ax.set_title('1: BTFR ratio (should be ~0)')
| ax.legend(fontsize=8)
|
| # 2: BTFR residual vs hR
| ax = axes[0, 1]
| m = np.isfinite(log_btfr_r) & np.isfinite(log_hR)
| ax.scatter(log_hR[m], log_btfr_r[m], s=10, alpha=0.5, c='steelblue', edgecolors='none')
| xf = np.linspace(log_hR[m].min(), log_hR[m].max(), 100)
| ax.plot(xf, sl_br*xf + int_br, 'r-', lw=2,
|         label=f'rho={rho_br_hR:.3f}, slope={sl_br:.3f}')
| ax.set_xlabel('log(hR / kpc)')
| ax.set_ylabel('log(BTFR ratio)')
| ax.set_title('2: BTFR residual vs hR')
| ax.legend(fontsize=8)
| ax.grid(True, alpha=0.3)
|
| # 3: η² vs Σ_bar
| ax = axes[0, 2]
| m3 = np.isfinite(log_eta_sq) & np.isfinite(log_GSbar)
| ax.scatter(log_GSbar[m3], log_eta_sq[m3], s=10, alpha=0.5, c='darkorange', edgecolors='none')
| xf = np.linspace(log_GSbar[m3].min(), log_GSbar[m3].max(), 100)
| ax.plot(xf, sl_eS*xf + int_eS, 'r-', lw=2,
|         label=f'slope={sl_eS:.3f} (cancel=-1.0)')
| ax.plot(xf, -1.0*xf + int_eS, 'b--', lw=1, label='exact cancellation')
| ax.set_xlabel('log(G*Sigma_bar / a0)')
| ax.set_ylabel('log(eta^2)')
| ax.set_title('3: Does eta cancel Sigma_bar?')
| ax.legend(fontsize=8)
| ax.grid(True, alpha=0.3)
|
| # 4: BTFR direct
| ax = axes[1, 0]
| m4 = np.isfinite(log_Mbar) & np.isfinite(log_vf4)
| ax.scatter(log_Mbar[m4], log_vf4[m4], s=10, alpha=0.5, c='green', edgecolors='none')
| xf = np.linspace(log_Mbar[m4].min(), log_Mbar[m4].max(), 100)
| ax.plot(xf, sl4*xf + int4, 'r-', lw=2, label=f'slope={sl4:.3f}, R^2={r4**2:.3f}')
| ax.plot(xf, 1.0*xf + int4, 'b--', lw=1, label='BTFR: slope=1')
| ax.set_xlabel('log(M_bar / M_sun) [direct]')
| ax.set_ylabel('log(vflat^4)')
| ax.set_title('4: BTFR quality')
| ax.legend(fontsize=8)
| ax.grid(True, alpha=0.3)
|
| # 5: c_fit vs gc
| ax = axes[1, 1]
| m5 = np.isfinite(log_c_fit) & np.isfinite(log_gc)
| if np.sum(m5) > 5:
|     ax.scatter(log_c_fit[m5], log_gc[m5], s=10, alpha=0.5, c='purple', edgecolors='none')
|     xf = np.linspace(log_c_fit[m5].min(), log_c_fit[m5].max(), 100)
|     if np.isfinite(r_cg):
|         ax.plot(xf, sl_cg*xf + int_cg, 'r-', lw=2,
|                 label=f'slope={sl_cg:.3f}, R^2={r_cg**2:.3f}')
|     ax.plot(xf, 1.0*xf, 'b--', lw=1, label='gc=c*a0 (slope=1)')
|     ax.set_xlabel('log(c_fit)')
|     ax.set_ylabel('log(gc/a0)')
|     ax.set_title('5: c_fit vs gc (eq.8 -> gc=c*a0?)')
|     ax.legend(fontsize=8)
|     ax.grid(True, alpha=0.3)
|
| # 6: Summary R² bar chart
| ax = axes[1, 2]
| labels = ['proxy¥nalpha=0.5', 'M_bar¥ndirect', 'Sigma_bar¥ndirect',
|           'R_BTFR¥n=vf^4/GM', 'c_fit¥n(eq.8)']
| r2s = [
```

```
    stats.linregress(log_GS0[np.isfinite(log_GS0)&np.isfinite(log_gc)],
                     log_gc[np.isfinite(log_GS0)&np.isfinite(log_gc)])[2]**2,
    stats.linregress(log_Mbar[np.isfinite(log_Mbar)&np.isfinite(log_gc)],
                     log_gc[np.isfinite(log_Mbar)&np.isfinite(log_gc)])[2]**2,
    r2**2 if np.isfinite(r2) else 0,
    r_rb**2 if np.isfinite(r_rb) else 0,
    r_cg**2 if np.isfinite(r_cg) else 0,
]
cols = ['steelblue', 'darkorange', 'green', 'crimson', 'purple']
ax.bar(range(len(labels)), r2s, color=cols, alpha=0.7, edgecolor='black', linewidth=0.5)
ax.set_xticks(range(len(labels)))
ax.set_xticklabels(labels, fontsize=8)
ax.set_ylabel('R^2')
ax.set_title('6: gc prediction comparison')
for i, v in enumerate(r2s):
    ax.text(i, v+0.01, f'{v:.3f}', ha='center', fontsize=8)
ax.grid(True, alpha=0.3, axis='y')

fig.suptitle(f'N-1 Layer 3: Self-consistency algebra (N={N})', fontsize=14, y=1.01)
fig.tight_layout()
fig.savefig(os.path.join(BASE, 'fig_N1_layer3.png'), dpi=150)
print('  -> fig_N1_layer3.png')

# ============================================================
# Save CSV
# ============================================================
outcsv = os.path.join(BASE, 'N1_layer3_results.csv')
cols_out = ['galaxy','gc_a0','log_gc','vflat','Yd','hR_kpc','log_hR',
            'log_GS0','log_Mbar','log_GSbar','log_R_BTFR','log_btfr_ratio',
            'log_eta_sq','c_fit','log_c_fit','eps_eq','U_pp']
with open(outcsv, 'w', encoding='utf-8') as f:
    f.write(','.join(cols_out)+'¥n')
    for r in results:
        f.write(','.join(str(r.get(c,'')) for c in cols_out)+'¥n')
print(f'  -> {outcsv}')

# ============================================================
# Verdict
# ============================================================
print('¥n' + '='*70)
print('VERDICT')
print('='*70)

print(f'¥n  TEST 1: BTFR residual vs hR')
if abs(rho_br_hR) > 0.3 and p_br_hR < 0.001:
    print(f'    >>> BTFR has hR dependence (rho={rho_br_hR:.3f})')
    print(f'    >>> Standard BTFR is NOT exact - hR modifies it')
elif abs(rho_br_hR) > 0.15 and p_br_hR < 0.05:
    print(f'    >>> Weak hR dependence in BTFR (rho={rho_br_hR:.3f})')
else:
    print(f'    >>> No significant hR dependence (rho={rho_br_hR:.3f})')

print(f'¥n  TEST 2: gc vs Sigma_bar')
print(f'    >>> Confirmed: R^2={r2**2:.3f} (unchanged from V-1改)')

print(f'¥n  TEST 3: eta cancellation')
if abs(sl_eS + 1.0) < 2*se_eS:
    print(f'    >>> eta DOES cancel Sigma_bar (slope={sl_eS:.3f} ~ -1.0)')
    print(f'    >>> This explains why gc vs Sigma_bar is flat!')
elif abs(sl_eS) > 0.3:
    print(f'    >>> eta PARTIALLY cancels Sigma_bar (slope={sl_eS:.3f})')
else:
    print(f'    >>> eta does NOT cancel Sigma_bar (slope={sl_eS:.3f})')

print(f'¥n  TEST 4: BTFR quality')
print(f'    >>> vflat^4 vs M_bar_direct: slope={sl4:.3f}, R^2={r4**2:.3f}')
if p_hR4 < 0.05:
    print(f'    >>> hR adds to BTFR (p={p_hR4:.4f}): BTFR has hR residual')
else:
    print(f'    >>> hR does NOT add to BTFR (p={p_hR4:.4f})')

print(f'¥n  TEST 5: c_fit (eq.8) vs gc')
if n5 > 10 and np.isfinite(r_cg):
    print(f'    >>> slope={sl_cg:.3f}, R^2={r_cg**2:.3f}')
    if abs(sl_cg - 1.0) < 2*se_cg:
        print(f'    >>> gc = c_fit * a0 is CONSISTENT')
    else:
        print(f'    >>> gc = c_fit * a0 is NOT consistent (slope != 1)')

print(f'¥n  SYNTHESIS:')
print(f'    The gc-Sigma_bar contradiction is resolved if:')
print(f'    (a) eta has Sigma_bar dependence that cancels -> Test 3')
print(f'    (b) BTFR itself has hR dependence -> Test 1 + Test 4')
print(f'    (c) Both effects combine')

print('¥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) -> gc^2/(a0*GS0)。

主要テスト

- [*] 健全性チェック: eta ~ O(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: R^2=0.000。eta vs vflat: R^2=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

```
# -*- coding: utf-8 -*-
"""
N-1 Layer 3 修正版: η²計算バグ修正 + Test 3 再実行
======================================================================
バグ: eta_sq = gc / (a0 * GS0)  ← gc の一乗
修正: eta_sq = gc^2 / (a0 * GS0)  ← gc の二乗

根拠:
  gc = eta * sqrt(a0 * GS0)
  gc^2 = eta^2 * a0 * GS0
  eta^2 = gc^2 / (a0 * GS0)

Test 1,2,4,5 はeta非依存なので前回結果を維持。
Test 3 のみ再実行。

実行: uv run --with scipy --with matplotlib python sparc_N1_layer3_fix.py

著者: 坂口 忍 (坂口製麺所)
日付: 2026年4月
"""
import os, sys, glob, warnings
import numpy as np
from scipy import stats

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import font_manager as _fm
for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf',
            r'C:¥Windows¥Fonts¥msgothic.ttc']:
    try: _fm.fontManager.addfont(_fp)
    except: pass
for fontname in ['IPAGothic', 'MS Gothic', 'DejaVu Sans']:
    try:
        plt.rcParams['font.family'] = fontname
        break
    except: continue
plt.rcParams['axes.unicode_minus'] = False
warnings.filterwarnings('ignore')

a0     = 1.2e-10
G_SI   = 6.674e-11
```

```
kpc_m  = 3.0857e19
Msun   = 1.989e30
pc_m   = 3.0857e16

BASE   = os.path.dirname(os.path.abspath(__file__))
ROTMOD = os.path.join(BASE, 'Rotmod_LTG')
PHASE1 = os.path.join(BASE, 'phase1', 'sparc_results.csv')
TA3    = os.path.join(BASE, 'TA3_gc_independent.csv')

for p, label in [(ROTMOD,'Rotmod_LTG'),(PHASE1,'sparc_results.csv'),
                 (TA3,'TA3_gc_independent.csv')]:
    if not os.path.exists(p):
        print(f'[ERROR] {label} not found: {p}'); sys.exit(1)

# ============================================================
# Loaders
# ============================================================
def load_csv(path):
    with open(path, 'r', encoding='utf-8-sig') as f:
        header = f.readline().strip()
    sep = ',' if ',' in header else None
    data = {}
    with open(path, 'r', encoding='utf-8-sig') as f:
        cols = [c.strip() for c in f.readline().strip().split(sep)]
        rows = []
        for line in f:
            line = line.strip()
            if not line: continue
            rows.append([p.strip() for p in line.split(sep)])
    for i, col in enumerate(cols):
        vals = []
        for row in rows:
            if i < len(row):
                try:    vals.append(float(row[i]))
                except: vals.append(row[i])
            else: vals.append(np.nan)
        data[col] = vals
    return data

def find_name_col(data):
    for c in ['galaxy','Galaxy','name','Name','GALAXY']:
        if c in data: return c
    for k,v in data.items():
        if isinstance(v[0], str): return k
    return list(data.keys())[0]

def get_key(info, candidates, default=None):
    for c in candidates:
        if c in info:
            try: return float(info[c])
            except: return info[c]
    return default

def load_rotmod(filepath):
    cols = [[] for _ in range(8)]
    with open(filepath, 'r') as f:
        for line in f:
            line = line.strip()
            if not line or line.startswith('#'): continue
            parts = line.split()
            if len(parts) < 6: continue
            try:
                for j in range(min(len(parts),8)):
                    cols[j].append(float(parts[j]))
                for j in range(len(parts),8):
                    cols[j].append(0.0)
            except ValueError: continue
    return tuple(np.array(c) for c in cols)

# ============================================================
# U(ε;c) functions
# ============================================================
def epsilon_eq(x, c):
    disc = (x + 2.0)**2 - 4.0*c
    if disc < 0: return np.nan
    return (-x + np.sqrt(disc)) / 2.0

def U_double_prime(eps, c):
    if eps >= 1.0: return np.nan
    return -1.0 + c / (1.0 - eps)**2

# ============================================================
# Load & process
# ============================================================
print('[1] Loading data...')
phase1 = load_csv(PHASE1)
ta3    = load_csv(TA3)
p1_nc  = find_name_col(phase1)
```

```
ta3_nc = find_name_col(ta3)

galaxy_info = {}
for i, name in enumerate(phase1[p1_nc]):
    name = str(name).strip()
    info = {}
    for k in phase1:
        if k == p1_nc: continue
        try:    info[k] = float(phase1[k][i])
        except: info[k] = phase1[k][i]
    galaxy_info[name] = info

for i, name in enumerate(ta3[ta3_nc]):
    name = str(name).strip()
    if name in galaxy_info:
        for k in ta3:
            if k == ta3_nc: continue
            try:    galaxy_info[name][k] = float(ta3[k][i])
            except: galaxy_info[name][k] = ta3[k][i]

print('[2] Processing galaxies...')
results = []
rotmod_files = sorted(glob.glob(os.path.join(ROTMOD, '*.dat')))

for fpath in rotmod_files:
    gname = os.path.splitext(os.path.basename(fpath))[0].replace('_rotmod','').strip()
    info = None
    for key in [gname, gname.upper(), gname.lower()]:
        if key in galaxy_info:
            info = galaxy_info[key]; break
    if info is None: continue

    ud    = get_key(info, ['upsilon_d','Upsilon_d','Ud','ud','Yd'])
    gc_a0 = get_key(info, ['gc_over_a0','gc/a0','gc_ratio'])
    vflat = get_key(info, ['vflat','Vflat','v_flat'])
    ttype = get_key(info, ['T','t_type','T_type','ttype'])
    log_mstar = get_key(info, ['log_Mstar','logMstar','log_M*'])
    log_sb = get_key(info, ['log_SBdisk','logSBdisk','SBdisk'])

    if ud is None or gc_a0 is None or vflat is None: continue
    if np.isnan(ud) or np.isnan(gc_a0) or gc_a0<=0 or vflat<=0 or ud<=0: continue

    try:
        rad, vobs, errv, vgas, vdisk, vbul, sbdisk, sbbul = load_rotmod(fpath)
    except: continue
    if len(rad) < 3: continue

    vds = np.sqrt(max(ud, 0.01)) * np.abs(vdisk)
    rpk_idx = np.argmax(vds)
    rpk = rad[rpk_idx]
    if rpk < 0.01 or rpk >= rad.max()*0.9: continue
    hR_kpc = rpk / 2.15
    hR_m   = hR_kpc * kpc_m

    vflat_ms = vflat * 1e3
    gc_si = gc_a0 * a0
    GS0_proxy = vflat_ms**2 / hR_m

    # M_bar direct (BT)
    vdisk_peak_ms = np.max(np.sqrt(max(ud, 0.01)) * np.abs(vdisk)) * 1e3
    M_disk = 2.0 * vdisk_peak_ms**2 * hR_m / (0.56 * G_SI)
    vgas_peak_ms = np.max(np.abs(vgas)) * 1e3
    M_gas = 2.0 * vgas_peak_ms**2 * hR_m / (0.56 * G_SI) if vgas_peak_ms > 0 else 0
    M_bar_direct = M_disk + M_gas
    M_bar_sun = M_bar_direct / Msun

    Sigma_bar = M_disk / (2 * np.pi * hR_m**2)
    G_Sigma_bar = G_SI * Sigma_bar

    # --- FIXED: η² = gc² / (a□ × GS0_proxy) ---
    eta_sq = gc_si**2 / (a0 * GS0_proxy) if GS0_proxy > 0 else np.nan
    log_eta_sq = np.log10(eta_sq) if np.isfinite(eta_sq) and eta_sq > 0 else np.nan

    # η (not η²)
    eta_val = np.sqrt(eta_sq) if np.isfinite(eta_sq) and eta_sq > 0 else np.nan
    log_eta = np.log10(eta_val) if np.isfinite(eta_val) and eta_val > 0 else np.nan

    # BTFR residual
    if M_bar_direct > 0:
        R_BTFR = vflat_ms**4 / (G_SI * M_bar_direct)
        btfr_ratio = R_BTFR / gc_si if gc_si > 0 else np.nan
    else:
        R_BTFR = np.nan
        btfr_ratio = np.nan

    # c_fit from eq(8)
    c_fit = np.nan
    U_pp = np.nan
```

```
    if (log_mstar is not None and ttype is not None and log_sb is not None
        and not np.isnan(log_mstar) and not np.isnan(ttype) and not np.isnan(log_sb)):
        log_m9 = log_mstar - 9.0
        log_c_fit_val = -0.894 + 0.278*log_m9 - 0.046*ttype + 0.153*log_sb
        c_fit = 10**log_c_fit_val
        x_rep = 0.5
        if 0 < c_fit < 10:
            eps_eq_val = epsilon_eq(x_rep, min(c_fit, 0.999))
            if np.isfinite(eps_eq_val) and 0 < eps_eq_val < 1:
                U_pp = U_double_prime(eps_eq_val, min(c_fit, 0.999))

    results.append({
        'galaxy':       gname,
        'gc_a0':        gc_a0,
        'log_gc':       np.log10(gc_a0),
        'vflat':        vflat,
        'Yd':           ud,
        'log_Yd':       np.log10(ud),
        'hR_kpc':       hR_kpc,
        'log_hR':       np.log10(hR_kpc),
        'log_vflat':    np.log10(vflat),
        'GS0_proxy':    GS0_proxy,
        'log_GS0':      np.log10(GS0_proxy/a0),
        'M_bar_sun':    M_bar_sun,
        'log_Mbar':     np.log10(M_bar_sun) if M_bar_sun > 0 else np.nan,
        'G_Sigma_bar':  G_Sigma_bar,
        'log_GSbar':    np.log10(G_Sigma_bar/a0) if G_Sigma_bar > 0 else np.nan,
        'eta_sq':       eta_sq,
        'log_eta_sq':   log_eta_sq,
        'eta':          eta_val,
        'log_eta':      log_eta,
        'btfr_ratio':   btfr_ratio,
        'log_btfr_r':   np.log10(btfr_ratio) if np.isfinite(btfr_ratio) and btfr_ratio > 0 else np.nan,
        'c_fit':        c_fit,
        'log_c_fit':    np.log10(c_fit) if np.isfinite(c_fit) and c_fit > 0 else np.nan,
        'U_pp':         U_pp,
    })

N = len(results)
print(f'  Processed: {N} galaxies')
if N < 10:
    print('[ERROR] Too few.'); sys.exit(1)

# ============================================================
# Arrays
# ============================================================
log_gc       = np.array([r['log_gc'] for r in results])
log_hR       = np.array([r['log_hR'] for r in results])
log_vflat    = np.array([r['log_vflat'] for r in results])
log_GS0      = np.array([r['log_GS0'] for r in results])
log_Mbar     = np.array([r['log_Mbar'] for r in results])
log_GSbar    = np.array([r['log_GSbar'] for r in results])
log_eta_sq   = np.array([r['log_eta_sq'] for r in results])
log_eta      = np.array([r['log_eta'] for r in results])
log_Yd       = np.array([r['log_Yd'] for r in results])
log_c_fit    = np.array([r['log_c_fit'] for r in results])
log_btfr_r   = np.array([r['log_btfr_r'] for r in results])
eta_arr      = np.array([r['eta'] for r in results])
c_fit_arr    = np.array([r['c_fit'] for r in results])
U_pp_arr     = np.array([r['U_pp'] for r in results])

# ============================================================
# Sanity check: η values
# ============================================================
print('¥n' + '='*70)
print('SANITY CHECK: eta values (FIXED)')
print('='*70)

m_eta = np.isfinite(log_eta)
print(f'¥n  N = {np.sum(m_eta)}')
print(f'  eta distribution:')
print(f'    median = {np.nanmedian(eta_arr[m_eta]):.4f}')
print(f'    IQR = [{np.nanpercentile(eta_arr[m_eta],25):.4f}, '
      f'{np.nanpercentile(eta_arr[m_eta],75):.4f}]')
print(f'    range = [{np.nanmin(eta_arr[m_eta]):.4f}, {np.nanmax(eta_arr[m_eta]):.4f}]')
print(f'    (5-5 reports eta0 ~ O(1). Should be ~0.1-1.0)')

print(f'¥n  eta^2 distribution:')
eta_sq_arr = np.array([r['eta_sq'] for r in results])
print(f'    median = {np.nanmedian(eta_sq_arr[m_eta]):.6f}')
print(f'    log10(eta^2) median = {np.nanmedian(log_eta_sq[m_eta]):.4f}')

# Verification: pick a typical galaxy
for r in results:
    if 0.2 < r['gc_a0'] < 0.3 and 80 < r['vflat'] < 120:
        print(f'¥n  Verification galaxy: {r["galaxy"]}')
        print(f'    gc = {r["gc_a0"]:.3f} a0 = {r["gc_a0"]*a0:.3e} m/s^2')
        print(f'    vflat = {r["vflat"]:.1f} km/s')
```

```
        print(f'    hR = {r["hR_kpc"]:.2f} kpc')
        print(f'    GS0 = {r["GS0_proxy"]:.3e} m/s^2')
        print(f'    eta^2 = gc^2/(a0*GS0) = {r["eta_sq"]:.6f}')
        print(f'    eta = {r["eta"]:.4f}')
        print(f'    Check: eta*sqrt(a0*GS0) = {r["eta"]*np.sqrt(a0*r["GS0_proxy"]):.3e} '
              f'vs gc = {r["gc_a0"]*a0:.3e}')
        break

# ============================================================
# TEST 3 CORRECTED: η structure
# ============================================================
print('\n' + '='*70)
print('TEST 3 CORRECTED: eta residual structure')
print('='*70)

# --- 3a: η vs Σ_bar ---
mask3s = np.isfinite(log_eta) & np.isfinite(log_GSbar)
rho_eS, p_eS = stats.spearmanr(log_eta[mask3s], log_GSbar[mask3s])
sl_eS, int_eS, r_eS, _, se_eS = stats.linregress(log_GSbar[mask3s], log_eta[mask3s])
print(f'\n  [3a] log(eta) vs log(G*Sigma_bar/a0):')
print(f'    N = {np.sum(mask3s)}')
print(f'    Spearman rho = {rho_eS:.4f}, p = {p_eS:.2e}')
print(f'    slope = {sl_eS:.4f} +/- {se_eS:.4f}, R^2 = {r_eS**2:.4f}')
print(f'    If eta cancels Sigma_bar: expect slope ~ -0.5')
print(f'    (because gc=eta*sqrt(a0*GS0) and gc~const*Sigma_bar^0')
print(f'     -> eta ~ Sigma_bar^(-0.5) to cancel)')
t_cancel = abs(sl_eS - (-0.5)) / se_eS
p_cancel = 2*(1-stats.t.cdf(t_cancel, df=np.sum(mask3s)-2))
print(f'    p(slope=-0.5) = {p_cancel:.4f}')

# --- 3b: η vs hR ---
mask3h = np.isfinite(log_eta) & np.isfinite(log_hR)
rho_eH, p_eH = stats.spearmanr(log_eta[mask3h], log_hR[mask3h])
sl_eH, int_eH, r_eH, _, se_eH = stats.linregress(log_hR[mask3h], log_eta[mask3h])
print(f'\n  [3b] log(eta) vs log(hR):')
print(f'    Spearman rho = {rho_eH:.4f}, p = {p_eH:.2e}')
print(f'    slope = {sl_eH:.4f} +/- {se_eH:.4f}, R^2 = {r_eH**2:.4f}')

# --- 3c: η vs vflat ---
mask3v = np.isfinite(log_eta) & np.isfinite(log_vflat)
rho_eV, p_eV = stats.spearmanr(log_eta[mask3v], log_vflat[mask3v])
sl_eV, int_eV, r_eV, _, se_eV = stats.linregress(log_vflat[mask3v], log_eta[mask3v])
print(f'\n  [3c] log(eta) vs log(vflat):')
print(f'    Spearman rho = {rho_eV:.4f}, p = {p_eV:.2e}')
print(f'    slope = {sl_eV:.4f} +/- {se_eV:.4f}, R^2 = {r_eV**2:.4f}')

# --- 3d: η vs Yd (known from 5-5: eta ~ Yd^-0.44) ---
mask3y = np.isfinite(log_eta) & np.isfinite(log_Yd)
rho_eY, p_eY = stats.spearmanr(log_eta[mask3y], log_Yd[mask3y])
sl_eY, int_eY, r_eY, _, se_eY = stats.linregress(log_Yd[mask3y], log_eta[mask3y])
print(f'\n  [3d] log(eta) vs log(Yd):')
print(f'    Spearman rho = {rho_eY:.4f}, p = {p_eY:.2e}')
print(f'    slope = {sl_eY:.4f} +/- {se_eY:.4f}')
print(f'    (5-5 predicts: slope ~ -0.44)')
t_yd = abs(sl_eY - (-0.44)) / se_eY
p_yd = 2*(1-stats.t.cdf(t_yd, df=np.sum(mask3y)-2))
print(f'    p(slope=-0.44) = {p_yd:.4f}')

# --- 3e: η vs M_bar direct ---
mask3m = np.isfinite(log_eta) & np.isfinite(log_Mbar)
rho_eM, p_eM = stats.spearmanr(log_eta[mask3m], log_Mbar[mask3m])
sl_eM, int_eM, r_eM, _, se_eM = stats.linregress(log_Mbar[mask3m], log_eta[mask3m])
print(f'\n  [3e] log(eta) vs log(M_bar_direct):')
print(f'    Spearman rho = {rho_eM:.4f}, p = {p_eM:.2e}')
print(f'    slope = {sl_eM:.4f} +/- {se_eM:.4f}, R^2 = {r_eM**2:.4f}')

# --- 3f: η vs c_fit ---
mask3c = np.isfinite(log_eta) & np.isfinite(log_c_fit)
if np.sum(mask3c) > 10:
    rho_eC, p_eC = stats.spearmanr(log_eta[mask3c], log_c_fit[mask3c])
    sl_eC, int_eC, r_eC, _, se_eC = stats.linregress(log_c_fit[mask3c], log_eta[mask3c])
    print(f'\n  [3f] log(eta) vs log(c_fit):')
    print(f'    Spearman rho = {rho_eC:.4f}, p = {p_eC:.2e}')
    print(f'    slope = {sl_eC:.4f} +/- {se_eC:.4f}, R^2 = {r_eC**2:.4f}')
    print(f'    (gc=c*a0 means eta=gc/sqrt(a0*GS0)=c*a0/sqrt(a0*GS0)=c*sqrt(a0/GS0))')
else:
    rho_eC = r_eC = sl_eC = np.nan
    print(f'\n  [3f] Too few c_fit values ({np.sum(mask3c)})')

# --- 3g: η vs GS0_proxy (the tautology check) ---
# Since eta = gc/sqrt(a0*GS0), and gc ~ GS0^0.5 (the geometric mean law),
# eta ~ GS0^0.5 / GS0^0.5 = constant (if law is exact)
# Any deviation shows eta's dependence structure
mask3g = np.isfinite(log_eta) & np.isfinite(log_GS0)
rho_eG, p_eG = stats.spearmanr(log_eta[mask3g], log_GS0[mask3g])
sl_eG, int_eG, r_eG, _, se_eG = stats.linregress(log_GS0[mask3g], log_eta[mask3g])
print(f'\n  [3g] log(eta) vs log(GS0_proxy/a0):')
print(f'    Spearman rho = {rho_eG:.4f}, p = {p_eG:.2e}')
```

```
print(f'    slope = {sl_eG:.4f} +/- {se_eG:.4f}, R^2 = {r_eG**2:.4f}')
print(f'    If gc=eta*sqrt(a0*GS0) exact with const eta: slope=0')
print(f'    Residual from 0.5 law: slope = {sl_eG:.4f} = alpha_eff - 0.5 = {sl_eG+0.5:.4f}')

# ============================================================
# TEST 5 RECHECK: c_fit vs gc
# ============================================================
print('¥n' + '='*70)
print('TEST 5 RECHECK: c_fit vs gc')
print('='*70)

mask5 = np.isfinite(log_c_fit) & np.isfinite(log_gc)
n5 = np.sum(mask5)
if n5 > 10:
    sl5, int5, r5, _, se5 = stats.linregress(log_c_fit[mask5], log_gc[mask5])
    rho5, p5 = stats.spearmanr(log_c_fit[mask5], log_gc[mask5])
    print(f'¥n  N = {n5}')
    print(f'  log(gc/a0) vs log(c_fit):')
    print(f'    slope = {sl5:.4f} +/- {se5:.4f}, R^2 = {r5**2:.4f}')
    print(f'    Spearman rho = {rho5:.4f}, p = {p5:.2e}')
    t5s = abs(sl5 - 1.0) / se5
    p5s = 2*(1-stats.t.cdf(t5s, df=n5-2))
    print(f'    p(slope=1.0) = {p5s:.4f}  (gc=c*a0 prediction)')

    # Does adding eta improve gc prediction from c_fit?
    mask5e = mask5 & np.isfinite(log_eta)
    if np.sum(mask5e) > 10:
        X5 = np.column_stack([log_c_fit[mask5e], log_eta[mask5e], np.ones(mask5e.sum())])
        y5 = log_gc[mask5e]
        b5, _, _, _ = np.linalg.lstsq(X5, y5, rcond=None)
        ss5r = np.sum((y5 - X5 @ b5)**2)
        ss5t = np.sum((y5 - y5.mean())**2)
        r2_5e = 1 - ss5r/ss5t
        mse5e = ss5r / (len(y5)-3)
        se5e = np.sqrt(np.diag(mse5e * np.linalg.inv(X5.T @ X5)))
        print(f'¥n  gc = c_fit^a * eta^b:')
        print(f'    c_fit^{b5[0]:.3f}+/-{se5e[0]:.3f}, eta^{b5[1]:.3f}+/-{se5e[1]:.3f}')
        print(f'    R^2 = {r2_5e:.4f} (vs c_fit alone {r5**2:.4f})')
else:
    print(f'  Too few c_fit values ({n5})')

# ============================================================
# SYNTHESIS: What determines eta?
# ============================================================
print('¥n' + '='*70)
print('SYNTHESIS: What determines eta?')
print('='*70)

print(f'¥n  {"Variable":<20s} | {"slope":>8s} +/- {"SE":>6s} | {"R^2":>6s} | {"rho":>6s} | {"p":>10s}')
print('  ' + '-'*75)
for label, sl, se, r2, rho, p in [
    ('G*Sigma_bar', sl_eS, se_eS, r_eS**2, rho_eS, p_eS),
    ('hR', sl_eH, se_eH, r_eH**2, rho_eH, p_eH),
    ('vflat', sl_eV, se_eV, r_eV**2, rho_eV, p_eV),
    ('Yd', sl_eY, se_eY, r_eY**2, rho_eY, p_eY),
    ('M_bar direct', sl_eM, se_eM, r_eM**2, rho_eM, p_eM),
    ('GS0_proxy', sl_eG, se_eG, r_eG**2, rho_eG, p_eG),
]:
    if np.isfinite(sl):
        print(f'  {label:<20s} | {sl:>8.4f} +/- {se:>6.4f} | {r2:>6.4f} | {rho:>6.4f} | {p:>10.2e}')

if np.isfinite(sl_eC):
    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}')

# ============================================================
# Figures
# ============================================================
print('¥n[3] Generating figures...')
fig, axes = plt.subplots(2, 3, figsize=(17, 11))

# 1: eta distribution
ax = axes[0, 0]
ev = eta_arr[np.isfinite(eta_arr) & (eta_arr > 0)]
ax.hist(np.log10(ev), bins=30, color='steelblue', alpha=0.7, edgecolor='white')
ax.axvline(np.log10(np.nanmedian(ev)), color='red', ls='--',
           label=f'median={np.nanmedian(ev):.4f}')
ax.set_xlabel('log10(eta)')
ax.set_ylabel('Count')
ax.set_title('1: eta distribution (FIXED)')
ax.legend(fontsize=8)

# 2: eta vs Sigma_bar
ax = axes[0, 1]
m = np.isfinite(log_eta) & np.isfinite(log_GSbar)
ax.scatter(log_GSbar[m], log_eta[m], s=10, alpha=0.5, c='darkorange', edgecolors='none')
xf = np.linspace(log_GSbar[m].min(), log_GSbar[m].max(), 100)
ax.plot(xf, sl_eS*xf + int_eS, 'r-', lw=2, label=f'slope={sl_eS:.3f}')
ax.plot(xf, -0.5*xf + int_eS, 'b--', lw=1.5, label='full cancel: -0.5')
```

```
ax.set_xlabel('log(G*Sigma_bar / a0)')
ax.set_ylabel('log(eta)')
ax.set_title(f'2: eta vs Sigma_bar (rho={rho_eS:.3f})')
ax.legend(fontsize=8)
ax.grid(True, alpha=0.3)

# 3: eta vs Yd
ax = axes[0, 2]
m = np.isfinite(log_eta) & np.isfinite(log_Yd)
ax.scatter(log_Yd[m], log_eta[m], s=10, alpha=0.5, c='green', edgecolors='none')
xf = np.linspace(log_Yd[m].min(), log_Yd[m].max(), 100)
ax.plot(xf, sl_eY*xf + int_eY, 'r-', lw=2, label=f'slope={sl_eY:.3f}')
ax.plot(xf, -0.44*xf + int_eY, 'b--', lw=1.5, label='5-5 prediction: -0.44')
ax.set_xlabel('log(Yd)')
ax.set_ylabel('log(eta)')
ax.set_title(f'3: eta vs Yd (rho={rho_eY:.3f})')
ax.legend(fontsize=8)
ax.grid(True, alpha=0.3)

# 4: eta vs GS0_proxy
ax = axes[1, 0]
m = np.isfinite(log_eta) & np.isfinite(log_GS0)
ax.scatter(log_GS0[m], log_eta[m], s=10, alpha=0.5, c='crimson', edgecolors='none')
xf = np.linspace(log_GS0[m].min(), log_GS0[m].max(), 100)
ax.plot(xf, sl_eG*xf + int_eG, 'r-', lw=2, label=f'slope={sl_eG:.3f}')
ax.axhline(np.nanmedian(log_eta[m]), color='grey', ls=':', label='const eta')
ax.set_xlabel('log(GS0_proxy / a0)')
ax.set_ylabel('log(eta)')
ax.set_title(f'4: eta vs proxy (rho={rho_eG:.3f})')
ax.legend(fontsize=8)
ax.grid(True, alpha=0.3)

# 5: c_fit vs gc
ax = axes[1, 1]
m5p = np.isfinite(log_c_fit) & np.isfinite(log_gc)
if np.sum(m5p) > 5:
    ax.scatter(log_c_fit[m5p], log_gc[m5p], s=10, alpha=0.5, c='purple', edgecolors='none')
    xf = np.linspace(log_c_fit[m5p].min(), log_c_fit[m5p].max(), 100)
    if np.isfinite(sl5):
        ax.plot(xf, sl5*xf + int5, 'r-', lw=2, label=f'slope={sl5:.3f}, R^2={r5**2:.3f}')
    ax.plot(xf, 1.0*xf, 'b--', lw=1, label='gc=c*a0')
    ax.set_xlabel('log(c_fit)')
    ax.set_ylabel('log(gc/a0)')
    ax.set_title('5: c_fit vs gc')
    ax.legend(fontsize=8)
    ax.grid(True, alpha=0.3)

# 6: R^2 summary for eta correlations
ax = axes[1, 2]
labels_bar = ['Sigma_bar', 'hR', 'vflat', 'Yd', 'M_bar', 'GS0_proxy']
r2s_bar = [r_eS**2, r_eH**2, r_eV**2, r_eY**2, r_eM**2, r_eG**2]
if np.isfinite(r_eC):
    labels_bar.append('c_fit')
    r2s_bar.append(r_eC**2)
cols_bar = ['darkorange','teal','crimson','green','steelblue','grey','purple'][:len(labels_bar)]
ax.bar(range(len(labels_bar)), r2s_bar, color=cols_bar, alpha=0.7,
       edgecolor='black', linewidth=0.5)
ax.set_xticks(range(len(labels_bar)))
ax.set_xticklabels(labels_bar, fontsize=8, rotation=15)
ax.set_ylabel('R^2 with log(eta)')
ax.set_title('6: What determines eta?')
for i, v in enumerate(r2s_bar):
    ax.text(i, v+0.005, f'{v:.3f}', ha='center', fontsize=8)
ax.grid(True, alpha=0.3, axis='y')

fig.suptitle(f'N-1 Layer 3 FIX: eta structure (N={N})', fontsize=14, y=1.01)
fig.tight_layout()
fig.savefig(os.path.join(BASE, 'fig_N1_layer3_fix.png'), dpi=150)
print('  -> fig_N1_layer3_fix.png')

# ============================================================
# Save CSV
# ============================================================
outcsv = os.path.join(BASE, 'N1_layer3_fix_results.csv')
cols_out = ['galaxy','gc_a0','log_gc','vflat','Yd','log_Yd','hR_kpc','log_hR',
            'log_GS0','log_Mbar','log_GSbar','eta','log_eta','log_eta_sq',
            'log_btfr_r','c_fit','log_c_fit','U_pp']
with open(outcsv, 'w', encoding='utf-8') as f:
    f.write(','.join(cols_out)+'¥n')
    for r in results:
        f.write(','.join(str(r.get(c,'')) for c in cols_out)+'¥n')
print(f'  -> {outcsv}')

# ============================================================
# Verdict
# ============================================================
print('¥n' + '='*70)
print('VERDICT (CORRECTED)')
```

```
print('='*70)

print(f'¥n  eta median = {np.nanmedian(eta_arr[m_eta]):.4f} (should be O(0.1-1))')

print(f'¥n  KEY: Does eta cancel Sigma_bar dependence?')
print(f'    log(eta) vs log(G*Sigma_bar/a0): slope = {sl_eS:.4f} +/- {se_eS:.4f}')
if abs(sl_eS - (-0.5)) < 2*se_eS:
    print(f'    >>> YES: slope consistent with -0.5 (full cancellation)')
    print(f'    >>> gc ~ eta*sqrt(a0*GS0) with eta~Sigma_bar^-0.5')
    print(f'    >>> means gc is INDEPENDENT of Sigma_bar (as observed)')
    print(f'    >>> N-1 Layer 3: RESOLVED (eta absorbs Sigma_bar)')
elif abs(sl_eS) > 0.2:
    print(f'    >>> PARTIAL: slope={sl_eS:.3f} (not -0.5 but significant)')
    print(f'    >>> eta partially cancels Sigma_bar')
else:
    print(f'    >>> NO: eta does not cancel Sigma_bar')

print(f'¥n  eta vs Yd: slope={sl_eY:.3f} (5-5 predicts -0.44)')
print(f'    p(slope=-0.44) = {p_yd:.4f}')

print(f'¥n  eta vs GS0_proxy: slope={sl_eG:.3f}')
print(f'    (deviation from 0 = deviation of alpha from 0.5)')
print(f'    effective alpha = 0.5 + {sl_eG:.4f} = {0.5+sl_eG:.4f}')

print('¥n[DONE]')
```