膜宇宙論モデル 隔離考察チャット

解析スクリプト全集（全23本・全文収録）

2026年4月 | 坂口製麺所

目次: 全23スクリプト

#	ファイル名	フェーズ	使用rs	ステータス
1	f_tanh_robustness.py	Step 0	rs2（V-1で確認）	Level B（rs2依存）
2	step1_f_functional_form.py	Step 1	rs2（V-1で確認）	Level B（rs2依存）
3	step2_rs_hR_relation.py	Step 2	rs2（V-1で確認）	撤回（rs2ベース、後に無効化）
4	step3_alpha_derivation.py	Step 3（旧版）	rs2 + 循環論法（f(v_flat) U字型を V_peak 設定に使用）	無効（C-1 循環論法）
5	step3_revised.py	Step 3（修正版）	非循環	参考（rs2ベースの alpha=0.883 自体が後に無効化）
6	delta_gamma_decompose.py	Step 3 延長	非循環	確定（合成モデルでは alpha 導出不能）
7	TA2_rs_ratio.py	T-A2	なし（理論計算）	Level B（定性的に確立）
8	TA2_direct.py	T-A2（直接計算）	rs2（V-1で確認）	Level B（rs2依存）
9	TA3_gc_measurement.py	T-A3	rs不使用（安全）	Level A（確立）
10	cond14_complexity.py	条件14検証	rs2（gamma残差の計算）	Level B（条件14不支持、ガスの間接効果あり）
11	D3_D1_fix.py	D-1/D-3	rs1/rs2 混在	確定（因果連鎖の構造を解明）
12	rs1_validity.py	rs1/rs2 問題	rs1	Level A（rs1/rs2問題の確立）
13	V1_rs_check.py	V-1/V-2	なし（診断スクリプト）	Level A（汚染範囲の確定）
14	V3_rs1_interpretation.py	V-3	rs1	Level A（rs1局所解問題の確立）
15	rs_best_all.py	rs_best	rs_best（新規生成）	Level B（R_max人工効果に注意）
16	Rmax_hR_separation.py	R_max分離	rs_best	Level A（分離不能の確定）
17	U2_gc_Ud_degeneracy.py	U-2	rs不使用（安全）	Level B（縮退は部分的、残差は系統誤差排除が未了）
18	U3_spatial_residuals.py	U-3	rs_best（参照点として使用、フィッティングには不使用）	Level B（条件b不充足: rs_best は特別でない）
19	levelA_promotion.py	Level A テスト	rs_best	Level B 維持（遷移スケールは h_R に追従）
20	rs_fixed_hR.py	MOND vs tanh	h_R固定（rsフィッティングなし）	Level A（MOND式(5)が中核、式(7)は冗長）
21	gc_predictive_model.py	g_c予測	rs不使用（安全）	Level A（g_c 予測モデル確立）
22	zero_param_test.py	ゼロparamテスト	rs不使用（安全）	Level A（インサンプル）
23	loo_cv.py	LOO-CV	rs不使用（安全）	Level A（最終確定）

1. f_tanh_robustness.py

項目	内容
フェーズ	Step 0
目的	f^(tanh) = v_bar^2(r_s)/v_flat^2 の堅牢性確認
使用rs	rs2（V-1で確認）
結果	f ~= 0.29。外挿非依存で堅牢。ただし rs2 ベース。
ステータス	Level B（rs2依存）

解析目的

f^(tanh)=0.262（前チャットの値）が外挿銀河の除外に依存しないか検証する。r_s_tanh の位置が観測データの最大半径 R_max を超える銀河（外挿）を除外し、f の中央値が変化するか確認。

ソースコード全文

```
#!/usr/bin/env python3
"""
f^(tanh) = v_bar^2(r_s_tanh) / v_flat^2 の堅牢性確認
外挿銀河（r_s_tanh &gt; R_max）を除外して再計算

実行: uv run --with scipy --with matplotlib --with numpy python f_tanh_robustness.py
作業ディレクトリ: D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン¥
"""

import csv
import os
import sys
import numpy as np
from scipy.interpolate import interp1d
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 設定
# ============================================================
RESULTS_CSV = "sparc_results.csv"
# .datファイルが同ディレクトリにある前提

# ============================================================
# 1. sparc_results.csv 読み込み
# ============================================================
galaxies = []
with open(RESULTS_CSV, "r", encoding="utf-8") as f:
    reader = csv.DictReader(f)
    print(f"[INFO] CSV columns: {reader.fieldnames}")
    for row in reader:
        galaxies.append(row)

print(f"[INFO] Total galaxies in CSV: {len(galaxies)}")

# --- カラム名を自動検出 ---
# r_s_tanh, v_flat, upsilon_d に該当するカラムを探す
sample_keys = list(galaxies[0].keys())
print(f"[INFO] Available columns: {sample_keys}")

# ============================================================
# 2. 各銀河の.datファイルを読み、R_max と v_bar(r) を構築
# ============================================================

def read_sparc_dat(galaxy_name):
    """
    SPARCの.datファイルを読む
    列: Rad Vobs errV Vgas Vdisk Vbul (標準形式)
    返り値: rad, v_gas, v_disk, v_bul (numpy arrays)
    """
    # ファイル名候補
    candidates = [
        f"{galaxy_name}_rotmod.dat",
        f"{galaxy_name}.dat",
    ]
    for fname in candidates:
        if os.path.exists(fname):
            data = np.loadtxt(fname, comments='#')
            rad = data[:, 0]      # kpc
            # Vobs = data[:, 1]
            # errV = data[:, 2]
            v_gas = data[:, 3]    # km/s
            v_disk = data[:, 4]   # km/s (Upsilon_d=1.0 での値)
            v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
            return rad, v_gas, v_disk, v_bul
    return None, None, None, None

def compute_vbar(rad, v_gas, v_disk, v_bul, upsilon_d):
    """
```

```
    v_bar^2 = upsilon_d * v_disk^2 + v_gas^2 + v_bul^2
    v_disk は Upsilon_d=1.0 での値なので、upsilon_d を掛ける
    注意: v_disk, v_gas は符号付き（負 = 反対方向の寄与）
    """
    # 符号を保持して二乗
    vd2 = np.sign(v_disk) * v_disk**2
    vg2 = np.sign(v_gas) * v_gas**2
    vb2 = np.sign(v_bul) * v_bul**2
    vbar2 = upsilon_d * vd2 + vg2 + vb2
    return vbar2  # これは v_bar^2（符号付き）


# ============================================================
# 3. メインループ
# ============================================================

# カラム名のマッピング（CSVの実際のカラム名に合わせて調整が必要な場合あり）
# まず候補を試す
def get_val(row, candidates, default=None):
    for c in candidates:
        if c in row and row[c].strip() != '':
            try:
                return float(row[c])
            except ValueError:
                pass
    return default

def get_str(row, candidates, default=''):
    for c in candidates:
        if c in row and row[c].strip() != '':
            return row[c].strip()
    return default


results_all = []    # 全銀河
results_inrange = [] # r_s_tanh &lt;= R_max（非外挿）
results_extrap = []  # r_s_tanh &gt; R_max（外挿）

n_no_dat = 0
n_interp_fail = 0

for gal in galaxies:
    name = get_str(gal, ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY'])
    if not name:
        # 最初のカラムを名前として使う
        name = list(gal.values())[0]

    rs = get_val(gal, ['r_s', 'rs', 'r_s_tanh', 'rs_tanh', 'r_s_kpc'])
    vf = get_val(gal, ['v_flat', 'vflat', 'Vflat', 'v_flat_kms'])
    ud = get_val(gal, ['upsilon_d', 'Upsilon_d', 'Ud', 'upsilon'])

    if rs is None or vf is None or ud is None:
        continue
    if vf &lt;= 0 or rs &lt;= 0:
        continue

    # .datファイル読み込み
    rad, v_gas, v_disk, v_bul = read_sparc_dat(name)
    if rad is None:
        n_no_dat += 1
        continue

    R_max = rad.max()
    is_extrap = (rs &gt; R_max)

    # v_bar^2(r_s_tanh) を補間で求める
    vbar2_arr = compute_vbar(rad, v_gas, v_disk, v_bul, ud)

    # 補間（外挿の場合も一応計算するが、フラグで区別）
    try:
        if is_extrap:
            # 外挿：fill_value で最後の値を使う（参考値）
            interp_func = interp1d(rad, vbar2_arr, kind='linear',
                                   fill_value=vbar2_arr[-1], bounds_error=False)
        else:
            interp_func = interp1d(rad, vbar2_arr, kind='linear',
                                   bounds_error=False, fill_value='extrapolate')

        vbar2_at_rs = interp_func(rs)
        if np.isnan(vbar2_at_rs):
            n_interp_fail += 1
            continue

        f_tanh = float(vbar2_at_rs) / (vf**2)

        entry = {
            'name': name,
            'rs': rs,
            'vflat': vf,
            'upsilon_d': ud,
            'R_max': R_max,
            'rs_over_Rmax': rs / R_max,
            'vbar2_at_rs': float(vbar2_at_rs),
            'f_tanh': f_tanh,
```

```
            'is_extrap': is_extrap,
        }
        results_all.append(entry)
        if is_extrap:
            results_extrap.append(entry)
        else:
            results_inrange.append(entry)

    except Exception as e:
        n_interp_fail += 1
        continue

# ============================================================
# 4. 結果表示
# ============================================================
print("¥n" + "="*70)
print("f^(tanh) = v_bar^2(r_s_tanh) / v_flat^2  堅牢性チェック")
print("="*70)

print(f"¥n総銀河数 (CSV) : {len(galaxies)}")
print(f".datファイルなし: {n_no_dat}")
print(f"補間失敗: {n_interp_fail}")
print(f"解析成功: {len(results_all)}")
print(f"  非外挿 (r_s &lt;= R_max) : {len(results_inrange)}")
print(f"  外挿   (r_s &gt;  R_max) : {len(results_extrap)}")

def print_stats(label, data_list):
    if not data_list:
        print(f"¥n--- {label}: データなし ---")
        return
    fs = [d['f_tanh'] for d in data_list]
    fs = np.array(fs)
    print(f"¥n--- {label} (N={len(fs)}) ---")
    print(f"  中央値:   {np.median(fs):.4f}")
    print(f"  平均値:   {np.mean(fs):.4f}")
    print(f"  標準偏差: {np.std(fs):.4f}")
    print(f"  25%ile:   {np.percentile(fs, 25):.4f}")
    print(f"  75%ile:   {np.percentile(fs, 75):.4f}")
    print(f"  最小値:   {np.min(fs):.4f}")
    print(f"  最大値:   {np.max(fs):.4f}")

    # v_bar(r_s) / v_flat の中央値
    vbar_ratio = np.sqrt(np.abs(fs)) * np.sign(fs)
    print(f"  → v_bar(r_s)/v_flat 中央値: {np.median(vbar_ratio):.4f}")

print_stats("全銀河", results_all)
print_stats("非外挿のみ (r_s ≦ R_max) ", results_inrange)
print_stats("外挿のみ (r_s &gt; R_max) ", results_extrap)

# ============================================================
# 5. 外挿度の分布
# ============================================================
print("¥n--- r_s / R_max の分布 ---")
ratios = [d['rs_over_Rmax'] for d in results_all]
ratios = np.array(ratios)
print(f"  中央値: {np.median(ratios):.3f}")
print(f"  r_s/R_max &gt; 1.0: {np.sum(ratios &gt; 1.0)}/{len(ratios)} ({100*np.sum(ratios &gt; 1.0)/len(ratios):.1f}%)")
print(f"  r_s/R_max &gt; 1.5: {np.sum(ratios &gt; 1.5)}/{len(ratios)} ({100*np.sum(ratios &gt; 1.5)/len(ratios):.1f}%)")
print(f"  r_s/R_max &gt; 2.0: {np.sum(ratios &gt; 2.0)}/{len(ratios)} ({100*np.sum(ratios &gt; 2.0)/len(ratios):.1f}%)")

# ============================================================
# 6. 追加解析：非外挿をさらにR_max &gt;= 1.2*r_s で絞る (安全マージン)
# ============================================================
results_safe = [d for d in results_all if d['rs_over_Rmax'] &lt;= 0.8]
print_stats("安全領域 (r_s ≦ 0.8×R_max) ", results_safe)

results_moderate = [d for d in results_all if 0.8 &lt; d['rs_over_Rmax'] &lt;= 1.0]
print_stats("境界領域 (0.8×R_max &lt; r_s ≦ R_max) ", results_moderate)

# ============================================================
# 7. v_flat 依存性チェック (非外挿のみ)
# ============================================================
if len(results_inrange) &gt;= 10:
    vf_arr = np.array([d['vflat'] for d in results_inrange])
    f_arr = np.array([d['f_tanh'] for d in results_inrange])

    # v_flat の四分位で分割
    q25, q50, q75 = np.percentile(vf_arr, [25, 50, 75])
    print(f"¥n--- v_flat 依存性チェック (非外挿のみ) ---")
    print(f"v_flat 四分位: Q1={q25:.1f}, Q2={q50:.1f}, Q3={q75:.1f} km/s")

    for label, mask in [
        (f"v_flat &lt; {q25:.0f}", vf_arr &lt; q25),
        (f"{q25:.0f} &lt;= v_flat &lt; {q50:.0f}", (vf_arr &gt;= q25) &amp; (vf_arr &lt; q50)),
        (f"{q50:.0f} &lt;= v_flat &lt; {q75:.0f}", (vf_arr &gt;= q50) &amp; (vf_arr &lt; q75)),
        (f"v_flat &gt;= {q75:.0f}", vf_arr &gt;= q75),
    ]:
        if np.sum(mask) &gt; 0:
            print(f"  {label}: N={np.sum(mask)}, f中央値={np.median(f_arr[mask]):.4f}")

# ============================================================
# 8. 図の作成
# ============================================================
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# (a) f^(tanh) のヒストグラム：全体 vs 非外挿
```

```
ax = axes[0, 0]
f_all = [d['f_tanh'] for d in results_all]
f_in = [d['f_tanh'] for d in results_inrange]
f_ex = [d['f_tanh'] for d in results_extrap]
bins = np.linspace(-0.5, 2.0, 50)
ax.hist(f_all, bins=bins, alpha=0.3, label=f'All (N={len(f_all)})', color='gray')
ax.hist(f_in, bins=bins, alpha=0.6, label=f'In-range (N={len(f_in)})', color='blue')
ax.hist(f_ex, bins=bins, alpha=0.4, label=f'Extrap (N={len(f_ex)})', color='red')
ax.axvline(0.262, color='k', ls='--', label='Median=0.262 (reported)')
if f_in:
    ax.axvline(np.median(f_in), color='blue', ls='--', label=f'In-range median={np.median(f_in):.3f}')
ax.set_xlabel('f^(tanh) = v_bar²(r_s) / v_flat²')
ax.set_ylabel('Count')
ax.set_title('(a) f^(tanh) distribution')
ax.legend(fontsize=8)

# (b) r_s / R_max の分布
ax = axes[0, 1]
ax.hist(ratios, bins=50, color='steelblue', edgecolor='white')
ax.axvline(1.0, color='red', ls='--', label='r_s = R_max')
ax.set_xlabel('r_s_tanh / R_max')
ax.set_ylabel('Count')
ax.set_title('(b) Extrapolation fraction')
ax.legend()

# (c) f^(tanh) vs v_flat (色分け)
ax = axes[1, 0]
for d in results_inrange:
    ax.scatter(d['vflat'], d['f_tanh'], c='blue', s=10, alpha=0.5)
for d in results_extrap:
    ax.scatter(d['vflat'], d['f_tanh'], c='red', s=10, alpha=0.5, marker='x')
ax.axhline(0.262, color='k', ls='--', alpha=0.5)
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('f^(tanh)')
ax.set_title('(c) f^(tanh) vs v_flat')
ax.legend(['In-range', 'Extrap', 'Median=0.262'], fontsize=8)

# (d) r_s/R_max vs f^(tanh)
ax = axes[1, 1]
rs_ratio_arr = [d['rs_over_Rmax'] for d in results_all]
f_arr_all = [d['f_tanh'] for d in results_all]
colors = ['red' if d['is_extrap'] else 'blue' for d in results_all]
ax.scatter(rs_ratio_arr, f_arr_all, c=colors, s=10, alpha=0.5)
ax.axvline(1.0, color='gray', ls='--')
ax.axhline(0.262, color='k', ls='--', alpha=0.5)
ax.set_xlabel('r_s_tanh / R_max')
ax.set_ylabel('f^(tanh)')
ax.set_title('(d) f^(tanh) vs extrapolation degree')

plt.tight_layout()
plt.savefig('f_tanh_robustness.png', dpi=150)
print(f"¥n[SAVED] f_tanh_robustness.png")

# ============================================================
# 9. CSV出力 (詳細)
# ============================================================
out_csv = "f_tanh_robustness_detail.csv"
with open(out_csv, "w", newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerow(['galaxy', 'r_s_tanh', 'v_flat', 'upsilon_d',
                     'R_max', 'rs_over_Rmax', 'vbar2_at_rs',
                     'f_tanh', 'is_extrap'])
    for d in sorted(results_all, key=lambda x: x['rs_over_Rmax']):
        writer.writerow([
            d['name'], f"{d['rs']:.3f}", f"{d['vflat']:.1f}",
            f"{d['upsilon_d']:.3f}", f"{d['R_max']:.2f}",
            f"{d['rs_over_Rmax']:.4f}", f"{d['vbar2_at_rs']:.2f}",
            f"{d['f_tanh']:.4f}", d['is_extrap']
        ])
print(f"[SAVED] {out_csv}")

print("¥n" + "="*70)
print("完了。次のステップ:")
print("  1. 非外挿の中央値が0.262からどれだけ変化するか確認")
print("  2. v_flat依存性がある場合、スケーリングの系統誤差を評価")
print("  3. 安全領域 (r_s ☒ 0.8×R_max) の結果が最も信頼できる")
print("="*70)
```

2. step1_f_functional_form.py

項目	内容
フェーズ	Step 1
目的	f(v_flat) の関数形特定
使用rs	rs2 (V-1で確認)
結果	U字型 (極小 v_flat ~= 74 km/s) が最良。Ud との強い相関 (rho=0.483)。
ステータス	Level B (rs2依存)

解析目的

f^(tanh) が v_flat の関数としてどのような形状を持つかを特定する。定数、べき乗、log二次 (U字型)、折れ線の4モデルを BIC で比較。Ud との相関も解析。

ソースコード全文

```
#!/usr/bin/env python3
"""
Step 1: f^(tanh)(v_flat) の関数形特定

前提: f_tanh_robustness.py を実行済み → f_tanh_robustness_detail.csv が存在
追加で sparc_results.csv から Y_d, grade, T_type 等を取得

実行: uv run --with scipy --with matplotlib --with numpy python step1_f_functional_form.py
"""

import csv
import os
import numpy as np
from scipy.optimize import curve_fit
from scipy.stats import spearmanr, pearsonr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 1. データ読み込み
# ============================================================

def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        cols = reader.fieldnames
        rows = list(reader)
    return cols, rows

# --- f_tanh_robustness_detail.csv ---
if os.path.exists("f_tanh_robustness_detail.csv"):
    _, detail_rows = load_csv("f_tanh_robustness_detail.csv")
    print(f"[INFO] f_tanh_robustness_detail.csv loaded: {len(detail_rows)} rows")
else:
    print("[ERROR] f_tanh_robustness_detail.csv not found. Run f_tanh_robustness.py first.")
    exit(1)

# --- sparc_results.csv (追加カラム取得用) ---
extra = {}
if os.path.exists("sparc_results.csv"):
    src_cols, src_rows = load_csv("sparc_results.csv")
    print(f"[INFO] sparc_results.csv columns: {src_cols}")
    for row in src_rows:
        # 銀河名をキーに
        name = None
        for c in ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY']:
            if c in row and row[c].strip():
                name = row[c].strip()
                break
        if name is None:
            name = list(row.values())[0].strip()
        extra[name] = row

# ============================================================
# 2. 解析用配列の構築
# ============================================================
names = []
v_flat = []
f_tanh = []
r_s = []
upsilon_d = []
is_extrap = []

# sparc_results.csvからの追加量 (あれば)
T_type_arr = []
grade_arr = []

for d in detail_rows:
```

```
    gal = d['galaxy'].strip()
    vf = float(d['v_flat'])
    ft = float(d['f_tanh'])
    rs = float(d['r_s_tanh'])
    ud = float(d['upsilon_d'])
    ext = d['is_extrap'].strip()

    # f &lt; 0 の銀河 (v_bar^2 が負 = ガス逆回転等) は除外検討
    names.append(gal)
    v_flat.append(vf)
    f_tanh.append(ft)
    r_s.append(rs)
    upsilon_d.append(ud)
    is_extrap.append(ext == 'True')

    # 追加量
    if gal in extra:
        row = extra[gal]
        tt = None
        for c in ['T_type', 'T', 'Ttype', 'hubble_type', 'type']:
            if c in row and row[c].strip():
                try:
                    tt = float(row[c])
                except ValueError:
                    pass
        T_type_arr.append(tt)

        gr = None
        for c in ['grade', 'Grade', 'quality']:
            if c in row and row[c].strip():
                gr = row[c].strip()
        grade_arr.append(gr)
    else:
        T_type_arr.append(None)
        grade_arr.append(None)

v_flat = np.array(v_flat)
f_tanh = np.array(f_tanh)
r_s = np.array(r_s)
upsilon_d = np.array(upsilon_d)
is_extrap = np.array(is_extrap)
N_total = len(v_flat)

# 非外挿マスク
mask_in = ~is_extrap
# f &gt; 0 マスク (対数フィット用)
mask_pos = f_tanh &gt; 0
# 主解析マスク：非外挿 &amp; f &gt; 0
mask_main = mask_in &amp; mask_pos

print(f"¥n[INFO] N_total={N_total}, 非外挿={mask_in.sum()}, f&gt;0={mask_pos.sum()}, 主解析={mask_main.sum()}")
print(f"[INFO] f&lt;=0 の銀河: {np.sum(f_tanh &lt;= 0)} 個")
if np.sum(f_tanh &lt;= 0) &gt; 0:
    for i in range(N_total):
        if f_tanh[i] &lt;= 0:
            print(f"       {names[i]}: f={f_tanh[i]:.4f}, v_flat={v_flat[i]:.1f}")

# ============================================================
# 3. 関数形のフィッティング (非外挿 &amp; f &gt; 0)
# ============================================================
lv = np.log10(v_flat[mask_main])
lf = np.log10(f_tanh[mask_main])
vf_m = v_flat[mask_main]
ft_m = f_tanh[mask_main]
N_fit = mask_main.sum()

print(f"¥n{'='*70}")
print(f"関数形フィッティング (N={N_fit}、非外挿・f&gt;0) ")
print(f"{'='*70}")

results = {}

# --- Model 1: 単純べき乗  log(f) = a + b*log(v) ---
p1 = np.polyfit(lv, lf, 1)
pred1 = np.polyval(p1, lv)
rss1 = np.sum((lf - pred1)**2)
bic1 = N_fit * np.log(rss1/N_fit) + 2*np.log(N_fit)
r1 = np.corrcoef(lf, pred1)[0,1]
results['power_law'] = {'params': p1, 'rss': rss1, 'bic': bic1, 'r': r1, 'k': 2}
print(f"¥nModel 1: べき乗  log(f) = {p1[1]:.3f} + {p1[0]:.3f}*log(v_flat)")
print(f"  → f ∝ v_flat^{p1[0]:.3f}")
print(f"  r={r1:.4f}, RSS={rss1:.3f}, BIC={bic1:.1f}")

# --- Model 2: log二次  log(f) = a + b*log(v) + c*log(v)^2 ---
p2 = np.polyfit(lv, lf, 2)
pred2 = np.polyval(p2, lv)
rss2 = np.sum((lf - pred2)**2)
bic2 = N_fit * np.log(rss2/N_fit) + 3*np.log(N_fit)
r2 = np.corrcoef(lf, pred2)[0,1]
results['log_quadratic'] = {'params': p2, 'rss': rss2, 'bic': bic2, 'r': r2, 'k': 3}
print(f"¥nModel 2: log二次  log(f) = {p2[2]:.3f} + {p2[1]:.3f}*log(v) + {p2[0]:.3f}*log(v)²")
lv_min = -p2[1]/(2*p2[0])
print(f"  極小点: log(v_flat) = {lv_min:.3f} → v_flat = {10**lv_min:.1f} km/s")
print(f"  r={r2:.4f}, RSS={rss2:.3f}, BIC={bic2:.1f}")
```

```
# --- Model 3: 定数（帰無仮説） ---
pred0 = np.mean(lf) * np.ones_like(lf)
rss0 = np.sum((lf - pred0)**2)
bic0 = N_fit * np.log(rss0/N_fit) + 1*np.log(N_fit)
results['constant'] = {'rss': rss0, 'bic': bic0, 'k': 1}
print(f"¥nModel 0: 定数  f = 10^{np.mean(lf):.3f} = {10**np.mean(lf):.3f}")
print(f"  RSS={rss0:.3f}, BIC={bic0:.1f}")

# --- Model 4: 折れ線（区分線形） log(f) = a + b1*(lv-lv_b) if lv&lt;lv_b else a + b2*(lv-lv_b) ---
def broken_line(lv, a, b1, b2, lv_break):
    return np.where(lv &lt; lv_break,
                    a + b1*(lv - lv_break),
                    a + b2*(lv - lv_break))

try:
    p4, _ = curve_fit(broken_line, lv, lf,
                      p0=[np.median(lf), -0.5, 0.5, 1.9],
                      maxfev=10000)
    pred4 = broken_line(lv, *p4)
    rss4 = np.sum((lf - pred4)**2)
    bic4 = N_fit * np.log(rss4/N_fit) + 4*np.log(N_fit)
    r4 = np.corrcoef(lf, pred4)[0,1]
    results['broken_line'] = {'params': p4, 'rss': rss4, 'bic': bic4, 'r': r4, 'k': 4}
    print(f"¥nModel 4: 折れ線  break at log(v)={p4[3]:.3f} → v_flat={10**p4[3]:.1f} km/s")
    print(f"  slope_low={p4[1]:.3f}, slope_high={p4[2]:.3f}")
    print(f"  r={r4:.4f}, RSS={rss4:.3f}, BIC={bic4:.1f}")
except Exception as e:
    print(f"¥nModel 4: 折れ線フィット失敗 ({e})")

# --- BIC比較 ---
print(f"¥n--- BIC比較（小さいほど良い） ---")
for name_m, res in sorted(results.items(), key=lambda x: x[1]['bic']):
    print(f"  {name_m:20s}: BIC={res['bic']:8.1f}  (k={res['k']})")

# ============================================================
# 4. Y_d との相関（U字型の原因探索）
# ============================================================
print(f"¥n{'='*70}")
print("Y_d および他パラメータとの相関")
print(f"{'='*70}")

ud_m = upsilon_d[mask_main]
rs_m = r_s[mask_main]

# f vs Y_d
rho_ud, p_ud = spearmanr(ft_m, ud_m)
print(f"¥nf vs Y_d:  Spearman ρ={rho_ud:.3f}, p={p_ud:.2e}")

# f vs r_s
rho_rs, p_rs = spearmanr(ft_m, rs_m)
print(f"f vs r_s:  Spearman ρ={rho_rs:.3f}, p={p_rs:.2e}")

# Y_d vs v_flat
rho_uv, p_uv = spearmanr(ud_m, vf_m)
print(f"Y_d vs v_flat: Spearman ρ={rho_uv:.3f}, p={p_uv:.2e}")

# --- 多変量：log(f) = a + b*log(v) + c*log(Y_d) ---
lud = np.log10(ud_m)
X = np.column_stack([np.ones(N_fit), lv, lud])
beta, res_mv, _, _ = np.linalg.lstsq(X, lf, rcond=None)
pred_mv = X @ beta
rss_mv = np.sum((lf - pred_mv)**2)
bic_mv = N_fit * np.log(rss_mv/N_fit) + 3*np.log(N_fit)
r_mv = np.corrcoef(lf, pred_mv)[0,1]
print(f"¥n多変量: log(f) = {beta[0]:.3f} + {beta[1]:.3f}*log(v) + {beta[2]:.3f}*log(Y_d)")
print(f"  r={r_mv:.4f}, RSS={rss_mv:.3f}, BIC={bic_mv:.1f}")
print(f"  → Y_d の寄与: 係数={beta[2]:.3f}")

# --- 多変量 + 二次項：log(f) = a + b*log(v) + c*log(v)^2 + d*log(Y_d) ---
X2 = np.column_stack([np.ones(N_fit), lv, lv**2, lud])
beta2, _, _, _ = np.linalg.lstsq(X2, lf, rcond=None)
pred_mv2 = X2 @ beta2
rss_mv2 = np.sum((lf - pred_mv2)**2)
bic_mv2 = N_fit * np.log(rss_mv2/N_fit) + 4*np.log(N_fit)
r_mv2 = np.corrcoef(lf, pred_mv2)[0,1]
print(f"¥n多変量+二次: log(f) = {beta2[0]:.3f} + {beta2[1]:.3f}*log(v) + {beta2[2]:.3f}*log(v)² + {beta2[3]:.3f}*log(Y_d)")
print(f"  r={r_mv2:.4f}, RSS={rss_mv2:.3f}, BIC={bic_mv2:.1f}")

# --- T_type との相関（あれば） ---
tt_valid = [(ft_m[i], T_type_arr[np.where(mask_main)[0][i]])
            for i in range(N_fit)
            if np.where(mask_main)[0][i] &lt; len(T_type_arr)
            and T_type_arr[np.where(mask_main)[0][i]] is not None]
if len(tt_valid) &gt;= 10:
    ft_tt = np.array([x[0] for x in tt_valid])
    tt_tt = np.array([x[1] for x in tt_valid])
    rho_tt, p_tt = spearmanr(ft_tt, tt_tt)
    print(f"¥nf vs T_type: Spearman ρ={rho_tt:.3f}, p={p_tt:.2e}, N={len(tt_valid)}")

# ============================================================
# 5. ビン別統計（より細かく）
# ============================================================
print(f"¥n{'='*70}")
print("v_flat ビン別 f^(tanh) 統計（非外挿・f&gt;0）")
print(f"{'='*70}")
```

```
# 等N分割 (8分割)
sort_idx = np.argsort(vf_m)
n_per_bin = N_fit // 8
print(f"¥n等N分割 (各ビン N≈{n_per_bin}) :")
print(f"{'v_flat範囲':&gt;20s}  {'N':&gt;4s}  {'f中央値':&gt;8s}  {'f平均':&gt;8s}  {'f_std':&gt;8s}  {'v_bar/v_flat':&gt;12s}")

bin_vf_centers = []
bin_f_medians = []

for i in range(8):
    i0 = i * n_per_bin
    i1 = (i+1) * n_per_bin if i &lt; 7 else N_fit
    idx = sort_idx[i0:i1]
    vf_bin = vf_m[idx]
    ft_bin = ft_m[idx]
    vr_bin = np.sqrt(np.abs(ft_bin)) * np.sign(ft_bin)
    label = f"{vf_bin.min():.0f}-{vf_bin.max():.0f}"
    print(f"{label:&gt;20s}  {len(idx):4d}  {np.median(ft_bin):8.3f}  {np.mean(ft_bin):8.3f}  {np.std(ft_bin):8.3f}  {np.median(vr_bin):12.3f}")
    bin_vf_centers.append(np.median(vf_bin))
    bin_f_medians.append(np.median(ft_bin))

# ============================================================
# 6. 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) f vs v_flat + フィット曲線
ax = axes[0, 0]
ax.scatter(vf_m, ft_m, s=8, alpha=0.4, c='steelblue')
ax.scatter(bin_vf_centers, bin_f_medians, s=60, c='red', zorder=5,
           edgecolors='black', label='Bin medians')
vv = np.linspace(vf_m.min(), vf_m.max(), 200)
lvv = np.log10(vv)
# べき乗
ax.plot(vv, 10**np.polyval(p1, lvv), 'g--', label=f'Power law (α={p1[0]:.2f})')
# log二次
ax.plot(vv, 10**np.polyval(p2, lvv), 'r-', label=f'Log-quad (min@{10**lv_min:.0f})')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('f^(tanh)')
ax.set_title('(a) f vs v_flat')
ax.legend(fontsize=7)
ax.set_ylim(-0.2, 2.0)

# (b) log-log
ax = axes[0, 1]
ax.scatter(lv, lf, s=8, alpha=0.4, c='steelblue')
ax.plot(lvv, np.polyval(p1, lvv), 'g--', label='Power law')
ax.plot(lvv, np.polyval(p2, lvv), 'r-', label='Log-quadratic')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(f)')
ax.set_title('(b) log-log')
ax.legend(fontsize=8)

# (c) 残差 vs v_flat (log二次)
ax = axes[0, 2]
resid2 = lf - np.polyval(p2, lv)
ax.scatter(lv, resid2, s=8, alpha=0.4, c='steelblue')
ax.axhline(0, color='gray', ls='--')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('Residual (log-quad)')
ax.set_title(f'(c) Log-quad residuals (σ={np.std(resid2):.3f})')

# (d) f vs Y_d
ax = axes[1, 0]
ax.scatter(ud_m, ft_m, s=8, alpha=0.4, c='orange')
ax.set_xlabel('Y_d')
ax.set_ylabel('f^(tanh)')
ax.set_title(f'(d) f vs Y_d (ρ={rho_ud:.3f})')

# (e) Y_d vs v_flat
ax = axes[1, 1]
ax.scatter(vf_m, ud_m, s=8, alpha=0.4, c='green')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('Y_d')
ax.set_title(f'(e) Y_d vs v_flat (ρ={rho_uv:.3f})')

# (f) 多変量フィットの実測 vs 予測
ax = axes[1, 2]
ax.scatter(pred_mv2, lf, s=8, alpha=0.4, c='purple')
xx = [min(lf.min(), pred_mv2.min()), max(lf.max(), pred_mv2.max())]
ax.plot(xx, xx, 'k--')
ax.set_xlabel('Predicted log(f)')
ax.set_ylabel('Observed log(f)')
ax.set_title(f'(f) Multivariate+quad (r={r_mv2:.3f})')

plt.tight_layout()
plt.savefig('step1_f_functional_form.png', dpi=150)
print(f"¥n[SAVED] step1_f_functional_form.png")

# ============================================================
# 7. 結論の自動出力
# ============================================================
print(f"¥n{'='*70}")
print("Step 1 結論")
print(f"{'='*70}")
```

```
# BIC最良モデル
best = min(results.items(), key=lambda x: x[1]['bic'])
print(f"¥n最良モデル (BIC) : {best[0]} (BIC={best[1]['bic']:.1f})")

# ΔBIC
for name_m, res in results.items():
    delta = res['bic'] - best[1]['bic']
    if delta &gt; 0:
        print(f"  vs {name_m}: ΔBIC = +{delta:.1f}")

print(f"¥n--- T-A1 導出への制約条件 ---")
print(f"f(v_flat) は定数ではない (定数モデルの BIC={results['constant']['bic']:.1f}) ")
if 'log_quadratic' in results:
    print(f"log二次が最良の場合: 極小点 v_flat ≈ {10**lv_min:.0f} km/s")
    print(f"  低質量側: f が高い (バリオン優勢の遷移) ")
    print(f"  中間質量: f が最低 (膜展開が支配的な遷移) ")
    print(f"  大質量側: f が再上昇 (バリオン深いポテンシャル井戸) ")
print(f"¥nY_d の寄与: 多変量での係数 = {beta[2]:.3f}")
if abs(beta[2]) &gt; 0.3:
    print(f"  → Y_d は f の説明に有意に寄与。fはバリオン構造に依存。")
else:
    print(f"  → Y_d の寄与は限定的。")

print(f"¥n→ Step 2 へ: r_s_tanh/h_R vs v_flat の解析で")
print(f"   バリオン分布 (h_R) の寄与を分離する。")
print(f"   h_R のデータソースを確認すること。")
```

3. step2_rs_hR_relation.py

項目	内容
フェーズ	Step 2
目的	r_s/h_R vs v_flat (alpha = beta + gamma の分解)
使用rs	rs2 (V-1で確認)
結果	alpha=0.883 = beta(0.600) + gamma(0.284)。ただし rs2 ベース。
ステータス	撤回 (rs2ベース、後に無効化)

解析目的

r_s のスケーリング alpha を、バリオンのサイズ-質量関係 beta (h_R ~ v^beta) と膜固有の寄与 gamma (r_s/h_R ~ v^gamma) に分解する。h_R は V_disk ピーク半径 / 2.15 から推定。

ソースコード全文

```
#!/usr/bin/env python3
"""
Step 2: r_s_tanh / h_R vs v_flat の解析
バリオン分布が r_s_tanh をどこまで決めているかを分離する

h_R の取得方法 (優先順):
  1. SPARC photometric table (SPARC_Lelli2016c.mrt 等) があれば R_d を使用
  2. .dat ファイルの V_disk(r) ピーク半径 r_peak から h_R ≈ r_peak / 2.15 を推定

実行: uv run --with scipy --with matplotlib --with numpy python step2_rs_hR_relation.py
"""

import csv
import os
import glob
import numpy as np
from scipy.optimize import curve_fit
from scipy.stats import spearmanr, pearsonr
from scipy.interpolate import interp1d
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 1. データ読み込み
# ============================================================

def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

# --- f_tanh_robustness_detail.csv (Step 0 出力) ---
if not os.path.exists("f_tanh_robustness_detail.csv"):
    print("[ERROR] f_tanh_robustness_detail.csv not found.")
    exit(1)
_, detail_rows = load_csv("f_tanh_robustness_detail.csv")
print(f"[INFO] detail.csv: {len(detail_rows)} rows")

# --- sparc_results.csv ---
extra = {}
src_cols = []
if os.path.exists("sparc_results.csv"):
    src_cols, src_rows = load_csv("sparc_results.csv")
    print(f"[INFO] sparc_results.csv columns: {src_cols}")
    for row in src_rows:
        name = None
        for c in ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY']:
            if c in row and row[c].strip():
                name = row[c].strip()
                break
        if name is None:
            name = list(row.values())[0].strip()
        extra[name] = row

# --- SPARC photometric table (h_R 直接取得、あれば) ---
photo_hR = {}
photo_candidates = [
    "SPARC_Lelli2016c.mrt",
    "SPARC_Lelli2016c.dat",
    "SPARC_properties.csv",
    "sparc_photo.csv",
    "Table1.dat",
    "table1.mrt",
]
photo_file = None
for fn in photo_candidates:
    if os.path.exists(fn):
        photo_file = fn
```

```
        break

if photo_file:
    print(f"[INFO] Photometric table found: {photo_file}")
    # MRT/datフォーマットの読み込みを試行
    try:
        with open(photo_file, "r", encoding="utf-8", errors='replace') as fh:
            lines = fh.readlines()
        # CSVか固定幅かを判定
        if ',' in lines[0]:
            _, prows = load_csv(photo_file)
            print(f"  CSV format, columns: {list(prows[0].keys()) if prows else 'empty'}")
            for row in prows:
                gname = None
                for c in ['Galaxy', 'galaxy', 'Name', 'name']:
                    if c in row:
                        gname = row[c].strip()
                        break
                hR = None
                for c in ['R_d', 'Rd', 'h_R', 'hR', 'Rdisk', 'scale_length']:
                    if c in row and row[c].strip():
                        try:
                            hR = float(row[c])
                        except ValueError:
                            pass
                if gname and hR and hR &gt; 0:
                    photo_hR[gname] = hR
        else:
            # MRT形式: 固定幅。SPARCの場合 Galaxy=col1-12, R_d は特定列
            # ヘッダ行をスキップして数値行を探す
            for line in lines:
                parts = line.split()
                if len(parts) &gt;= 6:
                    try:
                        # 典型: Galaxy T D(Mpc) e_D inc e_inc L R_d(kpc) ...
                        float(parts[2])  # Dが数値なら本体行
                        gname = parts[0]
                        # R_d の位置は不明→スキップ（正確なフォーマットが必要）
                    except (ValueError, IndexError):
                        pass
            print(f"  MRT format detected. Auto-parse may need adjustment.")
    except Exception as e:
        print(f"  [WARN] Failed to parse photometric table: {e}")

print(f"[INFO] Photometric h_R available for {len(photo_hR)} galaxies")

# ============================================================
# 2. V_disk ピークから h_R を推定
# ============================================================

def read_sparc_dat(galaxy_name):
    candidates = [f"{galaxy_name}_rotmod.dat", f"{galaxy_name}.dat"]
    for fname in candidates:
        if os.path.exists(fname):
            data = np.loadtxt(fname, comments='#')
            rad = data[:, 0]
            v_obs = data[:, 1]
            v_gas = data[:, 3]
            v_disk = data[:, 4]
            v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
            return rad, v_obs, v_gas, v_disk, v_bul
    return None, None, None, None, None


def estimate_hR_from_vdisk(rad, v_disk):
    """
    Freeman指数円盤: V_disk(r) のピークは r_peak ≈ 2.15 × h_R
    |V_disk| のピーク半径を見つけて h_R を推定する
    """
    if rad is None or len(rad) &lt; 3:
        return None, None

    abs_vd = np.abs(v_disk)

    # スプライン補間でピークを精密化
    if len(rad) &gt;= 4:
        try:
            from scipy.interpolate import UnivariateSpline
            r_fine = np.linspace(rad.min(), rad.max(), 500)
            spl = UnivariateSpline(rad, abs_vd, s=len(rad)*0.5, k=min(3, len(rad)-1))
            vd_fine = spl(r_fine)
            i_peak = np.argmax(vd_fine)
            r_peak = r_fine[i_peak]
        except Exception:
            i_peak = np.argmax(abs_vd)
            r_peak = rad[i_peak]
    else:
        i_peak = np.argmax(abs_vd)
        r_peak = rad[i_peak]

    # ピークが最外点にある場合（まだ上昇中）→ h_R は推定不能（下限のみ）
    if r_peak &gt;= rad.max() * 0.95:
        return None, r_peak  # h_R=None, r_peak返す

    h_R = r_peak / 2.15
```

```
    return h_R, r_peak


# ============================================================
# 3. メインループ
# ============================================================
results = []

for d in detail_rows:
    gal = d['galaxy'].strip()
    vf = float(d['v_flat'])
    ft = float(d['f_tanh'])
    rs = float(d['r_s_tanh'])
    ud = float(d['upsilon_d'])
    ext = d['is_extrap'].strip() == 'True'

    # h_R 取得 (優先: photo table → V_disk peak推定)
    h_R = None
    h_R_source = None
    r_peak = None

    if gal in photo_hR:
        h_R = photo_hR[gal]
        h_R_source = 'photo'

    # V_disk peak からの推定 (photo があっても比較用に計算)
    rad, v_obs, v_gas, v_disk, v_bul = read_sparc_dat(gal)
    h_R_est = None
    if rad is not None:
        h_R_est, r_peak = estimate_hR_from_vdisk(rad, v_disk)

    if h_R is None and h_R_est is not None:
        h_R = h_R_est
        h_R_source = 'vpeak'

    if h_R is None or h_R &lt;= 0:
        continue

    # r_s_tanh / h_R
    rs_over_hR = rs / h_R

    # r_s_tanh / (2.15 * h_R)  = r_s / r_peak (ピーク比)
    rs_over_rpeak = rs / (2.15 * h_R) if h_R &gt; 0 else None

    results.append({
        'name': gal,
        'v_flat': vf,
        'f_tanh': ft,
        'r_s': rs,
        'upsilon_d': ud,
        'h_R': h_R,
        'h_R_source': h_R_source,
        'r_peak': r_peak,
        'h_R_est': h_R_est,
        'rs_over_hR': rs_over_hR,
        'rs_over_rpeak': rs_over_rpeak,
        'is_extrap': ext,
    })

print(f"¥n[INFO] h_R 取得成功: {len(results)} / {len(detail_rows)} 銀河")
n_photo = sum(1 for r in results if r['h_R_source'] == 'photo')
n_vpeak = sum(1 for r in results if r['h_R_source'] == 'vpeak')
print(f"  photo table: {n_photo}, V_disk peak: {n_vpeak}")

# 非外挿のみのサブセット
res_in = [r for r in results if not r['is_extrap']]
print(f"  非外挿: {len(res_in)}")

# ============================================================
# 4. r_s_tanh / h_R の基本統計
# ============================================================
print(f"¥n{'='*70}")
print("r_s_tanh / h_R の基本統計")
print(f"{'='*70}")

def stats_block(label, data):
    arr = np.array([d['rs_over_hR'] for d in data])
    print(f"¥n--- {label} (N={len(arr)}) ---")
    print(f"  中央値: {np.median(arr):.3f}")
    print(f"  平均値: {np.mean(arr):.3f}")
    print(f"  標準偏差: {np.std(arr):.3f}")
    print(f"  25%ile: {np.percentile(arr, 25):.3f}")
    print(f"  75%ile: {np.percentile(arr, 75):.3f}")
    return arr

rh_all = stats_block("全銀河", results)
rh_in = stats_block("非外挿のみ", res_in)

# ============================================================
# 5. r_s/h_R vs v_flat の相関とフィッティング
# ============================================================
print(f"¥n{'='*70}")
print("r_s_tanh / h_R vs v_flat")
print(f"{'='*70}")
```

```
# 非外挿データで解析
vf_arr = np.array([d['v_flat'] for d in res_in])
rh_arr = np.array([d['rs_over_hR'] for d in res_in])
ft_arr = np.array([d['f_tanh'] for d in res_in])
rs_arr = np.array([d['r_s'] for d in res_in])
hR_arr = np.array([d['h_R'] for d in res_in])
ud_arr = np.array([d['upsilon_d'] for d in res_in])
N = len(vf_arr)

# 正の値のみ (log解析用)
mask_pos = (rh_arr &amp;gt; 0) &amp;amp; (vf_arr &amp;gt; 0)
lv = np.log10(vf_arr[mask_pos])
lrh = np.log10(rh_arr[mask_pos])
N_pos = mask_pos.sum()

# Pearson &amp;amp; Spearman
rp, pp = pearsonr(lv, lrh)
rs_corr, ps = spearmanr(vf_arr[mask_pos], rh_arr[mask_pos])
print(f"¥nlog(r_s/h_R) vs log(v_flat): Pearson r={rp:.4f}, p={pp:.2e}")
print(f"r_s/h_R vs v_flat:          Spearman ρ={rs_corr:.4f}, p={ps:.2e}")

# --- べき乗フィット: log(r_s/h_R) = a + b*log(v) ---
p1 = np.polyfit(lv, lrh, 1)
print(f"¥nべき乗: r_s/h_R ∝ v_flat^{p1[0]:.3f}")
print(f"  log(r_s/h_R) = {p1[1]:.3f} + {p1[0]:.3f}*log(v_flat)")

# --- log二次 ---
p2 = np.polyfit(lv, lrh, 2)
lv_min2 = -p2[1]/(2*p2[0]) if abs(p2[0]) &amp;gt; 1e-6 else None
print(f"¥nlog二次: {p2[2]:.3f} + {p2[1]:.3f}*log(v) + {p2[0]:.3f}*log(v)²")
if lv_min2:
    print(f"  極値点: v_flat ≈ {10**lv_min2:.1f} km/s")

# --- BIC比較 ---
pred1 = np.polyval(p1, lv)
pred2 = np.polyval(p2, lv)
pred0 = np.mean(lrh) * np.ones(N_pos)

rss0 = np.sum((lrh - pred0)**2)
rss1 = np.sum((lrh - pred1)**2)
rss2 = np.sum((lrh - pred2)**2)

bic0 = N_pos * np.log(rss0/N_pos) + 1*np.log(N_pos)
bic1 = N_pos * np.log(rss1/N_pos) + 2*np.log(N_pos)
bic2 = N_pos * np.log(rss2/N_pos) + 3*np.log(N_pos)

print(f"¥nBIC比較:")
print(f"  定数:     BIC={bic0:.1f}")
print(f"  べき乗:   BIC={bic1:.1f} (ΔBIC={bic1-min(bic0,bic1,bic2):.1f})")
print(f"  log二次:  BIC={bic2:.1f} (ΔBIC={bic2-min(bic0,bic1,bic2):.1f})")

# ============================================================
# 6. h_R vs v_flat のスケーリング (既知のバリオン関係の確認)
# ============================================================
print(f"¥n{'='*70}")
print("h_R vs v_flat (バリオンのサイズ-質量関係)")
print(f"{'='*70}")

lhR = np.log10(hR_arr[mask_pos])
p_hR = np.polyfit(lv, lhR, 1)
r_hR, p_hR_p = pearsonr(lv, lhR)
print(f"¥nh_R ∝ v_flat^{p_hR[0]:.3f}  (r={r_hR:.4f}, p={p_hR_p:.2e})")
print(f"  log(h_R) = {p_hR[1]:.3f} + {p_hR[0]:.3f}*log(v_flat)")

# ============================================================
# 7. α = 0.795 の分解
# ============================================================
print(f"¥n{'='*70}")
print("α = 0.795 の分解: r_s = (r_s/h_R) × h_R")
print(f"{'='*70}")

# r_s vs v_flat の直接フィット (確認)
lrs = np.log10(rs_arr[mask_pos])
p_rs = np.polyfit(lv, lrs, 1)
r_rs, p_rs_p = pearsonr(lv, lrs)
print(f"¥n直接フィット: r_s ∝ v_flat^{p_rs[0]:.3f}  (r={r_rs:.4f})")
print(f"¥n分解:")
print(f"  r_s ∝ v_flat^{p_rs[0]:.3f}  (直接フィット)")
print(f"  h_R ∝ v_flat^{p_hR[0]:.3f}  (バリオンサイズ-質量)")
print(f"  r_s/h_R ∝ v_flat^{p1[0]:.3f}  (膜遷移位置の質量依存性)")
print(f"  合計: {p_hR[0]:.3f} + {p1[0]:.3f} = {p_hR[0]+p1[0]:.3f}  (cf. 直接={p_rs[0]:.3f})")
print(f"¥n  → バリオン寄与 β = {p_hR[0]:.3f}")
print(f"  → 膜固有寄与   γ = {p1[0]:.3f}")
print(f"  → α = β + γ = {p_hR[0]+p1[0]:.3f}")

# ============================================================
# 8. ビン別統計
# ============================================================
print(f"¥n{'='*70}")
print("v_flat ビン別 r_s/h_R 統計 (非外挿)")
print(f"{'='*70}")

sort_idx = np.argsort(vf_arr[mask_pos])
n_per = N_pos // 6
bin_vf = []
```

```
bin_rh = []
bin_hR_med = []

print(f"¥n{'v_flat範囲':&gt;18s}  {'N':&gt;4s}  {'r_s/h_R中央値':&gt;12s}  {'h_R中央値':&gt;10s}  {'r_s中央値':&gt;10s}  {'f中央値':&gt;8s}")
for i in range(6):
    i0 = i * n_per
    i1 = (i+1) * n_per if i &lt; 5 else N_pos
    idx = sort_idx[i0:i1]
    vf_b = vf_arr[mask_pos][idx]
    rh_b = rh_arr[mask_pos][idx]
    hR_b = hR_arr[mask_pos][idx]
    rs_b = rs_arr[mask_pos][idx]
    ft_b = ft_arr[mask_pos][idx]
    label = f"{vf_b.min():.0f}-{vf_b.max():.0f}"
    print(f"{label:&gt;18s}  {len(idx):4d}  {np.median(rh_b):12.3f}  {np.median(hR_b):10.3f}  {np.median(rs_b):10.3f}  {np.median(ft_b):8.3f}")
    bin_vf.append(np.median(vf_b))
    bin_rh.append(np.median(rh_b))
    bin_hR_med.append(np.median(hR_b))

# ============================================================
# 9. 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) r_s/h_R vs v_flat (log-log)
ax = axes[0, 0]
ax.scatter(lv, lrh, s=8, alpha=0.4, c='steelblue')
ax.scatter(np.log10(bin_vf), np.log10(bin_rh), s=60, c='red', zorder=5,
           edgecolors='black', label='Bin medians')
vv = np.linspace(lv.min(), lv.max(), 200)
ax.plot(vv, np.polyval(p1, vv), 'g--', label=f'Power: γ={p1[0]:.3f}')
ax.plot(vv, np.polyval(p2, vv), 'r-', label=f'Log-quad')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(r_s / h_R)')
ax.set_title(f'(a) r_s/h_R vs v_flat (r={rp:.3f})')
ax.legend(fontsize=8)

# (b) h_R vs v_flat
ax = axes[0, 1]
ax.scatter(lv, lhR, s=8, alpha=0.4, c='orange')
ax.plot(vv, np.polyval(p_hR, vv), 'k--', label=f'h_R ∝ v^{p_hR[0]:.2f}')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(h_R) [kpc]')
ax.set_title(f'(b) h_R vs v_flat (β={p_hR[0]:.3f})')
ax.legend(fontsize=8)
# (c) r_s vs v_flat (直接、分解の確認)
ax = axes[0, 2]
ax.scatter(lv, lrs, s=8, alpha=0.4, c='green')
ax.plot(vv, np.polyval(p_rs, vv), 'k-', label=f'r_s ∝ v^{p_rs[0]:.2f} (direct)')
ax.plot(vv, np.polyval(p_hR, vv) + np.polyval(p1, vv), 'r--',
        label=f'h_R × (r_s/h_R) ∝ v^{p_hR[0]+p1[0]:.2f}')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(r_s) [kpc]')
ax.set_title(f'(c) α decomposition check')
ax.legend(fontsize=8)

# (d) r_s/h_R のヒストグラム
ax = axes[1, 0]
ax.hist(rh_arr[mask_pos], bins=40, color='steelblue', edgecolor='white')
ax.axvline(np.median(rh_arr[mask_pos]), color='red', ls='--',
           label=f'Median={np.median(rh_arr[mask_pos]):.2f}')
ax.axvline(2.2, color='gray', ls=':', label='r_s = r_peak (=2.2 h_R)')
ax.set_xlabel('r_s_tanh / h_R')
ax.set_ylabel('Count')
ax.set_title('(d) r_s/h_R distribution')
ax.legend(fontsize=8)
ax.set_xlim(0, max(15, np.percentile(rh_arr[mask_pos], 98)))

# (e) f vs r_s/h_R
ax = axes[1, 1]
ax.scatter(rh_arr[mask_pos], ft_arr[mask_pos], s=8, alpha=0.4, c='purple')
rho_frh, p_frh = spearmanr(rh_arr[mask_pos], ft_arr[mask_pos])
ax.set_xlabel('r_s_tanh / h_R')
ax.set_ylabel('f^(tanh)')
ax.set_title(f'(e) f vs r_s/h_R (ρ={rho_frh:.3f})')

# (f) r_s/h_R vs r_s/r_peak (ピーク比との関係)
ax = axes[1, 2]
# r_s / (2.15*h_R) = r_s/r_peak を表示
rs_rpeak = rs_arr[mask_pos] / (2.15 * hR_arr[mask_pos])
ax.scatter(vf_arr[mask_pos], rs_rpeak, s=8, alpha=0.4, c='teal')
ax.axhline(1.0, color='gray', ls='--', label='r_s = r_peak')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('r_s_tanh / r_peak')
ax.set_title(f'(f) r_s vs disk peak (median={np.median(rs_rpeak):.2f})')
ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('step2_rs_hR_relation.png', dpi=150)
print(f"¥n[SAVED] step2_rs_hR_relation.png")

# ============================================================
# 10. CSV出力
# ============================================================
out_csv = "step2_rs_hR_detail.csv"
```

```
with open(out_csv, "w", newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerow(['galaxy', 'v_flat', 'r_s_tanh', 'h_R', 'h_R_source',
                     'rs_over_hR', 'f_tanh', 'upsilon_d', 'is_extrap'])
    for d in sorted(results, key=lambda x: x['v_flat']):
        writer.writerow([
            d['name'], f"{d['v_flat']:.1f}", f"{d['r_s']:.3f}",
            f"{d['h_R']:.3f}", d['h_R_source'],
            f"{d['rs_over_hR']:.4f}", f"{d['f_tanh']:.4f}",
            f"{d['upsilon_d']:.3f}", d['is_extrap']
        ])
print(f"[SAVED] {out_csv}")

# ============================================================
# 11. 結論
# ============================================================
print(f"¥n{'='*70}")
print("Step 2 結論")
print(f"{'='*70}")

print(f"""
■ α = 0.795 の分解結果:

  r_s_tanh ∝ v_flat^α  where α = β + γ

  β = {p_hR[0]:.3f}  (h_R ∝ v_flat^β: バリオンのサイズ-質量関係)
  γ = {p1[0]:.3f}  (r_s/h_R ∝ v_flat^γ: 膜遷移位置の質量依存性)
  β + γ = {p_hR[0]+p1[0]:.3f}  (cf. 直接フィット α = {p_rs[0]:.3f})

■ 物理的解釈:
  β はバリオン分布のスケーリングで決まる（外部入力）
  γ はv3.0の方程式系から導出すべき量

  γ ≈ 0 → r_s/h_R = const（膜遷移は常にディスクの同じ相対位置）
  γ &gt; 0 → 大銀河ほど r_s がディスクの外側で起きる
  γ &lt; 0 → 小銀河ほど r_s がディスクの外側で起きる

■ r_s/h_R の中央値 = {np.median(rh_arr[mask_pos]):.2f}
  r_s/r_peak の中央値 = {np.median(rs_rpeak):.2f}
  → r_s_tanh は V_disk ピークの{np.median(rs_rpeak):.1f}倍の位置

■ f vs r_s/h_R の相関: Spearman ρ = {rho_frh:.3f}
  → {'fとr_s/h_Rは強く相関（U字型の原因がr_s/h_Rの変動にある）' if abs(rho_frh) &gt; 0.3 else 'fとr_s/h_Rの相関は弱い'}

■ T-A1への接続:
  Step 3では γ = {p1[0]:.3f} をv3.0の平衡条件から導出できるか検討する
  h_R ∝ v_flat^{p_hR[0]:.3f} は外部の観測的関係として扱う
""")
```

4. step3_alpha_derivation.py

項目	内容
フェーズ	Step 3（旧版）
目的	v3.0 + Freeman円盤から alpha の理論的導出
使用rs	rs2 + 循環論法（f(v_flat) U字型を V_peak 設定に使用）
結果	alpha_theory ˜= 0.80。C-1 循環論法で無効化。
ステータス	無効（C-1 循環論法）

解析目的

Freeman指数円盤の解析的 v_bar(r) を構築し、v3.0 の MOND 式で v_obs(r) を計算。tanh フィットで r_s_tanh を決定し、v_flat を系統的に変えて alpha を導出。

ソースコード全文

```
#!/usr/bin/env python3
"""
Step 3: v3.0 の平衡条件 + Freeman 円盤から α を理論的に導出

方法：
  1. Freeman指数円盤の解析的 v_bar(r) を構築
  2. v3.0 の MOND 式で g_obs(r) → v_obs(r) を計算
  3. tanh モデルをフィットして r_s_tanh を決定
  4. v_flat を系統的に変えて r_s_tanh(v_flat) のスケーリングを導出
  5. 観測値（α=0.883, β=0.600, γ=0.284）と比較

外部データ不要（純粋理論計算）

実行: uv run --with scipy --with matplotlib --with numpy python step3_alpha_derivation.py
"""

import numpy as np
from scipy.optimize import curve_fit, brentq
from scipy.special import i0, i1, k0, k1
from scipy.stats import pearsonr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 物理定数
# ============================================================
a0 = 1.2e-10  # m/s^2, MOND加速度
G = 4.3e-3    # (km/s)^2 pc / M_sun → 使わない（無次元化）
kpc_to_m = 3.086e19

# ============================================================
# 1. Freeman 指数円盤の回転速度
# ============================================================

def v_disk_freeman(r, h_R, V_peak):
    """
    Freeman指数円盤の回転速度（解析解）
    V_disk^2(r) = 4*pi*G*Sigma0*h_R * y^2 * [I0*K0 - I1*K1]
    ここで y = r/(2*h_R)

    V_peak で規格化：V_disk(r_peak) = V_peak
    r_peak ≈ 2.15 * h_R
    """
    y = r / (2.0 * h_R)
    # 安全な計算（y=0 で特異性回避）
    y = np.clip(y, 1e-6, 50.0)

    # Bessel 関数の組み合わせ
    bessel_term = y**2 * (i0(y)*k0(y) - i1(y)*k1(y))

    # r_peak でのBessel値を計算して規格化
    y_peak = 2.15 * h_R / (2.0 * h_R)  # = 1.075
    bessel_peak = y_peak**2 * (i0(y_peak)*k0(y_peak) - i1(y_peak)*k1(y_peak))

    v2 = V_peak**2 * bessel_term / bessel_peak
    return np.sqrt(np.maximum(v2, 0.0))


def v_gas_model(r, h_R, f_gas=0.3):
    """
    ガス円盤の簡易モデル：指数円盤の2倍のスケール長
    V_gas ∝ f_gas * V_disk(r, 2*h_R)
    """
    # ガスのスケール長は典型的にディスクの1.5-2倍
    h_gas = 2.0 * h_R
    y = r / (2.0 * h_gas)
    y = np.clip(y, 1e-6, 50.0)
    bessel_term = y**2 * (i0(y)*k0(y) - i1(y)*k1(y))
```

```
    y_peak = 1.075
    bessel_peak = y_peak**2 * (i0(y_peak)*k0(y_peak) - i1(y_peak)*k1(y_peak))
    v2 = (f_gas * 30.0)**2 * bessel_term / bessel_peak  # 簡易規格化
    return np.sqrt(np.maximum(v2, 0.0))


# ============================================================
# 2. v3.0 の MOND 式
# ============================================================

def g_obs_mond(g_N, g_c):
    """
    v3.0 式(5): g_obs = (g_N + sqrt(g_N^2 + 4*g_c*g_N)) / 2
    """
    return (g_N + np.sqrt(g_N**2 + 4*g_c*g_N)) / 2


def v_obs_from_mond(r_kpc, v_bar, g_c_ms2):
    """
    g_N = v_bar^2 / r  (SI単位に変換して計算)
    g_obs = MOND式
    v_obs = sqrt(r * g_obs)
    """
    r_m = r_kpc * kpc_to_m
    v_bar_ms = v_bar * 1e3  # km/s → m/s

    g_N = np.where(r_m &gt; 0, v_bar_ms**2 / r_m, 0.0)
    g_o = g_obs_mond(g_N, g_c_ms2)
    v_obs_ms = np.sqrt(r_m * g_o)
    return v_obs_ms / 1e3  # m/s → km/s


# ============================================================
# 3. tanh フィットモデル
# ============================================================

def tanh_model(r, v_flat, r_s):
    """v_c^2 = v_flat^2 * T(r, r_s), T = 0.5*(1+tanh((r-r_s)/r_s))"""
    T = 0.5 * (1.0 + np.tanh((r - r_s) / r_s))
    return np.sqrt(v_flat**2 * T)


def tanh_model_with_bar(r, v_flat, r_s, v_bar_arr):
    """v_c^2 = v_bar^2 + v_flat^2 * T(r, r_s)"""
    T = 0.5 * (1.0 + np.tanh((r - r_s) / r_s))
    return np.sqrt(v_bar_arr**2 + v_flat**2 * T)


def fit_tanh(r, v_obs, v_bar):
    """
    v_c^2(r) = v_bar^2(r) + v_flat^2 * T(r, r_s) を v_obs にフィット
    フィットパラメータ: v_flat, r_s
    """
    def model(r, vf, rs):
        T = 0.5 * (1.0 + np.tanh((r - rs) / rs))
        return np.sqrt(v_bar**2 + vf**2 * T)

    # 初期推定
    vf0 = v_obs[-1]
    rs0 = r[len(r)//3]

    try:
        popt, _ = curve_fit(model, r, v_obs, p0=[vf0, rs0],
                            bounds=([0.1, 0.01], [500, r[-1]*3]),
                            maxfev=5000)
        return popt[0], popt[1]  # v_flat_fit, r_s_fit
    except Exception:
        return None, None


# ============================================================
# 4. 合成銀河の生成とフィッティング
# ============================================================

# バリオンのスケーリング関係 (Step 2 の観測結果)
# h_R ∝ v_flat^0.600
# 規格化：v_flat=100 km/s のとき h_R ≈ 2.5 kpc (典型的)

def hR_from_vflat(vf):
    """h_R [kpc] = A * (v_flat/100)^0.600"""
    return 2.5 * (vf / 100.0)**0.600


def Vpeak_from_vflat(vf, Upsilon_d=0.5):
    """
    V_peak (ディスクのピーク回転速度)
    指数円盤の場合: V_peak^2 ∝ M_disk/h_R ∝ Upsilon_d * L / h_R
    Tully-Fisher: L ∝ v_flat^4, h_R ∝ v_flat^0.6
    → V_peak^2 ∝ v_flat^4 / v_flat^0.6 = v_flat^3.4
    → V_peak ∝ v_flat^1.7
    但し V_peak &lt; v_flat (バリオンだけでは回転曲線を説明できない)
    規格化は f ≈ 0.29 の中央値から: V_peak^2 ≈ f * v_flat^2 at r_peak
    ただし f は U字型なので v_flat 依存にする
    """
```

```
    # Step 1 の log二次モデル: log(f) = 5.80 - 6.89*log(v) + 1.84*log(v)^2
    lv = np.log10(vf)
    log_f = 5.80 - 6.89*lv + 1.84*lv**2
    f = 10**log_f
    f = np.clip(f, 0.05, 2.0)
    V_peak = np.sqrt(f) * vf
    return V_peak


print("="*70)
print("Step 3: v3.0 + Freeman円盤 → α の理論的導出")
print("="*70)

# --- v_flat の範囲 ---
vflat_grid = np.logspace(np.log10(20), np.log10(300), 40)

# --- g_c のシナリオ ---
scenarios = {
    'g_c=a0': a0,
    'g_c=0.5*a0': 0.5*a0,
    'g_c=2*a0': 2.0*a0,
}

all_results = {}

for scenario_name, g_c in scenarios.items():
    print(f"¥n--- シナリオ: {scenario_name} ---")

    rs_list = []
    vf_list = []
    hR_list = []
    rpeak_list = []
    f_list = []

    for vf in vflat_grid:
        h_R = hR_from_vflat(vf)
        V_peak = Vpeak_from_vflat(vf)
        r_peak = 2.15 * h_R

        # 半径グリッド
        r = np.linspace(0.1 * h_R, 15.0 * h_R, 300)

        # バリオン回転速度
        v_bar = v_disk_freeman(r, h_R, V_peak)

        # MOND による観測回転速度
        v_obs = v_obs_from_mond(r, v_bar, g_c)

        # tanh フィット
        vf_fit, rs_fit = fit_tanh(r, v_obs, v_bar)

        if vf_fit is None or rs_fit &lt;= 0:
            continue

        # f の計算
        v_bar_at_rs = np.interp(rs_fit, r, v_bar)
        f_val = (v_bar_at_rs / vf_fit)**2

        rs_list.append(rs_fit)
        vf_list.append(vf)
        hR_list.append(h_R)
        rpeak_list.append(r_peak)
        f_list.append(f_val)

    rs_arr = np.array(rs_list)
    vf_arr = np.array(vf_list)
    hR_arr = np.array(hR_list)
    rpeak_arr = np.array(rpeak_list)
    f_arr = np.array(f_list)
    N = len(rs_arr)

    if N &lt; 5:
        print(f"  フィット成功数不足: N={N}")
        continue

    # --- スケーリング解析 ---
    lv = np.log10(vf_arr)
    lrs = np.log10(rs_arr)
    lhR = np.log10(hR_arr)
    lrh = np.log10(rs_arr / hR_arr)

    # α: r_s ∝ v^α
    p_alpha = np.polyfit(lv, lrs, 1)
    # β: h_R ∝ v^β
    p_beta = np.polyfit(lv, lhR, 1)
    # γ: r_s/h_R ∝ v^γ
    p_gamma = np.polyfit(lv, lrh, 1)

    # r_s / r_peak
    rs_rpeak = rs_arr / rpeak_arr

    print(f"  N={N}")
    print(f"  α (r_s ~ v^α)   = {p_alpha[0]:.4f}")
    print(f"  β (h_R ~ v^β)   = {p_beta[0]:.4f}  (入力: 0.600)")
    print(f"  γ (r_s/h_R ~ v^γ) = {p_gamma[0]:.4f}")
```

```
    print(f"  β + γ = {p_beta[0]+p_gamma[0]:.4f}")
    print(f"  r_s/r_peak 中央値 = {np.median(rs_rpeak):.4f}")
    print(f"  r_s/r_peak 範囲  = {rs_rpeak.min():.3f} - {rs_rpeak.max():.3f}")
    print(f"  f 中央値 = {np.median(f_arr):.4f}")

    all_results[scenario_name] = {
        'vf': vf_arr, 'rs': rs_arr, 'hR': hR_arr, 'rpeak': rpeak_arr,
        'f': f_arr, 'alpha': p_alpha[0], 'beta': p_beta[0],
        'gamma': p_gamma[0], 'rs_rpeak': rs_rpeak,
        'p_alpha': p_alpha, 'p_gamma': p_gamma,
    }

# ============================================================
# 5. 観測値との比較
# ============================================================
print(f"¥n{'='*70}")
print("観測値との比較")
print(f"{'='*70}")

print(f"¥n{'シナリオ':&gt;15s}  {'α_theory':&gt;10s}  {'γ_theory':&gt;10s}  {'r_s/r_peak':&gt;10s}")
print(f"{'観測値':&gt;15s}  {'0.883':&gt;10s}  {'0.284':&gt;10s}  {'0.98':&gt;10s}")
print("-"*50)
for name, res in all_results.items():
    print(f"{name:&gt;15s}  {res['alpha']:10.4f}  {res['gamma']:10.4f}  {np.median(res['rs_rpeak']):10.4f}")

# ============================================================
# 6. 解析的導出の試み
# ============================================================
print(f"¥n{'='*70}")
print("解析的考察")
print(f"{'='*70}")

print("""
■ r_s_tanh ≈ r_peak の解析的理解

式(7): v_c²(r) = v_bar²(r) + v_flat² · T(r, r_s)
MOND:  v_obs² = r · g_obs = r · [g_N/2 + √(g_N²/4 + g_c·g_N)]

tanh フィットの r_s は「v_obs(r) の形状変化が最大の点」に対応。

Freeman 円盤では v_bar(r) は r_peak = 2.15·h_R でピーク。
r &lt; r_peak: v_bar 上昇 → v_obs も急上昇（バリオン駆動）
r &gt; r_peak: v_bar 下降、v_flat 成分が補償 → v_obs 平坦化

→ v_obs の曲率変化（凹→凸の変曲点）が r_peak 付近に来る
→ tanh(w·(r-r_s)/r_s) の変曲点は r = r_s
→ r_s_tanh ≈ r_peak（自然な帰結）

■ γ ≠ 0 の起源

深MOND極限 (g_N &lt;&lt; g_c) : v_obs ≈ (g_c·v_bar²/r)^{1/4} · r^{1/2}
  → 遷移半径は v_bar(r) のピークではなく、v_bar²/r のピークで決まる
  → r_peak(v_bar²/r) = r_peak(v_bar)·[補正項]

ニュートン極限 (g_N &gt;&gt; g_c) : v_obs ≈ v_bar
  → 遷移半径 = v_bar のピーク（そのまま）

大銀河: よりニュートン的 → r_s ≈ r_peak(v_bar)
小銀河: より深MOND → r_s ≈ r_peak(v_bar²/r) &gt; r_peak(v_bar)

→ 小銀河で r_s/r_peak がわずかに変化する
→ γ の符号と大きさは g_N/g_c の比率変化で決まる
""")

# ============================================================
# 7. g_N/g_c 比率の系統的変化の確認
# ============================================================
print(f"¥n{'='*70}")
print("g_N(r_peak)/g_c の v_flat 依存性")
print(f"{'='*70}")

if 'g_c=a0' in all_results:
    res = all_results['g_c=a0']
    vf_arr = res['vf']
    rpeak_arr = res['rpeak']

    # g_N(r_peak) の計算
    gN_at_rpeak = []
    for i in range(len(vf_arr)):
        vf = vf_arr[i]
        h_R = res['hR'][i]
        V_peak = Vpeak_from_vflat(vf)
        r_peak = rpeak_arr[i]

        v_bar_peak = V_peak  # V_disk のピーク値
        g_N = (v_bar_peak * 1e3)**2 / (r_peak * kpc_to_m)
        gN_at_rpeak.append(g_N / a0)

    gN_at_rpeak = np.array(gN_at_rpeak)
    print(f"¥nv_flat [km/s]  g_N(r_peak)/a0  regime")
    for vf_show in [25, 50, 75, 100, 150, 200, 300]:
        idx = np.argmin(np.abs(vf_arr - vf_show))
        regime = "deep MOND" if gN_at_rpeak[idx] &lt; 0.3 else ¥
                 "MOND" if gN_at_rpeak[idx] &lt; 3 else "Newtonian"
        print(f"  {vf_arr[idx]:6.0f}       {gN_at_rpeak[idx]:8.3f}        {regime}")
```

```
# ============================================================
# 8. 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

colors = {'g_c=a0': 'blue', 'g_c=0.5*a0': 'green', 'g_c=2*a0': 'orange'}

# (a) r_s vs v_flat (理論 vs 観測)
ax = axes[0, 0]
for name, res in all_results.items():
    lv = np.log10(res['vf'])
    lrs = np.log10(res['rs'])
    ax.plot(lv, lrs, '-', color=colors.get(name, 'gray'),
            label=f"{name}: α={res['alpha']:.3f}")
ax.plot([np.log10(20), np.log10(300)],
        [np.polyval([0.883, -1.5], np.log10(20)),
         np.polyval([0.883, -1.5], np.log10(300))],
        'k--', alpha=0.5, label='Observed α=0.883')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(r_s_tanh)')
ax.set_title('(a) r_s vs v_flat: theory')
ax.legend(fontsize=7)

# (b) r_s/h_R vs v_flat (γ の可視化)
ax = axes[0, 1]
for name, res in all_results.items():
    lv = np.log10(res['vf'])
    lrh = np.log10(res['rs'] / res['hR'])
    ax.plot(lv, lrh, '-', color=colors.get(name, 'gray'),
            label=f"{name}: γ={res['gamma']:.3f}")
ax.axhline(np.log10(2.11), color='k', ls='--', alpha=0.5, label='Obs median=2.11')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(r_s / h_R)')
ax.set_title('(b) γ: membrane contribution')
ax.legend(fontsize=7)

# (c) r_s / r_peak vs v_flat
ax = axes[0, 2]
for name, res in all_results.items():
    ax.plot(res['vf'], res['rs_rpeak'], '-', color=colors.get(name, 'gray'),
            label=name)
ax.axhline(0.98, color='k', ls='--', alpha=0.5, label='Obs median=0.98')
ax.axhline(1.0, color='gray', ls=':', alpha=0.3)
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('r_s / r_peak')
ax.set_title('(c) r_s_tanh / r_peak')
ax.legend(fontsize=8)

# (d) f(v_flat) の理論 vs 観測
ax = axes[1, 0]
for name, res in all_results.items():
    ax.plot(res['vf'], res['f'], '-', color=colors.get(name, 'gray'),
            label=name)
# 観測の U 字型
vv = np.logspace(np.log10(20), np.log10(300), 100)
lv_obs = np.log10(vv)
log_f_obs = 5.80 - 6.89*lv_obs + 1.84*lv_obs**2
ax.plot(vv, 10**log_f_obs, 'k--', label='Obs (log-quad)')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('f^(tanh)')
ax.set_title('(d) f(v_flat): theory vs obs')
ax.legend(fontsize=7)
ax.set_ylim(0, 1.5)

# (e) 合成銀河の回転曲線例 (3例)
ax = axes[1, 1]
for vf_show, col in [(30, 'blue'), (100, 'green'), (250, 'red')]:
    h_R = hR_from_vflat(vf_show)
    V_peak = Vpeak_from_vflat(vf_show)
    r = np.linspace(0.1*h_R, 12*h_R, 300)
    v_bar = v_disk_freeman(r, h_R, V_peak)
    v_obs = v_obs_from_mond(r, v_bar, a0)

    r_norm = r / h_R
    ax.plot(r_norm, v_bar, '--', color=col, alpha=0.4)
    ax.plot(r_norm, v_obs, '-', color=col,
            label=f'v_flat={vf_show}')
    ax.axvline(2.15, color='gray', ls=':', alpha=0.3)

ax.set_xlabel('r / h_R')
ax.set_ylabel('v [km/s]')
ax.set_title('(e) Synthetic RCs (dashed=baryonic)')
ax.legend(fontsize=8)

# (f) α の分解図 (棒グラフ)
ax = axes[1, 2]
if 'g_c=a0' in all_results:
    res = all_results['g_c=a0']
    labels = ['Observed', 'Theory (g_c=a0)']
    betas = [0.600, res['beta']]
    gammas = [0.284, res['gamma']]
    x = np.arange(len(labels))
    w = 0.35
    ax.bar(x, betas, w, label='β (baryonic)', color='steelblue')
    ax.bar(x, gammas, w, bottom=betas, label='γ (membrane)', color='coral')
```

```
    for i in range(len(labels)):
        ax.text(i, betas[i]+gammas[i]+0.02, f'α={betas[i]+gammas[i]:.3f}',
                ha='center', fontsize=9)
    ax.set_xticks(x)
    ax.set_xticklabels(labels)
    ax.set_ylabel('Exponent')
    ax.set_title('(f) α decomposition')
    ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('step3_alpha_derivation.png', dpi=150)
print(f"¥n[SAVED] step3_alpha_derivation.png")

# ============================================================
# 9. 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("Step 3 最終結論")
print(f"{'='*70}")

if 'g_c=a0' in all_results:
    res = all_results['g_c=a0']
    print(f"""
■ v3.0 + Freeman円盤 + h_R ∝ v^0.6 + f(v_flat) U字型 から:

  理論 α = {res['alpha']:.4f}  (観測 α = 0.883)
  理論 β = {res['beta']:.4f}  (観測 β = 0.600, 入力値)
  理論 γ = {res['gamma']:.4f}  (観測 γ = 0.284)
  r_s/r_peak = {np.median(res['rs_rpeak']):.4f}  (観測 = 0.98)

■ 一致度の評価:
  α: 理論 {res['alpha']:.3f} vs 観測 0.883 → 差 {abs(res['alpha']-0.883):.3f}
  γ: 理論 {res['gamma']:.3f} vs 観測 0.284 → 差 {abs(res['gamma']-0.284):.3f}

■ 結論:
""")
    if abs(res['alpha'] - 0.883) &lt; 0.1:
        print("  → α は v3.0 の MOND 式 + Freeman 円盤から導出可能")
        print("  → 外部入力は h_R ∝ v^0.6 と f(v_flat) のU字型のみ")
        print("  → 膜理論固有の新しい物理は不要（γ はMOND遷移の幾何学的効果）")
    elif abs(res['gamma'] - 0.284) &lt; 0.1:
        print("  → γ の大きさは再現。MOND遷移の幾何学的効果で説明可能。")
        print("  → α の差は β の入力精度に依存。")
    else:
        print("  → 理論と観測に有意な差がある。")
        print("  → 追加の物理（g_c の銀河依存性等）が必要な可能性。")

print(f"""
■ T-A1 の回答（最終版）:

  「なぜ r_s_tanh ∝ v_flat^0.795 か」

  答え：3つの要素の組み合わせ:

  (1) h_R ∝ v_flat^0.6（バリオンのサイズ-質量関係）
      → α の ~68% を説明。外部の観測的事実。

  (2) r_s_tanh ≈ r_peak = 2.15·h_R（v3.0の帰結）
      → v_obs(r) の変曲点が v_bar(r) のピークに一致。
      → MOND式の構造から自然に導かれる。

  (3) γ ≈ 0.28（MOND遷移の幾何学的効果）
      → 小銀河は深MOND極限 → v_obs の変曲点がわずかにずれる
      → 大銀河はニュートン的 → r_s ≈ r_peak（正確に一致）
      → この差が γ &gt; 0 を生む

  v3.0 の方程式系のみから導出可能な部分: (2) と (3)
  外部入力が必要な部分: (1) のバリオンスケーリング則
""")
```

5. step3_revised.py

項目	内容
フェーズ	Step 3（修正版）
目的	C-1 修正: V_peak を実測べき乗で設定
使用rs	非循環
結果	alpha = 0.650。旧版の 0.801 との差 0.15 は循環論法の影響。観測 0.883 との差 0.23。
ステータス	参考（rs2ベースの alpha=0.883 自体が後に無効化）

解析目的

C-1 循環論法を修正。V_peak を r_s に依存しない方法（SPARC .dat から直接測定）で設定し、alpha を再導出。

ソースコード全文

```
#!/usr/bin/env python3
"""
Step 3 修正版: C-1 循環論法の修正

問題: 旧Step3で V_peak = sqrt(f(v_flat)) * v_flat を使用していたが、
      f = v_bar^2(r_s)/v_flat^2 は r_s に依存 → 循環論法

修正: V_peak を SPARC .dat ファイルから直接測定（r_s に一切依存しない）
      V_peak = max(|v_bar(r)|) = max(sqrt(Y_d*v_disk^2 + v_gas^2 + v_bul^2))
      この V_peak vs v_flat の関係を実測し、合成モデルに入力

2段構成:
  Part A: SPARC から V_peak vs v_flat を実測（非循環的）
  Part B: 実測関係を使って合成銀河を生成し α を導出

実行: uv run --with scipy --with matplotlib --with numpy python step3_revised.py
"""

import csv
import os
import numpy as np
from scipy.optimize import curve_fit, minimize_scalar
from scipy.special import i0, i1, k0, k1
from scipy.stats import pearsonr, spearmanr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 定数
# ============================================================
a0_unit = 3703.0  # a0 in (km/s)^2 / kpc
kpc_to_m = 3.086e19

# ============================================================
# Part A: SPARC から V_peak, h_R_est を実測
# ============================================================
print("="*70)
print("Part A: SPARC 実測 V_peak, h_R")
print("="*70)

def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

# sparc_results.csv
src_cols, src_rows = load_csv("sparc_results.csv")
galaxies = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name:
        name = list(row.values())[0].strip()
    ud = get_val(row, ['ud', 'upsilon_d'])
    vf = get_val(row, ['vflat', 'v_flat'])
    rs = get_val(row, ['rs1', 'r_s'])
    if ud and vf and rs and vf &gt; 0:
        galaxies[name] = {'upsilon_d': ud, 'vflat': vf, 'rs_tanh': rs}

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None
```

```
# --- V_peak と h_R の実測 ---
meas = []

for name, gal in galaxies.items():
    data = read_dat(name)
    if data is None:
        continue
    rad = data[:, 0]
    v_obs = data[:, 1]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = gal['upsilon_d']

    # v_bar (全バリオン成分)
    vbar2 = ud * np.sign(v_disk)*v_disk**2 + ¥
            np.sign(v_gas)*v_gas**2 + ¥
            np.sign(v_bul)*v_bul**2
    vbar = np.sqrt(np.maximum(vbar2, 0.0))

    # v_disk のみ (Υ_d 適用)
    vdisk_scaled = np.sqrt(ud) * np.abs(v_disk)

    if len(rad) &lt; 3:
        continue

    # V_peak (v_bar 全体)
    V_peak_bar = np.max(vbar)
    i_peak_bar = np.argmax(vbar)
    r_peak_bar = rad[i_peak_bar]

    # V_peak (v_disk のみ)
    V_peak_disk = np.max(vdisk_scaled)
    i_peak_disk = np.argmax(vdisk_scaled)
    r_peak_disk = rad[i_peak_disk]

    # h_R 推定 (v_disk ピークから)
    peak_at_edge = (r_peak_disk &gt;= rad.max() * 0.9)
    h_R_est = r_peak_disk / 2.15 if not peak_at_edge else None

    # バルジの有無
    has_bulge = np.any(np.abs(v_bul) &gt; 1.0)

    meas.append({
        'name': name,
        'vflat': gal['vflat'],
        'rs_tanh': gal['rs_tanh'],
        'upsilon_d': ud,
        'V_peak_bar': V_peak_bar,
        'V_peak_disk': V_peak_disk,
        'r_peak_bar': r_peak_bar,
        'r_peak_disk': r_peak_disk,
        'h_R_est': h_R_est,
        'has_bulge': has_bulge,
        'peak_at_edge': peak_at_edge,
    })

print(f"実測成功: {len(meas)} 銀河")

# --- V_peak vs v_flat のフィット (非循環的) ---
vf_m = np.array([d['vflat'] for d in meas])
Vp_bar = np.array([d['V_peak_bar'] for d in meas])
Vp_disk = np.array([d['V_peak_disk'] for d in meas])

# log-log フィット: V_peak = A * v_flat^B
mask = (vf_m &gt; 0) &amp;amp; (Vp_disk &gt; 0)
lv = np.log10(vf_m[mask])
lVp_disk = np.log10(Vp_disk[mask])
lVp_bar = np.log10(Vp_bar[mask])

p_Vp_disk = np.polyfit(lv, lVp_disk, 1)
p_Vp_bar = np.polyfit(lv, lVp_bar, 1)
r_disk, _ = pearsonr(lv, lVp_disk)
r_bar, _ = pearsonr(lv, lVp_bar)

print(f"¥nV_peak(disk) ∝ v_flat^{p_Vp_disk[0]:.3f}  (r={r_disk:.3f})")
print(f"  log(V_peak) = {p_Vp_disk[1]:.3f} + {p_Vp_disk[0]:.3f}*log(v_flat)")

print(f"¥nV_peak(bar) ∝ v_flat^{p_Vp_bar[0]:.3f}  (r={r_bar:.3f})")
print(f"  log(V_peak) = {p_Vp_bar[1]:.3f} + {p_Vp_bar[0]:.3f}*log(v_flat)")

# V_peak / v_flat の統計
ratio_disk = Vp_disk / vf_m
ratio_bar = Vp_bar / vf_m
print(f"¥nV_peak(disk)/v_flat: 中央値={np.median(ratio_disk):.3f}, mean={np.mean(ratio_disk):.3f}")
print(f"V_peak(bar)/v_flat:  中央値={np.median(ratio_bar):.3f}, mean={np.mean(ratio_bar):.3f}")

# h_R vs v_flat (h_R 推定可能のみ)
hR_data = [(d['vflat'], d['h_R_est']) for d in meas if d['h_R_est'] is not None]
if hR_data:
    vf_hR = np.array([x[0] for x in hR_data])
    hR_arr = np.array([x[1] for x in hR_data])
    mask_hR = (vf_hR &gt; 0) &amp;amp; (hR_arr &gt; 0)
    p_hR = np.polyfit(np.log10(vf_hR[mask_hR]), np.log10(hR_arr[mask_hR]), 1)
    r_hR, _ = pearsonr(np.log10(vf_hR[mask_hR]), np.log10(hR_arr[mask_hR]))
```

```
    print(f"¥nh_R ∝ v_flat^{p_hR[0]:.3f}  (r={r_hR:.3f}, N={mask_hR.sum()})")


# ============================================================
# Part B: 合成銀河の生成と α 導出（修正版）
# ============================================================
print(f"¥n{'='*70}")
print("Part B: 合成銀河 → α 導出（V_peak 非循環版）")
print(f"{'='*70}")

# Freeman 円盤
def freeman_bessel(y):
    y = np.clip(y, 1e-8, 50.0)
    return y**2 * (i0(y)*k0(y) - i1(y)*k1(y))

y_fine = np.linspace(0.01, 5.0, 5000)
vd2_shape = freeman_bessel(y_fine)
i_vpeak_freeman = np.argmax(vd2_shape)
y_vpeak_freeman = y_fine[i_vpeak_freeman]
bessel_peak = vd2_shape[i_vpeak_freeman]

def v_disk_freeman(r, h_R, V_peak):
    y = r / (2.0 * h_R)
    y = np.clip(y, 1e-6, 50.0)
    v2 = V_peak**2 * freeman_bessel(y) / bessel_peak
    return np.sqrt(np.maximum(v2, 0.0))

def mond_gobs(gN, gc):
    return (gN + np.sqrt(gN**2 + 4*gc*gN)) / 2

def v_obs_from_mond(r_kpc, v_bar, g_c):
    r_m = r_kpc * kpc_to_m
    v_bar_ms = v_bar * 1e3
    gN = np.where(r_m &gt; 0, v_bar_ms**2 / r_m, 0.0)
    go = mond_gobs(gN, g_c)
    return np.sqrt(r_m * go) / 1e3

def fit_tanh(r, v_obs, v_bar):
    def model(r, vf, rs):
        T = 0.5 * (1.0 + np.tanh((r - rs) / rs))
        return np.sqrt(v_bar**2 + vf**2 * T)
    try:
        popt, _ = curve_fit(model, r, v_obs, p0=[v_obs[-1], r[len(r)//3]],
                            bounds=([0.1, 0.01], [500, r[-1]*3]), maxfev=5000)
        return popt[0], popt[1]
    except:
        return None, None


# --- V_peak の設定: 実測の V_peak(disk) ∝ v_flat^B を使用 ---
# これは r_s に依存しない（.dat の v_disk ピークから直接測定）
def Vpeak_nocirc(vf):
    """非循環的 V_peak: 実測べき乗関係"""
    return 10**np.polyval(p_Vp_disk, np.log10(vf))

def hR_from_vflat(vf):
    """h_R: 実測べき乗関係"""
    if hR_data:
        return 10**np.polyval(p_hR, np.log10(vf))
    return 2.5 * (vf / 100.0)**0.6


# --- 合成銀河グリッド ---
vflat_grid = np.logspace(np.log10(20), np.log10(300), 40)

scenarios = {
    'g_c=a0': 1.2e-10,
    'g_c=0.5a0': 0.6e-10,
    'g_c=2a0': 2.4e-10,
}

all_results = {}
for sc_name, g_c in scenarios.items():
    rs_list, vf_list, hR_list = [], [], []

    for vf in vflat_grid:
        h_R = hR_from_vflat(vf)
        V_peak = Vpeak_nocirc(vf)

        r = np.linspace(0.1*h_R, 15.0*h_R, 300)
        v_bar = v_disk_freeman(r, h_R, V_peak)
        v_obs = v_obs_from_mond(r, v_bar, g_c)

        vf_fit, rs_fit = fit_tanh(r, v_obs, v_bar)
        if vf_fit is None or rs_fit &lt;= 0:
            continue

        rs_list.append(rs_fit)
        vf_list.append(vf)
        hR_list.append(h_R)

    rs_arr = np.array(rs_list)
    vf_arr = np.array(vf_list)
    hR_arr = np.array(hR_list)
    N = len(rs_arr)
```

```
    if N &lt; 5:
        print(f"  {sc_name}: フィット不足 (N={N})")
        continue

    lv = np.log10(vf_arr)
    lrs = np.log10(rs_arr)
    lhR = np.log10(hR_arr)
    lrh = np.log10(rs_arr/hR_arr)

    p_alpha = np.polyfit(lv, lrs, 1)
    p_beta = np.polyfit(lv, lhR, 1)
    p_gamma = np.polyfit(lv, lrh, 1)

    rpeak_arr = 2.15 * hR_arr
    rs_rpeak = rs_arr / rpeak_arr

    print(f"¥n--- {sc_name} (N={N}) ---")
    print(f"  α (r_s ~ v^α)     = {p_alpha[0]:.4f}")
    print(f"  β (h_R ~ v^β)     = {p_beta[0]:.4f}")
    print(f"  γ (r_s/h_R ~ v^γ) = {p_gamma[0]:.4f}")
    print(f"  β + γ              = {p_beta[0]+p_gamma[0]:.4f}")
    print(f"  r_s/r_peak 中央値  = {np.median(rs_rpeak):.4f}")

    all_results[sc_name] = {
        'vf': vf_arr, 'rs': rs_arr, 'hR': hR_arr,
        'alpha': p_alpha[0], 'beta': p_beta[0], 'gamma': p_gamma[0],
        'rs_rpeak': rs_rpeak, 'p_alpha': p_alpha,
    }

# ============================================================
# Part C: 比較表
# ============================================================
print(f"¥n{'='*70}")
print("旧 Step 3 vs 修正版の比較")
print(f"{'='*70}")

print(f"¥n{'':&gt;20s}  {'α':&gt;8s}  {'β':&gt;8s}  {'γ':&gt;8s}  {'r_s/r_peak':&gt;10s}")
print(f"{'観測':&gt;20s}  {'0.883':&gt;8s}  {'0.600':&gt;8s}  {'0.284':&gt;8s}  {'0.98':&gt;10s}")
print(f"{'旧Step3 (循環)':&gt;20s}  {'0.801':&gt;8s}  {'0.600':&gt;8s}  {'0.201':&gt;8s}  {'0.571':&gt;10s}")
for name, res in all_results.items():
    print(f"{('修正版 '+name):&gt;20s}  {res['alpha']:8.3f}  {res['beta']:8.3f}  {res['gamma']:8.3f}  {np.median(res['rs_rpeak']):10.3f}")

# ============================================================
# Part D: V_peak 入力の比較
# ============================================================
print(f"¥n{'='*70}")
print("V_peak 入力の比較: 循環 vs 非循環")
print(f"{'='*70}")

vv = np.array([30, 50, 75, 100, 150, 200, 300])
print(f"¥n{'v_flat':&gt;8s}  {'V_peak(旧/循環)':&gt;16s}  {'V_peak(修正/実測)':&gt;18s}  {'差%':&gt;6s}")
for vf in vv:
    lv = np.log10(vf)
    # 旧: f(v_flat) U字型
    log_f_old = 5.80 - 6.89*lv + 1.84*lv**2
    f_old = np.clip(10**log_f_old, 0.05, 2.0)
    Vp_old = np.sqrt(f_old) * vf
    # 新: 実測べき乗
    Vp_new = Vpeak_nocirc(vf)
    diff = 100*(Vp_new - Vp_old)/Vp_old if Vp_old &gt; 0 else 0
    print(f"{vf:8.0f}  {Vp_old:16.1f}  {Vp_new:18.1f}  {diff:+6.1f}%")

# ============================================================
# Part E: 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) V_peak vs v_flat (実測)
ax = axes[0, 0]
ax.scatter(vf_m, Vp_disk, s=8, alpha=0.4, c='blue', label='V_peak(disk)')
ax.scatter(vf_m, Vp_bar, s=8, alpha=0.4, c='red', label='V_peak(bar)')
vv_line = np.linspace(vf_m.min(), vf_m.max(), 100)
ax.plot(vv_line, 10**np.polyval(p_Vp_disk, np.log10(vv_line)), 'b--',
        label=f'disk: ∝v^{p_Vp_disk[0]:.2f}')
ax.plot(vv_line, vv_line, 'k:', alpha=0.3, label='V_peak=v_flat')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('V_peak [km/s]')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title('(a) V_peak vs v_flat (SPARC direct)')
ax.legend(fontsize=7)

# (b) V_peak/v_flat vs v_flat (旧 vs 新の比較)
ax = axes[0, 1]
ax.scatter(vf_m, Vp_disk/vf_m, s=8, alpha=0.3, c='blue', label='SPARC measured')
vv2 = np.logspace(np.log10(20), np.log10(300), 100)
# 旧 (循環)
lv2 = np.log10(vv2)
f_old = 10**(5.80 - 6.89*lv2 + 1.84*lv2**2)
f_old = np.clip(f_old, 0.05, 2.0)
ax.plot(vv2, np.sqrt(f_old), 'r-', label='Old (circular f)')
# 新 (実測)
ax.plot(vv2, 10**np.polyval(p_Vp_disk, lv2)/vv2, 'g--', label='New (measured)')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('V_peak / v_flat')
```

```
ax.set_xscale('log')
ax.set_title('(b) V_peak/v_flat: old vs new')
ax.legend(fontsize=8)

# (c) r_s vs v_flat (理論、修正版)
ax = axes[0, 2]
colors = {'g_c=a0': 'blue', 'g_c=0.5a0': 'green', 'g_c=2a0': 'orange'}
for name, res in all_results.items():
    ax.plot(np.log10(res['vf']), np.log10(res['rs']), '-',
            color=colors.get(name, 'gray'),
            label=f"{name}: α={res['alpha']:.3f}")
# 観測の傾き
ax.plot([np.log10(20), np.log10(300)],
        np.polyval([0.883, np.polyval([0.883, -1.5], np.log10(20))- 0.883*np.log10(20)],
                   [np.log10(20), np.log10(300)]),
        'k--', alpha=0.3, label='Obs α=0.883')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(r_s)')
ax.set_title('(c) r_s vs v_flat (revised)')
ax.legend(fontsize=7)

# (d) α 分解の棒グラフ (旧 vs 新)
ax = axes[1, 0]
if 'g_c=a0' in all_results:
    res = all_results['g_c=a0']
    labels = ['Observed', 'Old (circular)', 'Revised']
    betas = [0.600, 0.600, res['beta']]
    gammas = [0.284, 0.201, res['gamma']]
    x = np.arange(len(labels))
    ax.bar(x, betas, 0.35, label='β (baryonic)', color='steelblue')
    ax.bar(x, gammas, 0.35, bottom=betas, label='γ (membrane)', color='coral')
    for i in range(len(labels)):
        ax.text(i, betas[i]+gammas[i]+0.02, f'α={betas[i]+gammas[i]:.3f}',
                ha='center', fontsize=9)
    ax.set_xticks(x)
    ax.set_xticklabels(labels, fontsize=8)
    ax.set_ylabel('Exponent')
    ax.set_title('(d) α decomposition: old vs revised')
    ax.legend(fontsize=8)

# (e) r_s/r_peak の比較
ax = axes[1, 1]
for name, res in all_results.items():
    ax.plot(res['vf'], res['rs_rpeak'], '-', color=colors.get(name, 'gray'),
            label=f"{name}: {np.median(res['rs_rpeak']):.2f}")
ax.axhline(0.98, color='black', ls='--', label='Obs=0.98')
ax.axhline(0.571, color='red', ls=':', label='Old=0.571')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('r_s / r_peak')
ax.set_title('(e) r_s/r_peak: revised')
ax.legend(fontsize=7)

# (f) h_R vs v_flat
ax = axes[1, 2]
if hR_data:
    ax.scatter(vf_hR, hR_arr, s=8, alpha=0.3, c='orange')
    vv3 = np.logspace(np.log10(vf_hR.min()), np.log10(vf_hR.max()), 100)
    ax.plot(vv3, 10**np.polyval(p_hR, np.log10(vv3)), 'k--',
            label=f'h_R ∝ v^{p_hR[0]:.2f}')
    ax.set_xlabel('v_flat [km/s]')
    ax.set_ylabel('h_R [kpc]')
    ax.set_xscale('log')
    ax.set_yscale('log')
    ax.set_title(f'(f) h_R vs v_flat (β={p_hR[0]:.3f})')
    ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('step3_revised.png', dpi=150)
print(f"¥n[SAVED] step3_revised.png")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("C-1 修正の結論")
print(f"{'='*70}")

if 'g_c=a0' in all_results:
    res = all_results['g_c=a0']
    print(f"""
■ V_peak の入力方法:
  旧: V_peak = sqrt(f(v_flat)) × v_flat  ← f は r_s 依存 (循環論法)
  新: V_peak = 実測べき乗 V_peak(disk) ∝ v_flat^{p_Vp_disk[0]:.3f}  ← r_s 不使用

■ α 導出結果の比較:
  旧Step3: α = {0.801:.3f}  (β=0.600, γ=0.201)
  修正版:  α = {res['alpha']:.3f}  (β={res['beta']:.3f}, γ={res['gamma']:.3f})
  観測:    α = 0.883  (β=0.600, γ=0.284)

■ 循環論法の影響:
  α の差: |旧-新| = {abs(0.801 - res['alpha']):.3f}
""")
    if abs(res['alpha'] - 0.801) &lt; 0.05:
        print("  → 循環論法の影響は小さい (&lt;0.05) 。結論は堅牢。")
    elif abs(res['alpha'] - 0.883) &lt; abs(0.801 - 0.883):
        print("  → 修正版の方が観測に近い。循環論法は α を歪めていた。")
```

```
    else:
        print("  → 修正版は観測から離れた。旧版の一致は偶然の打ち消し。")

    print(f"""
■ 修正後の T-A1 回答:

  α = β + γ
  β = {res['beta']:.3f}   (バリオンのサイズ-質量: h_R ∝ v^β、非循環)
  γ = {res['gamma']:.3f}   (MOND遷移の幾何学、非循環)
  α_theory = {res['alpha']:.3f}  vs  α_obs = 0.883

  V_peak(disk) ∝ v_flat^{p_Vp_disk[0]:.3f}   (非循環的実測)
  → C-1 循環論法は解消
""")
```

6. delta_gamma_decompose.py

項目	内容
フェーズ	Step 3 延長
目的	delta_gamma=0.172 の分解: ガス + g_c(v) + Ud(v) の効果
使用rs	非循環
結果	ガス追加で gamma が反転（+0.11 -> -0.05）。合成 Freeman モデルの限界が確定。
ステータス	確定（合成モデルでは alpha 導出不能）

解析目的

合成モデルにガス円盤、g_c の銀河依存性、Ud の v_flat 依存性を順に追加し、各効果が gamma にどれだけ寄与するかを分離。

ソースコード全文

```
#!/usr/bin/env python3
"""
Δγ = 0.172 の分解: 3つの効果を順に追加

Model 0: ディスクのみ, g_c=a0           (baseline, α=0.650)
Model 1: ディスク+ガス, g_c=a0           (+ガス効果)
Model 2: ディスク+ガス, g_c=g_c(v_flat)  (+g_c銀河依存)
Model 3: ディスク+ガス, g_c=g_c(v_flat), Υ_d(v_flat) で disk/gas比を変調

各ステップで α, β, γ を計算し、どの効果が Δγ にどれだけ寄与するか分離

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

import csv
import os
import numpy as np
from scipy.optimize import curve_fit
from scipy.special import i0, i1, k0, k1
from scipy.stats import pearsonr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 定数
# ============================================================
a0_SI = 1.2e-10  # m/s^2
a0_unit = 3703.0  # (km/s)^2 / kpc
kpc_to_m = 3.086e19

# ============================================================
# Part A: SPARC実測パラメータの取得
# ============================================================
print("="*70)
print("Part A: SPARC実測パラメータ")
print("="*70)

def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

src_cols, src_rows = load_csv("sparc_results.csv")
galaxies = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    ud = get_val(row, ['ud', 'upsilon_d'])
    vf = get_val(row, ['vflat', 'v_flat'])
    rs = get_val(row, ['rs1', 'r_s'])
    if ud and vf and rs and vf &gt; 0:
        galaxies[name] = {'upsilon_d': ud, 'vflat': vf, 'rs_tanh': rs}

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

# --- 各銀河から V_peak(disk), V_peak(gas), h_R(disk), h_R(gas), Υ_d を取得 ---
meas = []
for name, gal in galaxies.items():
    data = read_dat(name)
```

```
    if data is None:
        continue

    rad = data[:, 0]
    v_gas = data[:, 3]
    v_disk = data[:, 4]  # at Υ_d=1
    ud = gal['upsilon_d']

    if len(rad) &lt; 3:
        continue

    # ディスク (Υ_d 適用)
    vd_scaled = np.sqrt(ud) * np.abs(v_disk)
    Vpeak_disk = np.max(vd_scaled)
    i_pd = np.argmax(vd_scaled)
    r_peak_disk = rad[i_pd]

    # ガス (Υ_gas=1.33 が標準だが、SPARC は既に適用済み)
    vg = np.abs(v_gas)
    Vpeak_gas = np.max(vg) if np.max(vg) &gt; 0.1 else 0.0
    i_pg = np.argmax(vg) if Vpeak_gas &gt; 0 else 0
    r_peak_gas = rad[i_pg] if Vpeak_gas &gt; 0 else 0

    # h_R 推定 (V_disk ピークから)
    peak_at_edge = (r_peak_disk &gt;= rad.max() * 0.9)
    h_R_disk = r_peak_disk / 2.15 if not peak_at_edge else None

    # ガスのスケール長推定
    h_R_gas = r_peak_gas / 2.15 if Vpeak_gas &gt; 1.0 and r_peak_gas &gt; 0 else None

    # gas/disk 比
    gas_disk_ratio = Vpeak_gas / Vpeak_disk if Vpeak_disk &gt; 0 else 0

    meas.append({
        'name': name, 'vflat': gal['vflat'], 'upsilon_d': ud,
        'Vpeak_disk': Vpeak_disk, 'Vpeak_gas': Vpeak_gas,
        'r_peak_disk': r_peak_disk, 'r_peak_gas': r_peak_gas,
        'h_R_disk': h_R_disk, 'h_R_gas': h_R_gas,
        'gas_disk_ratio': gas_disk_ratio,
    })

print(f"実測成功: {len(meas)} 銀河")

# --- べき乗関係のフィット ---
vf_m = np.array([d['vflat'] for d in meas])
Vpd = np.array([d['Vpeak_disk'] for d in meas])
Vpg = np.array([d['Vpeak_gas'] for d in meas])
ud_m = np.array([d['upsilon_d'] for d in meas])
gdr = np.array([d['gas_disk_ratio'] for d in meas])

# V_peak(disk) vs v_flat
mask = (vf_m &gt; 0) &amp; (Vpd &gt; 0)
lv = np.log10(vf_m[mask])
p_Vpd = np.polyfit(lv, np.log10(Vpd[mask]), 1)
print(f"¥nV_peak(disk) ∝ v_flat^{p_Vpd[0]:.3f}")

# V_peak(gas) vs v_flat
mask_g = (vf_m &gt; 0) &amp; (Vpg &gt; 1.0)
if mask_g.sum() &gt; 10:
    p_Vpg = np.polyfit(np.log10(vf_m[mask_g]), np.log10(Vpg[mask_g]), 1)
    print(f"V_peak(gas) ∝ v_flat^{p_Vpg[0]:.3f}  (N={mask_g.sum()})")
else:
    p_Vpg = [0.3, 0.5]  # fallback
    print(f"V_peak(gas): データ不足、フォールバック使用")

# gas/disk ratio vs v_flat
mask_gd = (vf_m &gt; 0) &amp; (gdr &gt; 0.01)
if mask_gd.sum() &gt; 10:
    p_gdr = np.polyfit(np.log10(vf_m[mask_gd]), np.log10(gdr[mask_gd]), 1)
    r_gdr, _ = pearsonr(np.log10(vf_m[mask_gd]), np.log10(gdr[mask_gd]))
    print(f"gas/disk ratio ∝ v_flat^{p_gdr[0]:.3f}  (r={r_gdr:.3f})")
    print(f"  中央値: {np.median(gdr[mask_gd]):.3f}")
else:
    p_gdr = [-0.5, 0.5]
    r_gdr = 0
    print(f"gas/disk ratio: データ不足")

# Υ_d vs v_flat
mask_ud = (vf_m &gt; 0) &amp; (ud_m &gt; 0)
p_ud = np.polyfit(np.log10(vf_m[mask_ud]), np.log10(ud_m[mask_ud]), 1)
r_ud, _ = pearsonr(np.log10(vf_m[mask_ud]), np.log10(ud_m[mask_ud]))
print(f"Υ_d ∝ v_flat^{p_ud[0]:.3f}  (r={r_ud:.3f})")
print(f"  中央値: {np.median(ud_m):.3f}")

# h_R vs v_flat
hR_data = [(d['vflat'], d['h_R_disk']) for d in meas if d['h_R_disk'] is not None]
vf_hR = np.array([x[0] for x in hR_data])
hR_arr = np.array([x[1] for x in hR_data])
mask_hR = (vf_hR &gt; 0) &amp; (hR_arr &gt; 0)
p_hR = np.polyfit(np.log10(vf_hR[mask_hR]), np.log10(hR_arr[mask_hR]), 1)
print(f"h_R ∝ v_flat^{p_hR[0]:.3f}  (N={mask_hR.sum()})")

# ガスのスケール長比
hRg_data = [(d['h_R_gas'], d['h_R_disk']) for d in meas
```

```
            if d['h_R_gas'] is not None and d['h_R_disk'] is not None
            and d['h_R_gas'] &gt; 0 and d['h_R_disk'] &gt; 0]
if hRg_data:
    gas_scale_ratio = np.median([x[0]/x[1] for x in hRg_data])
    print(f"h_R(gas)/h_R(disk) 中央値: {gas_scale_ratio:.2f}  (N={len(hRg_data)})")
else:
    gas_scale_ratio = 2.0
    print(f"h_R(gas)/h_R(disk): データ不足、2.0 を使用")

# g_c(v_flat) from T-A3
# g_c/a0 ∝ v_flat^0.82
# 規格化: 中央値 0.825 at v_flat の中央値
vf_median = np.median(vf_m)
print(f"¥ng_c モデル: g_c/a0 = 0.825 × (v_flat/{vf_median:.0f})^0.82")


# ============================================================
# Part B: 合成銀河モデル (4段階)
# ============================================================
print(f"¥n{'='*70}")
print("Part B: 合成銀河モデル (4段階) ")
print(f"{'='*70}")

# Freeman 円盤
def freeman_bessel(y):
    y = np.clip(y, 1e-8, 50.0)
    return y**2 * (i0(y)*k0(y) - i1(y)*k1(y))

y_fine = np.linspace(0.01, 5.0, 5000)
vd2_shape = freeman_bessel(y_fine)
bessel_peak = np.max(vd2_shape)

def v_freeman(r, h_R, V_peak):
    y = r / (2.0 * h_R)
    y = np.clip(y, 1e-6, 50.0)
    v2 = V_peak**2 * freeman_bessel(y) / bessel_peak
    return np.sqrt(np.maximum(v2, 0.0))

def mond_gobs(gN, gc):
    return (gN + np.sqrt(gN**2 + 4*gc*gN)) / 2

def v_obs_from_mond(r_kpc, v_bar, g_c_SI):
    r_m = r_kpc * kpc_to_m
    v_bar_ms = v_bar * 1e3
    gN = np.where(r_m &gt; 0, v_bar_ms**2 / r_m, 0.0)
    go = mond_gobs(gN, g_c_SI)
    return np.sqrt(r_m * go) / 1e3

def fit_tanh(r, v_obs, v_bar):
    def model(r, vf, rs):
        T = 0.5 * (1.0 + np.tanh((r - rs) / rs))
        return np.sqrt(v_bar**2 + vf**2 * T)
    try:
        popt, _ = curve_fit(model, r, v_obs, p0=[v_obs[-1], r[len(r)//3]],
                            bounds=([0.1, 0.01], [500, r[-1]*3]), maxfev=5000)
        return popt[0], popt[1]
    except:
        return None, None


# --- パラメータ関数 ---
def get_hR(vf):
    return 10**np.polyval(p_hR, np.log10(vf))

def get_Vpeak_disk(vf):
    return 10**np.polyval(p_Vpd, np.log10(vf))

def get_Vpeak_gas(vf):
    return 10**np.polyval(p_Vpg, np.log10(vf))

def get_gas_disk_ratio(vf):
    return 10**np.polyval(p_gdr, np.log10(vf))

def get_Ud(vf):
    return 10**np.polyval(p_ud, np.log10(vf))

def get_gc(vf, model_type):
    """g_c [SI] を返す"""
    if model_type in ['gc_const', 'model0', 'model1']:
        return a0_SI
    else:
        # T-A3: g_c/a0 = 0.825 × (v/v_median)^0.82
        gc_a0 = 0.825 * (vf / vf_median)**0.82
        return gc_a0 * a0_SI


# --- 合成銀河の生成 ---
vflat_grid = np.logspace(np.log10(20), np.log10(300), 50)

model_configs = {
    'Model 0: disk only, g_c=a0': {
        'gas': False, 'gc_type': 'gc_const', 'ud_vary': False},
    'Model 1: disk+gas, g_c=a0': {
        'gas': True, 'gc_type': 'gc_const', 'ud_vary': False},
    'Model 2: disk+gas, g_c(v)': {
        'gas': True, 'gc_type': 'gc_vary', 'ud_vary': False},
```

```
    'Model 3: disk+gas, g_c(v), Υ_d(v)': {
        'gas': True, 'gc_type': 'gc_vary', 'ud_vary': True},
}

all_results = {}

for model_name, config in model_configs.items():
    rs_list, vf_list, hR_list = [], [], []

    for vf in vflat_grid:
        h_R = get_hR(vf)

        if config['ud_vary']:
            # Υ_d(v_flat) でディスクの V_peak を変調
            ud_ratio = get_Ud(vf) / np.median(ud_m)
            Vp_disk = get_Vpeak_disk(vf) * np.sqrt(np.clip(ud_ratio, 0.5, 3.0))
        else:
            Vp_disk = get_Vpeak_disk(vf)

        r = np.linspace(0.1*h_R, 15.0*h_R, 400)

        # ディスク成分
        v_disk = v_freeman(r, h_R, Vp_disk)

        # ガス成分
        if config['gas']:
            h_R_gas = gas_scale_ratio * h_R
            Vp_gas = get_Vpeak_gas(vf)
            # ガスの V_peak が非現実的に大きくならないようクリップ
            Vp_gas = min(Vp_gas, Vp_disk * 2.0)
            v_gas = v_freeman(r, h_R_gas, Vp_gas)
            v_bar = np.sqrt(v_disk**2 + v_gas**2)
        else:
            v_bar = v_disk

        # g_c
        gc = get_gc(vf, config['gc_type'])

        # MOND による v_obs
        v_obs = v_obs_from_mond(r, v_bar, gc)

        # tanh フィット
        vf_fit, rs_fit = fit_tanh(r, v_obs, v_bar)
        if vf_fit is None or rs_fit &lt;= 0:
            continue

        rs_list.append(rs_fit)
        vf_list.append(vf)
        hR_list.append(h_R)

    rs_a = np.array(rs_list)
    vf_a = np.array(vf_list)
    hR_a = np.array(hR_list)
    N = len(rs_a)

    if N &lt; 5:
        print(f"  {model_name}: N={N} (不足)")
        continue
    lv = np.log10(vf_a)
    lrs = np.log10(rs_a)
    lhR = np.log10(hR_a)
    lrh = np.log10(rs_a/hR_a)

    p_a = np.polyfit(lv, lrs, 1)
    p_b = np.polyfit(lv, lhR, 1)
    p_g = np.polyfit(lv, lrh, 1)

    rpeak = 2.15 * hR_a
    rs_rp = rs_a / rpeak

    all_results[model_name] = {
        'alpha': p_a[0], 'beta': p_b[0], 'gamma': p_g[0],
        'rs_rpeak_median': np.median(rs_rp),
        'vf': vf_a, 'rs': rs_a, 'hR': hR_a, 'N': N,
    }

    print(f"¥n{model_name} (N={N}):")
    print(f"  α={p_a[0]:.4f}  β={p_b[0]:.4f}  γ={p_g[0]:.4f}  r_s/r_peak={np.median(rs_rp):.3f}")

# ============================================================
# Part C: 差分解析
# ============================================================
print(f"¥n{'='*70}")
print("Δγ の分解")
print(f"{'='*70}")

model_names = list(all_results.keys())
print(f"¥n{'モデル':&gt;40s}  {'α':&gt;7s}  {'β':&gt;7s}  {'γ':&gt;7s}  {'Δα':&gt;7s}  {'Δγ':&gt;7s}")
print(f"{'観測':&gt;40s}  {'0.883':&gt;7s}  {'0.600':&gt;7s}  {'0.284':&gt;7s}  {'---':&gt;7s}  {'---':&gt;7s}")

prev_alpha = None
prev_gamma = None
for mn in model_names:
    res = all_results[mn]
    if prev_alpha is None:
```

```
        da = '---'
        dg = '---'
    else:
        da = f"{res['alpha']-prev_alpha:+.4f}"
        dg = f"{res['gamma']-prev_gamma:+.4f}"
    print(f"{mn:&gt;40s}  {res['alpha']:7.4f}  {res['beta']:7.4f}  {res['gamma']:7.4f}  {da:&gt;7s}  {dg:&gt;7s}")
    prev_alpha = res['alpha']
    prev_gamma = res['gamma']

# 寄与の計算
if len(all_results) &gt;= 4:
    keys = list(all_results.keys())
    g0 = all_results[keys[0]]['gamma']
    g1 = all_results[keys[1]]['gamma']
    g2 = all_results[keys[2]]['gamma']
    g3 = all_results[keys[3]]['gamma']
    g_obs = 0.284

    print(f"¥n--- γ の増分 ---")
    print(f"  ガス効果:     Δγ = {g1-g0:+.4f}")
    print(f"  g_c(v)効果:   Δγ = {g2-g1:+.4f}")
    print(f"  Υ_d(v)効果:   Δγ = {g3-g2:+.4f}")
    print(f"  合計追加:     Δγ = {g3-g0:+.4f}")
    print(f"  全体の差:     γ_obs - γ_Model3 = {g_obs-g3:+.4f}")
    print(f"  充填率:       {(g3-g0)/(g_obs-g0)*100:.1f}%  ({g_obs-g0:.4f} のうち {g3-g0:.4f} を説明)")

# ============================================================
# Part D: 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))
colors_m = ['gray', 'blue', 'green', 'red']

# (a) α 分解の棒グラフ
ax = axes[0, 0]
labels = ['Obs'] + [f'M{i}' for i in range(len(all_results))]
betas = [0.600] + [all_results[k]['beta'] for k in all_results]
gammas = [0.284] + [all_results[k]['gamma'] for k in all_results]
x = np.arange(len(labels))
ax.bar(x, betas, 0.4, label='β (baryonic)', color='steelblue')
ax.bar(x, gammas, 0.4, bottom=betas, label='γ (membrane/MOND)', color='coral')
for i in range(len(labels)):
    ax.text(i, betas[i]+gammas[i]+0.01, f'{betas[i]+gammas[i]:.3f}',
            ha='center', fontsize=8)
ax.set_xticks(x)
ax.set_xticklabels(labels, fontsize=8)
ax.set_ylabel('Exponent')
ax.set_title('(a) α = β + γ decomposition')
ax.legend(fontsize=8)

# (b) r_s vs v_flat (全モデル)
ax = axes[0, 1]
for i, (mn, res) in enumerate(all_results.items()):
    ax.plot(np.log10(res['vf']), np.log10(res['rs']),
            color=colors_m[i % len(colors_m)],
            label=f'M{i}: α={res["alpha"]:.3f}')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(r_s)')
ax.set_title('(b) r_s scaling by model')
ax.legend(fontsize=7)

# (c) r_s/h_R vs v_flat (全モデル)
ax = axes[0, 2]
for i, (mn, res) in enumerate(all_results.items()):
    lrh = np.log10(res['rs']/res['hR'])
    ax.plot(np.log10(res['vf']), lrh,
            color=colors_m[i % len(colors_m)],
            label=f'M{i}: γ={res["gamma"]:.3f}')
ax.axhline(np.log10(2.11), color='black', ls='--', alpha=0.5, label='Obs=2.11')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(r_s/h_R)')
ax.set_title('(c) γ by model')
ax.legend(fontsize=7)

# (d) gas/disk ratio vs v_flat
ax = axes[1, 0]
ax.scatter(vf_m[mask_gd], gdr[mask_gd], s=8, alpha=0.3, c='teal')
if mask_gd.sum() &gt; 10:
    vv = np.logspace(np.log10(20), np.log10(300), 100)
    ax.plot(vv, 10**np.polyval(p_gdr, np.log10(vv)), 'k--',
            label=f'∝v^{p_gdr[0]:.2f}')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('V_peak(gas) / V_peak(disk)')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title('(d) Gas/disk ratio')
ax.legend(fontsize=8)

# (e) Υ_d vs v_flat
ax = axes[1, 1]
ax.scatter(vf_m, ud_m, s=8, alpha=0.3, c='orange')
vv = np.logspace(np.log10(vf_m.min()), np.log10(vf_m.max()), 100)
ax.plot(vv, 10**np.polyval(p_ud, np.log10(vv)), 'k--',
        label=f'Υ_d ∝ v^{p_ud[0]:.2f}')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('Υ_d')
```

```
ax.set_xscale('log')
ax.set_title(f'(e) Υ_d vs v_flat (r={r_ud:.3f})')
ax.legend(fontsize=8)

# (f) Δγ の累積寄与
ax = axes[1, 2]
if len(all_results) &gt;= 4:
    keys = list(all_results.keys())
    steps = ['Baseline¥n(disk)', '+Gas', '+g_c(v)', '+Υ_d(v)', 'Observed']
    gamma_vals = [all_results[keys[0]]['gamma'],
                  all_results[keys[1]]['gamma'],
                  all_results[keys[2]]['gamma'],
                  all_results[keys[3]]['gamma'],
                  0.284]
    ax.plot(range(len(steps)), gamma_vals, 'bo-', markersize=8)
    ax.axhline(0.284, color='red', ls='--', label='γ_obs=0.284')
    ax.set_xticks(range(len(steps)))
    ax.set_xticklabels(steps, fontsize=8)
    ax.set_ylabel('γ')
    ax.set_title('(f) Cumulative γ contribution')
    ax.legend(fontsize=8)
    ax.set_ylim(0, 0.35)

plt.tight_layout()
plt.savefig('delta_gamma_decompose.png', dpi=150)
print(f"¥n[SAVED] delta_gamma_decompose.png")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("最終結論")
print(f"{'='*70}")

if len(all_results) &gt;= 4:
    keys = list(all_results.keys())
    print(f"""
■ Δγ = 0.172 の分解結果:

  γ_obs = 0.284 (観測)
  γ_M0  = {all_results[keys[0]]['gamma']:.4f} (ディスクのみ、g_c=a0)

  ガス効果:     +{all_results[keys[1]]['gamma']-all_results[keys[0]]['gamma']:.4f}
  g_c(v)効果:   +{all_results[keys[2]]['gamma']-all_results[keys[1]]['gamma']:.4f}
  Υ_d(v)効果:   +{all_results[keys[3]]['gamma']-all_results[keys[2]]['gamma']:.4f}
  ──────────────────────────
  合計説明:     +{all_results[keys[3]]['gamma']-all_results[keys[0]]['gamma']:.4f}
  残差:          {0.284-all_results[keys[3]]['gamma']:+.4f}

  γ_M3 = {all_results[keys[3]]['gamma']:.4f} (全効果込み)
  α_M3 = {all_results[keys[3]]['alpha']:.4f} (cf. α_obs = 0.883)

■ 充填率: {(all_results[keys[3]]['gamma']-all_results[keys[0]]['gamma'])/(0.284-all_results[keys[0]]['gamma'])*100:.0f}%

■ 判定:
""")
    fill = (all_results[keys[3]]['gamma']-all_results[keys[0]]['gamma'])/(0.284-all_results[keys[0]]['gamma'])
    residual = 0.284 - all_results[keys[3]]['gamma']

    if fill &gt; 0.8:
        print("  → Δγ の 80%以上が説明可能。v3.0 + バリオン物理で α をほぼ導出できる。")
    elif fill &gt; 0.5:
        print("  → Δγ の半分以上は既知の効果で説明。残差は合成モデルの精度限界内。")
    else:
        print(f"  → Δγ の {fill*100:.0f}% のみ説明。残差 {residual:.3f} は")
        print("    合成モデルの限界（単純化）か、v3.0 外の物理の寄与を示唆。")
```

7. TA2_rs_ratio.py

項目	内容
フェーズ	T-A2
目的	Freeman 幾何学から r_s/r_T3 の理論的説明
使用rs	なし（理論計算）
結果	定性的説明は成功。定量的再現は不完全（ガス成分省略のため）。
ステータス	Level B（定性的に確立）

解析目的

Freeman 指数円盤の g_N(r) ピーク位置と V_disk ピーク位置の比から、r_s_tanh/r_s^(T3) ~= 2.75 を理論的に説明する。

ソースコード全文

```
#!/usr/bin/env python3
"""
T-A2: r_s_tanh / r_s^(T3) ≈ 2.75 の理論的説明

Step 1-3 の結果:
  r_s_tanh ≈ r_peak = 2.15 × h_R (V_disk のピーク半径)

T-3 の定義:
  r_s^(T3) = g_N(r) = g_c を満たす半径 (g_c = a0 のとき)

→ 問題の帰着:
  r_peak / r_{g_N=a0} ≈ 2.75 を Freeman 円盤から導出する

実行: uv run --with scipy --with matplotlib --with numpy python TA2_rs_ratio.py
"""

import numpy as np
from scipy.special import i0, i1, k0, k1
from scipy.optimize import brentq
from scipy.stats import spearmanr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 物理定数
# ============================================================
a0 = 1.2e-10  # m/s^2
kpc_to_m = 3.086e19
G_SI = 6.674e-11  # m^3 kg^-1 s^-2
Msun = 1.989e30   # kg

# ============================================================
# 1. Freeman 指数円盤の解析的計算
# ============================================================

def freeman_bessel(y):
    """y^2 * [I0(y)K0(y) - I1(y)K1(y)]  (V_disk^2 の形状関数)"""
    y = np.clip(y, 1e-8, 50.0)
    return y**2 * (i0(y)*k0(y) - i1(y)*k1(y))


def freeman_gN_shape(y):
    """
    g_N(r) ∝ v^2/r ∝ y * [I0K0 - I1K1]  (加速度の形状関数)
    y = r / (2*h_R)
    """
    y = np.clip(y, 1e-8, 50.0)
    return y * (i0(y)*k0(y) - i1(y)*k1(y))


# --- 形状関数の特性点を計算 ---
y_fine = np.linspace(0.01, 5.0, 5000)
vdisk2_shape = freeman_bessel(y_fine)
gN_shape = freeman_gN_shape(y_fine)

# V_disk ピーク
i_vpeak = np.argmax(vdisk2_shape)
y_vpeak = y_fine[i_vpeak]
r_peak_over_hR = 2.0 * y_vpeak  # r_peak / h_R

# g_N ピーク
i_gpeak = np.argmax(gN_shape)
y_gpeak = y_fine[i_gpeak]
r_gNpeak_over_hR = 2.0 * y_gpeak

print("="*70)
print("Freeman 指数円盤の特性点")
print("="*70)
print(f"V_disk ピーク:  y = {y_vpeak:.4f}, r/h_R = {r_peak_over_hR:.4f}")
print(f"g_N ピーク:     y = {y_gpeak:.4f}, r/h_R = {r_gNpeak_over_hR:.4f}")
```

```
print(f"g_N ピーク / V_disk ピーク = {r_gNpeak_over_hR / r_peak_over_hR:.4f}")

# g_N のピーク値（規格化）
gN_shape_peak = gN_shape[i_gpeak]
vdisk2_shape_peak = vdisk2_shape[i_vpeak]

# ============================================================
# 2. g_N = a0 の交差半径を銀河パラメータの関数として計算
# ============================================================
def find_gN_crossing(Sigma0, h_R_kpc, g_c=a0):
    """
    Freeman 円盤で g_N(r) = g_c となる半径 r [kpc] を見つける

    g_N(r) = 2*pi*G*Sigma0 * freeman_gN_shape(y)
    ここで y = r/(2*h_R)

    Sigma0: 中心面密度 [kg/m^2]
    h_R_kpc: ディスクスケール長 [kpc]
    """
    h_R_m = h_R_kpc * kpc_to_m

    # g_N の振幅
    gN_amplitude = 2 * np.pi * G_SI * Sigma0  # [m/s^2] × shape function

    # g_N のピーク値
    gN_max = gN_amplitude * gN_shape_peak

    if gN_max &lt; g_c:
        return None, gN_max  # 交差なし（全域 g_N &lt; g_c）

    # 交差点を探す（g_N ピークの外側）
    def gN_minus_gc(y):
        return gN_amplitude * freeman_gN_shape(y) - g_c

    # ピーク外側での交差
    try:
        y_cross = brentq(gN_minus_gc, y_gpeak, 10.0)
        r_cross_kpc = 2.0 * y_cross * h_R_kpc
        return r_cross_kpc, gN_max
    except ValueError:
        return None, gN_max


# ============================================================
# 3. 系統的計算：Sigma0 を変えて比率を求める
# ============================================================

print(f"¥n{'='*70}")
print("Sigma0 を変えた系統的計算")
print(f"{'='*70}")

# 典型的な h_R = 2.5 kpc で固定し、Sigma0 を変える
h_R_fixed = 2.5  # kpc

# Sigma0 の範囲（M_sun/pc^2 → kg/m^2 に変換）
# 低面輝度: 10 M_sun/pc^2, 高面輝度: 2000 M_sun/pc^2
# Freeman値: ~140 M_sun/pc^2
Msun_per_pc2_to_kg_per_m2 = Msun / (3.086e16)**2

Sigma0_range_Msunpc2 = np.logspace(np.log10(30), np.log10(3000), 100)
Sigma0_range = Sigma0_range_Msunpc2 * Msun_per_pc2_to_kg_per_m2

ratios = []
gN_maxes = []
Sigma0_valid = []
r_cross_list = []

for S0, S0_Msun in zip(Sigma0_range, Sigma0_range_Msunpc2):
    r_cross, gN_max = find_gN_crossing(S0, h_R_fixed)
    gN_maxes.append(gN_max / a0)
    if r_cross is not None:
        r_peak = r_peak_over_hR * h_R_fixed
        ratio = r_peak / r_cross
        ratios.append(ratio)
        Sigma0_valid.append(S0_Msun)
        r_cross_list.append(r_cross)

ratios = np.array(ratios)
Sigma0_valid = np.array(Sigma0_valid)
r_cross_list = np.array(r_cross_list)
gN_maxes = np.array(gN_maxes)

print(f"¥n交差あり: {len(ratios)} / {len(Sigma0_range)}")
print(f"  Sigma0 範囲: {Sigma0_valid.min():.0f} - {Sigma0_valid.max():.0f} M_sun/pc^2")
print(f"¥nr_peak / r_{'{g_N=a0}'} の統計:")
print(f"  中央値: {np.median(ratios):.4f}")
print(f"  平均値: {np.mean(ratios):.4f}")
print(f"  範囲:   {ratios.min():.3f} - {ratios.max():.3f}")
print(f"¥n--- Sigma0 依存性の詳細 ---")
print(f"{'Sigma0 [M/pc2]':&gt;16s}  {'g_N_max/a0':&gt;10s}  {'r_cross [kpc]':&gt;13s}  {'r_peak/r_cross':&gt;14s}")
for S0_show in [50, 100, 150, 200, 500, 1000, 2000]:
    idx = np.argmin(np.abs(Sigma0_valid - S0_show))
    if abs(Sigma0_valid[idx] - S0_show) &lt; S0_show * 0.3:
        print(f"{Sigma0_valid[idx]:16.0f}  {gN_maxes[np.argmin(np.abs(Sigma0_range_Msunpc2 - S0_show))]:10.2f}  {r_cross_list[idx]:13.3f}  {ratios[idx]:14.4f}")
```

```
# ============================================================
# 4. h_R 依存性の確認
# ============================================================
print(f"¥n{'='*70}")
print("h_R 依存性の確認 (Sigma0 = 200 M_sun/pc^2 固定) ")
print(f"{'='*70}")

S0_fixed = 200 * Msun_per_pc2_to_kg_per_m2
hR_range = np.linspace(0.5, 8.0, 50)
ratios_hR = []
hR_valid = []

for hR in hR_range:
    r_cross, gN_max = find_gN_crossing(S0_fixed, hR)
    if r_cross is not None:
        r_peak = r_peak_over_hR * hR
        ratios_hR.append(r_peak / r_cross)
        hR_valid.append(hR)

ratios_hR = np.array(ratios_hR)
hR_valid = np.array(hR_valid)

if len(ratios_hR) &gt; 0:
    print(f"  中央値: {np.median(ratios_hR):.4f}")
    print(f"  範囲:   {ratios_hR.min():.3f} - {ratios_hR.max():.3f}")
    print(f"  → h_R には依存しない (比率は形状関数のみで決まる) ")

# ============================================================
# 5. 解析的導出
# ============================================================
print(f"¥n{'='*70}")
print("解析的導出")
print(f"{'='*70}")

print("""
■ Freeman 円盤の g_N(r):

  g_N(r) = 2πGΣ₀ · φ(y)
  φ(y) = y · [I₀(y)K₀(y) - I₁(y)K₁(y)]
  y = r / (2h_R)

■ 特性点:

  g_N ピーク: y_g ≈ {y_g:.3f}  →  r_g = {r_g:.3f} × h_R
  V_disk ピーク: y_v ≈ {y_v:.3f}  →  r_v = {r_v:.3f} × h_R

■ 比率 r_peak / r_{{g_N=a0}}:

  r_peak は Σ₀ に依存しない (形状関数のピーク = 固定)
  r_{{g_N=a0}} は Σ₀ に依存する (振幅が変わると交差点が移動)

  高Σ₀: g_N_max &gt;&gt; a0 → 交差点は外側 → r_peak/r_cross は小 (→1に近づく)
  低Σ₀: g_N_max ≈ a0 → 交差点はピーク近傍 → r_peak/r_cross は大

  → 比率は Σ₀ によって変化する。「2.75」は特定の Σ₀ 分布の中央値。
""".format(y_g=y_gpeak, r_g=r_gNpeak_over_hR, y_v=y_vpeak, r_v=r_peak_over_hR))

# ============================================================
# 6. SPARC N=46 銀河の Sigma0 分布から 2.75 を再現
# ============================================================
print(f"¥n{'='*70}")
print("N=46 銀河の選択バイアスの影響")
print(f"{'='*70}")

# T-3 r_s が存在するのは中~大質量銀河 (v_flat &gt; 100 km/s)
# これらの典型的な Sigma0 は？
# v_flat &gt; 100 の銀河は Sigma0 ≈ 100-500 M_sun/pc^2

# Step 1 の f(v_flat) U字型から推定:
# v_flat=100-200 では f ≈ 0.2-0.55
# V_peak = sqrt(f) * v_flat ≈ 45-150 km/s
# Sigma0 ∝ V_peak^2 * h_R / (2πG) → 具体的に計算
print("¥nv_flat 別の理論的比率:")
print(f"{'v_flat':&gt;8s}  {'h_R':&gt;6s}  {'V_peak':&gt;8s}  {'Sigma0_eff':&gt;10s}  {'r_peak/r_cross':&gt;14s}  {'r_cross':&gt;8s}")

vf_test = [80, 100, 120, 150, 200, 250]
ratio_for_vf = []

for vf in vf_test:
    h_R = 2.5 * (vf / 100.0)**0.6

    # f(v_flat) from Step 1
    lv = np.log10(vf)
    log_f = 5.80 - 6.89*lv + 1.84*lv**2
    f = 10**log_f
    f = np.clip(f, 0.05, 2.0)
    V_peak = np.sqrt(f) * vf  # km/s

    # V_peak^2 = 4πGΣ₀h_R × φ(y_peak) から Sigma0 を逆算
    V_peak_ms = V_peak * 1e3
    h_R_m = h_R * kpc_to_m
    phi_peak = freeman_bessel(y_vpeak)
    Sigma0 = V_peak_ms**2 / (4 * np.pi * G_SI * h_R_m * phi_peak)
    Sigma0_Msunpc2 = Sigma0 / Msun_per_pc2_to_kg_per_m2
```

```
    r_cross, gN_max = find_gN_crossing(Sigma0, h_R)
    if r_cross is not None:
        r_peak = r_peak_over_hR * h_R
        ratio = r_peak / r_cross
        ratio_for_vf.append(ratio)
        print(f"{vf:8.0f}  {h_R:6.2f}  {V_peak:8.1f}  {Sigma0_Msunpc2:10.0f}  {ratio:14.4f}  {r_cross:8.3f}")
    else:
        print(f"{vf:8.0f}  {h_R:6.2f}  {V_peak:8.1f}  {Sigma0_Msunpc2:10.0f}  {'交差なし':&gt;14s}  {'---':&gt;8s}")

if ratio_for_vf:
    print(f"¥n  v_flat=80-250 での中央値: {np.median(ratio_for_vf):.4f}")
    print(f"  観測値 (N=46) : 2.75")

# ============================================================
# 7. 逆問題: 2.75 を再現する Sigma0 を求める
# ============================================================
print(f"¥n{'='*70}")
print("逆問題: r_peak/r_cross = 2.75 を再現する条件")
print(f"{'='*70}")

target_ratio = 2.75

# Sigma0 を探索して比率=2.75になる値を見つける
def ratio_minus_target(log_S0):
    S0 = 10**log_S0 * Msun_per_pc2_to_kg_per_m2
    r_cross, _ = find_gN_crossing(S0, h_R_fixed)
    if r_cross is None:
        return -target_ratio  # 交差なし
    r_peak = r_peak_over_hR * h_R_fixed
    return r_peak / r_cross - target_ratio

try:
    log_S0_solution = brentq(ratio_minus_target, np.log10(30), np.log10(3000))
    S0_solution = 10**log_S0_solution
    print(f"  r_peak/r_cross = 2.75 のとき Sigma0 = {S0_solution:.1f} M_sun/pc^2")

    # この Sigma0 でのg_N_max/a0
    S0_SI = S0_solution * Msun_per_pc2_to_kg_per_m2
    gN_max_check = 2 * np.pi * G_SI * S0_SI * gN_shape_peak
    print(f"  g_N_max / a0 = {gN_max_check/a0:.3f}")
    print(f"  → g_N がギリギリ a0 を超える銀河（選択効果の境界付近）")
except Exception as e:
    print(f"  解なし: {e}")

# ============================================================
# 8. 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) Freeman 円盤の形状関数
ax = axes[0, 0]
y_plot = np.linspace(0.01, 4.0, 500)
ax.plot(y_plot*2, freeman_bessel(y_plot)/vdisk2_shape_peak, 'b-', label='V²_disk (normalized)')
ax.plot(y_plot*2, freeman_gN_shape(y_plot)/gN_shape_peak, 'r-', label='g_N (normalized)')
ax.axvline(r_peak_over_hR, color='blue', ls='--', alpha=0.5, label=f'r_peak={r_peak_over_hR:.2f}h_R')
ax.axvline(r_gNpeak_over_hR, color='red', ls='--', alpha=0.5, label=f'r_gNpeak={r_gNpeak_over_hR:.2f}h_R')
ax.set_xlabel('r / h_R')
ax.set_ylabel('Normalized amplitude')
ax.set_title('(a) Freeman disk shape functions')
ax.legend(fontsize=7)

# (b) g_N(r) プロファイル (異なるΣ□)
ax = axes[0, 1]
for S0_show, col, ls in [(50, 'blue', '-'), (150, 'green', '-'),
                         (500, 'orange', '-'), (1500, 'red', '-')]:
    S0_SI = S0_show * Msun_per_pc2_to_kg_per_m2
    gN = 2 * np.pi * G_SI * S0_SI * freeman_gN_shape(y_plot)
    ax.plot(y_plot*2, gN/a0, color=col, ls=ls, label=f'Σ□={S0_show}')
ax.axhline(1.0, color='black', ls='--', label='g_N = a0')
ax.axvline(r_peak_over_hR, color='gray', ls=':', alpha=0.5)
ax.set_xlabel('r / h_R')
ax.set_ylabel('g_N / a0')
ax.set_title('(b) g_N profiles')
ax.set_yscale('log')
ax.set_ylim(0.01, 100)
ax.legend(fontsize=7)

# (c) r_peak/r_cross vs Sigma0
ax = axes[0, 2]
ax.plot(Sigma0_valid, ratios, 'b-', linewidth=2)
ax.axhline(2.75, color='red', ls='--', label='Observed = 2.75')
ax.axhline(1.0, color='gray', ls=':', alpha=0.3)
ax.set_xlabel('Σ□ [M_sun/pc²]')
ax.set_ylabel('r_peak / r_{g_N=a0}')
ax.set_title('(c) Ratio vs surface density')
ax.set_xscale('log')
ax.legend(fontsize=8)
ax.set_ylim(0.5, 6)

# (d) 同上、g_N_max/a0 で横軸
ax = axes[1, 0]
# g_N_max/a0 と ratio の対応
gN_max_valid = []
for S0_Msun in Sigma0_valid:
    S0_SI = S0_Msun * Msun_per_pc2_to_kg_per_m2
```

```
    gN_max_valid.append(2 * np.pi * G_SI * S0_SI * gN_shape_peak / a0)
gN_max_valid = np.array(gN_max_valid)
ax.plot(gN_max_valid, ratios, 'b-', linewidth=2)
ax.axhline(2.75, color='red', ls='--', label='Observed = 2.75')
ax.set_xlabel('g_N_max / a0')
ax.set_ylabel('r_peak / r_{g_N=a0}')
ax.set_title('(d) Ratio vs g_N_max/a0')
ax.set_xscale('log')
ax.legend(fontsize=8)
ax.set_ylim(0.5, 6)

# (e) v_flat vs 理論的比率
ax = axes[1, 1]
if ratio_for_vf:
    ax.plot(vf_test[:len(ratio_for_vf)], ratio_for_vf, 'bo-', markersize=8)
    ax.axhline(2.75, color='red', ls='--', label='Observed = 2.75')
    ax.set_xlabel('v_flat [km/s]')
    ax.set_ylabel('r_peak / r_{g_N=a0}')
    ax.set_title('(e) Ratio vs v_flat (theory)')
    ax.legend(fontsize=8)

# (f) h_R 依存性
ax = axes[1, 2]
if len(ratios_hR) &gt; 0:
    ax.plot(hR_valid, ratios_hR, 'g-', linewidth=2)
    ax.axhline(2.75, color='red', ls='--', label='Observed = 2.75')
    ax.set_xlabel('h_R [kpc]')
    ax.set_ylabel('r_peak / r_{g_N=a0}')
    ax.set_title(f'(f) Ratio vs h_R (Σ□=200, median={np.median(ratios_hR):.2f})')
    ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('TA2_rs_ratio.png', dpi=150)
print(f"¥n[SAVED] TA2_rs_ratio.png")

# ============================================================
# 9. 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("T-A2 最終結論")
print(f"{'='*70}")
print(f"""
■ 問い: なぜ r_s_tanh / r_s^(T3) ≈ 2.75 か

■ 回答:

  r_s_tanh ≈ r_peak = {r_peak_over_hR:.2f} × h_R   (V_disk のピーク)
  r_s^(T3) = r_{{g_N=a0}}                          (g_N が a0 に等しくなる半径)

  Freeman 円盤では:
    g_N(r) のピークは r ≈ {r_gNpeak_over_hR:.2f} × h_R (V_disk ピークより内側)
    g_N = a0 の交差点は g_N ピークと V_disk ピークの間に来る
    → r_{{g_N=a0}} &lt; r_peak は幾何学的必然

  比率 r_peak / r_{{g_N=a0}} は Σ□ (中心面密度) に依存する:
    高Σ□: g_N_max &gt;&gt; a0 → 交差点は外側 → 比率は小 (→1)
    低Σ□: g_N_max ≈ a0 → 交差点はピーク近傍 → 比率は大

  N=46 銀河 (r_s^(T3)が存在する＝g_N_max &gt; a0 の中~大質量銀河) は
  g_N_max/a0 ≈ 1-10 の範囲にあり、この範囲での中央値が ≈ 2.75。

  → 「2.75」は Freeman 円盤の形状関数 + N=46 の選択バイアスから決まる
  → 膜固有の物理は関与しない
  → v3.0 の追加の説明は不要

■ 確立レベル:
  定性的説明 (r_{{g_N=a0}} &lt; r_peak は必然) → Level A
  定量的一致 (≈ 2.75 の数値再現) → Σ□ 分布に依存、Level B

■ 副産物:
  比率は h_R に依存しない (形状関数の比のみで決まる)
  比率は Σ□ (≈ 銀河の面輝度) で単調に変化する
  → 高面輝度銀河ほど r_s_tanh/r_s^(T3) は小さいはず (検証可能な予言)
""")
```

8. TA2_direct.py

項目	内容
フェーズ	T-A2（直接計算）
目的	SPARC 実データで r_peak, r_{g_N=a0}, r_s の3特性半径を比較
使用rs	rs2（V-1で確認）
結果	r_s/r_T3 ~= 2.13。r_peak/r_T3 vs g_N_max/a0: rho=-0.918。
ステータス	Level B（rs2依存）

解析目的

前スクリプトの合成モデル限界を克服し、.dat の実データから直接3つの特性半径を計算。比率の g_N_max/a0 依存性を調べる。

ソースコード全文

```
#!/usr/bin/env python3
"""
T-A2 改訂版: SPARC実データで r_s_tanh / r_s^(T3) ≈ 2.75 を直接検証

Freeman合成モデルではガス成分不足で定量再現できなかった。
→ 実データの v_bar(r)（ディスク+ガス+バルジ）から直接計算する。

3つの特性半径:
  r_s_tanh:   tanh フィットの遷移半径（sparc_results.csv から）
  r_peak:     v_bar(r) のピーク半径（.dat から）
  r_{g_N=a0}: g_N(r) = v_bar²(r)/r = a0 となる半径（.dat から）

実行: uv run --with scipy --with matplotlib --with numpy python TA2_direct.py
"""

import csv
import os
import numpy as np
from scipy.interpolate import interp1d
from scipy.stats import spearmanr, pearsonr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 定数
# ============================================================
a0_kms2_kpc = 3.8e-4  # a0 in (km/s)^2 / kpc  [1.2e-10 m/s^2 変換]
# 確認: a0 = 1.2e-10 m/s^2 = 1.2e-10 * (3.086e19 kpc/m) / (1e3 m/s / km/s)^2
#      = 1.2e-10 * 3.086e19 / 1e6 = 1.2e-10 * 3.086e13 = 3.70e3 ...
# 正確に: a0 [km^2/s^2/kpc] = 1.2e-10 [m/s^2] * 3.086e19 [m/kpc] / 1e6 [(m/s)^2/(km/s)^2]
#        = 1.2e-10 * 3.086e19 / 1e6 = 1.2 * 3.086e3 = 3703.2
# g_N = v^2/r の単位: v[km/s], r[kpc] → g_N [(km/s)^2/kpc]
# a0 = 3703 (km/s)^2/kpc ... これは大きすぎる。確認。

# a0 = 1.2e-10 m/s^2
# 1 kpc = 3.086e19 m
# v^2/r: (km/s)^2 / kpc = (1e3 m/s)^2 / (3.086e19 m) = 1e6/3.086e19 m/s^2 = 3.24e-14 m/s^2
# したがって a0 / (3.24e-14) = 1.2e-10/3.24e-14 = 3703 (km/s)^2/kpc
a0_unit = 3703.0  # a0 in (km/s)^2 / kpc

# ============================================================
# データ読み込み
# ============================================================
def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

# --- sparc_results.csv ---
if not os.path.exists("sparc_results.csv"):
    print("[ERROR] sparc_results.csv not found")
    exit(1)
src_cols, src_rows = load_csv("sparc_results.csv")
print(f"[INFO] sparc_results.csv: {len(src_rows)} rows, columns: {src_cols}")

# 辞書化
def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def get_str(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            return row[c].strip()
    return None

galaxies = {}
```

```
for row in src_rows:
    name = get_str(row, ['galaxy', 'Galaxy', 'name', 'Name', 'GALAXY'])
    if not name:
        name = list(row.values())[0].strip()
    rs = get_val(row, ['r_s', 'rs', 'r_s_tanh', 'rs_tanh'])
    vf = get_val(row, ['v_flat', 'vflat', 'Vflat'])
    ud = get_val(row, ['upsilon_d', 'Upsilon_d', 'Ud'])
    if rs and vf and ud and rs &gt; 0 and vf &gt; 0:
        galaxies[name] = {'rs_tanh': rs, 'vflat': vf, 'upsilon_d': ud}
print(f"[INFO] Valid galaxies: {len(galaxies)}")

# ============================================================
# .dat ファイルから特性半径を計算
# ============================================================

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            data = np.loadtxt(fn, comments='#')
            return data
    return None


def compute_radii(data, upsilon_d):
    """
    .dat ファイルから3つの特性半径を計算:
      r_peak:     |v_bar(r)| のピーク半径
      r_gN_a0:    g_N(r) = a0 となる最外半径（ピーク外側）
      g_N_max:    g_N の最大値
    """
    rad = data[:, 0]      # kpc
    v_gas = data[:, 3]    # km/s
    v_disk = data[:, 4]   # km/s (at Υ=1)
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)

    # v_bar^2 (符号保持)
    vbar2 = upsilon_d * np.sign(v_disk) * v_disk**2 + ¥
            np.sign(v_gas) * v_gas**2 + ¥
            np.sign(v_bul) * v_bul**2
    vbar = np.sqrt(np.maximum(vbar2, 0.0))

    if len(rad) &lt; 3 or rad.max() &lt; 0.1:
        return None

    # --- r_peak: v_bar のピーク ---
    # スプライン補間
    try:
        r_fine = np.linspace(rad.min(), rad.max(), 1000)
        vbar_interp = interp1d(rad, vbar, kind='linear', fill_value=0, bounds_error=False)
        vbar_fine = vbar_interp(r_fine)
        i_peak = np.argmax(vbar_fine)
        r_peak = r_fine[i_peak]
        v_peak = vbar_fine[i_peak]
    except:
        i_peak = np.argmax(vbar)
        r_peak = rad[i_peak]
        v_peak = vbar[i_peak]

    # ピークが最外点（まだ上昇中）の場合フラグ
    peak_at_edge = (r_peak &gt; rad.max() * 0.9)

    # --- g_N(r) = v_bar^2 / r ---
    # r &gt; 0 のデータのみ
    mask = rad &gt; 0.01
    rad_g = rad[mask]
    vbar_g = vbar[mask]
    gN = vbar_g**2 / rad_g  # (km/s)^2 / kpc

    # g_N のピーク
    i_gN_peak = np.argmax(gN)
    gN_max = gN[i_gN_peak]
    r_gN_peak = rad_g[i_gN_peak]

    # --- r_{g_N=a0}: g_N = a0 の交差点（ピーク外側）---
    r_gN_a0 = None
    if gN_max &gt; a0_unit:
        # ピーク外側で g_N = a0 を通過する点を探す
        for j in range(i_gN_peak, len(gN)-1):
            if gN[j] &gt;= a0_unit and gN[j+1] &lt; a0_unit:
                # 線形補間
                frac = (a0_unit - gN[j]) / (gN[j+1] - gN[j])
                r_gN_a0 = rad_g[j] + frac * (rad_g[j+1] - rad_g[j])
                break
        # ピーク内側の交差も探す（あれば）
        r_gN_a0_inner = None
        for j in range(i_gN_peak):
            if gN[j] &lt; a0_unit and gN[j+1] &gt;= a0_unit:
                frac = (a0_unit - gN[j]) / (gN[j+1] - gN[j])
                r_gN_a0_inner = rad_g[j] + frac * (rad_g[j+1] - rad_g[j])

    return {
        'r_peak': r_peak,
        'v_peak': v_peak,
        'peak_at_edge': peak_at_edge,
```

```
        'r_gN_peak': r_gN_peak,
        'gN_max': gN_max,
        'gN_max_over_a0': gN_max / a0_unit,
        'r_gN_a0': r_gN_a0,
        'rad': rad,
        'vbar': vbar,
        'gN': gN,
        'rad_g': rad_g,
    }


# ============================================================
# メインループ
# ============================================================
results_all = []
results_crossing = []  # g_N = a0 交差あり
results_no_crossing = []

for name, gal in galaxies.items():
    data = read_dat(name)
    if data is None:
        continue

    info = compute_radii(data, gal['upsilon_d'])
    if info is None:
        continue

    entry = {
        'name': name,
        'vflat': gal['vflat'],
        'rs_tanh': gal['rs_tanh'],
        'upsilon_d': gal['upsilon_d'],
        **{k: info[k] for k in ['r_peak', 'v_peak', 'peak_at_edge',
                                'r_gN_peak', 'gN_max', 'gN_max_over_a0',
                                'r_gN_a0']},
    }

    # 比率計算
    if info['r_gN_a0'] is not None and info['r_gN_a0'] &gt; 0:
        entry['rs_over_rT3'] = gal['rs_tanh'] / info['r_gN_a0']
        entry['rpeak_over_rT3'] = info['r_peak'] / info['r_gN_a0']
        entry['rs_over_rpeak'] = gal['rs_tanh'] / info['r_peak']
        results_crossing.append(entry)
    else:
        entry['rs_over_rT3'] = None
        entry['rpeak_over_rT3'] = None
        entry['rs_over_rpeak'] = gal['rs_tanh'] / info['r_peak'] if info['r_peak'] &gt; 0 else None
        results_no_crossing.append(entry)

    results_all.append(entry)

N_total = len(results_all)
N_cross = len(results_crossing)
N_no = len(results_no_crossing)

print(f"¥n{'='*70}")
print(f"結果概要")
print(f"{'='*70}")
print(f"解析成功: {N_total}")
print(f"  g_N=a0 交差あり: {N_cross} ({100*N_cross/N_total:.1f}%)")
print(f"  交差なし (全域MOND) : {N_no} ({100*N_no/N_total:.1f}%)")

# ============================================================
# 交差ありの銀河の解析
# ============================================================
print(f"¥n{'='*70}")
print(f"交差あり銀河の比率 (N={N_cross}) ")
print(f"{'='*70}")

rs_rT3 = np.array([d['rs_over_rT3'] for d in results_crossing])
rp_rT3 = np.array([d['rpeak_over_rT3'] for d in results_crossing])
rs_rp = np.array([d['rs_over_rpeak'] for d in results_crossing])
vf_cross = np.array([d['vflat'] for d in results_crossing])
gNmax = np.array([d['gN_max_over_a0'] for d in results_crossing])
print(f"¥n{'指標':&gt;25s}  {'中央値':&gt;8s}  {'平均':&gt;8s}  {'std':&gt;8s}  {'25%':&gt;8s}  {'75%':&gt;8s}")
for label, arr in [
    ('r_s_tanh / r_{g_N=a0}', rs_rT3),
    ('r_peak / r_{g_N=a0}', rp_rT3),
    ('r_s_tanh / r_peak', rs_rp),
    ('g_N_max / a0', gNmax),
]:
    print(f"{label:&gt;25s}  {np.median(arr):8.3f}  {np.mean(arr):8.3f}  {np.std(arr):8.3f}  {np.percentile(arr,25):8.3f}  {np.percentile(arr,75):8.3f}")

# ============================================================
# 分解: r_s/r_T3 = (r_s/r_peak) × (r_peak/r_T3)
# ============================================================
print(f"¥n{'='*70}")
print("比率の分解: r_s/r_T3 = (r_s/r_peak) × (r_peak/r_T3)")
print(f"{'='*70}")
print(f"  r_s_tanh/r_T3 中央値 = {np.median(rs_rT3):.3f}")
print(f"  = (r_s/r_peak) × (r_peak/r_T3)")
print(f"  = {np.median(rs_rp):.3f} × {np.median(rp_rT3):.3f}")
print(f"  = {np.median(rs_rp)*np.median(rp_rT3):.3f}")

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

```
# g_N_max/a0 依存性
# ============================================================
print(f"¥n{'='*70}")
print("比率の g_N_max/a0 依存性")
print(f"{'='*70}")

# ビン分割
sort_idx = np.argsort(gNmax)
n_per = max(N_cross // 4, 1)
print(f"¥n{'g_N_max/a0 範囲':&gt;20s}  {'N':&gt;4s}  {'r_s/r_T3':&gt;10s}  {'r_peak/r_T3':&gt;12s}  {'r_s/r_peak':&gt;10s}")
for i in range(4):
    i0 = i * n_per
    i1 = (i+1)*n_per if i &lt; 3 else N_cross
    idx = sort_idx[i0:i1]
    print(f"{gNmax[idx].min():.1f}-{gNmax[idx].max():.1f}:".rjust(20) +
          f"  {len(idx):4d}  {np.median(rs_rT3[idx]):10.3f}  {np.median(rp_rT3[idx]):12.3f}  {np.median(rs_rp[idx]):10.3f}")

# 相関
rho_gN, p_gN = spearmanr(gNmax, rs_rT3)
print(f"¥nr_s/r_T3 vs g_N_max/a0: Spearman ρ={rho_gN:.3f}, p={p_gN:.2e}")
rho_gN2, p_gN2 = spearmanr(gNmax, rp_rT3)
print(f"r_peak/r_T3 vs g_N_max/a0: Spearman ρ={rho_gN2:.3f}, p={p_gN2:.2e}")

# ============================================================
# v_flat 依存性
# ============================================================
print(f"¥n{'='*70}")
print("比率の v_flat 依存性")
print(f"{'='*70}")

rho_vf, p_vf = spearmanr(vf_cross, rs_rT3)
print(f"r_s/r_T3 vs v_flat: Spearman ρ={rho_vf:.3f}, p={p_vf:.2e}")

# ============================================================
# 全銀河での r_s/r_peak
# ============================================================
print(f"¥n{'='*70}")
print(f"全銀河での r_s_tanh / r_peak (N={N_total}) ")
print(f"{'='*70}")

rs_rp_all = np.array([d['rs_over_rpeak'] for d in results_all if d['rs_over_rpeak'] is not None])
print(f"  中央値: {np.median(rs_rp_all):.3f}")
print(f"  平均値: {np.mean(rs_rp_all):.3f}")
print(f"  std:    {np.std(rs_rp_all):.3f}")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) r_s/r_T3 のヒストグラム
ax = axes[0, 0]
ax.hist(rs_rT3, bins=30, color='steelblue', edgecolor='white', alpha=0.7)
ax.axvline(np.median(rs_rT3), color='red', ls='--',
           label=f'Median={np.median(rs_rT3):.2f}')
ax.axvline(2.75, color='black', ls=':', label='Previous=2.75')
ax.set_xlabel('r_s_tanh / r_{g_N=a0}')
ax.set_ylabel('Count')
ax.set_title(f'(a) r_s/r_T3 distribution (N={N_cross})')
ax.legend(fontsize=8)

# (b) r_peak/r_T3 のヒストグラム
ax = axes[0, 1]
ax.hist(rp_rT3, bins=30, color='coral', edgecolor='white', alpha=0.7)
ax.axvline(np.median(rp_rT3), color='red', ls='--',
           label=f'Median={np.median(rp_rT3):.2f}')
ax.set_xlabel('r_peak / r_{g_N=a0}')
ax.set_ylabel('Count')
ax.set_title(f'(b) r_peak/r_T3 (geometric factor)')
ax.legend(fontsize=8)

# (c) r_s/r_T3 vs g_N_max/a0
ax = axes[0, 2]
ax.scatter(gNmax, rs_rT3, s=15, alpha=0.5, c='steelblue', label='r_s/r_T3')
ax.scatter(gNmax, rp_rT3, s=15, alpha=0.5, c='coral', label='r_peak/r_T3')
ax.axhline(2.75, color='black', ls=':', alpha=0.5)
ax.set_xlabel('g_N_max / a0')
ax.set_ylabel('Ratio')
ax.set_title(f'(c) Ratio vs g_N_max/a0 (ρ={rho_gN:.2f})')
ax.set_xscale('log')
ax.legend(fontsize=8)

# (d) r_s/r_peak vs v_flat (全銀河)
ax = axes[1, 0]
vf_all = np.array([d['vflat'] for d in results_all if d['rs_over_rpeak'] is not None])
has_cross = np.array([d['r_gN_a0'] is not None for d in results_all if d['rs_over_rpeak'] is not None])
ax.scatter(vf_all[~has_cross], rs_rp_all[~has_cross], s=8, alpha=0.3, c='gray', label='No crossing')
ax.scatter(vf_all[has_cross], rs_rp_all[has_cross], s=15, alpha=0.5, c='blue', label='Has crossing')
ax.axhline(1.0, color='red', ls='--', alpha=0.5)
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('r_s_tanh / r_peak')
ax.set_title(f'(d) r_s/r_peak (all, median={np.median(rs_rp_all):.2f})')
ax.legend(fontsize=8)

# (e) 3つの半径の比較（交差あり銀河のみ）
```

```
ax = axes[1, 1]
rs_arr = np.array([d['rs_tanh'] for d in results_crossing])
rp_arr = np.array([d['r_peak'] for d in results_crossing])
rT3_arr = np.array([d['r_gN_a0'] for d in results_crossing])
ax.scatter(rT3_arr, rs_arr, s=15, alpha=0.5, c='steelblue', label='r_s_tanh')
ax.scatter(rT3_arr, rp_arr, s=15, alpha=0.5, c='coral', label='r_peak')
maxr = max(rs_arr.max(), rp_arr.max(), rT3_arr.max())
ax.plot([0, maxr], [0, maxr], 'k--', alpha=0.3)
ax.plot([0, maxr], [0, 2.75*maxr], 'gray', ls=':', alpha=0.3, label='×2.75')
ax.set_xlabel('r_{g_N=a0} [kpc]')
ax.set_ylabel('r [kpc]')
ax.set_title('(e) r_s, r_peak vs r_T3')
ax.legend(fontsize=8)

# (f) 分解の確認: r_s/r_T3 = (r_s/r_peak)*(r_peak/r_T3)
ax = axes[1, 2]
product = rs_rp * rp_rT3
ax.scatter(product, rs_rT3, s=15, alpha=0.5, c='green')
ax.plot([0, max(rs_rT3.max(), product.max())],
        [0, max(rs_rT3.max(), product.max())], 'k--')
ax.set_xlabel('(r_s/r_peak) × (r_peak/r_T3)')
ax.set_ylabel('r_s/r_T3 (direct)')
ax.set_title('(f) Decomposition check')

plt.tight_layout()
plt.savefig('TA2_direct.png', dpi=150)
print(f"¥n[SAVED] TA2_direct.png")

# ============================================================
# CSV出力
# ============================================================
out_csv = "TA2_three_radii.csv"
with open(out_csv, "w", newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerow(['galaxy', 'v_flat', 'r_s_tanh', 'r_peak', 'r_gN_a0',
                     'gN_max_over_a0', 'rs_over_rT3', 'rpeak_over_rT3',
                     'rs_over_rpeak', 'upsilon_d'])
    for d in sorted(results_all, key=lambda x: x['vflat']):
        writer.writerow([
            d['name'], f"{d['vflat']:.1f}",
            f"{d['rs_tanh']:.3f}", f"{d['r_peak']:.3f}",
            f"{d['r_gN_a0']:.3f}" if d['r_gN_a0'] else "NA",
            f"{d['gN_max_over_a0']:.3f}",
            f"{d['rs_over_rT3']:.4f}" if d['rs_over_rT3'] else "NA",
            f"{d['rpeak_over_rT3']:.4f}" if d.get('rpeak_over_rT3') else "NA",
            f"{d['rs_over_rpeak']:.4f}" if d['rs_over_rpeak'] else "NA",
            f"{d['upsilon_d']:.3f}",
        ])
print(f"[SAVED] {out_csv}")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("T-A2 最終結論")
print(f"{'='*70}")

print(f"""
■ 実データによる3つの特性半径の比較 (N_cross={N_cross}) :

  r_s_tanh / r_{{g_N=a0}}  中央値 = {np.median(rs_rT3):.3f}  (引き継ぎ値: 2.75)
  r_peak / r_{{g_N=a0}}    中央値 = {np.median(rp_rT3):.3f}
  r_s_tanh / r_peak       中央値 = {np.median(rs_rp):.3f}

■ 分解:
  r_s/r_T3 = (r_s/r_peak) × (r_peak/r_T3)
           ≈ {np.median(rs_rp):.2f} × {np.median(rp_rT3):.2f}

  第1因子 (r_s/r_peak ≈ {np.median(rs_rp):.2f}) : Step 2-3 で解明済み
  第2因子 (r_peak/r_T3 ≈ {np.median(rp_rT3):.2f}) :
    Freeman円盤の幾何学 (g_Nピーク位置 &lt; V_diskピーク位置)

■ g_N_max/a0 依存性:
  r_s/r_T3 vs g_N_max/a0: ρ = {rho_gN:.3f}
  → 負の相関なら「g_N_max が a0 にギリギリの銀河ほど比率が大きい」
  → 正の相関なら「g_N_max が大きい銀河ほど比率が大きい」

■ 全銀河の r_s/r_peak = {np.median(rs_rp_all):.3f} (N={len(rs_rp_all)})
  → Step 2 の結果 (0.98) との整合性確認

■ 結論の確立レベル:
  定性的 (r_T3 &lt; r_peak は必然) : Level A
  定量的 (比率 ≈ 2.75 の数値) : この結果で判定
""")
```

9. TA3_gc_measurement.py

項目	内容
フェーズ	T-A3
目的	g_c の独立測定（RAR フィット）
使用rs	rs不使用（安全）
結果	g_c 中央値 0.825 a0、CV=0.99。g_c ~ v_flat^0.82 (rho=0.546)。
ステータス	Level A（確立）

解析目的

各銀河の（g_N, g_obs）データに MOND 式をフィットし、g_c を唯一の自由パラメータとして決定。r_s を一切使用しない循環論法回避の測定方法。

ソースコード全文

```
#!/usr/bin/env python3
"""
T-A3: g_c の独立測定

方法: RAR (Radial Acceleration Relation) への直接フィッティング

  g_obs(r) = v_obs²(r) / r
  g_N(r)   = v_bar²(r) / r   (Υ_d 適用済み)

  v3.0 式(5): g_obs = (g_N + sqrt(g_N² + 4·g_c·g_N)) / 2

  g_c を各銀河で独立にフィット。r_s は一切使用しない。
  → 循環論法を完全回避した g_c の測定。

実行: uv run --with scipy --with matplotlib --with numpy python TA3_gc_measurement.py
"""

import csv
import os
import numpy as np
from scipy.optimize import minimize_scalar, curve_fit
from scipy.stats import spearmanr, pearsonr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 定数
# ============================================================
# a0 in (km/s)^2 / kpc
a0_unit = 3703.0

# ============================================================
# データ読み込み
# ============================================================
def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

# --- sparc_results.csv ---
src_cols, src_rows = load_csv("sparc_results.csv")
print(f"[INFO] sparc_results.csv: {len(src_rows)} rows")

galaxies = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name:
        name = list(row.values())[0].strip()
    ud = get_val(row, ['ud', 'upsilon_d', 'Upsilon_d'])
    vf = get_val(row, ['vflat', 'v_flat'])
    rs = get_val(row, ['rs1', 'r_s', 'rs'])
    if ud and vf and rs:
        galaxies[name] = {'upsilon_d': ud, 'vflat': vf, 'rs_tanh': rs}

print(f"[INFO] {len(galaxies)} galaxies loaded")

# ============================================================
# .dat ファイル読み込み
# ============================================================
def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
```

```
        if os.path.exists(fn):
            data = np.loadtxt(fn, comments='#')
            return data
    return None

# ============================================================
# MOND式フィッティング
# ============================================================
def mond_gobs(gN, gc):
    """v3.0 式(5): g_obs = (g_N + sqrt(g_N^2 + 4*gc*g_N)) / 2"""
    return (gN + np.sqrt(gN**2 + 4*gc*gN)) / 2

def fit_gc_single(gN_arr, gobs_arr, err_gobs=None):
    """
    単一銀河の (g_N, g_obs) データに対して g_c をフィット

    最小化: Σ [log(g_obs) - log(g_obs_model(g_N, g_c))]^2
    log空間でフィット (ダイナミックレンジが大きいため)
    """
    # g_N, g_obs &gt; 0 のデータのみ
    mask = (gN_arr &gt; 0) &amp; (gobs_arr &gt; 0)
    gN = gN_arr[mask]
    gobs = gobs_arr[mask]
    N = len(gN)

    if N &lt; 3:
        return None, None, None, N

    log_gobs = np.log10(gobs)

    def chi2(log_gc):
        gc = 10**log_gc
        model = mond_gobs(gN, gc)
        model = np.maximum(model, 1e-30)
        residuals = log_gobs - np.log10(model)
        return np.sum(residuals**2)

    # グリッドサーチ + 精密化
    log_gc_grid = np.linspace(-1.0, 2.0, 100)  # g_c/a0 = 0.1 ~ 100
    log_gc_grid_abs = log_gc_grid + np.log10(a0_unit)
    chi2_grid = [chi2(lg) for lg in log_gc_grid_abs]
    i_best = np.argmin(chi2_grid)

    # 精密化
    result = minimize_scalar(chi2,
                             bounds=(log_gc_grid_abs[max(0,i_best-5)],
                                     log_gc_grid_abs[min(len(log_gc_grid)-1,i_best+5)]),
                             method='bounded')

    gc_fit = 10**result.fun  # これは chi2 値、gc は：
    gc_fit = 10**result.x
    gc_over_a0 = gc_fit / a0_unit
    chi2_min = result.fun
    chi2_dof = chi2_min / max(N - 1, 1)

    # 信頼区間の推定 (Δχ² = 1 で 1σ)
    gc_err = None
    try:
        target = chi2_min + 1.0
        # 下限
        def f_lo(lg):
            return chi2(lg) - target
        from scipy.optimize import brentq
        lg_lo = brentq(f_lo, log_gc_grid_abs[0], result.x)
        lg_hi = brentq(f_lo, result.x, log_gc_grid_abs[-1])
        gc_err = (10**lg_hi / a0_unit - 10**lg_lo / a0_unit) / 2
    except:
        pass

    return gc_over_a0, chi2_dof, gc_err, N


# ============================================================
# メインループ
# ============================================================
results = []
n_fail = 0
n_no_dat = 0

for name, gal in galaxies.items():
    data = read_dat(name)
    if data is None:
        n_no_dat += 1
        continue

    rad = data[:, 0]      # kpc
    v_obs = data[:, 1]    # km/s
    err_v = data[:, 2]    # km/s
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)

    ud = gal['upsilon_d']

    # v_bar^2 (符号保持)
```

```
    vbar2 = ud * np.sign(v_disk) * v_disk**2 + ¥
            np.sign(v_gas) * v_gas**2 + ¥
            np.sign(v_bul) * v_bul**2

    # g_N と g_obs (r &gt; 0 のみ)
    mask = rad &gt; 0.01
    r = rad[mask]
    gN = np.maximum(vbar2[mask], 0) / r       # (km/s)^2 / kpc
    gobs = v_obs[mask]**2 / r                   # (km/s)^2 / kpc

    # フィット
    gc_a0, chi2_dof, gc_err, N_pts = fit_gc_single(gN, gobs)

    if gc_a0 is None:
        n_fail += 1
        continue

    results.append({
        'name': name,
        'vflat': gal['vflat'],
        'rs_tanh': gal['rs_tanh'],
        'upsilon_d': ud,
        'gc_over_a0': gc_a0,
        'gc_err': gc_err,
        'chi2_dof': chi2_dof,
        'N_pts': N_pts,
        # 比較用：循環論法の gc_ratio
        'gc_ratio_circular': gal['vflat']**2 / (gal['rs_tanh'] * a0_unit),
    })

N_total = len(results)
print(f"¥n[INFO] フィット成功: {N_total}, 失敗: {n_fail}, .datなし: {n_no_dat}")

# ============================================================
# 結果解析
# ============================================================
gc_arr = np.array([d['gc_over_a0'] for d in results])
vf_arr = np.array([d['vflat'] for d in results])
ud_arr = np.array([d['upsilon_d'] for d in results])
rs_arr = np.array([d['rs_tanh'] for d in results])
chi2_arr = np.array([d['chi2_dof'] for d in results])
gc_circ = np.array([d['gc_ratio_circular'] for d in results])

print(f"¥n{'='*70}")
print(f"g_c / a0 の分布 (N={N_total}) ")
print(f"{'='*70}")

# 外れ値除去 (chi2_dof &gt; 10 は不良フィット)
mask_good = chi2_arr &lt; 10
gc_good = gc_arr[mask_good]
N_good = mask_good.sum()
print(f"良好フィット (χ²/dof &lt; 10) : {N_good}/{N_total}")

print(f"¥n--- 全銀河 ---")
print(f"  中央値: {np.median(gc_arr):.4f}")
print(f"  平均:   {np.mean(gc_arr):.4f}")
print(f"  std:    {np.std(gc_arr):.4f}")
print(f"  CV:     {np.std(gc_arr)/np.mean(gc_arr):.3f}")
print(f"  25%ile: {np.percentile(gc_arr, 25):.4f}")
print(f"  75%ile: {np.percentile(gc_arr, 75):.4f}")

print(f"¥n--- 良好フィットのみ ---")
print(f"  中央値: {np.median(gc_good):.4f}")
print(f"  平均:   {np.mean(gc_good):.4f}")
print(f"  std:    {np.std(gc_good):.4f}")
print(f"  CV:     {np.std(gc_good)/np.mean(gc_good):.3f}")

# ============================================================
# g_c の普遍性検定
# ============================================================
print(f"¥n{'='*70}")
print("g_c の普遍性検定")
print(f"{'='*70}")

# g_c = a0 (=1) との比較
from scipy.stats import wilcoxon
try:
    stat, p_wilcox = wilcoxon(gc_good - 1.0)
    print(f"¥nWilcoxon検定 (H0: g_c = a0) : p = {p_wilcox:.2e}")
except Exception as e:
    print(f"¥nWilcoxon検定失敗: {e}")
    p_wilcox = None

# 分散の検定：CV が小さければ普遍的
print(f"¥nCV = {np.std(gc_good)/np.mean(gc_good):.3f}")
print(f"  CV &lt; 0.3: 概ね普遍的 (散布は測定誤差程度) ")
print(f"  CV &gt; 1.0: 銀河依存 (v1レポートのCV=4.41と比較) ")

# ============================================================
# g_c vs 銀河パラメータの相関
# ============================================================
print(f"¥n{'='*70}")
print("g_c vs 銀河パラメータ (良好フィットのみ) ")
print(f"{'='*70}")
```

```
vf_good = vf_arr[mask_good]
ud_good = ud_arr[mask_good]
rs_good = rs_arr[mask_good]

# g_c vs v_flat
rho_vf, p_vf = spearmanr(gc_good, vf_good)
print(f"¥ng_c vs v_flat:    Spearman ρ={rho_vf:.3f}, p={p_vf:.2e}")

# g_c vs Y_d
rho_ud, p_ud = spearmanr(gc_good, ud_good)
print(f"g_c vs Y_d:       Spearman ρ={rho_ud:.3f}, p={p_ud:.2e}")

# g_c vs r_s_tanh
rho_rs, p_rs = spearmanr(gc_good, rs_good)
print(f"g_c vs r_s_tanh:  Spearman ρ={rho_rs:.3f}, p={p_rs:.2e}")

# 循環論法の gc との比較
gc_circ_good = gc_circ[mask_good]
rho_circ, p_circ = spearmanr(gc_good, gc_circ_good)
print(f"¥ng_c(RAR) vs g_c(circular): Spearman ρ={rho_circ:.3f}, p={p_circ:.2e}")
print(f"  → ρ≈1 なら両者は同じ情報、ρ&lt;&lt;1 なら独立な測定")

# log-log フィット: g_c vs v_flat
mask_pos = (gc_good &gt; 0) &amp; (vf_good &gt; 0)
if mask_pos.sum() &gt; 10:
    lg = np.log10(gc_good[mask_pos])
    lv = np.log10(vf_good[mask_pos])
    p_fit = np.polyfit(lv, lg, 1)
    r_fit, _ = pearsonr(lv, lg)
    print(f"¥nlog(g_c/a0) = {p_fit[1]:.3f} + {p_fit[0]:.3f} × log(v_flat)")
    print(f"  → g_c ∝ v_flat^{p_fit[0]:.3f}  (r={r_fit:.3f})")
    print(f"  cf. 循環論法: g_c ∝ v_flat^1.32")

# ============================================================
# v_flat ビン別
# ============================================================
print(f"¥n{'='*70}")
print("v_flat ビン別 g_c/a0")
print(f"{'='*70}")

sort_idx = np.argsort(vf_good)
n_per = max(N_good // 6, 1)
print(f"¥n{'v_flat範囲':&gt;18s}  {'N':&gt;4s}  {'g_c/a0 中央値':&gt;12s}  {'g_c/a0 std':&gt;10s}  {'CV':&gt;6s}")

for i in range(6):
    i0 = i * n_per
    i1 = (i+1)*n_per if i &lt; 5 else N_good
    idx = sort_idx[i0:i1]
    vf_b = vf_good[idx]
    gc_b = gc_good[idx]
    cv_b = np.std(gc_b)/np.mean(gc_b) if np.mean(gc_b) &gt; 0 else 999
    label = f"{vf_b.min():.0f}-{vf_b.max():.0f}"
    print(f"{label:&gt;18s}  {len(idx):4d}  {np.median(gc_b):12.4f}  {np.std(gc_b):10.4f}  {cv_b:6.3f}")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) g_c/a0 のヒストグラム
ax = axes[0, 0]
ax.hist(gc_good, bins=40, color='steelblue', edgecolor='white', alpha=0.7)
ax.axvline(1.0, color='red', ls='--', linewidth=2, label='g_c = a0')
ax.axvline(np.median(gc_good), color='black', ls='-',
           label=f'Median={np.median(gc_good):.3f}')
ax.set_xlabel('g_c / a0')
ax.set_ylabel('Count')
ax.set_title(f'(a) g_c distribution (N={N_good})')
ax.legend(fontsize=8)

# (b) g_c vs v_flat (log-log)
ax = axes[0, 1]
ax.scatter(vf_good, gc_good, s=10, alpha=0.4, c='steelblue')
ax.axhline(1.0, color='red', ls='--', alpha=0.5)
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('g_c / a0')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title(f'(b) g_c vs v_flat (ρ={rho_vf:.3f})')

# log-log fit line
if mask_pos.sum() &gt; 10:
    vv = np.logspace(np.log10(vf_good.min()), np.log10(vf_good.max()), 100)
    ax.plot(vv, 10**np.polyval(p_fit, np.log10(vv)), 'k-', alpha=0.5,
            label=f'∝v^{p_fit[0]:.2f}')
    ax.legend(fontsize=8)

# (c) g_c vs Y_d
ax = axes[0, 2]
ax.scatter(ud_good, gc_good, s=10, alpha=0.4, c='orange')
ax.axhline(1.0, color='red', ls='--', alpha=0.5)
ax.set_xlabel('Y_d')
ax.set_ylabel('g_c / a0')
ax.set_yscale('log')
ax.set_title(f'(c) g_c vs Y_d (ρ={rho_ud:.3f})')
```

```
# (d) RAR: g_obs vs g_N (全銀河重ねがき + MOND曲線)
ax = axes[1, 0]
for d in results[:50]:  # 最初の50銀河
    data = read_dat(d['name'])
    if data is None:
        continue
    rad = data[:, 0]
    v_obs = data[:, 1]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = d['upsilon_d']
    vbar2 = ud * np.sign(v_disk)*v_disk**2 + np.sign(v_gas)*v_gas**2 + np.sign(v_bul)*v_bul**2
    mask = (rad &gt; 0.01) &amp; (vbar2 &gt; 0)
    if mask.sum() &lt; 2:
        continue
    gN = vbar2[mask] / rad[mask]
    gobs = v_obs[mask]**2 / rad[mask]
    ax.scatter(np.log10(gN/a0_unit), np.log10(gobs/a0_unit),
               s=1, alpha=0.1, c='gray')

# MOND曲線 (g_c = a0, 0.5a0, 2a0)
gN_line = np.logspace(-3, 2, 500) * a0_unit
for gc_val, col, label in [(1.0, 'red', 'g_c=a0'),
                           (0.5, 'blue', 'g_c=0.5a0'),
                           (2.0, 'green', 'g_c=2a0')]:
    gobs_line = mond_gobs(gN_line, gc_val * a0_unit)
    ax.plot(np.log10(gN_line/a0_unit), np.log10(gobs_line/a0_unit),
            color=col, label=label, linewidth=1.5)
ax.plot([-3, 2], [-3, 2], 'k:', alpha=0.3, label='g_obs=g_N')
ax.set_xlabel('log(g_N / a0)')
ax.set_ylabel('log(g_obs / a0)')
ax.set_title('(d) RAR with MOND curves')
ax.legend(fontsize=7)
ax.set_xlim(-3, 1.5)
ax.set_ylim(-2.5, 1.5)

# (e) g_c(RAR) vs g_c(circular)
ax = axes[1, 1]
ax.scatter(gc_circ_good, gc_good, s=10, alpha=0.4, c='purple')
maxval = max(gc_circ_good.max(), gc_good.max())
ax.plot([0, maxval], [0, maxval], 'k--', alpha=0.3)
ax.set_xlabel('g_c (circular) / a0')
ax.set_ylabel('g_c (RAR fit) / a0')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title(f'(e) RAR vs circular (ρ={rho_circ:.3f})')

# (f) χ²/dof のヒストグラム
ax = axes[1, 2]
ax.hist(chi2_arr[chi2_arr &lt; 20], bins=50, color='green', edgecolor='white', alpha=0.7)
ax.axvline(1.0, color='red', ls='--', label='χ²/dof = 1')
ax.set_xlabel('χ²/dof (log-space)')
ax.set_ylabel('Count')
ax.set_title(f'(f) Fit quality')
ax.legend()

plt.tight_layout()
plt.savefig('TA3_gc_measurement.png', dpi=150)
print(f"¥n[SAVED] TA3_gc_measurement.png")

# ============================================================
# CSV出力
# ============================================================
out_csv = "TA3_gc_independent.csv"
with open(out_csv, "w", newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerow(['galaxy', 'v_flat', 'rs_tanh', 'upsilon_d',
                     'gc_over_a0', 'gc_err', 'chi2_dof', 'N_pts',
                     'gc_circular'])
    for d in sorted(results, key=lambda x: x['vflat']):
        writer.writerow([
            d['name'], f"{d['vflat']:.1f}", f"{d['rs_tanh']:.3f}",
            f"{d['upsilon_d']:.3f}", f"{d['gc_over_a0']:.4f}",
            f"{d['gc_err']:.4f}" if d['gc_err'] else "NA",
            f"{d['chi2_dof']:.4f}", d['N_pts'],
            f"{d['gc_ratio_circular']:.4f}",
        ])
print(f"[SAVED] {out_csv}")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("T-A3 結論")
print(f"{'='*70}")

print(f"""
■ 独立な g_c 測定方法: RAR フィッティング

  g_obs = (g_N + sqrt(g_N² + 4·g_c·g_N)) / 2
  各銀河で g_c を唯一の自由パラメータとしてフィット
  r_s を一切使用しない → 循環論法を完全回避

■ g_c の分布:
```

```
  N = {N_good} (良好フィット)
  中央値: {np.median(gc_good):.4f} × a0
  CV:     {np.std(gc_good)/np.mean(gc_good):.3f}

■ g_c の普遍性:
  CV &lt; 0.3 → 概ね普遍 (散布は測定誤差程度)
  CV ≈ 1   → 銀河依存だが弱い
  CV &gt; 3   → 強い銀河依存

■ v_flat との相関:
  Spearman ρ = {rho_vf:.3f}
  → ρ ≈ 0: g_c は v_flat に依存しない (普遍的)
  → ρ &gt; 0.3: g_c は銀河質量に依存する

■ 循環論法の g_c との比較:
  ρ = {rho_circ:.3f}
  → ρ ≈ 1: RAR フィットは循環論法と同じ情報 (独立性なし)
  → ρ &lt; 0.5: RAR フィットは独立な測定

■ v3.0 への帰結:
  g_c ≈ a0 (普遍的) → v3.0 はそのまま維持
  g_c が銀河依存    → g_c の物理的起源の説明が必要
                      (条件14の塑性含有量？ Σ□ 依存？)
""")
```

10. cond14_complexity.py

項目	内容
フェーズ	条件14検証
目的	v_bar プロファイル複雑度と g_c の相関（条件14仮説テスト）
使用rs	rs2（gamma残差の計算）
結果	C_rms vs g_c: rho=0.048（無相関）。条件14の因果連鎖は不支持。gas -> C_rms -> gamma残差 の間接連鎖は有意。
ステータス	Level B（条件14不支持、ガスの間接効果あり）

解析目的

条件14（塑性領域 -> v_bar 複雑化 -> gamma の起源）仮説を検証。C_rms（Freeman逸脱度）、outer_excess 等を g_c と比較。

ソースコード全文

```
#!/usr/bin/env python3
"""
条件14仮説の検証：
v_bar(r) の「指数円盤からの逸脱度」が g_c, v_flat と相関するか

仮説: 塑性領域の分布 → v_bar の非指数的構造 → γ=0.284 の起源

指標:
  (1) C_profile: v_bar(r) のFreeman円盤からの逸脱度（RMS残差）
  (2) S_shape:   v_bar(r)/v_bar_Freeman(r) の勾配変化（形状複雑度）
  (3) r_eff_ratio: v_bar の有効半径 / Freeman の有効半径（拡がりの差）

検証:
  C_profile vs g_c(RAR) → 相関あれば条件14を支持
  C_profile vs v_flat   → 相関あれば γ の起源として候補
  C_profile vs (r_s実測 - r_s予測) → 残差を説明できるか

実行: uv run --with scipy --with matplotlib --with numpy python cond14_complexity.py

前提: TA3_gc_independent.csv（T-A3の出力）が同ディレクトリにあること
"""

import csv
import os
import numpy as np
from scipy.optimize import curve_fit
from scipy.special import i0, i1, k0, k1
from scipy.stats import spearmanr, pearsonr
from scipy.interpolate import interp1d
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 定数・ユーティリティ
# ============================================================
def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

# Freeman bessel
def freeman_bessel(y):
    y = np.clip(y, 1e-8, 50.0)
    return y**2 * (i0(y)*k0(y) - i1(y)*k1(y))

y_fine = np.linspace(0.01, 5.0, 5000)
vd2_shape = freeman_bessel(y_fine)
bessel_peak = np.max(vd2_shape)

def v_freeman_profile(r, h_R, V_peak):
    """Freeman指数円盤の回転速度"""
    y = r / (2.0 * h_R)
    y = np.clip(y, 1e-6, 50.0)
    v2 = V_peak**2 * freeman_bessel(y) / bessel_peak
    return np.sqrt(np.maximum(v2, 0.0))

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

```
# データ読み込み
# ============================================================
print("="*70)
print("データ読み込み")
print("="*70)

# sparc_results.csv
src_cols, src_rows = load_csv("sparc_results.csv")
galaxies = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    ud = get_val(row, ['ud', 'upsilon_d'])
    vf = get_val(row, ['vflat', 'v_flat'])
    rs = get_val(row, ['rs1', 'r_s'])
    if ud and vf and rs and vf &gt; 0:
        galaxies[name] = {'upsilon_d': ud, 'vflat': vf, 'rs_tanh': rs}

# g_c(RAR) from T-A3
gc_rar = {}
if os.path.exists("TA3_gc_independent.csv"):
    _, gc_rows = load_csv("TA3_gc_independent.csv")
    for row in gc_rows:
        gname = row.get('galaxy', '').strip()
        gc = get_val(row, ['gc_over_a0'])
        if gname and gc:
            gc_rar[gname] = gc
    print(f"g_c(RAR): {len(gc_rar)} galaxies")
else:
    print("[WARN] TA3_gc_independent.csv not found. g_c correlations skipped.")

print(f"SPARC: {len(galaxies)} galaxies")

# ============================================================
# v_bar プロファイル複雑度の計算
# ============================================================
print(f"¥n{'='*70}")
print("v_bar プロファイル複雑度の計算")
print(f"{'='*70}")

results = []

for name, gal in galaxies.items():
    data = read_dat(name)
    if data is None:
        continue

    rad = data[:, 0]
    v_obs = data[:, 1]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = gal['upsilon_d']

    # v_bar
    vbar2 = ud * np.sign(v_disk)*v_disk**2 + ¥
            np.sign(v_gas)*v_gas**2 + ¥
            np.sign(v_bul)*v_bul**2
    vbar = np.sqrt(np.maximum(vbar2, 0.0))

    # v_disk (Υ_d適用)
    vdisk = np.sqrt(ud) * np.abs(v_disk)

    if len(rad) &lt; 5 or np.max(vbar) &lt; 1.0:
        continue

    # --- Freeman フィット: v_disk に指数円盤をフィット ---
    V_peak_disk = np.max(vdisk)
    i_peak = np.argmax(vdisk)
    r_peak = rad[i_peak]
    h_R_est = r_peak / 2.15

    if h_R_est &lt; 0.01 or r_peak &gt;= rad.max() * 0.95:
        continue

    # Freeman 予測
    vbar_freeman = v_freeman_profile(rad, h_R_est, V_peak_disk)

    # --- 複雑度指標 (1): RMS逸脱度 ---
    # v_bar と Freeman の比 (V_peak で規格化)
    norm = max(V_peak_disk, 1.0)
    residual = (vbar - vbar_freeman) / norm
    C_rms = np.sqrt(np.mean(residual**2))

    # --- 複雑度指標 (2): 形状比 v_bar/v_freeman の非単調性 ---
    ratio_profile = vbar / np.maximum(vbar_freeman, 0.1)
    # 単調なら d(ratio)/dr は一定符号。符号変化の回数 = 複雑度
    d_ratio = np.diff(ratio_profile)
    sign_changes = np.sum(np.abs(np.diff(np.sign(d_ratio))) &gt; 0)
    S_shape = sign_changes / max(len(rad) - 2, 1)  # 規格化

    # --- 複雑度指標 (3): 有効半径の比 ---
    # v_bar^2 の累積分布の50%半径
    vbar2_cum = np.cumsum(vbar**2 * np.gradient(rad))
```

```
    vbar2_cum /= vbar2_cum[-1] if vbar2_cum[-1] &gt; 0 else 1
    try:
        r_eff_bar = np.interp(0.5, vbar2_cum, rad)
    except:
        r_eff_bar = rad[len(rad)//2]

    vfr2_cum = np.cumsum(vbar_freeman**2 * np.gradient(rad))
    vfr2_cum /= vfr2_cum[-1] if vfr2_cum[-1] &gt; 0 else 1
    try:
        r_eff_freeman = np.interp(0.5, vfr2_cum, rad)
    except:
        r_eff_freeman = rad[len(rad)//2]

    r_eff_ratio = r_eff_bar / r_eff_freeman if r_eff_freeman &gt; 0 else 1.0

    # --- 複雑度指標 (4): ガス寄与度 ---
    vgas_abs = np.abs(v_gas)
    gas_fraction = np.mean(vgas_abs**2 / np.maximum(vbar**2, 0.01))

    # --- 複雑度指標 (5): 外部勾配の逸脱 ---
    # r &gt; r_peak での v_bar の減衰率を Freeman と比較
    mask_outer = rad &gt; r_peak
    if np.sum(mask_outer) &gt; 3:
        r_out = rad[mask_outer]
        vbar_out = vbar[mask_outer]
        vfr_out = vbar_freeman[mask_outer]
        # log空間での勾配差
        if vbar_out[0] &gt; 0 and vfr_out[0] &gt; 0:
            slope_bar = np.polyfit(r_out/h_R_est, np.log10(np.maximum(vbar_out, 0.1)), 1)[0]
            slope_fr = np.polyfit(r_out/h_R_est, np.log10(np.maximum(vfr_out, 0.1)), 1)[0]
            outer_excess = slope_bar - slope_fr  # 正 → 実データの方が緩やかに減衰
        else:
            outer_excess = 0.0
    else:
        outer_excess = 0.0

    entry = {
        'name': name,
        'vflat': gal['vflat'],
        'rs_tanh': gal['rs_tanh'],
        'upsilon_d': ud,
        'h_R_est': h_R_est,
        'V_peak_disk': V_peak_disk,
        'C_rms': C_rms,
        'S_shape': S_shape,
        'r_eff_ratio': r_eff_ratio,
        'gas_fraction': gas_fraction,
        'outer_excess': outer_excess,
        'gc_rar': gc_rar.get(name, None),
    }
    results.append(entry)

N = len(results)
print(f"解析成功: {N} 銀河")

# ============================================================
# 相関解析
# ============================================================
print(f"¥n{'='*70}")
print("相関解析")
print(f"{'='*70}")

vf_arr = np.array([d['vflat'] for d in results])
C_rms_arr = np.array([d['C_rms'] for d in results])
S_shape_arr = np.array([d['S_shape'] for d in results])
r_eff_arr = np.array([d['r_eff_ratio'] for d in results])
gas_frac_arr = np.array([d['gas_fraction'] for d in results])
outer_arr = np.array([d['outer_excess'] for d in results])
rs_arr = np.array([d['rs_tanh'] for d in results])
hR_arr = np.array([d['h_R_est'] for d in results])

# g_c があるサブセット
gc_mask = np.array([d['gc_rar'] is not None for d in results])
gc_arr = np.array([d['gc_rar'] if d['gc_rar'] else 0 for d in results])
# rs/hR の実測 vs Freeman予測
rs_over_hR = rs_arr / hR_arr

# --- 主要相関 ---
indicators = [
    ('C_rms (RMS逸脱)', C_rms_arr),
    ('S_shape (形状複雑度)', S_shape_arr),
    ('r_eff_ratio (拡がり比)', r_eff_arr),
    ('gas_fraction', gas_frac_arr),
    ('outer_excess (外部超過)', outer_arr),
]

targets = [
    ('v_flat', vf_arr, None),
    ('r_s/h_R', rs_over_hR, None),
    ('log(g_c/a0)', np.log10(np.maximum(gc_arr, 0.01)), gc_mask),
]

print(f"¥n{'指標':&gt;25s}", end='')
for tname, _, _ in targets:
    print(f"  {'ρ('+tname+')':&gt;15s}", end='')
```

```
print()

for iname, iarr in indicators:
    print(f"{iname:&gt;25s}", end='')
    for tname, tarr, tmask in targets:
        if tmask is not None:
            rho, p = spearmanr(iarr[tmask], tarr[tmask])
            sig = '***' if p &lt; 0.001 else '**' if p &lt; 0.01 else '*' if p &lt; 0.05 else ''
        else:
            rho, p = spearmanr(iarr, tarr)
            sig = '***' if p &lt; 0.001 else '**' if p &lt; 0.01 else '*' if p &lt; 0.05 else ''
        print(f"  {rho:+.3f}{sig:&gt;4s}    ", end='')
    print()

# --- 核心の相関: outer_excess vs γ残差 ---
# γ残差 = log(r_s/h_R) - &lt;log(r_s/h_R)&gt; (v_flat 依存を除いた残差)
lv = np.log10(vf_arr)
lrh = np.log10(rs_over_hR)
p_rh = np.polyfit(lv, lrh, 1)
rh_residual = lrh - np.polyval(p_rh, lv)

print(f"¥n--- γ残差 (r_s/h_R のv_flat依存を除去) との相関 ---")
for iname, iarr in indicators:
    rho, p = spearmanr(iarr, rh_residual)
    sig = '***' if p &lt; 0.001 else '**' if p &lt; 0.01 else '*' if p &lt; 0.05 else ''
    print(f"  {iname:&gt;25s} vs γ残差: ρ={rho:+.3f}, p={p:.2e} {sig}")

# --- 条件14型5銀河の複雑度 ---
cond14_names = ['NGC1003', 'NGC2403', 'NGC2903', 'NGC6015', 'UGC00128']
print(f"¥n--- 条件14型銀河の複雑度 ---")
print(f"{'銀河':&gt;12s}  {'C_rms':&gt;6s}  {'S_shape':&gt;7s}  {'r_eff':&gt;6s}  {'gas_f':&gt;6s}  {'outer':&gt;6s}  {'g_c/a0':&gt;7s}")

cond14_found = []
for d in results:
    if d['name'] in cond14_names:
        gc_str = f"{d['gc_rar']:.3f}" if d['gc_rar'] else "NA"
        print(f"{d['name']:&gt;12s}  {d['C_rms']:6.3f}  {d['S_shape']:7.3f}  {d['r_eff_ratio']:6.3f}  {d['gas_fraction']:6.3f}  {d['outer_excess']:+6.3f}  {gc_str:&gt;7s}")
        cond14_found.append(d)

# 全体との比較
if cond14_found:
    print(f"¥n  条件14型 (N={len(cond14_found)}) vs 全体 (N={N}) :")
    for metric, arr in [('C_rms', C_rms_arr), ('outer_excess', outer_arr), ('gas_fraction', gas_frac_arr)]:
        c14_vals = [d[metric.lower() if metric != 'C_rms' else 'C_rms'] for d in cond14_found]
        # handle key matching
    for iname, iarr in indicators[:3]:
        key = [k for k, v in [('C_rms', C_rms_arr), ('S_shape', S_shape_arr), ('r_eff_ratio', r_eff_arr)] if k in iname.split()[0]][0] if iname.split()[0] in ['C_rms', 'S_shape', 'r_eff_r
        if key:
            c14_vals = [d[key] for d in cond14_found]
            if c14_vals:
                print(f"    {iname}: 条件14中央値={np.median(c14_vals):.3f}, 全体中央値={np.median(iarr):.3f}")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) C_rms vs v_flat
ax = axes[0, 0]
ax.scatter(vf_arr, C_rms_arr, s=8, alpha=0.4, c='steelblue')
for d in cond14_found:
    ax.scatter(d['vflat'], d['C_rms'], s=60, c='red', marker='*', zorder=5)
rho, _ = spearmanr(vf_arr, C_rms_arr)
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('C_rms (Freeman逸脱度)')
ax.set_title(f'(a) Profile complexity vs v_flat (ρ={rho:.3f})')

# (b) outer_excess vs v_flat
ax = axes[0, 1]
ax.scatter(vf_arr, outer_arr, s=8, alpha=0.4, c='teal')
for d in cond14_found:
    ax.scatter(d['vflat'], d['outer_excess'], s=60, c='red', marker='*', zorder=5)
ax.axhline(0, color='gray', ls='--', alpha=0.3)
rho_oe, _ = spearmanr(vf_arr, outer_arr)
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('Outer excess (slope difference)')
ax.set_title(f'(b) Outer excess vs v_flat (ρ={rho_oe:.3f})')

# (c) C_rms vs g_c (条件14の核心テスト)
ax = axes[0, 2]
if gc_mask.sum() &gt; 10:
    ax.scatter(gc_arr[gc_mask], C_rms_arr[gc_mask], s=8, alpha=0.4, c='purple')
    for d in cond14_found:
        if d['gc_rar']:
            ax.scatter(d['gc_rar'], d['C_rms'], s=60, c='red', marker='*', zorder=5)
    rho_gc, p_gc = spearmanr(gc_arr[gc_mask], C_rms_arr[gc_mask])
    ax.set_xlabel('g_c / a0 (RAR)')
    ax.set_ylabel('C_rms')
    ax.set_xscale('log')
    ax.set_title(f'(c) Complexity vs g_c (ρ={rho_gc:.3f}, p={p_gc:.2e})')

# (d) r_s/h_R vs C_rms (複雑度が r_s を決めるか)
ax = axes[1, 0]
ax.scatter(C_rms_arr, rs_over_hR, s=8, alpha=0.4, c='green')
for d in cond14_found:
```

```
    ax.scatter(d['C_rms'], d['rs_tanh']/d['h_R_est'], s=60, c='red', marker='*', zorder=5)
rho_rh, _ = spearmanr(C_rms_arr, rs_over_hR)
ax.set_xlabel('C_rms')
ax.set_ylabel('r_s / h_R')
ax.set_title(f'(d) r_s/h_R vs complexity (ρ={rho_rh:.3f})')

# (e) γ残差 vs outer_excess (核心テスト)
ax = axes[1, 1]
ax.scatter(outer_arr, rh_residual, s=8, alpha=0.4, c='orange')
for d in cond14_found:
    idx = [i for i, r in enumerate(results) if r['name'] == d['name']]
    if idx:
        ax.scatter(d['outer_excess'], rh_residual[idx[0]], s=60, c='red', marker='*', zorder=5)
rho_oe_res, p_oe_res = spearmanr(outer_arr, rh_residual)
ax.axhline(0, color='gray', ls='--', alpha=0.3)
ax.axvline(0, color='gray', ls='--', alpha=0.3)
ax.set_xlabel('Outer excess')
ax.set_ylabel('γ residual')
ax.set_title(f'(e) γ residual vs outer excess (ρ={rho_oe_res:.3f})')

# (f) ガス寄与度 vs v_flat
ax = axes[1, 2]
ax.scatter(vf_arr, gas_frac_arr, s=8, alpha=0.4, c='coral')
for d in cond14_found:
    ax.scatter(d['vflat'], d['gas_fraction'], s=60, c='red', marker='*', zorder=5)
rho_gf, _ = spearmanr(vf_arr, gas_frac_arr)
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('Gas fraction (v_gas²/v_bar²)')
ax.set_title(f'(f) Gas fraction vs v_flat (ρ={rho_gf:.3f})')

plt.tight_layout()
plt.savefig('cond14_complexity.png', dpi=150)
print(f"¥n[SAVED] cond14_complexity.png")

# ============================================================
# 結論
# ============================================================
print(f"¥n{'='*70}")
print("条件14仮説の検証結果")
print(f"{'='*70}")

# 核心の判定
print(f"""
■ 仮説: 塑性領域の分布 → v_bar の非指数的構造 → γ=0.284 の起源

■ 検証結果:

  (1) C_rms (Freeman逸脱度) vs g_c:
      → 相関があれば「g_c が低い銀河ほど v_bar が複雑」= 条件14を支持

  (2) outer_excess (外部勾配超過) vs γ残差:
      → 相関があれば「外側で v_bar が緩やかに減衰する銀河ほど r_s が外に出る」
      → 塑性領域が外側の v_bar を支えている可能性

  (3) 条件14型5銀河の位置:
      → 全体分布から外れていれば、二相構造が複雑度の源泉

■ 条件14仮説の支持/不支持の判定基準:

  支持: C_rms vs g_c に有意な負の相関 (低g_c = 多塑性 = 高複雑度)
        AND outer_excess vs γ残差に有意な正の相関
        AND 条件14型銀河が高複雑度側に位置

  不支持: 相関なし、または条件14型銀河が特別でない
""")
```

11. D3_D1_fix.py

項目	内容
フェーズ	D-1/D-3
目的	beta の統一計算 + gas -> C_rms -> gamma の因果連鎖検証
使用rs	rs1/rs2 混在
結果	gas -> gamma直接: 非有意(rho=0.11)。C_rms -> gamma(v制御後): 有意(rho=0.283)。
ステータス	確定（因果連鎖の構造を解明）

解析目的

Step 2 と Step 3 で beta が不整合（0.600 vs 0.537）だった問題を解消。同一サンプル・同一手法で beta を確定。gas_fraction -> gamma残差 の直接・間接連鎖を偏相関で検証。

ソースコード全文

```
#!/usr/bin/env python3
"""
D-3: gas_fraction → C_rms → γ残差 の因果連鎖を検証
D-1: β の不整合を解消（同一サンプル・同一手法で統一計算）

実行: uv run --with scipy --with matplotlib --with numpy python D3_D1_fix.py

前提: sparc_results.csv, .dat ファイル, f_tanh_robustness_detail.csv
"""

import csv
import os
import numpy as np
from scipy.special import i0, i1, k0, k1
from scipy.stats import spearmanr, pearsonr
from scipy.interpolate import interp1d
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# ユーティリティ
# ============================================================
def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

def freeman_bessel(y):
    y = np.clip(y, 1e-8, 50.0)
    return y**2 * (i0(y)*k0(y) - i1(y)*k1(y))

y_fine = np.linspace(0.01, 5.0, 5000)
bessel_peak = np.max(freeman_bessel(y_fine))

def v_freeman_profile(r, h_R, V_peak):
    y = r / (2.0 * h_R)
    y = np.clip(y, 1e-6, 50.0)
    v2 = V_peak**2 * freeman_bessel(y) / bessel_peak
    return np.sqrt(np.maximum(v2, 0.0))

# ============================================================
# データ読み込み
# ============================================================
src_cols, src_rows = load_csv("sparc_results.csv")
galaxies = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    ud = get_val(row, ['ud', 'upsilon_d'])
    vf = get_val(row, ['vflat', 'v_flat'])
    rs = get_val(row, ['rs1', 'r_s'])
    if ud and vf and rs and vf &gt; 0 and rs &gt; 0:
        galaxies[name] = {'upsilon_d': ud, 'vflat': vf, 'rs_tanh': rs}

print(f"[INFO] {len(galaxies)} galaxies")
```

```
# ============================================================
# 統一計算: 全銀河で h_R, r_peak, 複雑度, gas_fraction を一括取得
# ============================================================
results = []

for name, gal in galaxies.items():
    data = read_dat(name)
    if data is None:
        continue
    rad = data[:, 0]
    v_obs = data[:, 1]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = gal['upsilon_d']

    if len(rad) &lt; 5:
        continue

    # v_bar, v_disk_scaled
    vbar2 = ud * np.sign(v_disk)*v_disk**2 + ¥
            np.sign(v_gas)*v_gas**2 + ¥
            np.sign(v_bul)*v_bul**2
    vbar = np.sqrt(np.maximum(vbar2, 0.0))
    vdisk = np.sqrt(ud) * np.abs(v_disk)
    vgas = np.abs(v_gas)

    # --- h_R 推定 (V_disk ピーク / 2.15) ---
    V_peak_disk = np.max(vdisk)
    i_pd = np.argmax(vdisk)
    r_peak_disk = rad[i_pd]
    peak_at_edge = (r_peak_disk &gt;= rad.max() * 0.9)

    if peak_at_edge or V_peak_disk &lt; 1.0 or r_peak_disk &lt; 0.01:
        continue

    h_R = r_peak_disk / 2.15

    # --- r_s / h_R ---
    rs_over_hR = gal['rs_tanh'] / h_R

    # --- C_rms (Freeman逸脱度) ---
    vbar_freeman = v_freeman_profile(rad, h_R, V_peak_disk)
    norm = max(V_peak_disk, 1.0)
    residual = (vbar - vbar_freeman) / norm
    C_rms = np.sqrt(np.mean(residual**2))

    # --- gas_fraction ---
    gas_fraction = np.mean(vgas**2 / np.maximum(vbar**2, 0.01))

    # --- outer_excess ---
    mask_outer = rad &gt; r_peak_disk
    if np.sum(mask_outer) &gt; 3:
        r_out = rad[mask_outer]
        vbar_out = vbar[mask_outer]
        vfr_out = vbar_freeman[mask_outer]
        if vbar_out[0] &gt; 0.1 and vfr_out[0] &gt; 0.1:
            slope_bar = np.polyfit(r_out/h_R, np.log10(np.maximum(vbar_out, 0.1)), 1)[0]
            slope_fr = np.polyfit(r_out/h_R, np.log10(np.maximum(vfr_out, 0.1)), 1)[0]
            outer_excess = slope_bar - slope_fr
        else:
            outer_excess = 0.0
    else:
        outer_excess = 0.0

    # --- r_eff_ratio ---
    vbar2_cum = np.cumsum(vbar**2 * np.gradient(rad))
    if vbar2_cum[-1] &gt; 0:
        vbar2_cum /= vbar2_cum[-1]
        try: r_eff_bar = np.interp(0.5, vbar2_cum, rad)
        except: r_eff_bar = rad[len(rad)//2]
    else:
        r_eff_bar = rad[len(rad)//2]

    vfr2_cum = np.cumsum(vbar_freeman**2 * np.gradient(rad))
    if vfr2_cum[-1] &gt; 0:
        vfr2_cum /= vfr2_cum[-1]
        try: r_eff_fr = np.interp(0.5, vfr2_cum, rad)
        except: r_eff_fr = rad[len(rad)//2]
    else:
        r_eff_fr = rad[len(rad)//2]

    r_eff_ratio = r_eff_bar / r_eff_fr if r_eff_fr &gt; 0 else 1.0

    results.append({
        'name': name,
        'vflat': gal['vflat'],
        'rs_tanh': gal['rs_tanh'],
        'upsilon_d': ud,
        'h_R': h_R,
        'r_peak_disk': r_peak_disk,
        'rs_over_hR': rs_over_hR,
        'C_rms': C_rms,
        'gas_fraction': gas_fraction,
```

```
        'outer_excess': outer_excess,
        'r_eff_ratio': r_eff_ratio,
    })

N = len(results)
print(f"統一解析成功: {N} 銀河")

# 配列化
vf = np.array([d['vflat'] for d in results])
rs = np.array([d['rs_tanh'] for d in results])
hR = np.array([d['h_R'] for d in results])
rh = np.array([d['rs_over_hR'] for d in results])
C_rms = np.array([d['C_rms'] for d in results])
gas_f = np.array([d['gas_fraction'] for d in results])
outer = np.array([d['outer_excess'] for d in results])
r_eff = np.array([d['r_eff_ratio'] for d in results])

lv = np.log10(vf)
lrs = np.log10(rs)
lhR = np.log10(hR)
lrh = np.log10(rh)

# ============================================================
# D-1: β の統一計算
# ============================================================
print(f"¥n{'='*70}")
print("D-1: β の統一計算 (同一サンプル N={N}) ")
print(f"{'='*70}")

# α: r_s ∝ v^α
p_alpha = np.polyfit(lv, lrs, 1)
r_alpha, _ = pearsonr(lv, lrs)

# β: h_R ∝ v^β
p_beta = np.polyfit(lv, lhR, 1)
r_beta, _ = pearsonr(lv, lhR)

# γ: r_s/h_R ∝ v^γ
p_gamma = np.polyfit(lv, lrh, 1)
r_gamma, _ = pearsonr(lv, lrh)

print(f"¥n  α (r_s ~ v^α)     = {p_alpha[0]:.4f}  (r={r_alpha:.4f})")
print(f"  β (h_R ~ v^β)     = {p_beta[0]:.4f}  (r={r_beta:.4f})")
print(f"  γ (r_s/h_R ~ v^γ) = {p_gamma[0]:.4f}  (r={r_gamma:.4f})")
print(f"  β + γ              = {p_beta[0]+p_gamma[0]:.4f}  (cf. α={p_alpha[0]:.4f})")

# サブセット比較: Step 2 は「非外挿のみ」だった可能性
if os.path.exists("f_tanh_robustness_detail.csv"):
    _, det_rows = load_csv("f_tanh_robustness_detail.csv")
    extrap_set = set()
    for row in det_rows:
        if row.get('is_extrap', '').strip() == 'True':
            extrap_set.add(row.get('galaxy', '').strip())

    mask_inrange = np.array([d['name'] not in extrap_set for d in results])
    N_in = mask_inrange.sum()

    if N_in &gt; 20:
        lv_in = lv[mask_inrange]
        lrs_in = lrs[mask_inrange]
        lhR_in = lhR[mask_inrange]
        lrh_in = lrh[mask_inrange]

        p_a_in = np.polyfit(lv_in, lrs_in, 1)
        p_b_in = np.polyfit(lv_in, lhR_in, 1)
        p_g_in = np.polyfit(lv_in, lrh_in, 1)

        print(f"¥n--- 非外挿のみ (N={N_in}) ---")
        print(f"  α = {p_a_in[0]:.4f}")
        print(f"  β = {p_b_in[0]:.4f}")
        print(f"  γ = {p_g_in[0]:.4f}")
        print(f"  β + γ = {p_b_in[0]+p_g_in[0]:.4f}")

# 外挿のみ
mask_extrap = ~mask_inrange if 'mask_inrange' in dir() else np.zeros(N, dtype=bool)
N_ex = mask_extrap.sum()
if N_ex &gt; 10:
    lv_ex = lv[mask_extrap]
    lhR_ex = lhR[mask_extrap]
    p_b_ex = np.polyfit(lv_ex, lhR_ex, 1)
    print(f"¥n--- 外挿のみ (N={N_ex}) ---")
    print(f"  β = {p_b_ex[0]:.4f}")

# 以前の値との比較
print(f"¥n--- β の比較 ---")
print(f"  Step 2 報告値:       β = 0.600")
print(f"  Step 3 修正版報告値: β = 0.537")
print(f"  今回 (全サンプル) :  β = {p_beta[0]:.4f}  (N={N})")
if 'p_b_in' in dir():
    print(f"  今回 (非外挿のみ) :  β = {p_b_in[0]:.4f}  (N={N_in})")
print(f"¥n  → β の不整合の原因: サンプルフィルタリングの差")

# ============================================================
# D-3: gas → C_rms → γ残差 の因果連鎖検証
# ============================================================
```

```
print(f"¥n{'='*70}")
print("D-3: gas → C_rms → γ残差 の因果連鎖")
print(f"{'='*70}")

# γ残差の計算 (v_flat 依存を除去)
rh_residual = lrh - np.polyval(p_gamma, lv)

# --- 全ペア相関 ---
vars_list = [
    ('gas_fraction', gas_f),
    ('C_rms', C_rms),
    ('outer_excess', outer),
    ('r_eff_ratio', r_eff),
    ('γ残差', rh_residual),
    ('v_flat', vf),
    ('rs/hR', rh),
    ('Υ_d', np.array([d['upsilon_d'] for d in results])),
]

print(f"¥n--- 全ペア Spearman 相関行列 ---")
print(f"{'':&gt;16s}", end='')
for vn, _ in vars_list[:5]:
    print(f"  {vn:&gt;12s}", end='')
print()

for vn1, v1 in vars_list[:5]:
    print(f"{vn1:&gt;16s}", end='')
    for vn2, v2 in vars_list[:5]:
        rho, p = spearmanr(v1, v2)
        sig = '***' if p &lt; 0.001 else '**' if p &lt; 0.01 else '*' if p &lt; 0.05 else ''
        print(f"  {rho:+.3f}{sig:&gt;3s}  ", end='')
    print()

# --- 核心の3つの連鎖 ---
print(f"¥n--- 因果連鎖の検証 ---")

rho_gf_crms, p_gf_crms = spearmanr(gas_f, C_rms)
rho_crms_gamma, p_crms_gamma = spearmanr(C_rms, rh_residual)
rho_gf_gamma, p_gf_gamma = spearmanr(gas_f, rh_residual)
rho_gf_vf, p_gf_vf = spearmanr(gas_f, vf)

print(f"¥n  (a) gas_fraction → C_rms:     ρ={rho_gf_crms:+.3f}, p={p_gf_crms:.2e}")
print(f"  (b) C_rms → γ残差:            ρ={rho_crms_gamma:+.3f}, p={p_crms_gamma:.2e}")
print(f"  (c) gas_fraction → γ残差:      ρ={rho_gf_gamma:+.3f}, p={p_gf_gamma:.2e}")
print(f"  (d) gas_fraction → v_flat:     ρ={rho_gf_vf:+.3f}, p={p_gf_vf:.2e}")

# --- 偏相関: gas_fraction → γ残差, v_flat を制御 ---
# gas_f のv_flat依存を除去
p_gf_vf_fit = np.polyfit(lv, gas_f, 1)
gas_f_residual = gas_f - np.polyval(p_gf_vf_fit, lv)

rho_gf_gamma_partial, p_gf_gamma_partial = spearmanr(gas_f_residual, rh_residual)
print(f"¥n  (e) gas_fraction → γ残差 (v_flat制御後): ρ={rho_gf_gamma_partial:+.3f}, p={p_gf_gamma_partial:.2e}")

# --- C_rms のv_flat依存を除去した偏相関 ---
p_crms_vf = np.polyfit(lv, C_rms, 1)
C_rms_residual = C_rms - np.polyval(p_crms_vf, lv)
rho_crms_gamma_partial, p_crms_gamma_partial = spearmanr(C_rms_residual, rh_residual)
print(f"  (f) C_rms → γ残差 (v_flat制御後):        ρ={rho_crms_gamma_partial:+.3f}, p={p_crms_gamma_partial:.2e}")
# --- 判定 ---
print(f"¥n--- D-3 判定 ---")

chain_a = abs(rho_gf_crms) &gt; 0.15 and p_gf_crms &lt; 0.05
chain_b = abs(rho_crms_gamma) &gt; 0.15 and p_crms_gamma &lt; 0.05
chain_c = abs(rho_gf_gamma) &gt; 0.15 and p_gf_gamma &lt; 0.05

if chain_a and chain_b:
    print("  gas → C_rms → γ残差 の間接連鎖: 確認")
else:
    print("  gas → C_rms → γ残差 の間接連鎖: 不完全")
    if not chain_a:
        print(f"    gas → C_rms が弱い (ρ={rho_gf_crms:.3f})")
    if not chain_b:
        print(f"    C_rms → γ残差 が弱い (ρ={rho_crms_gamma:.3f})")

if chain_c:
    print(f"  gas → γ残差 の直接連鎖: 確認 (ρ={rho_gf_gamma:.3f})")
else:
    print(f"  gas → γ残差 の直接連鎖: 不支持 (ρ={rho_gf_gamma:.3f})")

if p_gf_gamma_partial &lt; 0.05:
    print(f"  gas → γ残差 (v_flat制御後): 有意 → ガスは v_flat 経由でない独立効果を持つ")
else:
    print(f"  gas → γ残差 (v_flat制御後): 非有意 → ガスの効果は v_flat に媒介されている")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) gas_fraction vs C_rms
ax = axes[0, 0]
ax.scatter(gas_f, C_rms, s=8, alpha=0.4, c='teal')
ax.set_xlabel('gas_fraction')
ax.set_ylabel('C_rms')
```

```
ax.set_title(f'(a) gas→C_rms (ρ={rho_gf_crms:.3f})')

# (b) gas_fraction vs γ残差
ax = axes[0, 1]
ax.scatter(gas_f, rh_residual, s=8, alpha=0.4, c='coral')
ax.axhline(0, color='gray', ls='--', alpha=0.3)
ax.set_xlabel('gas_fraction')
ax.set_ylabel('γ residual')
ax.set_title(f'(b) gas→γ residual (ρ={rho_gf_gamma:.3f})')

# (c) C_rms vs γ残差 (確認)
ax = axes[0, 2]
ax.scatter(C_rms, rh_residual, s=8, alpha=0.4, c='purple')
ax.axhline(0, color='gray', ls='--', alpha=0.3)
ax.set_xlabel('C_rms')
ax.set_ylabel('γ residual')
ax.set_title(f'(c) C_rms→γ residual (ρ={rho_crms_gamma:.3f})')

# (d) D-1: h_R vs v_flat (β の確認)
ax = axes[1, 0]
ax.scatter(lv, lhR, s=8, alpha=0.4, c='orange')
vv = np.linspace(lv.min(), lv.max(), 100)
ax.plot(vv, np.polyval(p_beta, vv), 'k-', label=f'All: β={p_beta[0]:.3f} (N={N})')
if 'p_b_in' in dir():
    ax.plot(vv, np.polyval(p_b_in, vv), 'b--', label=f'In-range: β={p_b_in[0]:.3f} (N={N_in})')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(h_R)')
ax.set_title('(d) D-1: β reconciliation')
ax.legend(fontsize=8)

# (e) α分解の確定版
ax = axes[1, 1]
labels_bar = ['Step2¥n(reported)', 'Step3rev¥n(reported)', f'Unified¥n(N={N})']
betas_bar = [0.600, 0.537, p_beta[0]]
gammas_bar = [0.284, 0.112, p_gamma[0]]
x = np.arange(len(labels_bar))
ax.bar(x, betas_bar, 0.4, label='β', color='steelblue')
ax.bar(x, gammas_bar, 0.4, bottom=betas_bar, label='γ', color='coral')
for i in range(len(labels_bar)):
    ax.text(i, betas_bar[i]+gammas_bar[i]+0.01, f'α={betas_bar[i]+gammas_bar[i]:.3f}',
            ha='center', fontsize=9)
ax.set_xticks(x)
ax.set_xticklabels(labels_bar, fontsize=8)
ax.set_ylabel('Exponent')
ax.set_title('(e) α decomposition reconciliation')
ax.legend(fontsize=8)

# (f) 因果連鎖のパス図 (テキスト)
ax = axes[1, 2]
ax.axis('off')
ax.set_xlim(0, 10)
ax.set_ylim(0, 10)

ax.text(1, 8, 'gas_fraction', fontsize=11, ha='center',
        bbox=dict(boxstyle='round', facecolor='lightblue'))
ax.text(5, 8, 'C_rms', fontsize=11, ha='center',
        bbox=dict(boxstyle='round', facecolor='lightyellow'))
ax.text(9, 8, 'γ残差', fontsize=11, ha='center',
        bbox=dict(boxstyle='round', facecolor='lightcoral'))
ax.text(5, 4, 'v_flat', fontsize=11, ha='center',
        bbox=dict(boxstyle='round', facecolor='lightgreen'))

# arrows with correlation values
ax.annotate('', xy=(3.5, 8), xytext=(2.2, 8),
            arrowprops=dict(arrowstyle='-&gt;', color='black'))
ax.text(2.8, 8.5, f'ρ={rho_gf_crms:+.2f}', fontsize=9, ha='center')

ax.annotate('', xy=(7.5, 8), xytext=(6.2, 8),
            arrowprops=dict(arrowstyle='-&gt;', color='black'))
ax.text(6.8, 8.5, f'ρ={rho_crms_gamma:+.2f}', fontsize=9, ha='center')

ax.annotate('', xy=(8.5, 7.3), xytext=(1.5, 7.3),
            arrowprops=dict(arrowstyle='-&gt;', color='gray', ls='dashed'))
ax.text(5, 6.8, f'direct: ρ={rho_gf_gamma:+.2f}', fontsize=9, ha='center', color='gray')

ax.annotate('', xy=(1, 5), xytext=(4.5, 4.5),
            arrowprops=dict(arrowstyle='-&gt;', color='green'))
ax.text(2.2, 4.5, f'ρ={rho_gf_vf:+.2f}', fontsize=8, ha='center', color='green')

ax.set_title('(f) Causal path diagram')

plt.tight_layout()
plt.savefig('D3_D1_fix.png', dpi=150)
print(f"¥n[SAVED] D3_D1_fix.png")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("D-1 / D-3 修正結果")
print(f"{'='*70}")

print(f"""
■ D-1: β の統一値 (同一サンプル N={N}、同一手法)
```

```
  β = {p_beta[0]:.4f}
  γ = {p_gamma[0]:.4f}
  α = β + γ = {p_beta[0]+p_gamma[0]:.4f}  (直接フィット α = {p_alpha[0]:.4f})
""")

if 'p_b_in' in dir():
    print(f"""  非外挿のみ (N={N_in}):
  β = {p_b_in[0]:.4f}
  γ = {p_g_in[0]:.4f}
  α = {p_b_in[0]+p_g_in[0]:.4f}
""")

print(f"""  Step 2 報告 β=0.600 vs Step 3 修正 β=0.537 の差 0.063 の原因:
  サンプルフィルタリング（非外挿 vs 全銀河）と
  h_R ピーク検出精度の差。
  統一サンプルでの確定値: β = {p_beta[0]:.4f}

■ D-3: gas → C_rms → γ残差 の因果連鎖

  gas → C_rms:      ρ={rho_gf_crms:+.3f} ({'有意' if chain_a else '非有意'})
  C_rms → γ残差:    ρ={rho_crms_gamma:+.3f} ({'有意' if chain_b else '非有意'})
  gas → γ残差:      ρ={rho_gf_gamma:+.3f} ({'有意' if chain_c else '非有意'})
  gas → γ残差(v制御): ρ={rho_gf_gamma_partial:+.3f} (p={p_gf_gamma_partial:.2e})
""")

if chain_a and chain_b and chain_c:
    print("  → 因果連鎖 gas → C_rms → γ残差 は全リンクが有意。")
    print("    「ガスが γ の主因」は直接的な統計的証拠あり。")
elif chain_c:
    print("  → gas → γ残差 の直接連鎖は有意だが、C_rms 経由は不完全。")
    print("    ガスの効果は C_rms (Freeman逸脱度) を必ずしも経由しない。")
elif chain_a and chain_b:
    print("  → 間接連鎖 gas → C_rms → γ残差 は有意だが直接連鎖は弱い。")
else:
    print("  → 「ガスが γ の主因」の統計的証拠は不十分。")
    print("    γ の起源は未特定のまま。")
```

12. rs1_validity.py

項目	内容
フェーズ	rs1/rs2 問題
目的	rs1 の物理的有意性の検証（rs1/rs2 問題の発見）
使用rs	rs1
結果	rs1 alpha ~= 0.3-0.6（フィルタ依存）。rs2 の 0.795 との差は 2段階フィットの効果。115/175銀河で |rs2-rs1| > 0.1 kpc。
ステータス	Level A（rs1/rs2問題の確立）

解析目的

sparc_results.csv の rs1（alpha=0.294）と rs2（alpha=0.795）の劇的な差の原因を調査。rs1 のフィルタ別スケーリングを計算。

ソースコード全文

```
#!/usr/bin/env python3
"""
rs1 の物理的有意性の検証

rs1 ≈ 0（下限張り付き）の銀河を除外し、
遷移半径が実際にデータで制約されている銀河だけで
α = d log(rs1) / d log(v_flat) を再計算する。

フィルタ基準:
  (A) rs1 &gt; 2 × r_min（最小半径の2倍以上）
  (B) rs1 &gt; 5 × r_min（より厳密）
  (C) rs1 &gt; 0.1 × R_max（データ範囲の10%以上）
  (D) rs1 &gt; 0.5 kpc（絶対値カット）

実行: uv run --with scipy --with matplotlib --with numpy python rs1_validity.py
"""

import csv
import os
import numpy as np
from scipy.stats import spearmanr, pearsonr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# データ読み込み
# ============================================================
def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

src_cols, src_rows = load_csv("sparc_results.csv")

results = []
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    rs1 = get_val(row, ['rs1'])
    rs2 = get_val(row, ['rs2'])
    vf = get_val(row, ['vflat'])
    ud = get_val(row, ['ud', 'upsilon_d'])
    chi2_1 = get_val(row, ['chi2_1s'])
    chi2_2 = get_val(row, ['chi2_2s'])
    daic = get_val(row, ['daic'])

    if rs1 is None or vf is None or vf &lt;= 0:
        continue

    data = read_dat(name)
    if data is None:
        continue

    rad = data[:, 0]
    r_min = rad[rad &gt; 0].min() if np.any(rad &gt; 0) else 0
```

```
    R_max = rad.max()
    N_pts = len(rad)

    results.append({
        'name': name,
        'rs1': rs1,
        'rs2': rs2 if rs2 else rs1,
        'vflat': vf,
        'upsilon_d': ud if ud else 0.5,
        'r_min': r_min,
        'R_max': R_max,
        'N_pts': N_pts,
        'chi2_1s': chi2_1,
        'chi2_2s': chi2_2,
        'daic': daic,
        'rs1_over_rmin': rs1 / r_min if r_min &gt; 0 else 0,
        'rs1_over_Rmax': rs1 / R_max if R_max &gt; 0 else 0,
    })

N_total = len(results)
print(f"[INFO] Total: {N_total} galaxies")

# 配列化
rs1 = np.array([d['rs1'] for d in results])
rs2 = np.array([d['rs2'] for d in results])
vf = np.array([d['vflat'] for d in results])
r_min = np.array([d['r_min'] for d in results])
R_max = np.array([d['R_max'] for d in results])
rs1_rmin = np.array([d['rs1_over_rmin'] for d in results])
rs1_Rmax = np.array([d['rs1_over_Rmax'] for d in results])

# ============================================================
# rs1 の分布診断
# ============================================================
print(f"¥n{'='*70}")
print("rs1 の分布診断")
print(f"{'='*70}")

print(f"¥nrs1 の基本統計:")
print(f"  中央値: {np.median(rs1):.3f} kpc")
print(f"  平均:   {np.mean(rs1):.3f} kpc")
print(f"  std:    {np.std(rs1):.3f} kpc")
print(f"  min:    {np.min(rs1):.4f} kpc")
print(f"  max:    {np.max(rs1):.3f} kpc")

print(f"¥nrs1 / r_min の分布:")
print(f"  中央値: {np.median(rs1_rmin):.2f}")
print(f"  rs1 &lt; r_min:     {np.sum(rs1 &lt; r_min):3d} ({100*np.sum(rs1 &lt; r_min)/N_total:.0f}%)")
print(f"  rs1 &lt; 2*r_min:   {np.sum(rs1 &lt; 2*r_min):3d} ({100*np.sum(rs1 &lt; 2*r_min)/N_total:.0f}%)")
print(f"  rs1 &lt; 5*r_min:   {np.sum(rs1 &lt; 5*r_min):3d} ({100*np.sum(rs1 &lt; 5*r_min)/N_total:.0f}%)")

print(f"¥nrs1 / R_max の分布:")
print(f"  中央値: {np.median(rs1_Rmax):.3f}")
print(f"  rs1 &lt; 0.1*R_max: {np.sum(rs1 &lt; 0.1*R_max):3d} ({100*np.sum(rs1 &lt; 0.1*R_max)/N_total:.0f}%)")
print(f"  rs1 &lt; 0.05*R_max:{np.sum(rs1 &lt; 0.05*R_max):3d} ({100*np.sum(rs1 &lt; 0.05*R_max)/N_total:.0f}%)")

print(f"¥n絶対値カット:")
print(f"  rs1 &lt; 0.1 kpc:   {np.sum(rs1 &lt; 0.1):3d} ({100*np.sum(rs1 &lt; 0.1)/N_total:.0f}%)")
print(f"  rs1 &lt; 0.5 kpc:   {np.sum(rs1 &lt; 0.5):3d} ({100*np.sum(rs1 &lt; 0.5)/N_total:.0f}%)")
print(f"  rs1 &lt; 1.0 kpc:   {np.sum(rs1 &lt; 1.0):3d} ({100*np.sum(rs1 &lt; 1.0)/N_total:.0f}%)")

# ============================================================
# 各フィルタでの α 再計算
# ============================================================
print(f"¥n{'='*70}")
print("各フィルタでの α (rs1 vs v_flat)")
print(f"{'='*70}")

filters = {
    'All (no filter)': np.ones(N_total, dtype=bool),
    'rs1 &gt; r_min': rs1 &gt; r_min,
    'rs1 &gt; 2*r_min': rs1 &gt; 2*r_min,
    'rs1 &gt; 5*r_min': rs1 &gt; 5*r_min,
    'rs1 &gt; 10*r_min': rs1 &gt; 10*r_min,
    'rs1 &gt; 0.05*R_max': rs1 &gt; 0.05*R_max,
    'rs1 &gt; 0.1*R_max': rs1 &gt; 0.1*R_max,
    'rs1 &gt; 0.2*R_max': rs1 &gt; 0.2*R_max,
    'rs1 &gt; 0.1 kpc': rs1 &gt; 0.1,
    'rs1 &gt; 0.5 kpc': rs1 &gt; 0.5,
    'rs1 &gt; 1.0 kpc': rs1 &gt; 1.0,
    'rs1 &gt; 2.0 kpc': rs1 &gt; 2.0,
}

print(f"¥n{'フィルタ':&gt;25s}  {'N':&gt;4s}  {'α(rs1)':&gt;8s}  {'r':&gt;6s}  {'p':&gt;10s}  {'rs1中央値':&gt;10s}  {'有意':&gt;4s}")
print("-"*80)

filter_results = {}
for fname, mask in filters.items():
    N_f = mask.sum()
    if N_f &lt; 10:
        print(f"{fname:&gt;25s}  {N_f:4d}  {'---':&gt;8s}")
        continue

    lv_f = np.log10(vf[mask])
```

```
    lrs_f = np.log10(rs1[mask])

    p_f = np.polyfit(lv_f, lrs_f, 1)
    r_f, p_val = pearsonr(lv_f, lrs_f)
    rho_f, p_spear = spearmanr(vf[mask], rs1[mask])

    sig = '***' if p_val &lt; 0.001 else '**' if p_val &lt; 0.01 else '*' if p_val &lt; 0.05 else ''
    print(f"{fname:&gt;25s}  {N_f:4d}  {p_f[0]:8.3f}  {r_f:6.3f}  {p_val:10.2e}  {np.median(rs1[mask]):10.3f}  {sig:&gt;4s}")

    filter_results[fname] = {
        'N': N_f, 'alpha': p_f[0], 'r': r_f, 'p': p_val,
        'rho': rho_f, 'mask': mask,
    }

# ============================================================
# rs1 ≈ 0 銀河 vs rs1 有意銀河の性質比較
# ============================================================
print(f"¥n{'='*70}")
print("rs1 ≈ 0 銀河 vs rs1 有意銀河の比較")
print(f"{'='*70}")

mask_low = rs1 &lt; 0.5  # 下限張り付き
mask_high = rs1 &gt; 2.0  # 明確に有意

print(f"¥n{'':&gt;20s}  {'rs1&lt;0.5 (低)':&gt;15s}  {'rs1&gt;2.0 (高)':&gt;15s}")
print(f"{'N':&gt;20s}  {mask_low.sum():15d}  {mask_high.sum():15d}")
print(f"{'v_flat 中央値':&gt;20s}  {np.median(vf[mask_low]):15.1f}  {np.median(vf[mask_high]):15.1f}")
print(f"{'R_max 中央値':&gt;20s}  {np.median(R_max[mask_low]):15.1f}  {np.median(R_max[mask_high]):15.1f}")
print(f"{'N_pts 中央値':&gt;20s}  {np.median([d['N_pts'] for d, m in zip(results, mask_low) if m]):15.0f}  {np.median([d['N_pts'] for d, m in zip(results, mask_high) if m]):15.0f}")
print(f"{'Y_d 中央値':&gt;20s}  {np.median([d['upsilon_d'] for d, m in zip(results, mask_low) if m]):15.3f}  {np.median([d['upsilon_d'] for d, m in zip(results, mask_high) if m]):15.3f}")

# rs1 低い銀河で rs2 がジャンプするか
rs2_low = rs2[mask_low]
rs1_low = rs1[mask_low]
jump = rs2_low / np.maximum(rs1_low, 0.001)
print(f"¥nrs1&lt;0.5 銀河での rs2/rs1:")
print(f"  中央値: {np.median(jump):.1f}")
print(f"  rs2/rs1 &gt; 10: {np.sum(jump &gt; 10)} 銀河")
print(f"  rs2/rs1 &gt; 100: {np.sum(jump &gt; 100)} 銀河")

# ============================================================
# rs2 のスケーリング（比較用）
# ============================================================
print(f"¥n{'='*70}")
print("rs2 のスケーリング（比較）")
print(f"{'='*70}")

mask_rs2 = rs2 &gt; 0
lrs2 = np.log10(rs2[mask_rs2])
lv2 = np.log10(vf[mask_rs2])
p_rs2 = np.polyfit(lv2, lrs2, 1)
r_rs2, p_rs2_p = pearsonr(lv2, lrs2)
print(f"  rs2: α={p_rs2[0]:.3f}, r={r_rs2:.3f}, p={p_rs2_p:.2e}")

# rs2 にも同じフィルタを適用
for fname in ['rs1 &gt; 0.5 kpc', 'rs1 &gt; 2.0 kpc']:
    if fname in filter_results:
        mask = filter_results[fname]['mask']
        N_f = mask.sum()
        lv_f = np.log10(vf[mask])
        lrs2_f = np.log10(rs2[mask])
        p_f2 = np.polyfit(lv_f, lrs2_f, 1)
        r_f2, p_f2p = pearsonr(lv_f, lrs2_f)
        print(f"  rs2 ({fname}): α={p_f2[0]:.3f}, r={r_f2:.3f}, N={N_f}")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) rs1 のヒストグラム（log スケール）
ax = axes[0, 0]
ax.hist(np.log10(np.maximum(rs1, 0.001)), bins=50, color='steelblue', edgecolor='white')
ax.axvline(np.log10(0.5), color='red', ls='--', label='0.5 kpc')
ax.axvline(np.log10(2.0), color='orange', ls='--', label='2.0 kpc')
ax.set_xlabel('log(rs1) [kpc]')
ax.set_ylabel('Count')
ax.set_title(f'(a) rs1 distribution (N={N_total})')
ax.legend(fontsize=8)

# (b) rs1 vs v_flat（全銀河 + フィルタ後）
ax = axes[0, 1]
ax.scatter(np.log10(vf), np.log10(np.maximum(rs1, 0.001)),
           s=8, alpha=0.3, c='gray', label='All')
if 'rs1 &gt; 2.0 kpc' in filter_results:
    m = filter_results['rs1 &gt; 2.0 kpc']['mask']
    ax.scatter(np.log10(vf[m]), np.log10(rs1[m]),
               s=15, alpha=0.6, c='blue', label='rs1&gt;2 kpc')
    # フィット線
    lv_f = np.log10(vf[m])
    lrs_f = np.log10(rs1[m])
    p_f = np.polyfit(lv_f, lrs_f, 1)
    vv = np.linspace(lv_f.min(), lv_f.max(), 100)
    ax.plot(vv, np.polyval(p_f, vv), 'b-',
            label=f'α={p_f[0]:.3f}')
```

```
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(rs1)')
ax.set_title('(b) rs1 vs v_flat')
ax.legend(fontsize=7)

# (c) rs1/r_min vs v_flat
ax = axes[0, 2]
ax.scatter(vf, rs1_rmin, s=8, alpha=0.4, c='teal')
ax.axhline(1.0, color='red', ls='--', alpha=0.5, label='rs1=r_min')
ax.axhline(2.0, color='orange', ls='--', alpha=0.5, label='rs1=2*r_min')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('rs1 / r_min')
ax.set_yscale('log')
ax.set_title('(c) rs1 resolution check')
ax.legend(fontsize=8)

# (d) α vs フィルタ閾値
ax = axes[1, 0]
thresholds = []
alphas = []
ns = []
for fname, res in filter_results.items():
    if 'kpc' in fname:
        try:
            thr = float(fname.split('&gt;')[1].strip().split()[0])
            thresholds.append(thr)
            alphas.append(res['alpha'])
            ns.append(res['N'])
        except:
            pass
if thresholds:
    sort_idx = np.argsort(thresholds)
    thresholds = np.array(thresholds)[sort_idx]
    alphas = np.array(alphas)[sort_idx]
    ns = np.array(ns)[sort_idx]
    ax.plot(thresholds, alphas, 'bo-')
    for i in range(len(thresholds)):
        ax.text(thresholds[i], alphas[i]+0.02, f'N={ns[i]}', fontsize=8, ha='center')
    ax.axhline(0, color='gray', ls='--', alpha=0.3)
    ax.axhline(0.795, color='red', ls='--', alpha=0.5, label='rs2 α=0.795')
ax.set_xlabel('rs1 threshold [kpc]')
ax.set_ylabel('α (rs1 vs v_flat)')
ax.set_title('(d) α vs filter threshold')
ax.legend(fontsize=8)

# (e) rs1 vs rs2
ax = axes[1, 1]
ax.scatter(rs1, rs2, s=8, alpha=0.4, c='purple')
maxv = max(rs1.max(), rs2.max())
ax.plot([0, maxv], [0, maxv], 'k--', alpha=0.3, label='rs1=rs2')
ax.set_xlabel('rs1 [kpc]')
ax.set_ylabel('rs2 [kpc]')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title('(e) rs1 vs rs2')
ax.legend(fontsize=8)

# (f) rs1/R_max のヒストグラム
ax = axes[1, 2]
ax.hist(rs1_Rmax, bins=50, color='coral', edgecolor='white')
ax.axvline(0.1, color='red', ls='--', label='10% of R_max')
ax.axvline(0.5, color='blue', ls='--', label='50% of R_max')
ax.set_xlabel('rs1 / R_max')
ax.set_ylabel('Count')
ax.set_title('(f) rs1 position in data range')
ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('rs1_validity.png', dpi=150)
print(f"¥n[SAVED] rs1_validity.png")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("rs1 の物理的有意性の検証結果")
print(f"{'='*70}")

# 最も厳しいフィルタでまだ有意なサンプルがあるか
best_filter = None
for fname in ['rs1 &gt; 2.0 kpc', 'rs1 &gt; 1.0 kpc', 'rs1 &gt; 0.5 kpc']:
    if fname in filter_results and filter_results[fname]['p'] &lt; 0.05:
        best_filter = fname
        break

if best_filter:
    res = filter_results[best_filter]
    print(f"""
■ rs1 に物理的情報がある銀河サブセット:

  フィルタ: {best_filter}
  N = {res['N']}
  α = {res['alpha']:.3f}  (r={res['r']:.3f}, p={res['p']:.2e})

  → rs1 が十分大きい銀河では v_flat との相関が存在する
```

```
""")
else:
    print(f"""
■ rs1 には v_flat との有意な相関なし

  すべてのフィルタで p &gt; 0.05
  → 1段階 tanh の遷移半径は v_flat に対して物理的情報を持たない
""")

print(f"""
■ rs1 ≈ 0 銀河の解釈:

  rs1 &lt; 0.5 kpc: {np.sum(rs1 &lt; 0.5)} 銀河 ({100*np.sum(rs1 &lt; 0.5)/N_total:.0f}%)
  → tanh 遷移がデータ範囲外 (r_s &lt; r_min)
  → v_c²(r) ≈ v_bar²(r) + v_flat² (定数加算モデル)
  → 膜展開は最内半径以下で完了済み

■ 引き継ぎメモへの影響:

  Level A (1) 「r_s ∝ v_flat^0.795」→ 撤回
    rs2 のスケーリングであり、rs2 の物理的意味が確立されていない

  代替の記述:
    rs1 (1段階 tanh) は v_flat とほぼ無相関 (α≈0.3, r≈0.19)
    rs1 の中央値 ≈ 2.5 kpc は銀河サイズに弱く依存する準普遍的スケール
    rs2 は v_flat と相関するが、フィッティングの人工相関の可能性が排除されていない
""")
```

13. V1_rs_check.py

項目	内容
フェーズ	V-1/V-2
目的	各CSVファイルが使用した r_s 列の特定
使用rs	なし（診断スクリプト）
結果	f_tanh, step2, TA2 -> rs2使用。TA3 -> rs1使用（但しRARはrs不使用で安全）。
ステータス	Level A（汚染範囲の確定）

解析目的

下流の全CSV（f_tanh_detail, TA2, TA3等）の r_s_tanh 列がrs1 と rs2 のどちらに一致するかを照合。汚染範囲を確定。

ソースコード全文

```
#!/usr/bin/env python3
"""
V-1 / V-2: 各スクリプトで使われた r_s の特定

f_tanh_robustness_detail.csv の r_s_tanh は rs1 か rs2 か？
TA2_three_radii.csv の r_s_tanh は rs1 か rs2 か？

実行: uv run --with scipy python V1_rs_check.py
"""

import csv
import os

def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

# ============================================================
# sparc_results.csv のrs1, rs2 を取得
# ============================================================
_, src_rows = load_csv("sparc_results.csv")
src = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name:
        name = list(row.values())[0].strip()
    try:
        rs1 = float(row.get('rs1', '0'))
        rs2 = float(row.get('rs2', '0'))
        src[name] = {'rs1': rs1, 'rs2': rs2}
    except:
        pass

print(f"sparc_results.csv: {len(src)} galaxies")

# ============================================================
# V-1: f_tanh_robustness_detail.csv
# ============================================================
print("¥n" + "="*70)
print("V-1: f_tanh_robustness_detail.csv")
print("="*70)

if os.path.exists("f_tanh_robustness_detail.csv"):
    cols_f, rows_f = load_csv("f_tanh_robustness_detail.csv")
    print(f"Columns: {cols_f}")

    match_rs1 = 0
    match_rs2 = 0
    match_neither = 0
    examples = []

    for row in rows_f:
        gal = row.get('galaxy', '').strip()
        try:
            rs_used = float(row.get('r_s_tanh', '0'))
        except:
            continue

        if gal in src:
            rs1 = src[gal]['rs1']
            rs2 = src[gal]['rs2']

            diff1 = abs(rs_used - rs1)
            diff2 = abs(rs_used - rs2)

            if diff1 &lt; 0.001:
                match_rs1 += 1
            elif diff2 &lt; 0.001:
                match_rs2 += 1
            else:
```

```
                match_neither += 1

            if len(examples) &lt; 10:
                examples.append({
                    'name': gal,
                    'rs_used': rs_used,
                    'rs1': rs1,
                    'rs2': rs2,
                    'diff1': diff1,
                    'diff2': diff2,
                })

    print(f"¥nマッチ結果:")
    print(f"  rs1 と一致: {match_rs1}")
    print(f"  rs2 と一致: {match_rs2}")
    print(f"  どちらとも不一致: {match_neither}")

    print(f"¥nサンプル比較 (先頭10銀河) :")
    print(f"{'Galaxy':&gt;15s}  {'rs_used':&gt;8s}  {'rs1':&gt;8s}  {'rs2':&gt;8s}  {'|diff1|':&gt;8s}  {'|diff2|':&gt;8s}  {'match':&gt;6s}")
    for ex in examples:
        m = 'rs1' if ex['diff1'] &lt; 0.001 else ('rs2' if ex['diff2'] &lt; 0.001 else '???')
        print(f"{ex['name']:&gt;15s}  {ex['rs_used']:8.3f}  {ex['rs1']:8.3f}  {ex['rs2']:8.3f}  {ex['diff1']:8.4f}  {ex['diff2']:8.4f}  {m:&gt;6s}")
else:
    print("f_tanh_robustness_detail.csv not found")

# ============================================================
# V-2: TA2_three_radii.csv
# ============================================================
print("¥n" + "="*70)
print("V-2: TA2_three_radii.csv")
print("="*70)

if os.path.exists("TA2_three_radii.csv"):
    cols_t, rows_t = load_csv("TA2_three_radii.csv")
    print(f"Columns: {cols_t}")

    match_rs1_t = 0
    match_rs2_t = 0
    match_neither_t = 0
    examples_t = []

    for row in rows_t:
        gal = row.get('galaxy', '').strip()
        try:
            rs_used = float(row.get('r_s_tanh', '0'))
        except:
            continue

        if gal in src:
            rs1 = src[gal]['rs1']
            rs2 = src[gal]['rs2']
            diff1 = abs(rs_used - rs1)
            diff2 = abs(rs_used - rs2)

            if diff1 &lt; 0.001:
                match_rs1_t += 1
            elif diff2 &lt; 0.001:
                match_rs2_t += 1
            else:
                match_neither_t += 1

            if len(examples_t) &lt; 5:
                examples_t.append({
                    'name': gal, 'rs_used': rs_used,
                    'rs1': rs1, 'rs2': rs2,
                })

    print(f"¥nマッチ結果:")
    print(f"  rs1 と一致: {match_rs1_t}")
    print(f"  rs2 と一致: {match_rs2_t}")
    print(f"  どちらとも不一致: {match_neither_t}")

    if examples_t:
        print(f"¥nサンプル:")
        for ex in examples_t:
            print(f"  {ex['name']:&gt;15s}: used={ex['rs_used']:.3f}, rs1={ex['rs1']:.3f}, rs2={ex['rs2']:.3f}")
else:
    print("TA2_three_radii.csv not found")

# ============================================================
# TA3 確認 (r_sを使わないことの確認)
# ============================================================
print("¥n" + "="*70)
print("V-2 補足: TA3_gc_independent.csv")
print("="*70)

if os.path.exists("TA3_gc_independent.csv"):
    cols_g, rows_g = load_csv("TA3_gc_independent.csv")
    print(f"Columns: {cols_g}")
    if 'rs_tanh' in cols_g or 'r_s_tanh' in cols_g or 'r_s' in cols_g:
        print("  → r_s 列が存在するが、RARフィット自体には使用していない (比較用のみ) ")
        # 念のため確認
        rs_col = 'rs_tanh' if 'rs_tanh' in cols_g else 'r_s_tanh' if 'r_s_tanh' in cols_g else 'r_s'
        match1, match2 = 0, 0
```

```
        for row in rows_g[:20]:
            gal = row.get('galaxy', '').strip()
            try:
                rs_used = float(row.get(rs_col, '0'))
            except:
                continue
            if gal in src:
                if abs(rs_used - src[gal]['rs1']) &lt; 0.001: match1 += 1
                elif abs(rs_used - src[gal]['rs2']) &lt; 0.001: match2 += 1
        print(f"  参考列の内容: rs1一致={match1}, rs2一致={match2} (先頭20銀河)")
    else:
        print("  → r_s 列なし (安全) ")
else:
    print("TA3_gc_independent.csv not found")

# ============================================================
# 結論
# ============================================================
print("¥n" + "="*70)
print("V-1 / V-2 結論")
print("="*70)

print(f"""
■ f_tanh_robustness_detail.csv:
  rs1一致: {match_rs1}, rs2一致: {match_rs2}, 不一致: {match_neither}
""")

if match_rs2 &gt; match_rs1:
    print("  → rs2 を使用。f^(tanh)=0.29 は rs2 ベースの結果。")
    print("     rs1≈0 の銀河では f が非物理的な値を取っている可能性は低い")
    print("      (rs2 は物理的範囲内のため) 。")
    print("     ただし rs2 自体の信頼性が未確立 → f の結果も Level B に格下げ。")
elif match_rs1 &gt; match_rs2:
    print("  → rs1 を使用。f^(tanh)=0.29 は rs1 ベース。")
    print("     rs1≈0 の銀河では f = v_bar²(~0)/v_flat² → 非物理的。")
    print("     → f=0.29 とU字型の信頼性に重大な影響。")
    print("     → rs1有意銀河のみで再計算が必要。")
else:
    print("  → 判定不能。手動で確認が必要。")
```

14. V3_rs1_interpretation.py

項目	内容
フェーズ	V-3
目的	rs1 ˜= 0 は下限張り付きか物理的ゼロかの判定
使用rs	rs1
結果	12/14 (86%) が局所解。真の極小は rs_best ˜= 1.9 kpc。
ステータス	Level A (rs1局所解問題の確立)

解析目的

rs1 < 0.5 kpc の銀河で chi2(rs) プロファイルをスキャンし、rs -> 0 が真の最適値か局所解かを分類。

ソースコード全文

```
#!/usr/bin/env python3
"""
V-3: rs1 ≈ 0 は「下限張り付き」か「物理的にゼロ」か

方法:
  rs1 &lt; 0.5 kpc の銀河に対して chi2(rs) のプロファイルを計算。
  rs を 0.001 ˜ R_max の範囲で系統的に変え、各点での chi2 を評価。

判定:
  chi2 が rs→0 で単調減少 → 解釈A (下限張り付き、真の最適値は rs⊠0)
  chi2 が rs≈0 で極小      → 解釈B (物理的にゼロ、遷移は最内側で完了)
  chi2 が rs&gt;0 で別の極小  → 解釈C (rs1 のフィッターが局所解に嵌まった)

実行: uv run --with scipy --with matplotlib --with numpy python V3_rs1_interpretation.py
"""

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

# ============================================================
# データ読み込み
# ============================================================
def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

_, src_rows = load_csv("sparc_results.csv")
galaxies = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    rs1 = get_val(row, ['rs1'])
    rs2 = get_val(row, ['rs2'])
    vf = get_val(row, ['vflat'])
    ud = get_val(row, ['ud'])
    if rs1 is not None and vf and ud:
        galaxies[name] = {'rs1': rs1, 'rs2': rs2, 'vflat': vf, 'upsilon_d': ud}

print(f"[INFO] {len(galaxies)} galaxies")

# ============================================================
# chi2 プロファイル計算
# ============================================================

def compute_chi2_profile(rad, v_obs, err_v, v_bar, rs_grid):
    """
    各 rs に対して v_flat を最適化し、chi2(rs) を返す
    v_c^2(r) = v_bar^2(r) + v_flat^2 * T(r, rs)
    T(r, rs) = 0.5 * [1 + tanh((r-rs)/rs)]
    """
    chi2_list = []
    vflat_list = []
```

```
    for rs in rs_grid:
        if rs &lt;= 0:
            # rs=0: T=1 everywhere → v_c^2 = v_bar^2 + v_flat^2
            T = np.ones_like(rad)
        else:
            T = 0.5 * (1.0 + np.tanh((rad - rs) / rs))

        # v_flat の最適値: min Σ((v_obs - v_model)^2 / err^2)
        # v_model = sqrt(v_bar^2 + vf^2 * T)
        # 1Dなのでグリッドサーチ
        best_chi2 = np.inf
        best_vf = 0

        for vf_try in np.linspace(1, max(v_obs)*1.5, 200):
            v_model = np.sqrt(v_bar**2 + vf_try**2 * T)
            weights = 1.0 / np.maximum(err_v, 1.0)**2
            chi2 = np.sum(weights * (v_obs - v_model)**2)
            if chi2 &lt; best_chi2:
                best_chi2 = chi2
                best_vf = vf_try

        chi2_list.append(best_chi2)
        vflat_list.append(best_vf)

    return np.array(chi2_list), np.array(vflat_list)


# ============================================================
# 銀河の選別と分析
# ============================================================

# rs1 &lt; 0.5 kpc の銀河（下限張り付き候補）
low_rs = [(n, g) for n, g in galaxies.items() if g['rs1'] &lt; 0.5]
# rs1 &gt; 3.0 kpc の銀河（比較用）
high_rs = [(n, g) for n, g in galaxies.items() if g['rs1'] &gt; 3.0]

print(f"¥nrs1 &lt; 0.5 kpc: {len(low_rs)} galaxies")
print(f"rs1 &gt; 3.0 kpc: {len(high_rs)} galaxies")

# ============================================================
# 全低rs銀河の chi2 プロファイルの分類
# ============================================================
print(f"¥n{'='*70}")
print("rs1 &lt; 0.5 kpc 銀河の chi2 プロファイル分類")
print(f"{'='*70}")

classification = {'monotone_decrease': [], 'minimum_at_zero': [],
                  'minimum_elsewhere': [], 'flat': [], 'failed': []}

all_profiles = []

for name, gal in low_rs:
    data = read_dat(name)
    if data is None:
        classification['failed'].append(name)
        continue

    rad = data[:, 0]
    v_obs = data[:, 1]
    err_v = data[:, 2]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = gal['upsilon_d']

    vbar2 = ud * np.sign(v_disk)*v_disk**2 + ¥
            np.sign(v_gas)*v_gas**2 + ¥
            np.sign(v_bul)*v_bul**2
    v_bar = np.sqrt(np.maximum(vbar2, 0.0))

    r_min = rad[rad &gt; 0].min() if np.any(rad &gt; 0) else 0.01
    R_max = rad.max()

    # rs グリッド: 0.001 ~ R_max
    rs_grid = np.concatenate([
        [0.001, 0.005, 0.01, 0.02, 0.05],
        np.linspace(0.1, min(R_max, 50), 80)
    ])
    rs_grid = np.sort(np.unique(rs_grid))

    chi2_prof, vf_prof = compute_chi2_profile(rad, v_obs, err_v, v_bar, rs_grid)

    # 分類
    i_min = np.argmin(chi2_prof)
    rs_best = rs_grid[i_min]
    chi2_min = chi2_prof[i_min]

    # chi2 の最初の5点（rs→0側）の傾き
    if len(chi2_prof) &gt; 5:
        slope_start = chi2_prof[4] - chi2_prof[0]
    else:
        slope_start = 0

    # chi2 の変動幅
```

```
    chi2_range = chi2_prof.max() - chi2_prof.min()
    relative_range = chi2_range / max(chi2_min, 1)

    if rs_best &lt; 0.1:
        if slope_start &lt; -0.1 * chi2_range:
            cat = 'monotone_decrease'
        else:
            cat = 'minimum_at_zero'
    elif relative_range &lt; 0.01:
        cat = 'flat'
    else:
        cat = 'minimum_elsewhere'

    classification[cat].append(name)

    all_profiles.append({
        'name': name,
        'vflat': gal['vflat'],
        'rs1': gal['rs1'],
        'rs2': gal['rs2'],
        'rs_best_profile': rs_best,
        'chi2_min': chi2_min,
        'chi2_at_0': chi2_prof[0],
        'relative_range': relative_range,
        'category': cat,
        'rs_grid': rs_grid,
        'chi2_prof': chi2_prof,
    })

print(f"¥n分類結果:")
for cat, names in classification.items():
    print(f"  {cat:&gt;25s}: {len(names):3d} 銀河")

# ============================================================
# 詳細統計
# ============================================================
print(f"¥n{'='*70}")
print("カテゴリ別の詳細")
print(f"{'='*70}")

for cat in ['monotone_decrease', 'minimum_at_zero', 'minimum_elsewhere', 'flat']:
    entries = [p for p in all_profiles if p['category'] == cat]
    if not entries:
        continue
    print(f"¥n--- {cat} (N={len(entries)}) ---")
    vf_arr = np.array([e['vflat'] for e in entries])
    rs1_arr = np.array([e['rs1'] for e in entries])
    rs_best_arr = np.array([e['rs_best_profile'] for e in entries])
    chi2_0 = np.array([e['chi2_at_0'] for e in entries])
    chi2_m = np.array([e['chi2_min'] for e in entries])

    print(f"  v_flat 中央値: {np.median(vf_arr):.1f} km/s")
    print(f"  rs1 中央値:    {np.median(rs1_arr):.4f} kpc")
    print(f"  rs_best 中央値: {np.median(rs_best_arr):.3f} kpc")
    print(f"  chi2(rs=0) / chi2_min 中央値: {np.median(chi2_0/np.maximum(chi2_m, 0.1)):.4f}")

    if len(entries) &lt;= 10:
        for e in entries:
            print(f"    {e['name']:&gt;15s}: rs1={e['rs1']:.3f}, rs_best={e['rs_best_profile']:.3f}, "
                  f"vflat={e['vflat']:.0f}, chi2_range={e['relative_range']:.4f}")

# ============================================================
# 比較: rs1 &gt; 3 kpc の銀河の chi2 プロファイル
# ============================================================
print(f"¥n{'='*70}")
print("比較: rs1 &gt; 3.0 kpc の銀河 (5例) ")
print(f"{'='*70}")

high_profiles = []
for name, gal in high_rs[:5]:
    data = read_dat(name)
    if data is None:
        continue
    rad = data[:, 0]
    v_obs = data[:, 1]
    err_v = data[:, 2]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = gal['upsilon_d']

    vbar2 = ud * np.sign(v_disk)*v_disk**2 + ¥
            np.sign(v_gas)*v_gas**2 + ¥
            np.sign(v_bul)*v_bul**2
    v_bar = np.sqrt(np.maximum(vbar2, 0.0))

    R_max = rad.max()
    rs_grid = np.linspace(0.01, min(R_max, 50), 100)
    chi2_prof, vf_prof = compute_chi2_profile(rad, v_obs, err_v, v_bar, rs_grid)

    i_min = np.argmin(chi2_prof)
    print(f"  {name:&gt;15s}: rs1={gal['rs1']:.2f}, rs_best={rs_grid[i_min]:.2f}, "
          f"chi2_range/min={chi2_prof.ptp()/max(chi2_prof[i_min],0.1):.3f}")

    high_profiles.append({
```

```
        'name': name, 'rs_grid': rs_grid, 'chi2_prof': chi2_prof,
        'rs1': gal['rs1'], 'vflat': gal['vflat'],
    })

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) 低rs銀河の chi2 プロファイル (個別、最大15本)
ax = axes[0, 0]
for p in all_profiles[:15]:
    chi2_norm = p['chi2_prof'] / max(p['chi2_prof'].min(), 0.1)
    ax.plot(p['rs_grid'], chi2_norm, alpha=0.4, linewidth=0.8)
ax.set_xlabel('r_s [kpc]')
ax.set_ylabel('χ² / χ²_min')
ax.set_title(f'(a) χ² profiles (rs1&lt;0.5, N={len(all_profiles)})')
ax.set_xlim(0, 20)
ax.set_ylim(0.9, 2.0)

# (b) 高rs銀河の chi2 プロファイル (比較)
ax = axes[0, 1]
for p in high_profiles:
    chi2_norm = p['chi2_prof'] / max(p['chi2_prof'].min(), 0.1)
    ax.plot(p['rs_grid'], chi2_norm, alpha=0.7, linewidth=1,
            label=f"{p['name']} (rs1={p['rs1']:.1f})")
ax.set_xlabel('r_s [kpc]')
ax.set_ylabel('χ² / χ²_min')
ax.set_title('(b) χ² profiles (rs1&gt;3, comparison)')
ax.set_xlim(0, 30)
ax.set_ylim(0.9, 3.0)
ax.legend(fontsize=6)

# (c) カテゴリ分布 (円グラフ)
ax = axes[0, 2]
cats = ['monotone_decrease', 'minimum_at_zero', 'minimum_elsewhere', 'flat', 'failed']
counts = [len(classification[c]) for c in cats]
labels = ['Monotone↓', 'Min@0', 'Min elsewhere', 'Flat', 'Failed']
colors = ['red', 'orange', 'green', 'gray', 'lightgray']
nonzero = [(l, c, co) for l, c, co in zip(labels, counts, colors) if c &gt; 0]
if nonzero:
    ax.pie([x[1] for x in nonzero], labels=[x[0] for x in nonzero],
           colors=[x[2] for x in nonzero], autopct='%1.0f%%')
ax.set_title(f'(c) Classification (N={sum(counts)})')

# (d) chi2(rs=0)/chi2_min のヒストグラム
ax = axes[1, 0]
ratios = [p['chi2_at_0']/max(p['chi2_min'], 0.1) for p in all_profiles]
ax.hist(ratios, bins=30, color='steelblue', edgecolor='white')
ax.axvline(1.0, color='red', ls='--', label='χ²(0)=χ²_min')
ax.set_xlabel('χ²(rs=0) / χ²_min')
ax.set_ylabel('Count')
ax.set_title('(d) How much worse is rs=0?')
ax.legend()

# (e) rs_best (profile) vs rs1 (original fit)
ax = axes[1, 1]
rs1_arr = np.array([p['rs1'] for p in all_profiles])
rs_best = np.array([p['rs_best_profile'] for p in all_profiles])
ax.scatter(rs1_arr, rs_best, s=15, alpha=0.5, c='purple')
maxv = max(rs1_arr.max(), rs_best.max(), 1)
ax.plot([0, maxv], [0, maxv], 'k--', alpha=0.3)
ax.set_xlabel('rs1 (original fit) [kpc]')
ax.set_ylabel('rs_best (profile scan) [kpc]')
ax.set_title('(e) Profile scan vs original fit')

# (f) 代表的な銀河3つの詳細プロファイル
ax = axes[1, 2]
# monotone, min@0, min_elsewhere から1つずつ
examples = []
for cat in ['monotone_decrease', 'minimum_at_zero', 'minimum_elsewhere']:
    for p in all_profiles:
        if p['category'] == cat:
            examples.append(p)
            break

colors_ex = ['red', 'orange', 'green']
for p, col in zip(examples, colors_ex):
    chi2_norm = p['chi2_prof'] / max(p['chi2_prof'].min(), 0.1)
    ax.plot(p['rs_grid'], chi2_norm, color=col, linewidth=1.5,
            label=f"{p['name']} ({p['category'][:8]})")
ax.set_xlabel('r_s [kpc]')
ax.set_ylabel('χ² / χ²_min')
ax.set_title('(f) Representative profiles')
ax.set_xlim(0, 25)
ax.legend(fontsize=7)

plt.tight_layout()
plt.savefig('V3_rs1_interpretation.png', dpi=150)
print(f"¥n[SAVED] V3_rs1_interpretation.png")

# ============================================================
# 結論
# ============================================================
print(f"¥n{'='*70}")
```

```
print("V-3 結論")
print(f"{'='*70}")

n_mono = len(classification['monotone_decrease'])
n_min0 = len(classification['minimum_at_zero'])
n_else = len(classification['minimum_elsewhere'])
n_flat = len(classification['flat'])
n_total_low = n_mono + n_min0 + n_else + n_flat

print(f"""
■ rs1 &lt; 0.5 kpc の {n_total_low} 銀河の chi2 プロファイル分類:

  単調減少 (rs→0 で chi2↓) : {n_mono} ({100*n_mono/max(n_total_low,1):.0f}%)
    → 解釈A: フィッターの下限に張り付き。真の最適値は rs≪0。
    → tanh 遷移は不要。v_c² = v_bar² + v_flat² で十分。

  rs≈0 で極小:              {n_min0} ({100*n_min0/max(n_total_low,1):.0f}%)
    → 解釈B: 物理的にゼロ。遷移は最内半径より内側で完了。

  rs&gt;0 に別の極小:           {n_else} ({100*n_else/max(n_total_low,1):.0f}%)
    → 解釈C: rs1 フィッターが局所解に嵌まった。真の rs は 0 より大きい。
    → rs_best と rs2 の関係を確認すべき。

  フラット (chi2 が rs にほぼ依存しない) : {n_flat} ({100*n_flat/max(n_total_low,1):.0f}%)
    → rs は制約されていない。tanh のどの rs でも同等のフィット。

■ 物理的意味:
""")

if n_mono + n_min0 &gt; n_else:
    print(f"  大多数 ({n_mono+n_min0}/{n_total_low}) で rs=0 が最適または最適に近い。")
    print(f"  → これらの銀河では膜遷移がデータ範囲外で完了済み。")
    print(f"  → 1段階 tanh は実質的に「定数加算モデル v²=v_bar²+v_flat²」。")
    print(f"  → rs1 の物理的情報は限定的。")
else:
    print(f"  {n_else}/{n_total_low} の銀河で rs&gt;0 に真の極小がある。")
    print(f"  → rs1 フィッターが局所解に嵌まった可能性。")
    print(f"  → これらの銀河では rs を再フィットする価値がある。")

if n_flat &gt; 0:
    print(f"¥n  フラット ({n_flat}銀河) : chi2 が rs にほぼ依存しない。")
    print(f"  → 回転曲線の形状が tanh 遷移を制約できない。")
```

15. rs_best_all.py

項目	内容
フェーズ	rs_best
目的	全175銀河の chi2 プロファイル大域スキャンで rs_best を決定
使用rs	rs_best（新規生成）
結果	rs_best alpha=0.635（r=0.589）。中央値 2.92 kpc。R_max との相関 rho=0.74。
ステータス	Level B（R_max人工効果に注意）

解析目的

rs1 の局所解問題を回避するため、全銀河で rs を 0.001 ~ R_max の範囲でスキャンし、chi2 の大域的最小点 rs_best を決定。

ソースコード全文

```
#!/usr/bin/env python3
"""
全175銀河の rs_best（chi2プロファイルの大域的最小点）を計算

V-3 で rs1≈0 が局所解と判明。全銀河に拡張し、
rs1 でも rs2 でもなく chi2 の大域的極小 rs_best のスケーリングを確定する。

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

import csv
import os
import numpy as np
from scipy.optimize import minimize_scalar, minimize
from scipy.stats import spearmanr, pearsonr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# データ読み込み
# ============================================================
def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

_, src_rows = load_csv("sparc_results.csv")
galaxies = []
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    rs1 = get_val(row, ['rs1'])
    rs2 = get_val(row, ['rs2'])
    vf = get_val(row, ['vflat'])
    ud = get_val(row, ['ud'])
    if rs1 is not None and rs2 is not None and vf and ud and vf &gt; 0:
        galaxies.append({
            'name': name, 'rs1': rs1, 'rs2': rs2,
            'vflat': vf, 'upsilon_d': ud,
        })

print(f"[INFO] {len(galaxies)} galaxies")

# ============================================================
# chi2 プロファイルスキャン
# ============================================================

def chi2_at_rs(rs, rad, v_obs, err_v, v_bar):
    """固定 rs での最適 v_flat における chi2"""
    if rs &lt;= 0.001:
        T = np.ones_like(rad)
    else:
        T = 0.5 * (1.0 + np.tanh((rad - rs) / rs))

    # v_flat の解析的最適化（線形最小二乗）
    # v_model = sqrt(v_bar^2 + vf^2 * T)
    # 非線形なので v_flat もグリッドサーチ
```

```
    w = 1.0 / np.maximum(err_v, 1.0)**2

    best_chi2 = np.inf
    best_vf = 0

    # 粗いグリッド
    vf_max = max(v_obs.max() * 1.5, 50)
    for vf in np.linspace(1, vf_max, 100):
        v_model = np.sqrt(np.maximum(v_bar**2 + vf**2 * T, 0.01))
        c2 = np.sum(w * (v_obs - v_model)**2)
        if c2 &lt; best_chi2:
            best_chi2 = c2
            best_vf = vf

    # 精密化
    for vf in np.linspace(max(best_vf - vf_max/50, 0.1), best_vf + vf_max/50, 50):
        v_model = np.sqrt(np.maximum(v_bar**2 + vf**2 * T, 0.01))
        c2 = np.sum(w * (v_obs - v_model)**2)
        if c2 &lt; best_chi2:
            best_chi2 = c2
            best_vf = vf

    return best_chi2, best_vf


def find_rs_best(rad, v_obs, err_v, v_bar, R_max):
    """chi2 プロファイルをスキャンして大域的最小の rs を見つける"""
    # 広いグリッド
    rs_grid = np.concatenate([
        np.array([0.001, 0.005, 0.01, 0.02, 0.05, 0.1]),
        np.linspace(0.2, min(R_max * 1.5, 80), 100),
    ])
    rs_grid = np.sort(np.unique(rs_grid))

    chi2_list = []
    vf_list = []
    for rs in rs_grid:
        c2, vf = chi2_at_rs(rs, rad, v_obs, err_v, v_bar)
        chi2_list.append(c2)
        vf_list.append(vf)

    chi2_arr = np.array(chi2_list)
    vf_arr = np.array(vf_list)

    i_min = np.argmin(chi2_arr)
    rs_best = rs_grid[i_min]
    chi2_min = chi2_arr[i_min]
    vf_best = vf_arr[i_min]

    # 精密化: 最小点周辺で細かくスキャン
    if 1 &lt; i_min &lt; len(rs_grid) - 1:
        rs_fine = np.linspace(rs_grid[max(i_min-3, 0)],
                              rs_grid[min(i_min+3, len(rs_grid)-1)], 50)
        for rs in rs_fine:
            c2, vf = chi2_at_rs(rs, rad, v_obs, err_v, v_bar)
            if c2 &lt; chi2_min:
                chi2_min = c2
                rs_best = rs
                vf_best = vf

    # chi2(rs=0) との比較
    chi2_0, vf_0 = chi2_at_rs(0.001, rad, v_obs, err_v, v_bar)

    N_pts = len(rad)
    chi2_dof = chi2_min / max(N_pts - 2, 1)

    return {
        'rs_best': rs_best,
        'vf_best': vf_best,
        'chi2_min': chi2_min,
        'chi2_dof': chi2_dof,
        'chi2_at_0': chi2_0,
        'improvement': (chi2_0 - chi2_min) / max(chi2_0, 0.1),
        'N_pts': N_pts,
    }


# ============================================================
# メインループ
# ============================================================
print(f"¥n全銀河の chi2 プロファイルスキャン中...")

results = []
n_done = 0

for gal in galaxies:
    data = read_dat(gal['name'])
    if data is None:
        continue
    rad = data[:, 0]
    v_obs = data[:, 1]
    err_v = data[:, 2]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
```

```
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = gal['upsilon_d']

    vbar2 = ud * np.sign(v_disk)*v_disk**2 + ¥
            np.sign(v_gas)*v_gas**2 + ¥
            np.sign(v_bul)*v_bul**2
    v_bar = np.sqrt(np.maximum(vbar2, 0.0))

    if len(rad) &lt; 3:
        continue

    R_max = rad.max()
    r_min = rad[rad &gt; 0].min() if np.any(rad &gt; 0) else 0.01

    fit = find_rs_best(rad, v_obs, err_v, v_bar, R_max)

    results.append({
        'name': gal['name'],
        'vflat': gal['vflat'],
        'rs1': gal['rs1'],
        'rs2': gal['rs2'],
        'upsilon_d': ud,
        'R_max': R_max,
        'r_min': r_min,
        **fit,
    })

    n_done += 1
    if n_done % 25 == 0:
        print(f"  {n_done}/{len(galaxies)} completed...")

N = len(results)
print(f"¥n完了: {N} 銀河")

# 配列化
rs_best = np.array([d['rs_best'] for d in results])
rs1 = np.array([d['rs1'] for d in results])
rs2 = np.array([d['rs2'] for d in results])
vf = np.array([d['vflat'] for d in results])
Rmax = np.array([d['R_max'] for d in results])
improve = np.array([d['improvement'] for d in results])

# ============================================================
# スケーリング解析
# ============================================================
print(f"¥n{'='*70}")
print("rs_best vs v_flat のスケーリング")
print(f"{'='*70}")

# 基本統計
print(f"¥nrs_best 基本統計:")
print(f"  中央値: {np.median(rs_best):.3f} kpc")
print(f"  平均:   {np.mean(rs_best):.3f} kpc")
print(f"  CV:     {np.std(rs_best)/np.mean(rs_best):.3f}")

# 全銀河
mask_pos = (rs_best &gt; 0.01) &amp;amp; (vf &gt; 0)
lv = np.log10(vf[mask_pos])
lrb = np.log10(rs_best[mask_pos])
lrs1 = np.log10(np.maximum(rs1[mask_pos], 0.001))
lrs2 = np.log10(np.maximum(rs2[mask_pos], 0.001))

p_best = np.polyfit(lv, lrb, 1)
r_best, p_best_p = pearsonr(lv, lrb)
rho_best, _ = spearmanr(vf[mask_pos], rs_best[mask_pos])

p_rs1 = np.polyfit(lv, lrs1, 1)
r_rs1, _ = pearsonr(lv, lrs1)

p_rs2 = np.polyfit(lv, lrs2, 1)
r_rs2, _ = pearsonr(lv, lrs2)

print(f"¥n{'指標':&gt;12s}  {'α':&gt;7s}  {'r':&gt;7s}  {'ρ(Spearman)':&gt;12s}  {'中央値[kpc]':&gt;12s}")
print(f"{'rs_best':&gt;12s}  {p_best[0]:7.3f}  {r_best:7.3f}  {rho_best:12.3f}  {np.median(rs_best):12.3f}")
print(f"{'rs1':&gt;12s}  {p_rs1[0]:7.3f}  {r_rs1:7.3f}  {'—':&gt;12s}  {np.median(rs1):12.3f}")
print(f"{'rs2':&gt;12s}  {p_rs2[0]:7.3f}  {r_rs2:7.3f}  {'—':&gt;12s}  {np.median(rs2):12.3f}")

# rs_best vs R_max (人工相関チェック)
rho_Rmax, p_Rmax = spearmanr(rs_best[mask_pos], Rmax[mask_pos])
p_rmax_fit = np.polyfit(np.log10(Rmax[mask_pos]), lrb, 1)
print(f"¥nrs_best vs R_max: ρ={rho_Rmax:.3f} (p={p_Rmax:.2e})")
print(f"  rs_best ∝ R_max^{p_rmax_fit[0]:.3f}")

# rs_best/R_max の分布
rbrm = rs_best / Rmax
print(f"  rs_best/R_max 中央値: {np.median(rbrm):.3f}")

# ============================================================
# rs_best vs rs1, rs2 の一致度
# ============================================================
print(f"¥n{'='*70}")
print("rs_best vs rs1, rs2 の比較")
print(f"{'='*70}")

diff_rs1 = np.abs(rs_best - rs1)
```

```
diff_rs2 = np.abs(rs_best - rs2)
closer_to_rs1 = np.sum(diff_rs1 &lt; diff_rs2)
closer_to_rs2 = np.sum(diff_rs2 &lt; diff_rs1)

print(f"¥nrs_best が rs1 に近い: {closer_to_rs1} ({100*closer_to_rs1/N:.0f}%)")
print(f"rs_best が rs2 に近い: {closer_to_rs2} ({100*closer_to_rs2/N:.0f}%)")

# rs_best/rs1, rs_best/rs2 の中央値
mask_rs1_pos = rs1 &gt; 0.01
mask_rs2_pos = rs2 &gt; 0.01
print(f"¥nrs_best/rs1 中央値: {np.median(rs_best[mask_rs1_pos]/rs1[mask_rs1_pos]):.3f}")
print(f"rs_best/rs2 中央値: {np.median(rs_best[mask_rs2_pos]/rs2[mask_rs2_pos]):.3f}")

# 相関
r_b1, _ = pearsonr(np.log10(np.maximum(rs_best, 0.01)), np.log10(np.maximum(rs1, 0.001)))
r_b2, _ = pearsonr(np.log10(np.maximum(rs_best, 0.01)), np.log10(np.maximum(rs2, 0.001)))
print(f"¥nlog相関: rs_best vs rs1: r={r_b1:.3f}")
print(f"log相関: rs_best vs rs2: r={r_b2:.3f}")

# ============================================================
# chi2 改善度の分布
# ============================================================
print(f"¥n{'='*70}")
print("rs=0 からの改善度")
print(f"{'='*70}")

print(f"  chi2 改善率中央値: {np.median(improve)*100:.1f}%")
print(f"  改善 &gt; 10%: {np.sum(improve &gt; 0.1)} ({100*np.sum(improve &gt; 0.1)/N:.0f}%)")
print(f"  改善 &gt; 30%: {np.sum(improve &gt; 0.3)} ({100*np.sum(improve &gt; 0.3)/N:.0f}%)")
print(f"  改善 &lt; 1%:  {np.sum(improve &lt; 0.01)} ({100*np.sum(improve &lt; 0.01)/N:.0f}%)")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) rs_best vs v_flat (log-log)
ax = axes[0, 0]
ax.scatter(lv, lrb, s=10, alpha=0.4, c='blue', label='rs_best')
vv = np.linspace(lv.min(), lv.max(), 100)
ax.plot(vv, np.polyval(p_best, vv), 'b-', linewidth=2,
        label=f'rs_best: α={p_best[0]:.3f}')
ax.plot(vv, np.polyval(p_rs1, vv), 'gray', ls='--',
        label=f'rs1: α={p_rs1[0]:.3f}')
ax.plot(vv, np.polyval(p_rs2, vv), 'r', ls=':',
        label=f'rs2: α={p_rs2[0]:.3f}')
ax.set_xlabel('log(v_flat)')
ax.set_ylabel('log(r_s) [kpc]')
ax.set_title(f'(a) Scaling comparison (N={mask_pos.sum()})')
ax.legend(fontsize=8)

# (b) rs_best vs rs1
ax = axes[0, 1]
ax.scatter(rs1, rs_best, s=10, alpha=0.4, c='steelblue')
maxv = max(rs1.max(), rs_best.max())
ax.plot([0, maxv], [0, maxv], 'k--', alpha=0.3)
ax.set_xlabel('rs1 [kpc]')
ax.set_ylabel('rs_best [kpc]')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title(f'(b) rs_best vs rs1 (r={r_b1:.3f})')

# (c) rs_best vs rs2
ax = axes[0, 2]
ax.scatter(rs2, rs_best, s=10, alpha=0.4, c='coral')
maxv = max(rs2.max(), rs_best.max())
ax.plot([0, maxv], [0, maxv], 'k--', alpha=0.3)
ax.set_xlabel('rs2 [kpc]')
ax.set_ylabel('rs_best [kpc]')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title(f'(c) rs_best vs rs2 (r={r_b2:.3f})')

# (d) rs_best vs R_max (人工相関チェック)
ax = axes[1, 0]
ax.scatter(Rmax, rs_best, s=10, alpha=0.4, c='green')
ax.plot([0, Rmax.max()], [0, Rmax.max()], 'k--', alpha=0.2)
ax.set_xlabel('R_max [kpc]')
ax.set_ylabel('rs_best [kpc]')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title(f'(d) rs_best vs R_max (ρ={rho_Rmax:.3f})')

# (e) rs_best のヒストグラム
ax = axes[1, 1]
ax.hist(np.log10(np.maximum(rs_best, 0.001)), bins=40,
        color='blue', alpha=0.5, label='rs_best')
ax.hist(np.log10(np.maximum(rs1, 0.001)), bins=40,
        color='gray', alpha=0.3, label='rs1')
ax.hist(np.log10(np.maximum(rs2, 0.001)), bins=40,
        color='red', alpha=0.3, label='rs2')
ax.set_xlabel('log(r_s) [kpc]')
ax.set_ylabel('Count')
ax.set_title('(e) Distribution comparison')
ax.legend(fontsize=8)
```

```
# (f) 改善度 vs v_flat
ax = axes[1, 2]
ax.scatter(vf, improve*100, s=10, alpha=0.4, c='purple')
ax.axhline(10, color='red', ls='--', alpha=0.5, label='10% improvement')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('χ² improvement from rs=0 [%]')
ax.set_title('(f) Improvement over rs=0')
ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('rs_best_all.png', dpi=150)
print(f"¥n[SAVED] rs_best_all.png")

# ============================================================
# CSV出力
# ============================================================
out_csv = "rs_best_all.csv"
with open(out_csv, "w", newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerow(['galaxy', 'v_flat', 'rs1', 'rs2', 'rs_best',
                     'vf_best', 'chi2_dof', 'improvement_pct',
                     'R_max', 'rs_best_over_Rmax', 'upsilon_d'])
    for d in sorted(results, key=lambda x: x['vflat']):
        writer.writerow([
            d['name'], f"{d['vflat']:.1f}",
            f"{d['rs1']:.4f}", f"{d['rs2']:.3f}", f"{d['rs_best']:.4f}",
            f"{d['vf_best']:.1f}", f"{d['chi2_dof']:.4f}",
            f"{d['improvement']*100:.1f}",
            f"{d['R_max']:.2f}", f"{d['rs_best']/d['R_max']:.4f}",
            f"{d['upsilon_d']:.3f}",
        ])
print(f"[SAVED] {out_csv}")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("rs_best スケーリングの最終結論")
print(f"{'='*70}")

print(f"""
■ 3つの r_s 推定値のスケーリング:
  rs1 (1段階fit) :    α = {p_rs1[0]:.3f}  (r={r_rs1:.3f})  ← 局所解汚染あり
  rs2 (2段階fit) :    α = {p_rs2[0]:.3f}  (r={r_rs2:.3f})  ← 信頼性未確立
  rs_best (profile) : α = {p_best[0]:.3f}  (r={r_best:.3f})  ← 大域的最適

■ rs_best の特性:
  中央値: {np.median(rs_best):.3f} kpc
  rs_best は rs1 より rs2 に近い: {closer_to_rs2}/{N} ({100*closer_to_rs2/N:.0f}%)
  rs_best vs R_max: ρ = {rho_Rmax:.3f}

■ 判定:
""")

if abs(p_best[0] - p_rs2[0]) &lt; 0.1:
    print(f"  rs_best ≈ rs2 のスケーリング → rs2 が正しい r_s に近い")
    print(f"  α ≈ {p_best[0]:.2f} が真のスケーリング")
elif abs(p_best[0] - p_rs1[0]) &lt; 0.1:
    print(f"  rs_best ≈ rs1 のスケーリング → rs1 の α=0.3 が真")
else:
    print(f"  rs_best は rs1 と rs2 の中間: α = {p_best[0]:.3f}")
    print(f"  どちらの既存推定も正確ではない")

if rho_Rmax &gt; 0.6:
    print(f"¥n  ★警告: rs_best vs R_max の相関が強い (ρ={rho_Rmax:.3f})")
    print(f"  → rs_best がデータ範囲に引っ張られる人工効果の可能性")
else:
    print(f"¥n  rs_best vs R_max の相関は中程度 (ρ={rho_Rmax:.3f})")
    print(f"  → データ範囲の人工効果は支配的ではない")
```

16. Rmax_hR_separation.py

項目	内容
フェーズ	R_max分離
目的	rs_best のスケーリングから v_flat, h_R, R_max の寄与を偏相関で分離
使用rs	rs_best
結果	2変数同時制御で全変数が非有意（p>0.05）。3変数の分離不能が確定。R^2=0.28。
ステータス	Level A（分離不能の確定）

解析目的

rs_best vs R_max の強い相関（rho=0.74）が物理的（h_R 経由）か人工的（データ範囲の効果）かを偏相関と多重回帰で判定。

ソースコード全文

```
#!/usr/bin/env python3
"""
R_max vs h_R の分離: rs_best のスケーリングの真偽判定

偏相関で分離:
  (A) rs_best vs v_flat | h_R を制御  → 真の γ
  (B) rs_best vs R_max | h_R を制御   → 人工効果の有無
  (C) rs_best vs v_flat | R_max を制御 → データ範囲を除いた信号
  (D) rs_best vs h_R | v_flat を制御   → バリオンスケールの独立効果

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

import csv
import os
import numpy as np
from scipy.stats import spearmanr, pearsonr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# データ読み込み
# ============================================================
def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

# rs_best_all.csv
_, rb_rows = load_csv("rs_best_all.csv")
rb_dict = {}
for row in rb_rows:
    name = row.get('galaxy', '').strip()
    rb = get_val(row, ['rs_best'])
    vf = get_val(row, ['v_flat'])
    Rm = get_val(row, ['R_max'])
    ud = get_val(row, ['upsilon_d'])
    rs1 = get_val(row, ['rs1'])
    rs2 = get_val(row, ['rs2'])
    if name and rb and vf and Rm:
        rb_dict[name] = {'rs_best': rb, 'vflat': vf, 'R_max': Rm,
                         'upsilon_d': ud, 'rs1': rs1, 'rs2': rs2}

# h_R を .dat から取得
results = []
for name, d in rb_dict.items():
    data = read_dat(name)
    if data is None:
        continue

    rad = data[:, 0]
    v_disk = data[:, 4]
    ud = d['upsilon_d'] if d['upsilon_d'] else 0.5

    vdisk = np.sqrt(ud) * np.abs(v_disk)
    if len(rad) &lt; 3 or np.max(vdisk) &lt; 0.1:
        continue
```

```
    i_peak = np.argmax(vdisk)
    r_peak = rad[i_peak]

    if r_peak &gt;= rad.max() * 0.9 or r_peak &lt; 0.01:
        continue

    h_R = r_peak / 2.15
    results.append({
        'name': name,
        'rs_best': d['rs_best'],
        'vflat': d['vflat'],
        'R_max': d['R_max'],
        'h_R': h_R,
        'rs1': d['rs1'],
        'rs2': d['rs2'],
    })

N = len(results)
print(f"[INFO] N={N} galaxies with rs_best, v_flat, R_max, h_R")

# 配列化 (log空間)
rs_b = np.log10(np.array([d['rs_best'] for d in results]))
vf = np.log10(np.array([d['vflat'] for d in results]))
Rm = np.log10(np.array([d['R_max'] for d in results]))
hR = np.log10(np.array([d['h_R'] for d in results]))
rs1 = np.log10(np.maximum(np.array([d['rs1'] for d in results]), 0.001))
rs2 = np.log10(np.maximum(np.array([d['rs2'] for d in results]), 0.001))

# ============================================================
# 偏相関関数
# ============================================================
def partial_corr(x, y, z):
    """x vs y, z を制御した偏相関 (Pearson) """
    # x, y, z から z の効果を線形回帰で除去
    px = np.polyfit(z, x, 1)
    py = np.polyfit(z, y, 1)
    x_res = x - np.polyval(px, z)
    y_res = y - np.polyval(py, z)
    r, p = pearsonr(x_res, y_res)
    return r, p, x_res, y_res


def partial_corr_2(x, y, z1, z2):
    """x vs y, z1 と z2 を同時に制御"""
    Z = np.column_stack([np.ones(len(z1)), z1, z2])
    bx, _, _, _ = np.linalg.lstsq(Z, x, rcond=None)
    by, _, _, _ = np.linalg.lstsq(Z, y, rcond=None)
    x_res = x - Z @ bx
    y_res = y - Z @ by
    r, p = pearsonr(x_res, y_res)
    return r, p

# ============================================================
# 基本相関 (ゼロ次)
# ============================================================
print(f"¥n{'='*70}")
print(f"ゼロ次相関 (制御なし、log空間、N={N}) ")
print(f"{'='*70}")

pairs = [
    ('rs_best', 'v_flat', rs_b, vf),
    ('rs_best', 'h_R', rs_b, hR),
    ('rs_best', 'R_max', rs_b, Rm),
    ('v_flat', 'h_R', vf, hR),
    ('v_flat', 'R_max', vf, Rm),
    ('h_R', 'R_max', hR, Rm),
]

print(f"¥n{'pair':&gt;25s}  {'r':&gt;7s}  {'p':&gt;10s}")
for label_x, label_y, x, y in pairs:
    r, p = pearsonr(x, y)
    print(f"{label_x+' vs '+label_y:&gt;25s}  {r:+7.3f}  {p:10.2e}")

# ============================================================
# 偏相関 (1変数制御)
# ============================================================
print(f"¥n{'='*70}")
print("偏相関 (1変数制御) ")
print(f"{'='*70}")

partials = [
    ('rs_best vs v_flat', '| h_R', rs_b, vf, hR),
    ('rs_best vs v_flat', '| R_max', rs_b, vf, Rm),
    ('rs_best vs h_R', '| v_flat', rs_b, hR, vf),
    ('rs_best vs h_R', '| R_max', rs_b, hR, Rm),
    ('rs_best vs R_max', '| v_flat', rs_b, Rm, vf),
    ('rs_best vs R_max', '| h_R', rs_b, Rm, hR),
]

print(f"¥n{'partial':&gt;35s}  {'r':&gt;7s}  {'p':&gt;10s}  {'判定':&gt;20s}")
for label, ctrl, x, y, z in partials:
    r, p, _, _ = partial_corr(x, y, z)
    sig = '***' if p &lt; 0.001 else '**' if p &lt; 0.01 else '*' if p &lt; 0.05 else 'ns'
    if 'v_flat' in label.split('vs')[1] and 'h_R' in ctrl:
```

```
        judge = f'γ の真の値'
    elif 'R_max' in label.split('vs')[1] and 'h_R' in ctrl:
        judge = f'人工効果の有無'
    elif 'h_R' in label.split('vs')[1] and 'v_flat' in ctrl:
        judge = f'バリオンの独立効果'
    else:
        judge = ''

    print(f"{label+' '+ctrl:&gt;35s}  {r:+7.3f}  {p:10.2e} {sig:&gt;3s}  {judge:&gt;20s}")

# ============================================================
# 偏相関 (2変数同時制御)
# ============================================================
print(f"¥n{'='*70}")
print("偏相関 (2変数同時制御) ")
print(f"{'='*70}")

partials_2 = [
    ('rs_best vs v_flat', '| h_R, R_max', rs_b, vf, hR, Rm),
    ('rs_best vs h_R', '| v_flat, R_max', rs_b, hR, vf, Rm),
    ('rs_best vs R_max', '| v_flat, h_R', rs_b, Rm, vf, hR),
]

print(f"¥n{'partial':&gt;40s}  {'r':&gt;7s}  {'p':&gt;10s}")
for label, ctrl, x, y, z1, z2 in partials_2:
    r, p = partial_corr_2(x, y, z1, z2)
    sig = '***' if p &lt; 0.001 else '**' if p &lt; 0.01 else '*' if p &lt; 0.05 else 'ns'
    print(f"{label+' '+ctrl:&gt;40s}  {r:+7.3f}  {p:10.2e} {sig:&gt;3s}")

# ============================================================
# α の分解 (多重回帰)
# ============================================================
print(f"¥n{'='*70}")
print("多重回帰: log(rs_best) = a + b·log(v_flat) + c·log(h_R) + d·log(R_max)")
print(f"{'='*70}")

X = np.column_stack([np.ones(N), vf, hR, Rm])
beta, residuals, _, _ = np.linalg.lstsq(X, rs_b, rcond=None)
pred = X @ beta
r_multi = np.corrcoef(rs_b, pred)[0, 1]
rss = np.sum((rs_b - pred)**2)

print(f"¥n  切片:     {beta[0]:+.3f}")
print(f"  v_flat:   {beta[1]:+.3f}  ← γ (v_flat の独立効果) ")
print(f"  h_R:      {beta[2]:+.3f}  ← バリオンスケールの効果")
print(f"  R_max:    {beta[3]:+.3f}  ← データ範囲の人工効果")
print(f"  R² = {r_multi**2:.4f}")

# 各変数の寄与度 (標準化係数)
std_vf = np.std(vf)
std_hR = np.std(hR)
std_Rm = np.std(Rm)
std_rb = np.std(rs_b)
print(f"¥n  標準化係数:")
print(f"  v_flat:   {beta[1]*std_vf/std_rb:+.3f}")
print(f"  h_R:      {beta[2]*std_hR/std_rb:+.3f}")
print(f"  R_max:    {beta[3]*std_Rm/std_rb:+.3f}")

# v_flat のみ vs h_R のみ vs R_max のみ
for label, x_var in [('v_flat only', vf), ('h_R only', hR), ('R_max only', Rm)]:
    X1 = np.column_stack([np.ones(N), x_var])
    b1, _, _, _ = np.linalg.lstsq(X1, rs_b, rcond=None)
    pred1 = X1 @ b1
    r1 = np.corrcoef(rs_b, pred1)[0, 1]
    print(f"¥n  {label}: slope={b1[1]:.3f}, R²={r1**2:.4f}")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) rs_best vs v_flat | h_R
r_a, p_a, xr_a, yr_a = partial_corr(rs_b, vf, hR)
ax = axes[0, 0]
ax.scatter(xr_a, yr_a, s=10, alpha=0.4, c='blue')
ax.axhline(0, color='gray', ls='--', alpha=0.3)
ax.axvline(0, color='gray', ls='--', alpha=0.3)
p_line = np.polyfit(xr_a, yr_a, 1)
xx = np.linspace(xr_a.min(), xr_a.max(), 100)
ax.plot(xx, np.polyval(p_line, xx), 'r-', linewidth=1.5)
ax.set_xlabel('log(rs_best) residual | h_R')
ax.set_ylabel('log(v_flat) residual | h_R')
ax.set_title(f'(a) rs vs v_flat | h_R¥nr={r_a:.3f} (γ true)')

# (b) rs_best vs R_max | h_R
r_b, p_b, xr_b, yr_b = partial_corr(rs_b, Rm, hR)
ax = axes[0, 1]
ax.scatter(xr_b, yr_b, s=10, alpha=0.4, c='red')
ax.axhline(0, color='gray', ls='--', alpha=0.3)
ax.axvline(0, color='gray', ls='--', alpha=0.3)
p_line = np.polyfit(xr_b, yr_b, 1)
ax.plot(xx, np.polyval(p_line, xx[:len(xx)]), 'k-', linewidth=1.5)
ax.set_xlabel('log(rs_best) residual | h_R')
ax.set_ylabel('log(R_max) residual | h_R')
ax.set_title(f'(b) rs vs R_max | h_R¥nr={r_b:.3f} (artifact?)')
```

```
# (c) rs_best vs h_R | v_flat
r_c, p_c, xr_c, yr_c = partial_corr(rs_b, hR, vf)
ax = axes[0, 2]
ax.scatter(xr_c, yr_c, s=10, alpha=0.4, c='green')
ax.axhline(0, color='gray', ls='--', alpha=0.3)
ax.axvline(0, color='gray', ls='--', alpha=0.3)
ax.set_xlabel('log(rs_best) residual | v_flat')
ax.set_ylabel('log(h_R) residual | v_flat')
ax.set_title(f'(c) rs vs h_R | v_flat¥nr={r_c:.3f} (baryon)')

# (d) 多重回帰の係数
ax = axes[1, 0]
labels_bar = ['v_flat¥n(γ)', 'h_R¥n(baryon)', 'R_max¥n(artifact)']
coeffs = [beta[1], beta[2], beta[3]]
colors_bar = ['blue', 'green', 'red']
bars = ax.bar(range(3), coeffs, color=colors_bar, alpha=0.7)
ax.axhline(0, color='black', linewidth=0.5)
for i, (b, c) in enumerate(zip(bars, coeffs)):
    ax.text(i, c + 0.02 * np.sign(c), f'{c:.3f}', ha='center', fontsize=10)
ax.set_xticks(range(3))
ax.set_xticklabels(labels_bar)
ax.set_ylabel('Regression coefficient')
ax.set_title('(d) Multiple regression coefficients')

# (e) 実測 vs 予測 (多重回帰)
ax = axes[1, 1]
ax.scatter(pred, rs_b, s=10, alpha=0.4, c='purple')
ax.plot([pred.min(), pred.max()], [pred.min(), pred.max()], 'k--')
ax.set_xlabel('Predicted log(rs_best)')
ax.set_ylabel('Observed log(rs_best)')
ax.set_title(f'(e) Multi-regression fit (R²={r_multi**2:.3f})')

# (f) v_flat, h_R, R_max の相互相関
ax = axes[1, 2]
corr_matrix = np.corrcoef(np.vstack([vf, hR, Rm]))
im = ax.imshow(corr_matrix, cmap='RdBu_r', vmin=-1, vmax=1)
ax.set_xticks([0, 1, 2])
ax.set_yticks([0, 1, 2])
ax.set_xticklabels(['v_flat', 'h_R', 'R_max'])
ax.set_yticklabels(['v_flat', 'h_R', 'R_max'])
for i in range(3):
    for j in range(3):
        ax.text(j, i, f'{corr_matrix[i,j]:.2f}', ha='center', va='center', fontsize=12)
plt.colorbar(im, ax=ax, shrink=0.8)
ax.set_title('(f) Predictor correlations')

plt.tight_layout()
plt.savefig('Rmax_hR_separation.png', dpi=150)
print(f"¥n[SAVED] Rmax_hR_separation.png")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("R_max vs h_R 分離の結論")
print(f"{'='*70}")

r_vf_hR, p_vf_hR, _, _ = partial_corr(rs_b, vf, hR)
r_Rm_hR, p_Rm_hR, _, _ = partial_corr(rs_b, Rm, hR)
r_hR_vf, p_hR_vf, _, _ = partial_corr(rs_b, hR, vf)

print(f"""
■ 偏相関の結果:

  rs_best vs v_flat | h_R :  r={r_vf_hR:+.3f} (p={p_vf_hR:.2e})  ← γ の真の値
  rs_best vs R_max | h_R :   r={r_Rm_hR:+.3f} (p={p_Rm_hR:.2e})  ← 人工効果
  rs_best vs h_R | v_flat :  r={r_hR_vf:+.3f} (p={p_hR_vf:.2e})  ← バリオンスケール

■ 多重回帰の係数:
  v_flat: {beta[1]:+.3f}  h_R: {beta[2]:+.3f}  R_max: {beta[3]:+.3f}

■ 判定:
""")

if abs(r_Rm_hR) &lt; 0.15 or p_Rm_hR &gt; 0.05:
    print("  R_max の人工効果は非有意。α=0.635 は物理的スケーリング。")
    print(f"  rs_best ∝ h_R^{beta[2]:.2f} × v_flat^{beta[1]:.2f}")
elif abs(r_Rm_hR) &gt; abs(r_vf_hR) and abs(r_Rm_hR) &gt; abs(r_hR_vf):
    print("  ★R_max の人工効果が支配的。α=0.635 の信頼性は低い。")
    print("  rs_best はデータ範囲に引っ張られている。")
elif abs(r_hR_vf) &gt; abs(r_Rm_hR):
    print("  h_R (バリオンスケール) が支配的。R_max 効果は従属的。")
    print(f"  → rs_best ∝ h_R が本質。R_max 相関は h_R 経由。")
    print(f"  → α=0.635 のうち大部分はバリオン由来 (γ≈{beta[1]:.2f}) 。")
else:
    print("  h_R と R_max の両方が寄与。完全な分離は困難。")
    print(f"  保守的推定: γ = {beta[1]:.2f} ± {abs(beta[3]):.2f}")
```

17. U2_gc_Ud_degeneracy.py

項目	内容
フェーズ	U-2
目的	g_c-Ud 縮退の定量化 + RAR 残差構造解析
使用rs	rs不使用（安全）
結果	g_c x Ud CV=1.10 < g_c CV=1.43（部分的縮退）。RAR残差 vs log(g_N): rho=-0.176 (p=5e-25)。
ステータス	Level B（縮退は部分的、残差は系統誤差排除が未了）

解析目的

Part A: (g_c, Ud) の2パラメータ RAR フィットで縮退を定量化。Part B: 全データ点の RAR 残差を g_N ビン別に集計し系統的パターンを検出。

ソースコード全文

```
#!/usr/bin/env python3
"""
U-2: g_c - Υ_d 縮退の解消 + RAR残差構造解析

Part A: (g_c, Υ_d) 2パラメータ RAR フィット
  深MOND極限: g_obs ≈ √(g_c · Υ_d · g_N(Υ=1))
  → g_c と Υ_d の積のみ制約 → 縮退
  Υ_d を自由にして、g_c&lt;&lt;a0 が Υ_d 制約の人工産物かを判定

Part B: RAR 残差構造
  g_obs - g_obs(MOND, g_c_fit) の系統的パターン
  → 膜効果が MOND からの「ずれ」として現れるか

r_s を一切使わない解析

実行: uv run --with scipy --with matplotlib --with numpy python U2_gc_Ud_degeneracy.py
"""

import csv
import os
import numpy as np
from scipy.optimize import minimize
from scipy.stats import spearmanr, pearsonr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# 定数・ユーティリティ
# ============================================================
a0_unit = 3703.0  # a0 in (km/s)^2 / kpc

def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

def mond_gobs(gN, gc):
    return (gN + np.sqrt(gN**2 + 4*gc*gN)) / 2

# ============================================================
# データ読み込み
# ============================================================
_, src_rows = load_csv("sparc_results.csv")
galaxies = []
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    ud = get_val(row, ['ud'])
    vf = get_val(row, ['vflat'])
    if ud and vf and vf &gt; 0:
        galaxies.append({'name': name, 'upsilon_d_fixed': ud, 'vflat': vf})

print(f"[INFO] {len(galaxies)} galaxies")

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

```
# Part A: 2パラメータ (g_c, Υ_d) フィット
# ============================================================
print(f"¥n{'='*70}")
print("Part A: (g_c, Υ_d) 2パラメータフィット")
print(f"{'='*70}")

def fit_gc_ud(rad, v_obs, err_v, v_disk, v_gas, v_bul):
    """
    (g_c, Υ_d) を同時フィット
    g_N(r) = [Υ_d · v_disk² + v_gas² + v_bul²] / r
    g_obs(r) = v_obs² / r
    MOND: g_obs = (g_N + √(g_N² + 4·g_c·g_N)) / 2
    """
    mask = rad &gt; 0.01
    r = rad[mask]
    vo = v_obs[mask]
    ev = np.maximum(err_v[mask], 1.0)
    vd = v_disk[mask]
    vg = v_gas[mask]
    vb = v_bul[mask]
    N = len(r)

    if N &lt; 4:
        return None

    g_obs = vo**2 / r  # (km/s)^2 / kpc
    w = 1.0 / ev**2

    def neg_loglik(params):
        log_gc, log_ud = params
        gc = 10**log_gc * a0_unit
        ud = 10**log_ud

        if ud &lt; 0.05 or ud &gt; 10.0:
            return 1e20

        gN = (ud * np.sign(vd)*vd**2 + np.sign(vg)*vg**2 + np.sign(vb)*vb**2) / r
        gN = np.maximum(gN, 1e-10)

        g_model = mond_gobs(gN, gc)

        # log空間残差
        log_ratio = np.log10(np.maximum(g_obs, 1e-10)) - np.log10(np.maximum(g_model, 1e-10))
        return np.sum(log_ratio**2)

    # グリッドサーチ初期値
    best = (1e20, 0, 0)
    for lgc in np.linspace(-1.0, 1.5, 15):
        for lud in np.linspace(-1.0, 0.7, 12):
            val = neg_loglik([lgc, lud])
            if val &lt; best[0]:
                best = (val, lgc, lud)

    # 精密化
    try:
        result = minimize(neg_loglik, [best[1], best[2]],
                          method='Nelder-Mead',
                          options={'maxiter': 3000, 'xatol': 0.01, 'fatol': 0.001})
        gc_fit = 10**result.x[0]
        ud_fit = 10**result.x[1]
        chi2 = result.fun
    except:
        gc_fit = 10**best[1]
        ud_fit = 10**best[2]
        chi2 = best[0]

    # 1パラメータフィット (Υ_d固定) も計算
    def neg_loglik_1p(params, ud_fixed):
        log_gc = params[0]
        gc = 10**log_gc * a0_unit
        gN = (ud_fixed * np.sign(vd)*vd**2 + np.sign(vg)*vg**2 + np.sign(vb)*vb**2) / r
        gN = np.maximum(gN, 1e-10)
        g_model = mond_gobs(gN, gc)
        log_ratio = np.log10(np.maximum(g_obs, 1e-10)) - np.log10(np.maximum(g_model, 1e-10))
        return np.sum(log_ratio**2)

    # Υ_d 固定版
    best_1p = min([(neg_loglik_1p([lgc], ud_fit), lgc) for lgc in np.linspace(-1.0, 1.5, 30)])

    # g_N の動的範囲
    ud_for_range = ud_fit
    gN_all = (ud_for_range * np.sign(vd)*vd**2 + np.sign(vg)*vg**2 + np.sign(vb)*vb**2) / r
    gN_all = np.maximum(gN_all, 1e-10)
    gN_range = np.log10(gN_all.max() / gN_all.min()) if gN_all.min() &gt; 0 else 0

    # 残差
    gN_best = (ud_fit * np.sign(vd)*vd**2 + np.sign(vg)*vg**2 + np.sign(vb)*vb**2) / r
    gN_best = np.maximum(gN_best, 1e-10)
    g_model_best = mond_gobs(gN_best, gc_fit * a0_unit)
    residuals = np.log10(np.maximum(g_obs, 1e-10)) - np.log10(np.maximum(g_model_best, 1e-10))

    return {
        'gc_2p': gc_fit,
        'ud_2p': ud_fit,
        'chi2_2p': chi2 / max(N-2, 1),
```

```
        'gc_ud_product': gc_fit * ud_fit,
        'gN_range': gN_range,
        'N_pts': N,
        'r': r,
        'gN': gN_best / a0_unit,
        'gobs': g_obs / a0_unit,
        'residuals': residuals,
    }


results = []
n_done = 0

for gal in galaxies:
    data = read_dat(gal['name'])
    if data is None:
        continue

    rad = data[:, 0]
    v_obs = data[:, 1]
    err_v = data[:, 2]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)

    fit = fit_gc_ud(rad, v_obs, err_v, v_disk, v_gas, v_bul)
    if fit is None:
        continue

    results.append({
        'name': gal['name'],
        'vflat': gal['vflat'],
        'ud_fixed': gal['upsilon_d_fixed'],
        **fit,
    })

    n_done += 1
    if n_done % 25 == 0:
        print(f"  {n_done} completed...")

N = len(results)
print(f"¥n完了: {N} galaxies")

# 配列化
gc_2p = np.array([d['gc_2p'] for d in results])
ud_2p = np.array([d['ud_2p'] for d in results])
ud_fixed = np.array([d['ud_fixed'] for d in results])
vf = np.array([d['vflat'] for d in results])
gc_ud = np.array([d['gc_ud_product'] for d in results])
gN_range = np.array([d['gN_range'] for d in results])

# ============================================================
# 縮退の診断
# ============================================================
print(f"¥n{'='*70}")
print("縮退の診断")
print(f"{'='*70}")

print(f"¥n--- g_c (2パラメータ) ---")
print(f"  中央値: {np.median(gc_2p):.3f} a0")
print(f"  CV: {np.std(gc_2p)/np.mean(gc_2p):.3f}")

print(f"¥n--- Υ_d (2パラメータ) vs Υ_d (固定) ---")
print(f"  Υ_d(2p) 中央値: {np.median(ud_2p):.3f}")
print(f"  Υ_d(fixed) 中央値: {np.median(ud_fixed):.3f}")
print(f"  Υ_d(2p)/Υ_d(fixed) 中央値: {np.median(ud_2p/ud_fixed):.3f}")

print(f"¥n--- g_c × Υ_d 積 (縮退パラメータ) ---")
print(f"  中央値: {np.median(gc_ud):.3f}")
print(f"  CV: {np.std(gc_ud)/np.mean(gc_ud):.3f}")

# g_c(2p) vs v_flat
rho_gc_vf, p_gc_vf = spearmanr(gc_2p, vf)
print(f"¥n--- g_c(2p) vs v_flat ---")
print(f"  Spearman ρ = {rho_gc_vf:.3f} (p={p_gc_vf:.2e})")
print(f"  cf. g_c(1p, Υ_d固定): ρ=0.546")
# g_c×Υ_d vs v_flat
rho_prod_vf, p_prod_vf = spearmanr(gc_ud, vf)
print(f"¥n--- g_c×Υ_d vs v_flat ---")
print(f"  Spearman ρ = {rho_prod_vf:.3f} (p={p_prod_vf:.2e})")

# gN 動的範囲 vs 縮退の程度
print(f"¥n--- g_N 動的範囲と縮退 ---")
print(f"  g_N range (log decades) 中央値: {np.median(gN_range):.2f}")

# 動的範囲で分割
mask_narrow = gN_range &lt; 1.0  # &lt; 1 decade
mask_wide = gN_range &gt; 1.5    # &gt; 1.5 decades
print(f"  狭い範囲 (&lt;1 decade) : N={mask_narrow.sum()}, g_c中央値={np.median(gc_2p[mask_narrow]):.3f}, Υ_d中央値={np.median(ud_2p[mask_narrow]):.3f}")
print(f"  広い範囲 (&gt;1.5 decades) : N={mask_wide.sum()}, g_c中央値={np.median(gc_2p[mask_wide]):.3f}, Υ_d中央値={np.median(ud_2p[mask_wide]):.3f}")

# v_flat ビン別
print(f"¥n--- v_flat ビン別 (g_c, Υ_d) ---")
sort_idx = np.argsort(vf)
n_per = max(N // 5, 1)
```

```
print(f"{'v_flat範囲':&gt;18s}  {'N':&gt;4s}  {'g_c/a0':&gt;8s}  {'Υ_d(2p)':&gt;8s}  {'Υ_d(fix)':&gt;8s}  {'g_c×Υ_d':&gt;8s}  {'gN_range':&gt;8s}")
for i in range(5):
    i0 = i * n_per
    i1 = (i+1)*n_per if i &lt; 4 else N
    idx = sort_idx[i0:i1]
    label = f"{vf[idx].min():.0f}-{vf[idx].max():.0f}"
    print(f"{label:&gt;18s}  {len(idx):4d}  {np.median(gc_2p[idx]):8.3f}  {np.median(ud_2p[idx]):8.3f}  {np.median(ud_fixed[idx]):8.3f}  {np.median(gc_ud[idx]):8.3f}  {np.median(gN_range[

# ============================================================
# Part B: RAR 残差構造
# ============================================================
print(f"¥n{'='*70}")
print("Part B: RAR 残差構造")
print(f"{'='*70}")

# 全データ点を集約
all_gN = []
all_gobs = []
all_resid = []
all_vf_tag = []

for d in results:
    all_gN.extend(d['gN'].tolist())
    all_gobs.extend(d['gobs'].tolist())
    all_resid.extend(d['residuals'].tolist())
    all_vf_tag.extend([d['vflat']] * len(d['gN']))

all_gN = np.array(all_gN)
all_gobs = np.array(all_gobs)
all_resid = np.array(all_resid)
all_vf_tag = np.array(all_vf_tag)

# g_N ビン別の残差
print(f"¥n全データ点: {len(all_gN)}")
mask_valid = (all_gN &gt; 1e-4) &amp;amp; np.isfinite(all_resid)
gN_v = all_gN[mask_valid]
resid_v = all_resid[mask_valid]
vf_v = all_vf_tag[mask_valid]

log_gN = np.log10(gN_v)
print(f"有効データ点: {mask_valid.sum()}")

# ビン別残差
n_bins = 10
sort_gN = np.argsort(log_gN)
n_per_bin = len(log_gN) // n_bins

print(f"¥n{'log(gN/a0)範囲':&gt;20s}  {'N':&gt;6s}  {'残差中央値':&gt;10s}  {'残差std':&gt;10s}")
bin_centers = []
bin_medians = []
bin_stds = []

for i in range(n_bins):
    i0 = i * n_per_bin
    i1 = (i+1)*n_per_bin if i &lt; n_bins-1 else len(log_gN)
    idx = sort_gN[i0:i1]
    label = f"{log_gN[idx].min():.2f} to {log_gN[idx].max():.2f}"
    med = np.median(resid_v[idx])
    std = np.std(resid_v[idx])
    print(f"{label:&gt;20s}  {len(idx):6d}  {med:+10.4f}  {std:10.4f}")
    bin_centers.append(np.median(log_gN[idx]))
    bin_medians.append(med)
    bin_stds.append(std)

bin_centers = np.array(bin_centers)
bin_medians = np.array(bin_medians)

# 残差の系統的パターン検出
rho_resid_gN, p_resid_gN = spearmanr(log_gN, resid_v)
print(f"¥n残差 vs log(gN): ρ={rho_resid_gN:.3f} (p={p_resid_gN:.2e})")

# v_flat 依存性
rho_resid_vf, p_resid_vf = spearmanr(vf_v, resid_v)
print(f"残差 vs v_flat: ρ={rho_resid_vf:.3f} (p={p_resid_vf:.2e})")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) g_c(2p) vs g_c×Υ_d
ax = axes[0, 0]
ax.scatter(gc_ud, gc_2p, s=10, alpha=0.4, c='steelblue')
ax.axhline(1.0, color='red', ls='--', alpha=0.5)
ax.set_xlabel('g_c × Υ_d')
ax.set_ylabel('g_c / a0 (2-param)')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title('(a) g_c vs g_c×Υ_d (degeneracy)')

# (b) Υ_d(2p) vs Υ_d(fixed)
ax = axes[0, 1]
ax.scatter(ud_fixed, ud_2p, s=10, alpha=0.4, c='orange')
maxud = max(ud_fixed.max(), ud_2p.max())
ax.plot([0, maxud], [0, maxud], 'k--', alpha=0.3)
```

```
ax.set_xlabel('Υ_d (fixed, 1-param)')
ax.set_ylabel('Υ_d (free, 2-param)')
ax.set_title(f'(b) Υ_d comparison')

# (c) g_c(2p) vs v_flat
ax = axes[0, 2]
ax.scatter(vf, gc_2p, s=10, alpha=0.4, c='purple')
ax.axhline(1.0, color='red', ls='--', alpha=0.5)
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('g_c / a0 (2-param)')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title(f'(c) g_c(2p) vs v_flat (ρ={rho_gc_vf:.3f})')

# (d) RAR with 2p fit
ax = axes[1, 0]
ax.scatter(np.log10(gN_v), np.log10(all_gobs[mask_valid]/a0_unit*a0_unit),
           s=0.5, alpha=0.05, c='gray')
# MOND曲線
gN_line = np.logspace(-3, 1.5, 500)
gobs_line = mond_gobs(gN_line, 1.0)  # g_c=a0
ax.plot(np.log10(gN_line), np.log10(gobs_line), 'r-', linewidth=1.5, label='g_c=a0')
ax.plot([-3, 1.5], [-3, 1.5], 'k:', alpha=0.3)
ax.set_xlabel('log(g_N/a0)')
ax.set_ylabel('log(g_obs/a0)')
ax.set_xlim(-3, 1.5)
ax.set_ylim(-2.5, 1.5)
ax.set_title('(d) RAR')
ax.legend(fontsize=8)

# (e) 残差 vs log(g_N) ― ビン別
ax = axes[1, 1]
ax.scatter(log_gN, resid_v, s=0.5, alpha=0.05, c='gray')
ax.errorbar(bin_centers, bin_medians, yerr=bin_stds, fmt='ro-', markersize=5, linewidth=1.5)
ax.axhline(0, color='black', ls='--', alpha=0.5)
ax.set_xlabel('log(g_N/a0)')
ax.set_ylabel('Residual (log g_obs - log g_model)')
ax.set_title(f'(e) RAR residual (ρ={rho_resid_gN:.3f})')
ax.set_ylim(-0.5, 0.5)

# (f) g_c×Υ_d vs v_flat
ax = axes[1, 2]
ax.scatter(vf, gc_ud, s=10, alpha=0.4, c='teal')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('g_c × Υ_d')
ax.set_xscale('log')
ax.set_yscale('log')
rho_label = f'ρ={rho_prod_vf:.3f}'
ax.set_title(f'(f) g_c×Υ_d vs v_flat ({rho_label})')

plt.tight_layout()
plt.savefig('U2_gc_Ud_degeneracy.png', dpi=150)
print(f"¥n[SAVED] U2_gc_Ud_degeneracy.png")

# ============================================================
# CSV出力
# ============================================================
out_csv = "U2_gc_Ud_2param.csv"
with open(out_csv, "w", newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerow(['galaxy', 'v_flat', 'gc_2p', 'ud_2p', 'ud_fixed',
                     'gc_x_ud', 'gN_range', 'chi2_dof'])
    for d in sorted(results, key=lambda x: x['vflat']):
        writer.writerow([
            d['name'], f"{d['vflat']:.1f}",
            f"{d['gc_2p']:.4f}", f"{d['ud_2p']:.4f}", f"{d['ud_fixed']:.3f}",
            f"{d['gc_ud_product']:.4f}", f"{d['gN_range']:.3f}",
            f"{d['chi2_2p']:.4f}",
        ])
print(f"[SAVED] {out_csv}")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("U-2 結論")
print(f"{'='*70}")

print(f"""
■ Part A: g_c - Υ_d 縮退

  g_c(2パラメータ): 中央値 = {np.median(gc_2p):.3f} a0, CV = {np.std(gc_2p)/np.mean(gc_2p):.3f}
  Υ_d(2パラメータ): 中央値 = {np.median(ud_2p):.3f}
  g_c × Υ_d (縮退積) : 中央値 = {np.median(gc_ud):.3f}, CV = {np.std(gc_ud)/np.mean(gc_ud):.3f}

  g_c(2p) vs v_flat: ρ = {rho_gc_vf:.3f}
  g_c×Υ_d vs v_flat: ρ = {rho_prod_vf:.3f}

  縮退の判定:
    g_c×Υ_d の CV が g_c(2p) の CV より小さい
    → 真に制約されているのは積 g_c×Υ_d
    → g_c と Υ_d は個別には制約されていない

■ Part B: RAR 残差構造
```

```
  残差 vs log(g_N): ρ = {rho_resid_gN:.3f} (p={p_resid_gN:.2e})
  残差 vs v_flat:   ρ = {rho_resid_vf:.3f} (p={p_resid_vf:.2e})

  残差に系統的パターンがあれば → MOND からのずれ = 膜効果の候補
  パターンがなければ → MOND が十分な記述
""")
```

18. U3_spatial_residuals.py

項目	内容
フェーズ	U-3
目的	RAR 残差の銀河内空間構造（tanh 遷移の直接証拠テスト）
使用rs	rs_best（参照点として使用、フィッティングには不使用）
結果	dResid=-0.181 (p=5.9e-9)。83%の銀河で内側<外側。バルジ有無で持続。
ステータス	Level B（条件b不充足: rs_best は特別でない）

解析目的

各銀河で g_obs/g_MOND の残差プロファイルを r/rs_best で規格化してスタッキング。r < rs_best で負、r > rs_best で 0 のパターン（tanh 予測）を検証。

ソースコード全文

```
#!/usr/bin/env python3
"""
U-3: RAR残差の銀河内空間構造 → tanh遷移の直接証拠を探す

方法:
  各銀河で g_obs(r) / g_MOND(r, g_c) の残差プロファイルを計算
  r/rs_best で規格化してスタッキング

予測（v3.0 tanh遷移が正しい場合）:
  r &lt;&lt; r_s: 残差 &lt; 0（MOND過大予測、膜未展開域）
  r &gt;&gt; r_s: 残差 ≈ 0（MOND正確、膜展開完了域）
  遷移は r ≈ r_s 付近

系統誤差の制御:
  バルジあり/なし銀河の比較
  データ点数 vs 残差パターン
  g_c を銀河ごとに個別フィット vs 一律 a0

r_s を参照点として使うが、フィッティングには使わない（RARフィットはrs不使用）

実行: uv run --with scipy --with matplotlib --with numpy python U3_spatial_residuals.py

前提: TA3_gc_independent.csv, rs_best_all.csv
"""

import csv
import os
import numpy as np
from scipy.stats import spearmanr, pearsonr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# ユーティリティ
# ============================================================
def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

def mond_gobs(gN, gc):
    return (gN + np.sqrt(gN**2 + 4*gc*gN)) / 2

a0_unit = 3703.0

# ============================================================
# データ読み込み
# ============================================================
_, src_rows = load_csv("sparc_results.csv")
gal_info = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    ud = get_val(row, ['ud'])
    vf = get_val(row, ['vflat'])
```

```
    if ud and vf:
        gal_info[name] = {'ud': ud, 'vflat': vf}

# g_c (RAR)
gc_dict = {}
if os.path.exists("TA3_gc_independent.csv"):
    _, gc_rows = load_csv("TA3_gc_independent.csv")
    for row in gc_rows:
        name = row.get('galaxy', '').strip()
        gc = get_val(row, ['gc_over_a0'])
        if name and gc:
            gc_dict[name] = gc

# rs_best
rs_dict = {}
if os.path.exists("rs_best_all.csv"):
    _, rb_rows = load_csv("rs_best_all.csv")
    for row in rb_rows:
        name = row.get('galaxy', '').strip()
        rb = get_val(row, ['rs_best'])
        if name and rb:
            rs_dict[name] = rb

print(f"[INFO] galaxies: {len(gal_info)}, g_c: {len(gc_dict)}, rs_best: {len(rs_dict)}")

# ============================================================
# 各銀河の残差プロファイル計算
# ============================================================
all_profiles = []

for name, info in gal_info.items():
    if name not in gc_dict or name not in rs_dict:
        continue

    data = read_dat(name)
    if data is None:
        continue

    rad = data[:, 0]
    v_obs = data[:, 1]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = info['ud']
    gc = gc_dict[name] * a0_unit
    rs = rs_dict[name]

    mask = rad &gt; 0.01
    r = rad[mask]
    vo = v_obs[mask]
    vd = v_disk[mask]
    vg = v_gas[mask]
    vb = v_bul[mask]

    if len(r) &lt; 5 or rs &lt;= 0:
        continue

    # g_N, g_obs
    gN = (ud * np.sign(vd)*vd**2 + np.sign(vg)*vg**2 + np.sign(vb)*vb**2) / r
    gN = np.maximum(gN, 1e-10)
    g_obs = vo**2 / r

    # g_MOND
    g_mond = mond_gobs(gN, gc)

    # 残差 (log空間)
    resid = np.log10(np.maximum(g_obs, 1e-10)) - np.log10(np.maximum(g_mond, 1e-10))

    # r/rs_best で規格化
    r_norm = r / rs

    # バルジの有無
    has_bulge = np.any(np.abs(vb) &gt; 1.0)

    # 内側/外側の残差
    mask_inner = r_norm &lt; 0.5
    mask_outer = r_norm &gt; 2.0
    mask_transit = (r_norm &gt;= 0.5) &amp;amp; (r_norm &lt;= 2.0)

    resid_inner = np.median(resid[mask_inner]) if mask_inner.sum() &gt; 1 else np.nan
    resid_outer = np.median(resid[mask_outer]) if mask_outer.sum() &gt; 1 else np.nan
    resid_transit = np.median(resid[mask_transit]) if mask_transit.sum() &gt; 1 else np.nan

    # 残差の勾配 (r/rs に対する傾き)
    if len(r_norm) &gt; 3:
        try:
            slope = np.polyfit(r_norm, resid, 1)[0]
        except:
            slope = np.nan
    else:
        slope = np.nan

    all_profiles.append({
        'name': name,
        'vflat': info['vflat'],
```

```
        'gc': gc_dict[name],
        'rs_best': rs,
        'has_bulge': has_bulge,
        'N_pts': len(r),
        'r_norm': r_norm,
        'resid': resid,
        'gN_over_a0': gN / a0_unit,
        'resid_inner': resid_inner,
        'resid_outer': resid_outer,
        'resid_transit': resid_transit,
        'slope': slope,
    })

N = len(all_profiles)
print(f"解析成功: {N} galaxies")

# ============================================================
# スタッキング解析
# ============================================================
print(f"¥n{'='*70}")
print("スタッキング解析: 残差 vs r/rs_best")
print(f"{'='*70}")

# 全データ点を集約
all_r_norm = np.concatenate([p['r_norm'] for p in all_profiles])
all_resid = np.concatenate([p['resid'] for p in all_profiles])
all_bulge = np.concatenate([np.full(len(p['r_norm']), p['has_bulge']) for p in all_profiles])

# ビン別残差 (全銀河)
n_bins = 12
mask_valid = np.isfinite(all_r_norm) &amp;amp; np.isfinite(all_resid) &amp;amp; (all_r_norm &amp;gt; 0)
rn_v = all_r_norm[mask_valid]
res_v = all_resid[mask_valid]
bul_v = all_bulge[mask_valid]

log_rn = np.log10(rn_v)
sort_idx = np.argsort(log_rn)
n_per = len(log_rn) // n_bins

print(f"¥n全データ点: {mask_valid.sum()}")
print(f"¥n{'r/rs範囲':&amp;gt;18s}  {'N':&amp;gt;6s}  {'残差中央値':&amp;gt;10s}  {'残差std':&amp;gt;10s}  {'予測':&amp;gt;10s}")

bin_centers = []
bin_medians = []
bin_stds = []
bin_ns = []

for i in range(n_bins):
    i0 = i * n_per
    i1 = (i+1)*n_per if i &amp;lt; n_bins-1 else len(log_rn)
    idx = sort_idx[i0:i1]
    rn_bin = rn_v[idx]
    res_bin = res_v[idx]
    med = np.median(res_bin)
    std = np.std(res_bin)

    # 予測
    rn_med = np.median(rn_bin)
    if rn_med &amp;lt; 0.5:
        pred = "&amp;lt; 0"
    elif rn_med &amp;gt; 2.0:
        pred = "≈ 0"
    else:
        pred = "遷移域"

    label = f"{rn_bin.min():.2f}-{rn_bin.max():.2f}"
    print(f"{label:&amp;gt;18s}  {len(idx):6d}  {med:+10.4f}  {std:10.4f}  {pred:&amp;gt;10s}")

    bin_centers.append(np.median(log_rn[idx]))
    bin_medians.append(med)
    bin_stds.append(std)
    bin_ns.append(len(idx))

bin_centers = np.array(bin_centers)
bin_medians = np.array(bin_medians)
bin_stds = np.array(bin_stds)
# ============================================================
# バルジあり/なしの比較
# ============================================================
print(f"¥n{'='*70}")
print("バルジあり vs なしの比較")
print(f"{'='*70}")

n_bulge = sum(1 for p in all_profiles if p['has_bulge'])
n_no_bulge = N - n_bulge
print(f"バルジあり: {n_bulge}, なし: {n_no_bulge}")

for label, mask_b in [("バルジなし", ~bul_v), ("バルジあり", bul_v)]:
    rn_sub = rn_v[mask_b]
    res_sub = res_v[mask_b]

    if len(rn_sub) &amp;lt; 50:
        print(f"¥n  {label}: データ不足 (N={len(rn_sub)})")
        continue
```

```
    mask_in = rn_sub &amp;lt; 0.5
    mask_out = rn_sub &amp;gt; 2.0

    med_in = np.median(res_sub[mask_in]) if mask_in.sum() &amp;gt; 5 else np.nan
    med_out = np.median(res_sub[mask_out]) if mask_out.sum() &amp;gt; 5 else np.nan

    print(f"¥n  {label} (N_pts={len(rn_sub)}):")
    print(f"    r/rs &amp;lt; 0.5 (内側): 残差中央値 = {med_in:+.4f} (N={mask_in.sum()})")
    print(f"    r/rs &amp;gt; 2.0 (外側): 残差中央値 = {med_out:+.4f} (N={mask_out.sum()})")
    if not np.isnan(med_in) and not np.isnan(med_out):
        print(f"    差 (内側-外側): {med_in - med_out:+.4f}")

# ============================================================
# 銀河ごとの残差勾配
# ============================================================
print(f"¥n{'='*70}")
print("銀河ごとの残差勾配 (d(resid)/d(r/rs))")
print(f"{'='*70}")

slopes = np.array([p['slope'] for p in all_profiles if not np.isnan(p['slope'])])
vf_slopes = np.array([p['vflat'] for p in all_profiles if not np.isnan(p['slope'])])
bulge_slopes = np.array([p['has_bulge'] for p in all_profiles if not np.isnan(p['slope'])])

print(f"¥n残差勾配の分布 (N={len(slopes)}):")
print(f"  中央値: {np.median(slopes):+.4f}")
print(f"  平均:   {np.mean(slopes):+.4f}")
print(f"  正の勾配 (外側で残差上昇) : {np.sum(slopes &amp;gt; 0)} ({100*np.sum(slopes &amp;gt; 0)/len(slopes):.0f}%)")

# tanh予測: 勾配 &amp;gt; 0 (内側で負、外側でゼロに回復)
from scipy.stats import wilcoxon
try:
    stat, p_wilcox = wilcoxon(slopes)
    print(f"  Wilcoxon検定 (H0: 勾配=0) : p={p_wilcox:.2e}")
except:
    p_wilcox = None

# バルジ有無での勾配差
slopes_bul = slopes[bulge_slopes]
slopes_no = slopes[~bulge_slopes]
print(f"¥n  バルジなし: 勾配中央値 = {np.median(slopes_no):+.4f} (N={len(slopes_no)})")
print(f"  バルジあり: 勾配中央値 = {np.median(slopes_bul):+.4f} (N={len(slopes_bul)})")

# ============================================================
# 内側-外側の残差差 (Δresid)
# ============================================================
print(f"¥n{'='*70}")
print("Δresid = resid(r/rs&amp;lt;0.5) - resid(r/rs&amp;gt;2.0)")
print(f"{'='*70}")

delta_resid = []
vf_delta = []
bulge_delta = []

for p in all_profiles:
    if not np.isnan(p['resid_inner']) and not np.isnan(p['resid_outer']):
        delta_resid.append(p['resid_inner'] - p['resid_outer'])
        vf_delta.append(p['vflat'])
        bulge_delta.append(p['has_bulge'])

delta_resid = np.array(delta_resid)
vf_delta = np.array(vf_delta)
bulge_delta = np.array(bulge_delta)

print(f"¥nN = {len(delta_resid)} 銀河 (内側+外側の両方にデータあり) ")
print(f"  Δresid 中央値: {np.median(delta_resid):+.4f}")
print(f"  Δresid &amp;lt; 0 (内側で MOND 過大予測) : {np.sum(delta_resid &amp;lt; 0)} ({100*np.sum(delta_resid &amp;lt; 0)/len(delta_resid):.0f}%)")

# tanh予測: Δresid &amp;lt; 0
try:
    stat2, p_wilcox2 = wilcoxon(delta_resid)
    print(f"  Wilcoxon検定 (H0: Δresid=0) : p={p_wilcox2:.2e}")
except:
    p_wilcox2 = None

# バルジ有無
if np.sum(~bulge_delta) &amp;gt; 5 and np.sum(bulge_delta) &amp;gt; 5:
    print(f"¥n  バルジなし: Δresid中央値 = {np.median(delta_resid[~bulge_delta]):+.4f} (N={np.sum(~bulge_delta)})")
    print(f"  バルジあり: Δresid中央値 = {np.median(delta_resid[bulge_delta]):+.4f} (N={np.sum(bulge_delta)})")

# Δresid vs v_flat
rho_dv, p_dv = spearmanr(vf_delta, delta_resid)
print(f"¥n  Δresid vs v_flat: ρ={rho_dv:.3f} (p={p_dv:.2e})")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) スタッキングプロファイル (全銀河)
ax = axes[0, 0]
ax.scatter(log_rn, res_v, s=0.3, alpha=0.03, c='gray')
ax.errorbar(bin_centers, bin_medians, yerr=np.array(bin_stds)/np.sqrt(np.array(bin_ns)),
            fmt='ro-', markersize=5, linewidth=1.5, capsize=3)
ax.axhline(0, color='black', ls='--', alpha=0.5)
ax.axvline(0, color='blue', ls=':', alpha=0.5, label='r = rs_best')
```

```
ax.set_xlabel('log(r / rs_best)')
ax.set_ylabel('Residual (log g_obs - log g_MOND)')
ax.set_title(f'(a) Stacked residual profile (N={N})')
ax.set_xlim(-1.5, 1.5)
ax.set_ylim(-0.4, 0.4)
ax.legend(fontsize=8)

# (b) バルジなし銀河のみ
ax = axes[0, 1]
mask_nb = ~bul_v
if mask_nb.sum() &gt; 100:
    log_rn_nb = log_rn[mask_nb]
    res_nb = res_v[mask_nb]
    ax.scatter(log_rn_nb, res_nb, s=0.3, alpha=0.03, c='gray')

    sort_nb = np.argsort(log_rn_nb)
    n_per_nb = len(log_rn_nb) // 10
    bc_nb, bm_nb, bs_nb, bn_nb = [], [], [], []
    for i in range(10):
        i0 = i * n_per_nb
        i1 = (i+1)*n_per_nb if i &lt; 9 else len(log_rn_nb)
        idx = sort_nb[i0:i1]
        bc_nb.append(np.median(log_rn_nb[idx]))
        bm_nb.append(np.median(res_nb[idx]))
        bs_nb.append(np.std(res_nb[idx]))
        bn_nb.append(len(idx))
    ax.errorbar(bc_nb, bm_nb, yerr=np.array(bs_nb)/np.sqrt(np.array(bn_nb)),
                fmt='go-', markersize=5, linewidth=1.5, capsize=3)

ax.axhline(0, color='black', ls='--', alpha=0.5)
ax.axvline(0, color='blue', ls=':', alpha=0.5)
ax.set_xlabel('log(r / rs_best)')
ax.set_ylabel('Residual')
ax.set_title('(b) No-bulge galaxies only')
ax.set_xlim(-1.5, 1.5)
ax.set_ylim(-0.4, 0.4)

# (c) バルジあり銀河のみ
ax = axes[0, 2]
mask_b = bul_v
if mask_b.sum() &gt; 100:
    log_rn_b = log_rn[mask_b]
    res_b = res_v[mask_b]
    ax.scatter(log_rn_b, res_b, s=0.3, alpha=0.03, c='gray')
    sort_b = np.argsort(log_rn_b)
    n_per_b = len(log_rn_b) // 10
    bc_b, bm_b, bs_b, bn_b = [], [], [], []
    for i in range(10):
        i0 = i * n_per_b
        i1 = (i+1)*n_per_b if i &lt; 9 else len(log_rn_b)
        idx = sort_b[i0:i1]
        bc_b.append(np.median(log_rn_b[idx]))
        bm_b.append(np.median(res_b[idx]))
        bs_b.append(np.std(res_b[idx]))
        bn_b.append(len(idx))
    ax.errorbar(bc_b, bm_b, yerr=np.array(bs_b)/np.sqrt(np.array(bn_b)),
                fmt='mo-', markersize=5, linewidth=1.5, capsize=3)

ax.axhline(0, color='black', ls='--', alpha=0.5)
ax.axvline(0, color='blue', ls=':', alpha=0.5)
ax.set_xlabel('log(r / rs_best)')
ax.set_ylabel('Residual')
ax.set_title('(c) Bulge galaxies only')
ax.set_xlim(-1.5, 1.5)
ax.set_ylim(-0.4, 0.4)

# (d) Δresid のヒストグラム
ax = axes[1, 0]
ax.hist(delta_resid, bins=30, color='steelblue', edgecolor='white', alpha=0.7)
ax.axvline(0, color='red', ls='--', linewidth=2)
ax.axvline(np.median(delta_resid), color='black', ls='-',
           label=f'Median={np.median(delta_resid):+.3f}')
ax.set_xlabel('Δresid (inner - outer)')
ax.set_ylabel('Count')
p_str = f'p={p_wilcox2:.2e}' if p_wilcox2 else 'N/A'
ax.set_title(f'(d) Δresid distribution ({p_str})')
ax.legend(fontsize=8)

# (e) 残差勾配のヒストグラム
ax = axes[1, 1]
ax.hist(slopes[~bulge_slopes], bins=25, alpha=0.5, color='green', label='No bulge')
ax.hist(slopes[bulge_slopes], bins=25, alpha=0.5, color='purple', label='Bulge')
ax.axvline(0, color='red', ls='--')
ax.set_xlabel('Residual slope (d(resid)/d(r/rs))')
ax.set_ylabel('Count')
ax.set_title('(e) Residual slope by bulge')
ax.legend(fontsize=8)

# (f) 個別銀河の残差プロファイル (代表5銀河)
ax = axes[1, 2]
# 最大データ点数の5銀河を選択
top5 = sorted(all_profiles, key=lambda x: x['N_pts'], reverse=True)[:5]
colors_5 = ['blue', 'red', 'green', 'orange', 'purple']
for p, col in zip(top5, colors_5):
    ax.plot(np.log10(p['r_norm']), p['resid'], '-', color=col, alpha=0.6, linewidth=1,
```

```
            label=f"{p['name']} (N={p['N_pts']})")
ax.axhline(0, color='black', ls='--', alpha=0.5)
ax.axvline(0, color='gray', ls=':', alpha=0.5)
ax.set_xlabel('log(r / rs_best)')
ax.set_ylabel('Residual')
ax.set_title('(f) Individual profiles (top 5 by N_pts)')
ax.set_xlim(-1.5, 1.5)
ax.set_ylim(-0.5, 0.5)
ax.legend(fontsize=6)

plt.tight_layout()
plt.savefig('U3_spatial_residuals.png', dpi=150)
print(f"¥n[SAVED] U3_spatial_residuals.png")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("U-3 結論")
print(f"{'='*70}")

print(f"""
■ tanh遷移の予測 vs 観測:

  予測: r &lt; r_s で残差 &lt; 0, r &gt; r_s で残差 ≈ 0

  スタッキング結果:
    r/rs &lt; 0.5 (内側) : 残差 = {np.median([p['resid_inner'] for p in all_profiles if not np.isnan(p['resid_inner'])]):+.4f}
    r/rs &gt; 2.0 (外側) : 残差 = {np.median([p['resid_outer'] for p in all_profiles if not np.isnan(p['resid_outer'])]):+.4f}
    Δresid (内側-外側) : {np.median(delta_resid):+.4f}
    Δresid &lt; 0 の銀河: {np.sum(delta_resid &lt; 0)}/{len(delta_resid)} ({100*np.sum(delta_resid &lt; 0)/len(delta_resid):.0f}%)
    Wilcoxon p = {p_wilcox2:.2e if p_wilcox2 else 'N/A'}

  残差勾配 (正 = 外側で回復) :
    中央値 = {np.median(slopes):+.4f}
    正の勾配: {np.sum(slopes &gt; 0)}/{len(slopes)} ({100*np.sum(slopes &gt; 0)/len(slopes):.0f}%)
    Wilcoxon p = {p_wilcox:.2e if p_wilcox else 'N/A'}

■ 判定:
""")

if p_wilcox2 and p_wilcox2 &lt; 0.01 and np.median(delta_resid) &lt; -0.02:
    print("  Δresid &lt; 0 が有意 → 内側でMOND過大予測のパターンを確認")
    if np.sum(~bulge_delta) &gt; 10:
        med_nb = np.median(delta_resid[~bulge_delta])
        if med_nb &lt; -0.02:
            print(f"  バルジなし銀河でも Δresid = {med_nb:+.4f} → バルジ系統誤差ではない")
            print("  → tanh 遷移の直接的証拠として Level B に分類可能")
        else:
            print(f"  バルジなし銀河では Δresid = {med_nb:+.4f} → バルジ系統誤差の可能性")
elif p_wilcox2 and p_wilcox2 &lt; 0.05:
    print("  弱い有意性。系統誤差の排除が必要。")
else:
    print("  Δresid ≈ 0 (有意でない) → tanh 遷移の空間的痕跡は検出されず")
    print("  → MOND式(5)の全半径一様適用でも空間的パターンは生じない")
```

19. levelA_promotion.py

項目	内容
フェーズ	Level A テスト
目的	空間残差パターンの Level A 昇格テスト
使用rs	rs_best
結果	(a) PASS: Ud 3.24倍変動が必要（非物理的）。(b) FAIL: r/h_R の方がシャープ。
ステータス	Level B 維持（遷移スケールは h_R に追従）

解析目的

(a) Ud(r) 代替モデルで残差が消えるか（非物理的変動なら条件充足）。(b) r/rs_best 以外の規格化（r/h_R, r/R_max等）でもパターンが出るか。

ソースコード全文

```
#!/usr/bin/env python3
"""
Level A 昇格条件の検証

(a) 代替モデル排除:
    Y_d(r) を各半径で自由にして残差を再計算
    → Y_d(r) が物理的に合理的な範囲 (0.1-3.0) で残差が消えるか？
    → 消えるなら tanh ではなく Y_d 勾配が原因
    → 消えないなら tanh が必要

(b) rs_best 依存性の検証:
    残差を r/rs_best 以外の規格化でスタッキング
    → r/h_R, r/R_max, r [kpc] で同じパターンが出るか？
    → rs_best でのみシャープなら遷移点としての意味がある
    → どの規格化でも同じなら rs_best は特別ではない

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

import csv
import os
import numpy as np
from scipy.stats import wilcoxon, spearmanr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
# ユーティリティ
# ============================================================
def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

def mond_gobs(gN, gc):
    return (gN + np.sqrt(gN**2 + 4*gc*gN)) / 2

a0_unit = 3703.0

# ============================================================
# データ読み込み
# ============================================================
_, src_rows = load_csv("sparc_results.csv")
gal_info = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    ud = get_val(row, ['ud'])
    vf = get_val(row, ['vflat'])
    if ud and vf:
        gal_info[name] = {'ud': ud, 'vflat': vf}

gc_dict = {}
if os.path.exists("TA3_gc_independent.csv"):
    _, gc_rows = load_csv("TA3_gc_independent.csv")
```

```
    for row in gc_rows:
        name = row.get('galaxy', '').strip()
        gc = get_val(row, ['gc_over_a0'])
        if name and gc: gc_dict[name] = gc

rs_dict = {}
if os.path.exists("rs_best_all.csv"):
    _, rb_rows = load_csv("rs_best_all.csv")
    for row in rb_rows:
        name = row.get('galaxy', '').strip()
        rb = get_val(row, ['rs_best'])
        if name and rb: rs_dict[name] = rb

# ============================================================
# 各銀河のデータ収集
# ============================================================
profiles = []

for name, info in gal_info.items():
    if name not in gc_dict or name not in rs_dict:
        continue
    data = read_dat(name)
    if data is None:
        continue

    rad = data[:, 0]
    v_obs = data[:, 1]
    v_disk = data[:, 4]
    v_gas = data[:, 3]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = info['ud']
    gc = gc_dict[name] * a0_unit
    rs = rs_dict[name]

    mask = rad &gt; 0.01
    r = rad[mask]
    vo = v_obs[mask]
    vd = v_disk[mask]
    vg = v_gas[mask]
    vb = v_bul[mask]

    if len(r) &lt; 5 or rs &lt;= 0:
        continue

    g_obs = vo**2 / r
    gN_base = (np.sign(vd)*vd**2 + np.sign(vg)*vg**2 + np.sign(vb)*vb**2) / r
    gN = (ud * np.sign(vd)*vd**2 + np.sign(vg)*vg**2 + np.sign(vb)*vb**2) / r
    gN = np.maximum(gN, 1e-10)
    gN_base_abs = np.maximum(np.abs(gN_base), 1e-10)

    g_mond = mond_gobs(gN, gc)
    resid = np.log10(np.maximum(g_obs, 1e-10)) - np.log10(np.maximum(g_mond, 1e-10))

    # h_R 推定
    vdisk_scaled = np.sqrt(ud) * np.abs(vd)
    i_peak = np.argmax(vdisk_scaled)
    r_peak = r[i_peak]
    h_R = r_peak / 2.15 if r_peak &lt; r.max() * 0.9 and r_peak &gt; 0.01 else None

    # (a) Υ_d(r) の逆算: g_obs = MOND(Υ_d(r)·gN_base, gc) を満たす Υ_d(r)
    # g_obs = (Υ_d·gN_base + √((Υ_d·gN_base)² + 4·gc·Υ_d·gN_base))/2
    # 数値的に各点で Υ_d を求める
    ud_local = []
    for j in range(len(r)):
        gb = gN_base_abs[j]
        go = g_obs[j]
        if gb &lt; 1e-10 or go &lt; 1e-10:
            ud_local.append(np.nan)
            continue
        # 二分探索
        ud_lo, ud_hi = 0.01, 20.0
        for _ in range(50):
            ud_mid = (ud_lo + ud_hi) / 2
            gn_try = ud_mid * gb
            gm_try = mond_gobs(gn_try, gc)
            if gm_try &lt; go:
                ud_lo = ud_mid
            else:
                ud_hi = ud_mid
        ud_local.append((ud_lo + ud_hi) / 2)
    ud_local = np.array(ud_local)

    profiles.append({
        'name': name, 'r': r, 'resid': resid, 'rs': rs,
        'h_R': h_R, 'R_max': r.max(), 'vflat': info['vflat'],
        'ud_local': ud_local, 'ud_fixed': ud,
        'g_obs': g_obs, 'gN': gN, 'gc': gc,
    })

N = len(profiles)
print(f"[INFO] {N} galaxies processed")

# ============================================================
# 条件(a): Υ_d(r) の物理的合理性
# ============================================================
```

```
print(f"¥n{'='*70}")
print("条件(a): Υ_d(r) 逆算の物理的合理性")
print(f"{'='*70}")

# 各銀河の Υ_d(r) を r/rs で規格化してスタッキング
all_rn_a = []
all_ud_ratio = []

for p in profiles:
    mask = np.isfinite(p['ud_local']) &amp;amp; (p['ud_local'] &amp;gt; 0)
    if mask.sum() &amp;lt; 3:
        continue
    r_norm = p['r'][mask] / p['rs']
    ud_ratio = p['ud_local'][mask] / p['ud_fixed']
    all_rn_a.extend(r_norm.tolist())
    all_ud_ratio.extend(ud_ratio.tolist())

all_rn_a = np.array(all_rn_a)
all_ud_ratio = np.array(all_ud_ratio)

# ビン別 Υ_d(r)/Υ_d_fixed
mask_v = np.isfinite(all_rn_a) &amp;amp; np.isfinite(all_ud_ratio) &amp;amp; (all_rn_a &amp;gt; 0)
rn_a = all_rn_a[mask_v]
udr = all_ud_ratio[mask_v]
log_rn_a = np.log10(rn_a)

sort_a = np.argsort(log_rn_a)
n_bins = 10
n_per = len(log_rn_a) // n_bins

print(f"¥n{'r/rs範囲':&amp;gt;18s}  {'Υ_d(r)/Υ_d_fixed':&amp;gt;18s}  {'物理性':&amp;gt;10s}")
bc_a, bm_a = [], []
for i in range(n_bins):
    i0 = i * n_per
    i1 = (i+1)*n_per if i &amp;lt; n_bins-1 else len(log_rn_a)
    idx = sort_a[i0:i1]
    med = np.median(udr[idx])
    rn_med = 10**np.median(log_rn_a[idx])
    phys = "OK" if 0.3 &amp;lt; med &amp;lt; 3.0 else "★要注意"
    print(f"{rn_med:.2f}".rjust(18) + f"  {med:18.3f}  {phys:&amp;gt;10s}")
    bc_a.append(np.median(log_rn_a[idx]))
    bm_a.append(med)

bc_a = np.array(bc_a)
bm_a = np.array(bm_a)

# Υ_d(r) の変動幅
ud_range_per_gal = []
for p in profiles:
    mask = np.isfinite(p['ud_local']) &amp;amp; (p['ud_local'] &amp;gt; 0)
    if mask.sum() &amp;gt; 3:
        ud_range_per_gal.append(p['ud_local'][mask].max() / p['ud_local'][mask].min())

print(f"¥n銀河内 Υ_d(r) の最大/最小比:")
print(f"  中央値: {np.median(ud_range_per_gal):.2f}")
print(f"  &amp;gt; 3倍: {np.sum(np.array(ud_range_per_gal) &amp;gt; 3)}/{len(ud_range_per_gal)} ({100*np.sum(np.array(ud_range_per_gal) &amp;gt; 3)/len(ud_range_per_gal):.0f}%)")
print(f"  &amp;gt; 5倍: {np.sum(np.array(ud_range_per_gal) &amp;gt; 5)}/{len(ud_range_per_gal)} ({100*np.sum(np.array(ud_range_per_gal) &amp;gt; 5)/len(ud_range_per_gal):.0f}%)")

# ============================================================
# 条件(b): 異なる規格化でのスタッキング
# ============================================================
print(f"¥n{'='*70}")
print("条件(b): 異なる規格化での残差パターン比較")
print(f"{'='*70}")

# 4つの規格化
normalizations = {}

for p in profiles:
    for norm_name, norm_val in [
        ('r/rs_best', p['rs']),
        ('r/h_R', p['h_R']),
        ('r/R_max', p['R_max']),
        ('r [kpc]', 1.0),
    ]:
        if norm_val is None or norm_val &amp;lt;= 0:
            continue
        r_norm = p['r'] / norm_val
        if norm_name not in normalizations:
            normalizations[norm_name] = {'r_norm': [], 'resid': []}
        normalizations[norm_name]['r_norm'].extend(r_norm.tolist())
        normalizations[norm_name]['resid'].extend(p['resid'].tolist())

# 各規格化での Δresid と遷移のシャープネス
print(f"¥n{'規格化':&amp;gt;12s}  {'N_pts':&amp;gt;7s}  {'Δresid':&amp;gt;8s}  {'遷移幅':&amp;gt;8s}  {'ρ(slope)':&amp;gt;10s}")

norm_results = {}
for norm_name, data in normalizations.items():
    rn = np.array(data['r_norm'])
    res = np.array(data['resid'])
    mask = np.isfinite(rn) &amp;amp; np.isfinite(res) &amp;amp; (rn &amp;gt; 0)
    rn = rn[mask]
    res = res[mask]
    log_rn = np.log10(rn)
```

```
    if len(rn) &lt; 100:
        continue

    # ビン別
    sort_idx = np.argsort(log_rn)
    n_per_b = len(log_rn) // 10
    bc, bm = [], []
    for i in range(10):
        i0 = i * n_per_b
        i1 = (i+1)*n_per_b if i &lt; 9 else len(log_rn)
        idx = sort_idx[i0:i1]
        bc.append(np.median(log_rn[idx]))
        bm.append(np.median(res[idx]))
    bc = np.array(bc)
    bm = np.array(bm)

    # Δresid: 下位25% vs 上位25%
    q25 = np.percentile(log_rn, 25)
    q75 = np.percentile(log_rn, 75)
    inner = res[log_rn &lt; q25]
    outer = res[log_rn &gt; q75]
    delta = np.median(inner) - np.median(outer)

    # 遷移のシャープネス: ビン間の最大残差変化
    diffs = np.diff(bm)
    sharpness = np.max(np.abs(diffs)) if len(diffs) &gt; 0 else 0

    # 全体の傾き
    rho, p_rho = spearmanr(log_rn, res)

    print(f"{norm_name:&gt;12s}  {len(rn):7d}  {delta:+8.4f}  {sharpness:8.4f}  {rho:+10.3f}")

    norm_results[norm_name] = {
        'bc': bc, 'bm': bm, 'delta': delta,
        'sharpness': sharpness, 'rho': rho,
        'rn': rn, 'res': res,
    }

# ============================================================
# 条件(a)の判定
# ============================================================
print(f"¥n{'='*70}")
print("条件(a)の判定")
print(f"{'='*70}")

inner_ud = bm_a[bc_a &lt; -0.3] if np.any(bc_a &lt; -0.3) else np.array([])
outer_ud = bm_a[bc_a &gt; 0.3] if np.any(bc_a &gt; 0.3) else np.array([])

if len(inner_ud) &gt; 0 and len(outer_ud) &gt; 0:
    print(f"¥n  内側 (r/rs &lt; 0.5) の Y_d(r)/Y_d_fixed 中央値: {np.median(inner_ud):.3f}")
    print(f"  外側 (r/rs &gt; 2.0) の Y_d(r)/Y_d_fixed 中央値: {np.median(outer_ud):.3f}")
    ud_gradient = np.median(inner_ud) - np.median(outer_ud)
    print(f"  Y_d 勾配 (内-外) : {ud_gradient:+.3f}")

    if abs(np.median(inner_ud) - 1.0) &lt; 0.3 and abs(np.median(outer_ud) - 1.0) &lt; 0.3:
        print("¥n  → Y_d(r) ≈ Y_d_fixed (全半径で一定に近い) ")
        print("  → 残差パターンは Y_d 勾配では説明できない → 条件(a) 充足")
    elif np.median(inner_ud) &lt; 0.5:
        print("¥n  → 内側で Y_d(r) &lt;&lt; Y_d_fixed (バリオン質量の過大見積もり) ")
        print("  → Y_d 勾配で残差を説明できる可能性 → 条件(a) 不充足")
    else:
        print(f"¥n  → Y_d(r) は {np.median(inner_ud):.2f}〜{np.median(outer_ud):.2f} で変動")
        if np.median(ud_range_per_gal) &gt; 3:
            print("  → 物理的に非合理的な変動幅 → 条件(a) 部分的に充足")
        else:
            print("  → 物理的に合理的な範囲 → 条件(a) 不充足 (Y_d勾配が代替説明として残る) ")

# ============================================================
# 条件(b)の判定
# ============================================================
print(f"¥n{'='*70}")
print("条件(b)の判定")
print(f"{'='*70}")

if 'r/rs_best' in norm_results:
    rs_sharp = norm_results['r/rs_best']['sharpness']
    rs_delta = abs(norm_results['r/rs_best']['delta'])

    print(f"¥n  r/rs_best: シャープネス={rs_sharp:.4f}, |Δresid|={rs_delta:.4f}")

    others_sharper = 0
    others_stronger = 0
    for nn, nr in norm_results.items():
        if nn == 'r/rs_best':
            continue
        print(f"  {nn}: シャープネス={nr['sharpness']:.4f}, |Δresid|={abs(nr['delta']):.4f}")
        if nr['sharpness'] &gt; rs_sharp:
            others_sharper += 1
        if abs(nr['delta']) &gt; rs_delta:
            others_stronger += 1

    if others_sharper == 0 and others_stronger == 0:
        print("¥n  → r/rs_best が最もシャープかつ最大Δresid → 条件(b) 充足")
        print("  → rs_best は遷移点として物理的に特別")
    elif others_sharper == 0:
```

```
        print("¥n  → r/rs_best が最もシャープだが ΔresidΔ は他と同等 → 条件(b) 部分的")
    else:
        print("¥n  → 他の規格化の方がシャープまたは強い → 条件(b) 不充足")
        print("  → 残差パターンは rs_best に特有ではない")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))
colors_norm = {'r/rs_best': 'red', 'r/h_R': 'green', 'r/R_max': 'blue', 'r [kpc]': 'orange'}

# (a) Υ_d(r)/Υ_d_fixed のスタッキング
ax = axes[0, 0]
ax.scatter(log_rn_a[:3000], udr[:3000], s=0.3, alpha=0.05, c='gray')
ax.plot(bc_a, bm_a, 'ro-', markersize=5, linewidth=1.5)
ax.axhline(1.0, color='black', ls='--', alpha=0.5, label='Υ_d = fixed')
ax.axvline(0, color='blue', ls=':', alpha=0.5, label='r = rs_best')
ax.set_xlabel('log(r / rs_best)')
ax.set_ylabel('Υ_d(r) / Υ_d_fixed')
ax.set_title('(a) Required Υ_d(r) to remove residuals')
ax.set_ylim(0, 4)
ax.set_xlim(-1.5, 1.5)
ax.legend(fontsize=8)

# (b)-(e) 4つの規格化でのスタッキング比較
for i, (nn, col) in enumerate([('r/rs_best', 'red'), ('r/h_R', 'green'),
                               ('r/R_max', 'blue'), ('r [kpc]', 'orange')]):
    if nn not in norm_results:
        continue
    nr = norm_results[nn]
    ax_idx = (0, 1) if i == 0 else (0, 2) if i == 1 else (1, 0) if i == 2 else (1, 1)
    ax = axes[ax_idx[0], ax_idx[1]]

    ax.scatter(np.log10(nr['rn'][:3000]), nr['res'][:3000], s=0.3, alpha=0.03, c='gray')
    ax.plot(nr['bc'], nr['bm'], 'o-', color=col, markersize=5, linewidth=1.5)
    ax.axhline(0, color='black', ls='--', alpha=0.5)
    ax.set_xlabel(f'log({nn})')
    ax.set_ylabel('Residual')
    sharp = nr['sharpness']
    delta = nr['delta']
    ax.set_title(f'({chr(98+i)}) {nn} (sharp={sharp:.3f}, Δ={delta:+.3f})')
    ax.set_ylim(-0.3, 0.3)

# (f) シャープネス比較の棒グラフ
ax = axes[1, 2]
nn_list = [nn for nn in ['r/rs_best', 'r/h_R', 'r/R_max', 'r [kpc]'] if nn in norm_results]
sharps = [norm_results[nn]['sharpness'] for nn in nn_list]
deltas = [abs(norm_results[nn]['delta']) for nn in nn_list]
x = np.arange(len(nn_list))
w = 0.35
ax.bar(x - w/2, sharps, w, label='Sharpness', color='steelblue')
ax.bar(x + w/2, deltas, w, label='|Δresid|', color='coral')
ax.set_xticks(x)
ax.set_xticklabels([nn.replace('r/', '') for nn in nn_list], fontsize=8)
ax.set_ylabel('Value')
ax.set_title('(f) Normalization comparison')
ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('levelA_promotion.png', dpi=150)
print(f"¥n[SAVED] levelA_promotion.png")

# ============================================================
# 最終判定
# ============================================================
print(f"¥n{'='*70}")
print("Level A 昇格の最終判定")
print(f"{'='*70}")

print(f"""
■ 条件(a): 代替モデル（Υ_d勾配）の排除
  Υ_d(r) の銀河内変動幅中央値: {np.median(ud_range_per_gal):.2f}倍

■ 条件(b): rs_best の特異性
  スタッキングのシャープネス比較（4規格化）

■ 総合判定:
  条件(a) と (b) の両方が充足 → Level A に昇格
  いずれか不充足 → Level B を維持（ただし有望な候補）
""")
```

20. rs_fixed_hR.py

項目	内容
フェーズ	MOND vs tanh
目的	rs=h_R 固定モデル vs MOND vs 定数加算の比較
使用rs	h_R固定（rsフィッティングなし）
結果	MOND勝利（dAIC=+3.9 vs tanh）。ただし定数加算は大幅劣後（dAIC=-93）。
ステータス	Level A（MOND式(5)が中核、式(7)は冗長）

解析目的

式(7) の rs を h_R に固定し（rs フィッティング問題を回避）、MOND式(5)（g_c free）と公平に比較。式(7) が観測的に冗長かを判定。

ソースコード全文

```
#!/usr/bin/env python3
"""
rs = h_R 固定モデルの検証

3+1 モデルを全銀河で比較:

  Model A: MOND (g_c 自由, 1param)
    v_obs = sqrt(r · MOND(g_N, g_c))

  Model B: tanh + rs=2.15·h_R 固定 (v_flat 自由, 1param)
    v_c² = v_bar² + v_flat² · T(r, 2.15·h_R)

  Model C: 定数加算 (v_flat 自由, 1param, rs=0 相当)
    v_c² = v_bar² + v_flat²

  Model D: tanh + rs=h_R + g_c 自由 (v_flat, g_c 自由, 2param)
    g_obs = MOND(g_N, g_c), v_c² = v_bar² + v_flat² · T(r, 2.15·h_R)

全モデル同一パラメータ数 (A,B,C: 1自由、D: 2自由) で chi2 比較

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

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

# ============================================================
a0_unit = 3703.0

def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

def mond_gobs(gN, gc):
    return (gN + np.sqrt(gN**2 + 4*gc*gN)) / 2

# ============================================================
# データ読み込み
# ============================================================
_, src_rows = load_csv("sparc_results.csv")
galaxies = []
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    ud = get_val(row, ['ud'])
    vf = get_val(row, ['vflat'])
    if ud and vf and vf &gt; 0:
        galaxies.append({'name': name, 'ud': ud, 'vflat_orig': vf})

print(f"[INFO] {len(galaxies)} galaxies")
```

```
# ============================================================
# フィッティング関数
# ============================================================

def chi2_mond(gc_a0, r, v_obs, err_v, gN):
    """Model A: MOND with free g_c"""
    gc = gc_a0 * a0_unit
    gN_pos = np.maximum(gN, 1e-10)
    g_model = mond_gobs(gN_pos, gc)
    v_model = np.sqrt(np.maximum(r * g_model, 0.01))
    w = 1.0 / np.maximum(err_v, 1.0)**2
    return np.sum(w * (v_obs - v_model)**2)

def chi2_tanh(vf, r, v_obs, err_v, v_bar, rs):
    """Model B: tanh with fixed rs"""
    T = 0.5 * (1.0 + np.tanh((r - rs) / np.maximum(rs, 0.01)))
    v_model = np.sqrt(np.maximum(v_bar**2 + vf**2 * T, 0.01))
    w = 1.0 / np.maximum(err_v, 1.0)**2
    return np.sum(w * (v_obs - v_model)**2)

def chi2_const(vf, r, v_obs, err_v, v_bar):
    """Model C: constant addition"""
    v_model = np.sqrt(np.maximum(v_bar**2 + vf**2, 0.01))
    w = 1.0 / np.maximum(err_v, 1.0)**2
    return np.sum(w * (v_obs - v_model)**2)

def fit_1param_grid(func, param_range, *args):
    """1パラメータのグリッドサーチ + 精密化"""
    best = (1e30, param_range[0])
    for p in param_range:
        c = func(p, *args)
        if c &lt; best[0]:
            best = (c, p)
    # 精密化
    dp = (param_range[1] - param_range[0]) * 2
    fine = np.linspace(max(best[1]-dp, param_range[0]), min(best[1]+dp, param_range[-1]), 50)
    for p in fine:
        c = func(p, *args)
        if c &lt; best[0]:
            best = (c, p)
    return best[1], best[0]

# ============================================================
# メインループ
# ============================================================
print("¥n全銀河のフィッティング中...")

results = []
n_done = 0

for gal in galaxies:
    data = read_dat(gal['name'])
    if data is None:
        continue

    rad = data[:, 0]
    v_obs = data[:, 1]
    err_v = data[:, 2]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = gal['ud']

    mask = rad &gt; 0.01
    r = rad[mask]
    vo = v_obs[mask]
    ev = err_v[mask]
    vd = v_disk[mask]
    vg = v_gas[mask]
    vb = v_bul[mask]

    if len(r) &lt; 5:
        continue

    N_pts = len(r)
    vbar2 = ud * np.sign(vd)*vd**2 + np.sign(vg)*vg**2 + np.sign(vb)*vb**2
    v_bar = np.sqrt(np.maximum(vbar2, 0.0))
    gN = np.maximum(vbar2 / r, 1e-10)

    # h_R 推定
    vdisk_scaled = np.sqrt(ud) * np.abs(vd)
    i_peak = np.argmax(vdisk_scaled)
    r_peak = r[i_peak]
    h_R = r_peak / 2.15

    if h_R &lt; 0.01 or r_peak &gt;= r.max() * 0.95:
        h_R = np.median(r) / 2.15  # フォールバック

    rs_fixed = 2.15 * h_R  # = r_peak

    # --- Model A: MOND (g_c free) ---
    gc_range = np.logspace(-1.5, 1.5, 80)
    gc_best, chi2_A = fit_1param_grid(chi2_mond, gc_range, r, vo, ev, gN)

    # --- Model B: tanh, rs=h_R, v_flat free ---
```

```
    vf_range = np.linspace(1, max(vo)*2, 80)
    vf_B, chi2_B = fit_1param_grid(chi2_tanh, vf_range, r, vo, ev, v_bar, rs_fixed)

    # --- Model C: constant addition, v_flat free ---
    vf_C, chi2_C = fit_1param_grid(chi2_const, vf_range, r, vo, ev, v_bar)

    # --- Model D: tanh + g_c free (2 params) ---
    # gN に g_c を適用した v_bar_eff を使うのではなく、
    # tanh の遷移構造 + MOND式の加速度修正 を組み合わせる
    # 簡易版: v_c² = v_bar² + v_flat² · T(r, rs) と g_c を独立にフィット
    # → まず v_flat を上のModel Bで固定し、g_c で残差を最適化
    # 実装: 2D グリッド
    best_D = (1e30, 0, 0)
    for gc_try in np.logspace(-1, 1, 20):
        for vf_try in np.linspace(max(1, vf_B*0.5), vf_B*1.5, 20):
            gN_gc = np.maximum(vbar2 / r, 1e-10)
            g_mond = mond_gobs(gN_gc, gc_try * a0_unit)
            v_mond = np.sqrt(np.maximum(r * g_mond, 0.01))
            T = 0.5 * (1.0 + np.tanh((r - rs_fixed) / np.maximum(rs_fixed, 0.01)))
            # Model D: 内側は MOND なし、外側は MOND
            # v_c² = v_bar² + (v_mond² - v_bar²) · T  ← 段階的に MOND へ遷移
            v_model_D = np.sqrt(np.maximum(v_bar**2 + (v_mond**2 - v_bar**2) * T, 0.01))
            w = 1.0 / np.maximum(ev, 1.0)**2
            c2 = np.sum(w * (vo - v_model_D)**2)
            if c2 &lt; best_D[0]:
                best_D = (c2, gc_try, vf_try)

    chi2_D = best_D[0]
    gc_D = best_D[1]

    # chi2/dof
    dof_1p = max(N_pts - 1, 1)
    dof_2p = max(N_pts - 2, 1)

    # ΔAIC (B vs A, B vs C)
    aic_A = chi2_A + 2 * 1
    aic_B = chi2_B + 2 * 1
    aic_C = chi2_C + 2 * 1
    aic_D = chi2_D + 2 * 2

    results.append({
        'name': gal['name'],
        'vflat': gal['vflat_orig'],
        'h_R': h_R,
        'N_pts': N_pts,
        'chi2_A': chi2_A / dof_1p,
        'chi2_B': chi2_B / dof_1p,
        'chi2_C': chi2_C / dof_1p,
        'chi2_D': chi2_D / dof_2p,
        'gc_A': gc_best,
        'vf_B': vf_B,
        'vf_C': vf_C,
        'gc_D': gc_D,
        'daic_BA': aic_B - aic_A,
        'daic_BC': aic_B - aic_C,
        'daic_DA': aic_D - aic_A,
    })

    n_done += 1
    if n_done % 25 == 0:
        print(f"  {n_done} completed...")

N = len(results)
print(f"¥n完了: {N} galaxies")

# ============================================================
# 結果解析
# ============================================================
chi2_A = np.array([d['chi2_A'] for d in results])
chi2_B = np.array([d['chi2_B'] for d in results])
chi2_C = np.array([d['chi2_C'] for d in results])
chi2_D = np.array([d['chi2_D'] for d in results])
daic_BA = np.array([d['daic_BA'] for d in results])
daic_BC = np.array([d['daic_BC'] for d in results])
daic_DA = np.array([d['daic_DA'] for d in results])
vf_arr = np.array([d['vflat'] for d in results])

print(f"¥n{'='*70}")
print("モデル比較（全銀河）")
print(f"{'='*70}")

print(f"¥n{'モデル':&gt;25s}  {'chi2/dof中央値':&gt;14s}  {'説明':&gt;30s}")
print(f"{'A: MOND (g_c free)':&gt;25s}  {np.median(chi2_A):14.3f}  {'式(5)全半径、g_c自由':&gt;30s}")
print(f"{'B: tanh, rs=h_R':&gt;25s}  {np.median(chi2_B):14.3f}  {'式(7) rs=r_peak固定、v_flat自由':&gt;30s}")
print(f"{'C: 定数加算':&gt;25s}  {np.median(chi2_C):14.3f}  {'v²=v_bar²+v_flat²':&gt;30s}")
print(f"{'D: tanh+MOND遷移':&gt;25s}  {np.median(chi2_D):14.3f}  {'内→外でバリオン→MOND遷移、2param':&gt;30s}")

# B vs A
B_better_A = np.sum(daic_BA &lt; 0)
B_worse_A = np.sum(daic_BA &gt; 0)
print(f"¥n--- B(tanh) vs A(MOND) ---")
print(f"  ΔAIC(B-A) 中央値: {np.median(daic_BA):+.2f}")
print(f"  B が A より良い: {B_better_A}/{N} ({100*B_better_A/N:.0f}%)")
print(f"  A が B より良い: {B_worse_A}/{N} ({100*B_worse_A/N:.0f}%)")
print(f"  ΔAIC &lt; -2 (B有意に良い) : {np.sum(daic_BA &lt; -2)} ({100*np.sum(daic_BA &lt; -2)/N:.0f}%)")
```

```
print(f"  ΔAIC &gt; +2 (A有意に良い) : {np.sum(daic_BA &gt; 2)} ({100*np.sum(daic_BA &gt; 2)/N:.0f}%)")

# B vs C
B_better_C = np.sum(daic_BC &lt; 0)
print(f"¥n--- B(tanh) vs C(定数) ---")
print(f"  ΔAIC(B-C) 中央値: {np.median(daic_BC):+.2f}")
print(f"  B が C より良い: {B_better_C}/{N} ({100*B_better_C/N:.0f}%)")
print(f"  ΔAIC &lt; -2 (B有意に良い) : {np.sum(daic_BC &lt; -2)} ({100*np.sum(daic_BC &lt; -2)/N:.0f}%)")

# D vs A
D_better_A = np.sum(daic_DA &lt; 0)
print(f"¥n--- D(tanh+MOND遷移) vs A(MOND) ---")
print(f"  ΔAIC(D-A) 中央値: {np.median(daic_DA):+.2f}")
print(f"  D が A より良い: {D_better_A}/{N} ({100*D_better_A/N:.0f}%)")

# v_flat 依存性
print(f"¥n--- v_flat ビン別 ---")
sort_vf = np.argsort(vf_arr)
n_per = max(N // 5, 1)
print(f"{'v_flat':&gt;12s}  {'N':&gt;4s}  {'chi2_A':&gt;8s}  {'chi2_B':&gt;8s}  {'chi2_C':&gt;8s}  {'B&lt;A %':&gt;6s}  {'B&lt;C %':&gt;6s}")
for i in range(5):
    i0 = i * n_per
    i1 = (i+1)*n_per if i &lt; 4 else N
    idx = sort_vf[i0:i1]
    vf_b = vf_arr[idx]
    label = f"{vf_b.min():.0f}-{vf_b.max():.0f}"
    ba_pct = 100*np.sum(daic_BA[idx] &lt; 0)/len(idx)
    bc_pct = 100*np.sum(daic_BC[idx] &lt; 0)/len(idx)
    print(f"{label:&gt;12s}  {len(idx):4d}  {np.median(chi2_A[idx]):8.3f}  {np.median(chi2_B[idx]):8.3f}  {np.median(chi2_C[idx]):8.3f}  {ba_pct:5.0f}%  {bc_pct:5.0f}%")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) chi2/dof の箱ひげ図
ax = axes[0, 0]
bp = ax.boxplot([chi2_A, chi2_B, chi2_C, chi2_D],
                labels=['A:MOND', 'B:tanh¥nrs=hR', 'C:const', 'D:tanh¥n+MOND'],
                showfliers=False, patch_artist=True)
colors_bp = ['lightcoral', 'lightblue', 'lightgray', 'lightgreen']
for patch, color in zip(bp['boxes'], colors_bp):
    patch.set_facecolor(color)
ax.set_ylabel('χ²/dof')
ax.set_title('(a) Model comparison')
ax.set_ylim(0, max(np.median(chi2_A), np.median(chi2_B), np.median(chi2_C)) * 3)

# (b) ΔAIC(B-A) のヒストグラム
ax = axes[0, 1]
ax.hist(daic_BA, bins=50, color='steelblue', edgecolor='white', alpha=0.7)
ax.axvline(0, color='red', ls='--', linewidth=2)
ax.axvline(np.median(daic_BA), color='black', ls='-',
           label=f'Median={np.median(daic_BA):+.1f}')
ax.set_xlabel('ΔAIC (B - A)')
ax.set_ylabel('Count')
ax.set_title(f'(b) tanh(rs=hR) vs MOND')
ax.legend(fontsize=8)

# (c) ΔAIC(B-C) のヒストグラム
ax = axes[0, 2]
ax.hist(daic_BC, bins=50, color='coral', edgecolor='white', alpha=0.7)
ax.axvline(0, color='red', ls='--', linewidth=2)
ax.axvline(np.median(daic_BC), color='black', ls='-',
           label=f'Median={np.median(daic_BC):+.1f}')
ax.set_xlabel('ΔAIC (B - C)')
ax.set_ylabel('Count')
ax.set_title(f'(c) tanh(rs=hR) vs constant')
ax.legend(fontsize=8)

# (d) chi2_B vs chi2_A (銀河ごと)
ax = axes[1, 0]
ax.scatter(chi2_A, chi2_B, s=10, alpha=0.4, c='steelblue')
maxchi = max(np.percentile(chi2_A, 95), np.percentile(chi2_B, 95))
ax.plot([0, maxchi], [0, maxchi], 'k--', alpha=0.3)
ax.set_xlabel('χ²/dof (A: MOND)')
ax.set_ylabel('χ²/dof (B: tanh rs=hR)')
ax.set_title('(d) Per-galaxy comparison B vs A')
ax.set_xlim(0, maxchi)
ax.set_ylim(0, maxchi)

# (e) ΔAIC(B-A) vs v_flat
ax = axes[1, 1]
ax.scatter(vf_arr, daic_BA, s=10, alpha=0.4, c='green')
ax.axhline(0, color='red', ls='--')
ax.axhline(-2, color='gray', ls=':', alpha=0.5)
ax.axhline(2, color='gray', ls=':', alpha=0.5)
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('ΔAIC (B - A)')
ax.set_title('(e) ΔAIC vs v_flat')

# (f) 代表銀河のフィット比較 (3例)
ax = axes[1, 2]
# ΔAIC(B-A) が最も負 (B最良) 、ゼロ付近、最も正 (A最良) の3銀河
i_best_B = np.argmin(daic_BA)
i_neutral = np.argmin(np.abs(daic_BA))
```

```
i_best_A = np.argmax(daic_BA)

for idx, col, ls in [(i_best_B, 'blue', '-'), (i_neutral, 'gray', '--'), (i_best_A, 'red', ':')]:
    d = results[idx]
    data = read_dat(d['name'])
    if data is None:
        continue
    rad = data[:, 0]
    v_obs = data[:, 1]
    # 正規化して重ね描き
    vmax = max(v_obs)
    ax.plot(rad, v_obs/vmax, col+ls, alpha=0.7,
            label=f"{d['name']} (ΔAIC={d['daic_BA']:+.0f})")

ax.set_xlabel('r [kpc]')
ax.set_ylabel('v / v_max')
ax.set_title('(f) Representative galaxies')
ax.legend(fontsize=7)

plt.tight_layout()
plt.savefig('rs_fixed_hR.png', dpi=150)
print(f"¥n[SAVED] rs_fixed_hR.png")

# ============================================================
# CSV 出力
# ============================================================
out_csv = "rs_fixed_hR_comparison.csv"
with open(out_csv, "w", newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerow(['galaxy', 'v_flat', 'h_R', 'N_pts',
                     'chi2_A', 'chi2_B', 'chi2_C', 'chi2_D',
                     'daic_BA', 'daic_BC', 'daic_DA',
                     'gc_A', 'vf_B', 'vf_C', 'gc_D'])
    for d in sorted(results, key=lambda x: x['daic_BA']):
        writer.writerow([
            d['name'], f"{d['vflat']:.1f}", f"{d['h_R']:.3f}", d['N_pts'],
            f"{d['chi2_A']:.4f}", f"{d['chi2_B']:.4f}",
            f"{d['chi2_C']:.4f}", f"{d['chi2_D']:.4f}",
            f"{d['daic_BA']:.2f}", f"{d['daic_BC']:.2f}", f"{d['daic_DA']:.2f}",
            f"{d['gc_A']:.4f}", f"{d['vf_B']:.1f}",
            f"{d['vf_C']:.1f}", f"{d['gc_D']:.4f}",
        ])
print(f"[SAVED] {out_csv}")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("rs = h_R 固定モデルの最終結論")
print(f"{'='*70}")

print(f"""
■ モデル比較 (N={N}) :

  Model A (MOND, g_c free):     chi2/dof中央値 = {np.median(chi2_A):.3f}
  Model B (tanh, rs=hR, vf free): chi2/dof中央値 = {np.median(chi2_B):.3f}
  Model C (定数加算, vf free):    chi2/dof中央値 = {np.median(chi2_C):.3f}

  B vs A: ΔAIC中央値 = {np.median(daic_BA):+.2f}
    B が有意に良い (ΔAIC&lt;-2) : {np.sum(daic_BA &lt; -2)}/{N} ({100*np.sum(daic_BA &lt; -2)/N:.0f}%)
    A が有意に良い (ΔAIC&gt;+2) : {np.sum(daic_BA &gt; 2)}/{N} ({100*np.sum(daic_BA &gt; 2)/N:.0f}%)

  B vs C: ΔAIC中央値 = {np.median(daic_BC):+.2f}
    B が有意に良い (ΔAIC&lt;-2) : {np.sum(daic_BC &lt; -2)}/{N} ({100*np.sum(daic_BC &lt; -2)/N:.0f}%)

■ 判定:
""")

if np.median(daic_BA) &lt; -2:
    print("  ★ tanh(rs=hR) が MOND より有意に良い → v3.0 の空間遷移の観測的支持")
    print(f"    rs をフィッティングせず h_R に固定しても改善が持続する")
elif np.median(daic_BA) &gt; 2:
    print("  MOND が tanh(rs=hR) より良い → 空間遷移は不要")
else:
    print("  tanh(rs=hR) と MOND は同等 → 空間遷移の優位性は確立されない")

if np.median(daic_BC) &lt; -2:
    print(f"  tanh(rs=hR) は定数加算より有意に良い → h_R での遷移構造が重要")
elif np.median(daic_BC) &gt; 2:
    print(f"  定数加算の方が tanh より良い → 遷移構造は不要")
else:
    print(f"  tanh と定数加算は同等")
```

21. gc_predictive_model.py

項目	内容
フェーズ	g_c予測
目的	g_c の予測モデル構築（銀河パラメータからの多変量回帰）
使用rs	rs不使用（安全）
結果	R^2=0.737、残差0.23 dex。dBIC=-192 vs g_c=a0。
ステータス	Level A（g_c 予測モデル確立）

解析目的

g_c(RAR) を v_flat, h_R, Sigma0 等の銀河パラメータで予測するモデルを構築。段階的変数追加で BIC 最良モデルを選択。g_c=a0（MOND）との比較。

ソースコード全文

```
#!/usr/bin/env python3
"""
g_c の予測モデル構築

g_c(RAR) を銀河の物理量で予測する:
  候補変数: Σ□, h_R, v_flat, V_peak, gas_fraction, M_bar_proxy

多変量回帰で g_c = f(galaxy properties) を確立し、
純粋MOND (g_c=a0) との差別化を図る。

r_s を一切使わない。

実行: uv run --with scipy --with matplotlib --with numpy python gc_predictive_model.py

前提: TA3_gc_independent.csv
"""

import csv
import os
import numpy as np
from scipy.stats import spearmanr, pearsonr
from itertools import combinations
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
a0_unit = 3703.0

def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

# ============================================================
# データ収集
# ============================================================
print("="*70)
print("データ収集")
print("="*70)

# sparc_results.csv
_, src_rows = load_csv("sparc_results.csv")
gal_info = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    ud = get_val(row, ['ud'])
    vf = get_val(row, ['vflat'])
    if ud and vf and vf &gt; 0:
        gal_info[name] = {'ud': ud, 'vflat': vf}

# g_c (RAR)
gc_dict = {}
if os.path.exists("TA3_gc_independent.csv"):
    _, gc_rows = load_csv("TA3_gc_independent.csv")
```

```
    for row in gc_rows:
        name = row.get('galaxy', '').strip()
        gc = get_val(row, ['gc_over_a0'])
        if name and gc and gc &gt; 0:
            gc_dict[name] = gc

# 銀河パラメータの計算
results = []

for name, info in gal_info.items():
    if name not in gc_dict:
        continue
    data = read_dat(name)
    if data is None:
        continue

    rad = data[:, 0]
    v_obs = data[:, 1]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = info['ud']

    if len(rad) &lt; 5:
        continue

    mask = rad &gt; 0.01
    r = rad[mask]
    vd = np.sqrt(ud) * np.abs(v_disk[mask])
    vg = np.abs(v_gas[mask])
    vb = np.abs(v_bul[mask]) if len(v_bul) &gt; 0 else np.zeros(mask.sum())

    vbar2 = ud * np.sign(v_disk[mask])*v_disk[mask]**2 + ¥
            np.sign(v_gas[mask])*v_gas[mask]**2 + ¥
            np.sign(v_bul[mask])*v_bul[mask]**2
    v_bar = np.sqrt(np.maximum(vbar2, 0.0))

    # V_peak (disk)
    V_peak_disk = np.max(vd)
    i_peak = np.argmax(vd)
    r_peak = r[i_peak]

    # h_R
    if r_peak &gt;= r.max() * 0.9 or r_peak &lt; 0.01:
        continue
    h_R = r_peak / 2.15

    # V_peak (total baryonic)
    V_peak_bar = np.max(v_bar)

    # Σ□ proxy: V_peak² / h_R (∝ central surface density)
    Sigma0_proxy = V_peak_disk**2 / h_R

    # gas fraction: mean(v_gas² / v_bar²)
    gas_frac = np.mean(vg**2 / np.maximum(v_bar**2, 0.01))

    # M_bar proxy: v_flat□ / a0 (Tully-Fisher)
    M_bar_proxy = info['vflat']**4 / a0_unit

    # disk dominance: V_peak_disk / v_flat
    disk_dominance = V_peak_disk / info['vflat']

    # compactness: V_peak / h_R
    compactness = V_peak_bar / h_R

    # g_N max / a0 (maximum baryonic acceleration)
    gN_max = np.max(v_bar**2 / r) / a0_unit

    # バルジの有無
    has_bulge = np.any(np.abs(v_bul[mask]) &gt; 1.0)

    results.append({
        'name': name,
        'gc': gc_dict[name],
        'log_gc': np.log10(gc_dict[name]),
        'vflat': info['vflat'],
        'ud': ud,
        'h_R': h_R,
        'V_peak_disk': V_peak_disk,
        'V_peak_bar': V_peak_bar,
        'Sigma0': Sigma0_proxy,
        'gas_frac': gas_frac,
        'M_bar': M_bar_proxy,
        'disk_dom': disk_dominance,
        'compactness': compactness,
        'gN_max': gN_max,
        'has_bulge': has_bulge,
    })

N = len(results)
print(f"解析成功: {N} galaxies")
# 配列化
gc = np.array([d['gc'] for d in results])
log_gc = np.array([d['log_gc'] for d in results])
```

```
predictors = {
    'log_vflat': np.log10(np.array([d['vflat'] for d in results])),
    'log_hR': np.log10(np.array([d['h_R'] for d in results])),
    'log_Sigma0': np.log10(np.array([d['Sigma0'] for d in results])),
    'log_Vpeak': np.log10(np.array([d['V_peak_disk'] for d in results])),
    'gas_frac': np.array([d['gas_frac'] for d in results]),
    'log_Mbar': np.log10(np.array([d['M_bar'] for d in results])),
    'log_Ud': np.log10(np.array([d['ud'] for d in results])),
    'disk_dom': np.array([d['disk_dom'] for d in results]),
    'log_compact': np.log10(np.array([d['compactness'] for d in results])),
    'log_gNmax': np.log10(np.maximum(np.array([d['gN_max'] for d in results]), 1e-4)),
}

# ============================================================
# 単変量相関
# ============================================================
print(f"¥n{'='*70}")
print("単変量相関: log(g_c/a0) vs 各予測変数")
print(f"{'='*70}")

print(f"¥n{'変数':&gt;15s}  {'Pearson r':&gt;10s}  {'Spearman ρ':&gt;12s}  {'p':&gt;10s}")
single_results = {}
for pname, parr in predictors.items():
    mask = np.isfinite(parr) &amp;amp; np.isfinite(log_gc)
    if mask.sum() &lt; 20:
        continue
    rp, pp = pearsonr(parr[mask], log_gc[mask])
    rs, ps = spearmanr(parr[mask], log_gc[mask])
    sig = '***' if pp &lt; 0.001 else '**' if pp &lt; 0.01 else '*' if pp &lt; 0.05 else ''
    print(f"{pname:&gt;15s}  {rp:+10.3f}  {rs:+12.3f}  {pp:10.2e} {sig}")
    single_results[pname] = {'r': rp, 'rho': rs, 'p': pp}

# ============================================================
# 最良の単変量モデル
# ============================================================
print(f"¥n{'='*70}")
print("最良の単変量モデル")
print(f"{'='*70}")

best_single = max(single_results.items(), key=lambda x: abs(x[1]['r']))
best_name = best_single[0]
best_arr = predictors[best_name]
mask_bs = np.isfinite(best_arr) &amp;amp; np.isfinite(log_gc)

p_best = np.polyfit(best_arr[mask_bs], log_gc[mask_bs], 1)
pred_best = np.polyval(p_best, best_arr[mask_bs])
rss_best = np.sum((log_gc[mask_bs] - pred_best)**2)
R2_best = 1 - rss_best / np.sum((log_gc[mask_bs] - np.mean(log_gc[mask_bs]))**2)

print(f"¥n  最良変数: {best_name}")
print(f"  log(g_c/a0) = {p_best[1]:.3f} + {p_best[0]:.3f} × {best_name}")
print(f"  R² = {R2_best:.4f}")
print(f"  残差 std = {np.std(log_gc[mask_bs] - pred_best):.4f} dex")

# ============================================================
# 多変量回帰 (段階的)
# ============================================================
print(f"¥n{'='*70}")
print("多変量回帰 (段階的追加) ")
print(f"{'='*70}")

# 相関の強い順にソート
sorted_preds = sorted(single_results.items(), key=lambda x: abs(x[1]['r']), reverse=True)
pred_names_sorted = [x[0] for x in sorted_preds]

# 全データが有効な共通マスク
common_mask = np.ones(N, dtype=bool)
for pname in pred_names_sorted[:6]:
    common_mask &amp;amp;= np.isfinite(predictors[pname])
common_mask &amp;amp;= np.isfinite(log_gc)
N_common = common_mask.sum()
print(f"¥n共通有効データ: {N_common}/{N}")

y = log_gc[common_mask]
# 段階的に変数追加
print(f"¥n{'変数数':&gt;6s}  {'追加変数':&gt;15s}  {'R²':&gt;8s}  {'ΔR²':&gt;8s}  {'BIC':&gt;10s}  {'残差std':&gt;10s}")

best_bic = np.inf
best_model = None
prev_R2 = 0

for n_vars in range(1, min(7, len(pred_names_sorted)+1)):
    # 前のモデルの変数 + 残りから1つ追加
    if n_vars == 1:
        # 全単変量から最良を選択
        best_combo = None
        best_R2_combo = -1
        for pname in pred_names_sorted:
            X = np.column_stack([np.ones(N_common), predictors[pname][common_mask]])
            b, _, _, _ = np.linalg.lstsq(X, y, rcond=None)
            pred = X @ b
            R2 = 1 - np.sum((y - pred)**2) / np.sum((y - np.mean(y))**2)
            if R2 &gt; best_R2_combo:
                best_R2_combo = R2
                best_combo = [pname]
```

```
                best_b = b
                best_pred = pred
    else:
        # 前の最良モデルに1変数追加
        prev_vars = best_combo.copy()
        best_R2_combo = -1
        for pname in pred_names_sorted:
            if pname in prev_vars:
                continue
            test_vars = prev_vars + [pname]
            X = np.column_stack([np.ones(N_common)] + [predictors[v][common_mask] for v in test_vars])
            b, _, _, _ = np.linalg.lstsq(X, y, rcond=None)
            pred = X @ b
            R2 = 1 - np.sum((y - pred)**2) / np.sum((y - np.mean(y))**2)
            if R2 &gt; best_R2_combo:
                best_R2_combo = R2
                best_combo = test_vars
                best_b = b
                best_pred = pred

    k = n_vars + 1
    rss = np.sum((y - best_pred)**2)
    bic = N_common * np.log(rss/N_common) + k * np.log(N_common)
    residual_std = np.std(y - best_pred)
    delta_R2 = best_R2_combo - prev_R2

    added = best_combo[-1] if n_vars &gt; 0 else best_combo[0]
    print(f"{n_vars:6d}  {added:&gt;15s}  {best_R2_combo:8.4f}  {delta_R2:+8.4f}  {bic:10.2f}  {residual_std:10.4f}")

    if bic &lt; best_bic:
        best_bic = bic
        best_model = {
            'vars': best_combo.copy(),
            'coeffs': best_b.copy(),
            'R2': best_R2_combo,
            'bic': bic,
            'residual_std': residual_std,
            'pred': best_pred.copy(),
        }

    prev_R2 = best_R2_combo

# ============================================================
# 最良モデルの詳細
# ============================================================
print(f"¥n{'='*70}")
print("最良モデル (BIC最小) ")
print(f"{'='*70}")

bm = best_model
print(f"¥n  変数: {bm['vars']}")
print(f"  R² = {bm['R2']:.4f}")
print(f"  BIC = {bm['bic']:.2f}")
print(f"  残差 std = {bm['residual_std']:.4f} dex")

print(f"¥n  係数:")
print(f"    切片: {bm['coeffs'][0]:+.4f}")
for i, vname in enumerate(bm['vars']):
    print(f"    {vname}: {bm['coeffs'][i+1]:+.4f}")
# 予測式
eq = f"log(g_c/a0) = {bm['coeffs'][0]:.3f}"
for i, vname in enumerate(bm['vars']):
    eq += f" {bm['coeffs'][i+1]:+.3f}·{vname}"
print(f"¥n  予測式: {eq}")

# ============================================================
# g_c = a0 (定数モデル) との比較
# ============================================================
print(f"¥n{'='*70}")
print("g_c 予測モデル vs g_c = a0 (定数) ")
print(f"{'='*70}")

rss_const = np.sum((y - 0)**2)  # g_c = a0 → log(g_c/a0) = 0
rss_model = np.sum((y - bm['pred'])**2)
bic_const = N_common * np.log(rss_const/N_common) + 0  # 0 params

print(f"¥n  g_c = a0:        RSS={rss_const:.2f}, BIC={bic_const:.2f}")
print(f"  予測モデル:       RSS={rss_model:.2f}, BIC={bm['bic']:.2f}")
print(f"  ΔBIC = {bm['bic'] - bic_const:.2f}")

F_stat = ((rss_const - rss_model) / len(bm['vars'])) / (rss_model / (N_common - len(bm['vars']) - 1))
print(f"  F統計量 = {F_stat:.2f}")

if bm['bic'] &lt; bic_const:
    print(f"¥n  → 予測モデルが g_c=a0 より有意に良い")
    print(f"    膜宇宙論の g_c(galaxy) モデルは MOND の g_c=a0 を超える")
else:
    print(f"¥n  → g_c = a0 の方がBICが良い (予測モデルは過剰適合) ")

# ============================================================
# v_flat ビン別の予測精度
# ============================================================
print(f"¥n{'='*70}")
print("v_flat ビン別の予測精度")
print(f"{'='*70}")
```

```
vf_common = np.array([d['vflat'] for d in results])[common_mask]
gc_common = gc[common_mask]
gc_pred = 10**bm['pred']

sort_vf = np.argsort(vf_common)
n_per = max(N_common // 5, 1)
print(f"¥n{'v_flat':&gt;12s}  {'N':&gt;4s}  {'g_c実測':&gt;8s}  {'g_c予測':&gt;8s}  {'g_c=a0':&gt;6s}  {'|err|予測':&gt;10s}  {'|err|a0':&gt;10s}")

for i in range(5):
    i0 = i * n_per
    i1 = (i+1)*n_per if i &lt; 4 else N_common
    idx = sort_vf[i0:i1]
    vf_b = vf_common[idx]
    gc_b = gc_common[idx]
    gcp_b = gc_pred[idx]
    label = f"{vf_b.min():.0f}-{vf_b.max():.0f}"
    err_pred = np.median(np.abs(np.log10(gcp_b) - np.log10(gc_b)))
    err_a0 = np.median(np.abs(np.log10(gc_b)))
    print(f"{label:&gt;12s}  {len(idx):4d}  {np.median(gc_b):8.3f}  {np.median(gcp_b):8.3f}  {'1.000':&gt;6s}  {err_pred:10.3f}  {err_a0:10.3f}")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) 最良単変量: log_gc vs best predictor
ax = axes[0, 0]
ba = best_arr[mask_bs]
ax.scatter(ba, log_gc[mask_bs], s=10, alpha=0.4, c='steelblue')
xx = np.linspace(ba.min(), ba.max(), 100)
ax.plot(xx, np.polyval(p_best, xx), 'r-', linewidth=1.5)
r_val = single_results[best_name]['r']
ax.set_xlabel(best_name)
ax.set_ylabel('log(g_c / a0)')
ax.set_title(f'(a) Best single: {best_name} (r={r_val:.3f})')

# (b) 実測 vs 予測 (最良モデル)
ax = axes[0, 1]
ax.scatter(bm['pred'], y, s=10, alpha=0.4, c='green')
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--')
ax.set_xlabel('Predicted log(g_c/a0)')
ax.set_ylabel('Observed log(g_c/a0)')
ax.set_title(f'(b) Best model (R²={bm["R2"]:.3f})')

# (c) 残差 vs v_flat
ax = axes[0, 2]
resid = y - bm['pred']
ax.scatter(vf_common, resid, s=10, alpha=0.4, c='orange')
ax.axhline(0, color='black', ls='--')
rho_res, _ = spearmanr(vf_common, resid)
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('Residual (obs - pred)')
ax.set_title(f'(c) Residual vs v_flat (ρ={rho_res:.3f})')

# (d) g_c 実測 vs g_c=a0 vs 予測
ax = axes[1, 0]
ax.scatter(vf_common, gc_common, s=10, alpha=0.4, c='steelblue', label='Measured')
ax.scatter(vf_common, gc_pred, s=10, alpha=0.4, c='red', label='Predicted')
ax.axhline(1.0, color='gray', ls='--', label='g_c = a0')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('g_c / a0')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title('(d) g_c: measured vs predicted vs a0')
ax.legend(fontsize=8)

# (e) 相関係数の棒グラフ
ax = axes[1, 1]
sorted_r = sorted(single_results.items(), key=lambda x: abs(x[1]['r']), reverse=True)
names_bar = [x[0] for x in sorted_r[:8]]
r_vals = [x[1]['r'] for x in sorted_r[:8]]
colors_bar = ['steelblue' if r &gt; 0 else 'coral' for r in r_vals]
ax.barh(range(len(names_bar)), r_vals, color=colors_bar)
ax.set_yticks(range(len(names_bar)))
ax.set_yticklabels(names_bar, fontsize=8)
ax.set_xlabel('Pearson r with log(g_c)')
ax.set_title('(e) Predictor ranking')
ax.axvline(0, color='black', linewidth=0.5)

# (f) 予測精度の比較: モデル vs a0
ax = axes[1, 2]
err_model = np.abs(y - bm['pred'])
err_a0 = np.abs(y)
ax.hist(err_model, bins=30, alpha=0.5, color='green', label=f'Model (med={np.median(err_model):.3f})')
ax.hist(err_a0, bins=30, alpha=0.5, color='gray', label=f'g_c=a0 (med={np.median(err_a0):.3f})')
ax.set_xlabel('|log(g_c_obs) - log(g_c_pred)| [dex]')
ax.set_ylabel('Count')
ax.set_title('(f) Prediction error comparison')
ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('gc_predictive_model.png', dpi=150)
print(f"¥n[SAVED] gc_predictive_model.png")

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

```
# CSV出力
# ============================================================
out_csv = "gc_predictive_model.csv"
with open(out_csv, "w", newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    header = ['galaxy', 'v_flat', 'gc_measured', 'gc_predicted'] + list(predictors.keys())
    writer.writerow(header)
    for i, d in enumerate(results):
        if not common_mask[i]:
            continue
        idx_cm = np.sum(common_mask[:i])
        row_data = [d['name'], f"{d['vflat']:.1f}",
                    f"{d['gc']:.4f}", f"{10**bm['pred'][idx_cm]:.4f}"]
        for pname in predictors:
            row_data.append(f"{predictors[pname][i]:.4f}")
        writer.writerow(row_data)
print(f"[SAVED] {out_csv}")

# ============================================================
# 最終結論
# ============================================================
print(f"¥n{'='*70}")
print("g_c 予測モデルの最終結論")
print(f"{'='*70}")

print(f"""
■ 最良の予測変数: {best_name} (r={single_results[best_name]['r']:.3f})
■ 最良モデル: {eq}
  R² = {bm['R2']:.4f}
  残差 std = {bm['residual_std']:.4f} dex

■ g_c = a0 (MOND) との比較:
  ΔBIC = {bm['bic'] - bic_const:.2f}
  (負なら予測モデルが優位、正なら g_c=a0 が優位)

■ 予測誤差の比較:
  予測モデル: |error| 中央値 = {np.median(err_model):.3f} dex
  g_c = a0:   |error| 中央値 = {np.median(err_a0):.3f} dex
  改善率: {(1 - np.median(err_model)/np.median(err_a0))*100:.0f}%

■ 物理的意味:
""")

if bm['bic'] &lt; bic_const - 10:
    print("  g_c は銀河パラメータで予測可能 → MOND の g_c=a0 を明確に超える")
    print(f"  → 膜の「臨界加速度」は普遍定数ではなく銀河環境の関数")
    print(f"  → 条件14（膜の物性の銀河依存性）の間接的支持")
elif bm['bic'] &lt; bic_const:
    print("  g_c の予測モデルが g_c=a0 よりわずかに良い")
    print("  → g_c の銀河依存性は存在するが、追加パラメータに見合う改善は限定的")
else:
    print("  g_c = a0 の方が良い → g_c の銀河依存性は過剰適合")
    print("  → 純粋 MOND で十分")
```

22. zero_param_test.py

項目	内容
フェーズ	ゼロparamテスト
目的	g_c(predicted) vs g_c=a0 のゼロパラメータ回転曲線フィット比較
使用rs	rs不使用（安全）
結果	dAIC=-57、70%で改善。矮小銀河で97%。
ステータス	Level A（インサンプル）

解析目的

予測式で g_c を算出（0 param）し、MOND（g_c=a0, 0 param）と回転曲線の chi2
を直接比較。同じパラメータ数での公平なテスト。

ソースコード全文

```
#!/usr/bin/env python3
"""
ゼロパラメータ予測テスト: 膜宇宙論 vs MOND の最終判定

3モデルを全銀河の回転曲線に適用:

  Model A: g_c = a0（純粋MOND、0 param）
  Model B: g_c = 予測式（銀河パラメータから算出、0 param）
  Model C: g_c = free（各銀河で個別フィット、1 param）

g_c の予測式（前スクリプトで確立）:
  log(gc/a0) = -2.175 + 2.015·log(vf) - 0.046·log(Vp)
               - 1.294·log(hR) + 0.138·disk_dom
               - 1.024·log(compact) - 0.217·log(Ud)

B が A より有意に良い → 膜宇宙論が MOND を超える
B ≈ C → 予測モデルが g_c の情報を完全に捉えている

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

import csv
import os
import numpy as np
from scipy.optimize import minimize_scalar
from scipy.stats import wilcoxon
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
a0_unit = 3703.0

def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

def mond_gobs(gN, gc):
    return (gN + np.sqrt(gN**2 + 4*gc*gN)) / 2

def chi2_mond_fixed(gc_a0, r, v_obs, err_v, gN):
    """固定g_cでのchi2 (0 param) """
    gc = gc_a0 * a0_unit
    gN_pos = np.maximum(gN, 1e-10)
    g_model = mond_gobs(gN_pos, gc)
    v_model = np.sqrt(np.maximum(r * g_model, 0.01))
    w = 1.0 / np.maximum(err_v, 1.0)**2
    return np.sum(w * (v_obs - v_model)**2)

def fit_gc_free(r, v_obs, err_v, gN):
    """g_c自由フィット（1 param）"""
    def neg(log_gc):
        gc = 10**log_gc * a0_unit
        gN_pos = np.maximum(gN, 1e-10)
        g_model = mond_gobs(gN_pos, gc)
```

```
        v_model = np.sqrt(np.maximum(r * g_model, 0.01))
        w = 1.0 / np.maximum(err_v, 1.0)**2
        return np.sum(w * (v_obs - v_model)**2)

    result = minimize_scalar(neg, bounds=(-2.0, 2.0), method='bounded')
    return 10**result.x, result.fun

# ============================================================
# g_c 予測式の係数
# ============================================================
# 前スクリプトの最良モデル
COEFFS = {
    'intercept': -2.175,
    'log_vflat': 2.015,
    'log_Vpeak': -0.046,
    'log_hR': -1.294,
    'disk_dom': 0.138,
    'log_compact': -1.024,
    'log_Ud': -0.217,
}

def predict_gc(vflat, V_peak, h_R, disk_dom, compactness, Ud):
    """予測式から g_c/a0 を計算"""
    log_gc = (COEFFS['intercept']
              + COEFFS['log_vflat'] * np.log10(vflat)
              + COEFFS['log_Vpeak'] * np.log10(max(V_peak, 0.1))
              + COEFFS['log_hR'] * np.log10(max(h_R, 0.01))
              + COEFFS['disk_dom'] * disk_dom
              + COEFFS['log_compact'] * np.log10(max(compactness, 0.01))
              + COEFFS['log_Ud'] * np.log10(max(Ud, 0.01)))
    return 10**log_gc

# ============================================================
# データ読み込みと銀河パラメータ計算
# ============================================================
_, src_rows = load_csv("sparc_results.csv")
gal_info = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    ud = get_val(row, ['ud'])
    vf = get_val(row, ['vflat'])
    if ud and vf and vf &gt; 0:
        gal_info[name] = {'ud': ud, 'vflat': vf}

print(f"[INFO] {len(gal_info)} galaxies")

# ============================================================
# メインループ
# ============================================================
print("¥nゼロパラメータテスト実行中...")

results = []
n_done = 0

for name, info in gal_info.items():
    data = read_dat(name)
    if data is None:
        continue

    rad = data[:, 0]
    v_obs = data[:, 1]
    err_v = data[:, 2]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = info['ud']
    vf = info['vflat']

    mask = rad &gt; 0.01
    r = rad[mask]
    vo = v_obs[mask]
    ev = err_v[mask]
    vd = v_disk[mask]
    vg = v_gas[mask]
    vb = v_bul[mask]
    N_pts = len(r)

    if N_pts &lt; 5:
        continue

    vbar2 = ud * np.sign(vd)*vd**2 + np.sign(vg)*vg**2 + np.sign(vb)*vb**2
    gN = np.maximum(vbar2 / r, 1e-10)

    # 銀河パラメータ
    vdisk_scaled = np.sqrt(ud) * np.abs(vd)
    V_peak = np.max(vdisk_scaled)
    i_peak = np.argmax(vdisk_scaled)
    r_peak = r[i_peak]

    if r_peak &gt;= r.max() * 0.9 or r_peak &lt; 0.01 or V_peak &lt; 0.1:
        h_R = np.median(r) / 2.15
        V_peak = max(V_peak, 0.1)
    else:
```

```
        h_R = r_peak / 2.15

    v_bar = np.sqrt(np.maximum(vbar2, 0.0))
    V_peak_bar = np.max(v_bar)
    disk_dom = V_peak / vf
    compactness = V_peak_bar / h_R

    # --- Model A: g_c = a0 (0 param) ---
    chi2_A = chi2_mond_fixed(1.0, r, vo, ev, gN)

    # --- Model B: g_c = predicted (0 param) ---
    gc_pred = predict_gc(vf, V_peak, h_R, disk_dom, compactness, ud)
    chi2_B = chi2_mond_fixed(gc_pred, r, vo, ev, gN)

    # --- Model C: g_c = free (1 param) ---
    gc_free, chi2_C = fit_gc_free(r, vo, ev, gN)

    # chi2/dof
    dof_0p = max(N_pts, 1)
    dof_1p = max(N_pts - 1, 1)

    # AIC (パラメータ数で補正)
    aic_A = chi2_A + 2 * 0  # 0 param
    aic_B = chi2_B + 2 * 0  # 0 param
    aic_C = chi2_C + 2 * 1  # 1 param

    results.append({
        'name': name,
        'vflat': vf,
        'N_pts': N_pts,
        'gc_pred': gc_pred,
        'gc_free': gc_free,
        'chi2_A': chi2_A / dof_0p,
        'chi2_B': chi2_B / dof_0p,
        'chi2_C': chi2_C / dof_1p,
        'chi2_raw_A': chi2_A,
        'chi2_raw_B': chi2_B,
        'chi2_raw_C': chi2_C,
        'aic_A': aic_A,
        'aic_B': aic_B,
        'aic_C': aic_C,
        'daic_BA': aic_B - aic_A,
        'daic_CA': aic_C - aic_A,
    })

    n_done += 1
    if n_done % 25 == 0:
        print(f"  {n_done} completed...")

N = len(results)
print(f"¥n完了: {N} galaxies")

# 配列化
chi2_A = np.array([d['chi2_A'] for d in results])
chi2_B = np.array([d['chi2_B'] for d in results])
chi2_C = np.array([d['chi2_C'] for d in results])
daic_BA = np.array([d['daic_BA'] for d in results])
daic_CA = np.array([d['daic_CA'] for d in results])
vf_arr = np.array([d['vflat'] for d in results])
gc_pred = np.array([d['gc_pred'] for d in results])
gc_free = np.array([d['gc_free'] for d in results])

# ============================================================
# 結果解析
# ============================================================
print(f"¥n{'='*70}")
print("ゼロパラメータ予測テスト結果")
print(f"{'='*70}")

print(f"¥n{'モデル':&gt;30s}  {'param':&gt;5s}  {'chi2/dof中央値':&gt;14s}")
print(f"{'A: g_c = a0 (MOND)':&gt;30s}  {'0':&gt;5s}  {np.median(chi2_A):14.3f}")
print(f"{'B: g_c = predicted':&gt;30s}  {'0':&gt;5s}  {np.median(chi2_B):14.3f}")
print(f"{'C: g_c = free':&gt;30s}  {'1':&gt;5s}  {np.median(chi2_C):14.3f}")

# B vs A (核心の比較、同じパラメータ数)
B_better = np.sum(daic_BA &lt; 0)
A_better = np.sum(daic_BA &gt; 0)
B_sig_better = np.sum(daic_BA &lt; -2)
A_sig_better = np.sum(daic_BA &gt; 2)

print(f"¥n--- B(predicted) vs A(MOND) --- [同じパラメータ数: 0]")
print(f"  ΔAIC(B-A) 中央値: {np.median(daic_BA):+.2f}")
print(f"  B が良い: {B_better}/{N} ({100*B_better/N:.0f}%)")
print(f"  A が良い: {A_better}/{N} ({100*A_better/N:.0f}%)")
print(f"  B が有意に良い (ΔAIC&lt;-2): {B_sig_better}/{N} ({100*B_sig_better/N:.0f}%)")
print(f"  A が有意に良い (ΔAIC&gt;+2): {A_sig_better}/{N} ({100*A_sig_better/N:.0f}%)")

# Wilcoxon符号順位検定
try:
    stat_BA, p_BA = wilcoxon(chi2_A - chi2_B)  # 正なら B が良い
    print(f"  Wilcoxon検定 (A vs B) : p = {p_BA:.2e}")
except:
    p_BA = None

# B vs C (予測 vs 自由フィット)
```

```
print(f"¥n--- B(predicted, 0p) vs C(free, 1p) ---")
print(f"  chi2/dof中央値: B={np.median(chi2_B):.3f}, C={np.median(chi2_C):.3f}")
print(f"  B/C 比中央値: {np.median(chi2_B/chi2_C):.3f}")

# C vs A
C_better_A = np.sum(daic_CA &amp;lt; 0)
print(f"¥n--- C(free, 1p) vs A(MOND, 0p) ---")
print(f"  ΔAIC(C-A) 中央値: {np.median(daic_CA):+.2f}")
print(f"  C が良い: {C_better_A}/{N} ({100*C_better_A/N:.0f}%)")

# v_flat ビン別
print(f"¥n--- v_flat ビン別 ---")
sort_vf = np.argsort(vf_arr)
n_per = max(N // 5, 1)
print(f"{'v_flat':&gt;12s}  {'N':&gt;4s}  {'chi2_A':&gt;8s}  {'chi2_B':&gt;8s}  {'chi2_C':&gt;8s}  {'B&lt;A%':&gt;6s}  {'med ΔAIC':&gt;10s}")

for i in range(5):
    i0 = i * n_per
    i1 = (i+1)*n_per if i &lt; 4 else N
    idx = sort_vf[i0:i1]
    vf_b = vf_arr[idx]
    label = f"{vf_b.min():.0f}-{vf_b.max():.0f}"
    ba_pct = 100*np.sum(daic_BA[idx] &lt; 0)/len(idx)
    med_daic = np.median(daic_BA[idx])
    print(f"{label:&gt;12s}  {len(idx):4d}  {np.median(chi2_A[idx]):8.3f}  {np.median(chi2_B[idx]):8.3f}  {np.median(chi2_C[idx]):8.3f}  {ba_pct:5.0f}%  {med_daic:+10.2f}")

# gc_pred vs gc_free の相関
r_pf, p_pf = np.corrcoef(np.log10(gc_pred), np.log10(gc_free))[0,1], 0
from scipy.stats import pearsonr as pr
r_pf, p_pf = pr(np.log10(gc_pred), np.log10(gc_free))
print(f"¥n--- g_c(predicted) vs g_c(free) ---")
print(f"  Pearson r = {r_pf:.3f} (p={p_pf:.2e})")
print(f"  g_c(pred)/g_c(free) 中央値: {np.median(gc_pred/gc_free):.3f}")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) chi2/dof の箱ひげ図
ax = axes[0, 0]
bp = ax.boxplot([chi2_A, chi2_B, chi2_C],
                labels=['A: g_c=a0¥n(MOND, 0p)', 'B: g_c=pred¥n(0p)', 'C: g_c=free¥n(1p)'],
                showfliers=False, patch_artist=True)
for patch, color in zip(bp['boxes'], ['lightcoral', 'lightblue', 'lightgreen']):
    patch.set_facecolor(color)
ax.set_ylabel('χ²/dof')
ax.set_title(f'(a) Model comparison (N={N})')

# (b) ΔAIC(B-A) ヒストグラム
ax = axes[0, 1]
ax.hist(daic_BA, bins=50, color='steelblue', edgecolor='white', alpha=0.7)
ax.axvline(0, color='red', ls='--', linewidth=2)
ax.axvline(np.median(daic_BA), color='black', ls='-',
           label=f'Median={np.median(daic_BA):+.1f}')
ax.set_xlabel('ΔAIC (B - A)')
ax.set_ylabel('Count')
p_str = f', p={p_BA:.1e}' if p_BA else ''
ax.set_title(f'(b) Predicted vs MOND{p_str}')
ax.legend(fontsize=8)

# (c) ΔAIC(B-A) vs v_flat
ax = axes[0, 2]
ax.scatter(vf_arr, daic_BA, s=10, alpha=0.4, c='steelblue')
ax.axhline(0, color='red', ls='--')
ax.axhline(-2, color='gray', ls=':', alpha=0.5)
ax.axhline(2, color='gray', ls=':', alpha=0.5)
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('ΔAIC (B - A)')
ax.set_title('(c) Improvement vs v_flat')

# (d) g_c(pred) vs g_c(free)
ax = axes[1, 0]
ax.scatter(gc_free, gc_pred, s=10, alpha=0.4, c='purple')
maxv = max(gc_pred.max(), gc_free.max())
ax.plot([0.01, maxv], [0.01, maxv], 'k--', alpha=0.3)
ax.axhline(1.0, color='gray', ls=':', alpha=0.3)
ax.axvline(1.0, color='gray', ls=':', alpha=0.3)
ax.set_xlabel('g_c (free fit) / a0')
ax.set_ylabel('g_c (predicted) / a0')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title(f'(d) g_c: predicted vs free (r={r_pf:.3f})')

# (e) chi2_B vs chi2_A (銀河ごと)
ax = axes[1, 1]
ax.scatter(chi2_A, chi2_B, s=10, alpha=0.4, c='green')
maxchi = max(np.percentile(chi2_A, 95), np.percentile(chi2_B, 95))
ax.plot([0, maxchi], [0, maxchi], 'k--', alpha=0.3)
ax.set_xlabel('χ²/dof (A: MOND)')
ax.set_ylabel('χ²/dof (B: predicted)')
ax.set_title('(e) Per-galaxy: B vs A')
ax.set_xlim(0, maxchi)
ax.set_ylim(0, maxchi)
```

```
# (f) g_c の3つの推定値 vs v_flat
ax = axes[1, 2]
ax.scatter(vf_arr, gc_free, s=8, alpha=0.3, c='blue', label='g_c(free)')
ax.scatter(vf_arr, gc_pred, s=8, alpha=0.3, c='red', label='g_c(pred)')
ax.axhline(1.0, color='gray', ls='--', linewidth=2, label='g_c = a0')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('g_c / a0')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title('(f) g_c: free vs predicted vs a0')
ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('zero_param_test.png', dpi=150)
print(f"¥n[SAVED] zero_param_test.png")

# ============================================================
# CSV出力
# ============================================================
out_csv = "zero_param_test.csv"
with open(out_csv, "w", newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    writer.writerow(['galaxy', 'v_flat', 'gc_pred', 'gc_free',
                     'chi2_A', 'chi2_B', 'chi2_C',
                     'daic_BA', 'daic_CA'])
    for d in sorted(results, key=lambda x: x['daic_BA']):
        writer.writerow([
            d['name'], f"{d['vflat']:.1f}",
            f"{d['gc_pred']:.4f}", f"{d['gc_free']:.4f}",
            f"{d['chi2_A']:.4f}", f"{d['chi2_B']:.4f}", f"{d['chi2_C']:.4f}",
            f"{d['daic_BA']:.2f}", f"{d['daic_CA']:.2f}",
        ])
print(f"[SAVED] {out_csv}")

# ============================================================
# 最終判定
# ============================================================
print(f"¥n{'='*70}")
print("★ ゼロパラメータ予測テスト: 最終判定 ★")
print(f"{'='*70}")

print(f"""
■ テスト結果:

  Model A (g_c = a0, MOND):    chi2/dof = {np.median(chi2_A):.3f}
  Model B (g_c = predicted):    chi2/dof = {np.median(chi2_B):.3f}
  Model C (g_c = free, 1p):    chi2/dof = {np.median(chi2_C):.3f}

  B vs A: ΔAIC中央値 = {np.median(daic_BA):+.2f}
  B が A より良い銀河: {B_better}/{N} ({100*B_better/N:.0f}%)
  Wilcoxon p = {p_BA:.2e if p_BA else 'N/A'}

■ 判定:
""")

if np.median(daic_BA) &lt; -2 and B_better &gt; N*0.6:
    print("  ★★★ 膜宇宙論が MOND を超える ★★★")
    print(f"  g_c(galaxy) の予測モデルが、自由パラメータなしで")
    print(f"  g_c=a0 (純粋MOND) より有意に良い回転曲線フィットを提供する。")
    print(f"  → 膜の「臨界加速度」は銀河のバリオン構造の関数である。")
    print(f"  → Level A に確立。")
elif np.median(daic_BA) &lt; 0 and B_better &gt; N*0.5:
    print("  ★★ 膜宇宙論が MOND よりやや良い ★★")
    print(f"  予測モデルは MOND を平均的に改善するが、ΔAIC は小さい。")
    print(f"  → Level B (示唆的だが決定的ではない) 。")
elif abs(np.median(daic_BA)) &lt; 2:
    print("  ★ 膜宇宙論と MOND は同等 ★")
    print(f"  予測モデルは MOND と統計的に区別できない。")
    print(f"  → g_c の銀河依存性は回転曲線フィットの改善に寄与しない。")
else:
    print("  MOND (g_c=a0) が予測モデルより良い。")
    print(f"  → g_c の銀河依存性は過剰適合の可能性。")

print(f"""
■ g_c 予測の品質:
  g_c(pred) vs g_c(free): r = {r_pf:.3f}
  g_c(pred)/g_c(free) 中央値 = {np.median(gc_pred/gc_free):.3f}
  → 予測が free フィットにどれだけ近いか
""")
```

23. loo_cv.py

項目	内容
フェーズ	LOO-CV
目的	Leave-One-Out 交差検証による過剰適合排除
使用rs	rs不使用（安全）
結果	dAIC=-54（LOO）。劣化6%のみ。p=6.7e-9。過剰適合なし。
ステータス	Level A（最終確定）

解析目的

各銀河を1つずつ除外して残り N-1 銀河で係数を再推定。除外した銀河の g_c を予測し回転曲線を評価。インサンプルの結果が汎化可能かを最終確認。

ソースコード全文

```
#!/usr/bin/env python3
"""
LOO-CV: g_c 予測モデルの交差検証

各銀河を1つずつ除外し、残り N-1 銀河で係数を再推定。
除外した銀河の g_c を予測し、回転曲線の chi2 を計算。
→ インサンプルのバイアスを完全に排除した予測精度の評価。

比較:
  A: g_c = a0 (MOND、0 param)
  B_in: g_c = predicted (インサンプル、前スクリプトの結果)
  B_loo: g_c = predicted (LOO-CV)
  C: g_c = free (1 param)

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

import csv
import os
import numpy as np
from scipy.optimize import minimize_scalar
from scipy.stats import wilcoxon, pearsonr
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================
a0_unit = 3703.0

def load_csv(path):
    with open(path, "r", encoding="utf-8") as fh:
        reader = csv.DictReader(fh)
        return reader.fieldnames, list(reader)

def get_val(row, candidates):
    for c in candidates:
        if c in row and row[c].strip():
            try: return float(row[c])
            except: pass
    return None

def read_dat(name):
    for fn in [f"{name}_rotmod.dat", f"{name}.dat"]:
        if os.path.exists(fn):
            return np.loadtxt(fn, comments='#')
    return None

def mond_gobs(gN, gc):
    return (gN + np.sqrt(gN**2 + 4*gc*gN)) / 2

# ============================================================
# データ収集 (gc_predictive_model.py と同一)
# ============================================================
_, src_rows = load_csv("sparc_results.csv")
gal_info = {}
for row in src_rows:
    name = row.get('galaxy', '').strip()
    if not name: name = list(row.values())[0].strip()
    ud = get_val(row, ['ud'])
    vf = get_val(row, ['vflat'])
    if ud and vf and vf &gt; 0:
        gal_info[name] = {'ud': ud, 'vflat': vf}

gc_dict = {}
if os.path.exists("TA3_gc_independent.csv"):
    _, gc_rows = load_csv("TA3_gc_independent.csv")
    for row in gc_rows:
        name = row.get('galaxy', '').strip()
        gc = get_val(row, ['gc_over_a0'])
```

```
        if name and gc and gc &gt; 0:
            gc_dict[name] = gc

# 銀河パラメータ＋回転曲線データの収集
all_data = []

for name, info in gal_info.items():
    if name not in gc_dict:
        continue
    data = read_dat(name)
    if data is None:
        continue

    rad = data[:, 0]
    v_obs = data[:, 1]
    err_v = data[:, 2]
    v_gas = data[:, 3]
    v_disk = data[:, 4]
    v_bul = data[:, 5] if data.shape[1] &gt; 5 else np.zeros_like(rad)
    ud = info['ud']

    mask = rad &gt; 0.01
    r = rad[mask]
    vo = v_obs[mask]
    ev = err_v[mask]
    vd = v_disk[mask]
    vg = v_gas[mask]
    vb = v_bul[mask]

    if len(r) &lt; 5:
        continue

    vbar2 = ud * np.sign(vd)*vd**2 + np.sign(vg)*vg**2 + np.sign(vb)*vb**2
    gN = np.maximum(vbar2 / r, 1e-10)

    vdisk_scaled = np.sqrt(ud) * np.abs(vd)
    V_peak = np.max(vdisk_scaled)
    i_peak = np.argmax(vdisk_scaled)
    r_peak = r[i_peak]

    if r_peak &gt;= r.max() * 0.9 or r_peak &lt; 0.01 or V_peak &lt; 0.1:
        h_R = np.median(r) / 2.15
        V_peak = max(V_peak, 0.1)
    else:
        h_R = r_peak / 2.15

    v_bar = np.sqrt(np.maximum(vbar2, 0.0))
    V_peak_bar = np.max(v_bar)
    disk_dom = V_peak / info['vflat']
    compactness = V_peak_bar / h_R

    # 予測変数ベクトル
    pred_vec = np.array([
        np.log10(info['vflat']),
        np.log10(max(V_peak, 0.1)),
        np.log10(max(h_R, 0.01)),
        disk_dom,
        np.log10(max(compactness, 0.01)),
        np.log10(max(ud, 0.01)),
    ])

    if not np.all(np.isfinite(pred_vec)):
        continue

    all_data.append({
        'name': name,
        'vflat': info['vflat'],
        'gc_true': gc_dict[name],
        'log_gc': np.log10(gc_dict[name]),
        'pred_vec': pred_vec,
        'r': r, 'vo': vo, 'ev': ev, 'gN': gN,
    })

N = len(all_data)
print(f"[INFO] {N} galaxies for LOO-CV")

# ============================================================
# LOO-CV
# ============================================================
print(f"¥nLOO-CV 実行中 ({N} iterations) ...")

# 予測変数行列と目的変数
X_all = np.column_stack([np.ones(N)] + [d['pred_vec'] for d in all_data])
# X_all の形状を修正
X_all = np.zeros((N, 7))
X_all[:, 0] = 1.0
for i, d in enumerate(all_data):
    X_all[i, 1:] = d['pred_vec']

y_all = np.array([d['log_gc'] for d in all_data])

loo_results = []

for i in range(N):
    # i番目を除外
```

```
    mask_train = np.ones(N, dtype=bool)
    mask_train[i] = False

    X_train = X_all[mask_train]
    y_train = y_all[mask_train]

    # 回帰係数の再推定
    beta, _, _, _ = np.linalg.lstsq(X_train, y_train, rcond=None)

    # i番目の g_c を予測
    log_gc_pred = X_all[i] @ beta
    gc_pred_loo = 10**log_gc_pred

    # 回転曲線の chi2
    d = all_data[i]
    r, vo, ev, gN = d['r'], d['vo'], d['ev'], d['gN']
    w = 1.0 / np.maximum(ev, 1.0)**2

    # Model A: g_c = a0
    gm_A = mond_gobs(gN, 1.0 * a0_unit)
    vm_A = np.sqrt(np.maximum(r * gm_A, 0.01))
    chi2_A = np.sum(w * (vo - vm_A)**2)

    # Model B_loo: g_c = predicted (LOO)
    gm_B = mond_gobs(gN, gc_pred_loo * a0_unit)
    vm_B = np.sqrt(np.maximum(r * gm_B, 0.01))
    chi2_B = np.sum(w * (vo - vm_B)**2)

    # Model C: g_c = free
    def neg_chi2(log_gc):
        gc = 10**log_gc * a0_unit
        gm = mond_gobs(gN, gc)
        vm = np.sqrt(np.maximum(r * gm, 0.01))
        return np.sum(w * (vo - vm)**2)
    res = minimize_scalar(neg_chi2, bounds=(-2.0, 2.0), method='bounded')
    chi2_C = res.fun
    gc_free = 10**res.x

    N_pts = len(r)

    loo_results.append({
        'name': d['name'],
        'vflat': d['vflat'],
        'gc_true': d['gc_true'],
        'gc_pred_loo': gc_pred_loo,
        'gc_free': gc_free,
        'chi2_A': chi2_A / N_pts,
        'chi2_B': chi2_B / N_pts,
        'chi2_C': chi2_C / max(N_pts-1, 1),
        'daic_BA': chi2_B - chi2_A,  # 同じparam数→ΔAICはΔchi2
        'N_pts': N_pts,
    })

    if (i+1) % 25 == 0:
        print(f"  {i+1}/{N} completed...")

print(f"完了")

# ============================================================
# 結果解析
# ============================================================
chi2_A = np.array([d['chi2_A'] for d in loo_results])
chi2_B = np.array([d['chi2_B'] for d in loo_results])
chi2_C = np.array([d['chi2_C'] for d in loo_results])
daic_BA = np.array([d['daic_BA'] for d in loo_results])
vf = np.array([d['vflat'] for d in loo_results])
gc_pred_loo = np.array([d['gc_pred_loo'] for d in loo_results])
gc_free = np.array([d['gc_free'] for d in loo_results])
gc_true = np.array([d['gc_true'] for d in loo_results])

print(f"¥n{'='*70}")
print("LOO-CV 結果")
print(f"{'='*70}")

print(f"¥n{'モデル':&gt;30s}  {'param':&gt;5s}  {'chi2/dof中央値':&gt;14s}")
print(f"{'A: g_c = a0 (MOND)':&gt;30s}  {'0':&gt;5s}  {np.median(chi2_A):14.3f}")
print(f"{'B_loo: g_c = predicted (LOO)':&gt;30s}  {'0':&gt;5s}  {np.median(chi2_B):14.3f}")
print(f"{'C: g_c = free':&gt;30s}  {'1':&gt;5s}  {np.median(chi2_C):14.3f}")

# B_loo vs A
B_better = np.sum(daic_BA &lt; 0)
B_sig = np.sum(daic_BA &lt; -2)
A_sig = np.sum(daic_BA &gt; 2)

print(f"¥n--- B_loo vs A (同じパラメータ数: 0) ---")
print(f"  ΔAIC(B-A) 中央値: {np.median(daic_BA):+.2f}")
print(f"  B が良い: {B_better}/{N} ({100*B_better/N:.0f}%)")
print(f"  B が有意に良い (ΔAIC&lt;-2): {B_sig}/{N} ({100*B_sig/N:.0f}%)")
print(f"  A が有意に良い (ΔAIC&gt;+2): {A_sig}/{N} ({100*A_sig/N:.0f}%)")

try:
    stat, p_wilcox = wilcoxon(chi2_A - chi2_B)
    print(f"  Wilcoxon p = {p_wilcox:.2e}")
except:
    p_wilcox = None
```

```
# インサンプル vs LOO の比較
print(f"¥n--- インサンプル vs LOO-CV の劣化 ---")
print(f"  インサンプル: ΔAIC中央値 = -57.4, B良い = 71%")
print(f"  LOO-CV:       ΔAIC中央値 = {np.median(daic_BA):+.1f}, B良い = {100*B_better/N:.0f}%")

# g_c 予測精度
r_pred, p_pred = pearsonr(np.log10(gc_pred_loo), np.log10(gc_true))
err_pred = np.abs(np.log10(gc_pred_loo) - np.log10(gc_true))
err_a0 = np.abs(np.log10(gc_true))
print(f"¥n--- g_c 予測精度 (LOO) ---")
print(f"  g_c(LOO) vs g_c(RAR true): r = {r_pred:.3f}")
print(f"  |error| 中央値(LOO): {np.median(err_pred):.3f} dex")
print(f"  |error| 中央値(a0):  {np.median(err_a0):.3f} dex")
print(f"  改善率: {(1-np.median(err_pred)/np.median(err_a0))*100:.0f}%")

# v_flat ビン別
print(f"¥n--- v_flat ビン別 (LOO-CV) ---")
sort_vf = np.argsort(vf)
n_per = max(N // 5, 1)
print(f"{'v_flat':&gt;12s}  {'N':&gt;4s}  {'chi2_A':&gt;8s}  {'chi2_B':&gt;8s}  {'B&lt;A%':&gt;6s}  {'ΔAIC':&gt;8s}")
for i in range(5):
    i0 = i * n_per
    i1 = (i+1)*n_per if i &lt; 4 else N
    idx = sort_vf[i0:i1]
    label = f"{vf[idx].min():.0f}-{vf[idx].max():.0f}"
    ba_pct = 100*np.sum(daic_BA[idx] &lt; 0)/len(idx)
    print(f"{label:&gt;12s}  {len(idx):4d}  {np.median(chi2_A[idx]):8.3f}  {np.median(chi2_B[idx]):8.3f}  {ba_pct:5.0f}%  {np.median(daic_BA[idx]):+8.1f}")

# ============================================================
# 図の作成
# ============================================================
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# (a) chi2 箱ひげ図
ax = axes[0, 0]
bp = ax.boxplot([chi2_A, chi2_B, chi2_C],
                labels=['A: MOND¥n(g_c=a0)', 'B: Predicted¥n(LOO-CV)', 'C: Free¥n(1 param)'],
                showfliers=False, patch_artist=True)
for patch, color in zip(bp['boxes'], ['lightcoral', 'lightblue', 'lightgreen']):
    patch.set_facecolor(color)
ax.set_ylabel('χ²/dof')
ax.set_title(f'(a) LOO-CV Model Comparison (N={N})')

# (b) ΔAIC ヒストグラム
ax = axes[0, 1]
ax.hist(daic_BA, bins=50, color='steelblue', edgecolor='white', alpha=0.7)
ax.axvline(0, color='red', ls='--', linewidth=2)
ax.axvline(np.median(daic_BA), color='black', ls='-',
           label=f'LOO median={np.median(daic_BA):+.1f}')
ax.axvline(-57.4, color='gray', ls=':', alpha=0.5,
           label='In-sample median=-57.4')
ax.set_xlabel('ΔAIC (B_loo - A)')
ax.set_ylabel('Count')
ax.set_title('(b) LOO-CV vs In-sample')
ax.legend(fontsize=7)

# (c) ΔAIC vs v_flat
ax = axes[0, 2]
ax.scatter(vf, daic_BA, s=10, alpha=0.4, c='steelblue')
ax.axhline(0, color='red', ls='--')
ax.set_xlabel('v_flat [km/s]')
ax.set_ylabel('ΔAIC (B_loo - A)')
ax.set_title('(c) LOO improvement vs v_flat')

# (d) g_c(LOO predicted) vs g_c(true)
ax = axes[1, 0]
ax.scatter(gc_true, gc_pred_loo, s=10, alpha=0.4, c='green')
maxv = max(gc_true.max(), gc_pred_loo.max())
ax.plot([0.01, maxv], [0.01, maxv], 'k--', alpha=0.3)
ax.set_xlabel('g_c (RAR measured) / a0')
ax.set_ylabel('g_c (LOO predicted) / a0')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_title(f'(d) LOO prediction (r={r_pred:.3f})')

# (e) chi2_B vs chi2_A (銀河ごと)
ax = axes[1, 1]
ax.scatter(chi2_A, chi2_B, s=10, alpha=0.4, c='purple')
maxchi = max(np.percentile(chi2_A, 95), np.percentile(chi2_B, 95))
ax.plot([0, maxchi], [0, maxchi], 'k--', alpha=0.3)
ax.set_xlabel('χ²/dof (MOND)')
ax.set_ylabel('χ²/dof (LOO predicted)')
ax.set_title('(e) Per-galaxy LOO comparison')
ax.set_xlim(0, maxchi)
ax.set_ylim(0, maxchi)

# (f) 予測誤差の比較
ax = axes[1, 2]
ax.hist(err_pred, bins=30, alpha=0.5, color='green',
        label=f'LOO (med={np.median(err_pred):.3f})')
ax.hist(err_a0, bins=30, alpha=0.5, color='gray',
        label=f'g_c=a0 (med={np.median(err_a0):.3f})')
ax.set_xlabel('|log(g_c_pred) - log(g_c_true)| [dex]')
ax.set_ylabel('Count')
ax.set_title('(f) g_c prediction error')
```

```
ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('loo_cv.png', dpi=150)
print(f"¥n[SAVED] loo_cv.png")

# ============================================================
# 最終判定
# ============================================================
print(f"¥n{'='*70}")
print("★ LOO-CV 最終判定 ★")
print(f"{'='*70}")

print(f"""
■ LOO-CV結果:

  MOND (g_c=a0):     chi2/dof = {np.median(chi2_A):.3f}
  Predicted (LOO):    chi2/dof = {np.median(chi2_B):.3f}
  Free fit (1p):      chi2/dof = {np.median(chi2_C):.3f}

  LOO ΔAIC中央値 = {np.median(daic_BA):+.1f}
  B が良い: {B_better}/{N} ({100*B_better/N:.0f}%)
  Wilcoxon p = {p_wilcox:.2e if p_wilcox else 'N/A'}

■ インサンプルからの劣化:
  ΔAIC: -57.4 → {np.median(daic_BA):+.1f}
  B良い%: 71% → {100*B_better/N:.0f}%
  g_c予測精度: r=0.908(in) → r={r_pred:.3f}(LOO)

■ 最終判定:
""")

if np.median(daic_BA) &lt; -10 and B_better &gt; N*0.6 and p_wilcox and p_wilcox &lt; 0.001:
    print("  ★★★ LOO-CVでも膜宇宙論が MOND を明確に超える ★★★")
    print(f"  過剰適合ではない。g_c(galaxy) モデルは汎化可能。")
    print(f"  → Level A に確立。")
elif np.median(daic_BA) &lt; -2 and B_better &gt; N*0.55:
    print("  ★★ LOO-CVで膜宇宙論の優位性が持続 ★★")
    print(f"  インサンプルより弱まったが依然として有意。")
    print(f"  → Level A（条件付き）。")
elif np.median(daic_BA) &lt; 0 and B_better &gt; N*0.5:
    print("  ★ 弱い優位性が LOO-CV でも残存 ★")
    print(f"  → Level B。")
else:
    print("  LOO-CV で改善が消失。インサンプルの結果は過剰適合。")
    print(f"  → g_c = a0 (MOND) で十分。")
```