膜宇宙論モデル

観測データ解析スクリプト集

理論導出 + 弱重力レンズ + 独立検証

全20スクリプト 完全収録

2026年4月 | 坂口製麺所

本書は膜宇宙論モデルの検証に使用した全解析スクリプトを、目的・結果・全文とともに収録したものである。

目次

A. 幾何平均法則の確立（理論導出）

B. 弱重力レンズ解析（HSC-SSP Y3）

C. 独立データセット検証（N-2）

A. 幾何平均法則の確立（理論導出）

gc_sigma0_theory_test.py

解析目的	G*Sigma0 基本相関の検証。slope=0.55 の発見（slope=1 ではない）。幾何平均法則の出発点。
結果	A13 確立。r=0.725, slope=0.55。

```
"""
g_c ∝ G·Σ■·F(c) ■■■■■■■■■■
========================================
■■■■:
  g_c = η · G · Σ■ · h_R · 2(1 − √c)
  where c = c(galaxy type), Σ■ ∝ v_flat² / h_R

■■■■:
  (1) c(Σ■) ■■■■■■■■■■
  (2) G·Σ■ ■■■ g_c ■■■■■■■■■■■■
  (3) F(c)·Σ■ ■■■■■■■■■
  (4) ■■■■ 1.3 ■ c(Σ■) ■■■■■■■■■■

■■: uv run --with scipy --with matplotlib --with numpy python gc_sigma0_theory_test.py
"""

import numpy as np
import pandas as pd
from scipy import stats
from pathlib import Path
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ======================================================================
# 0. ■■■■■■■■ &amp;amp; ■■■■■■
# ======================================================================
work_dir = Path(".")

# --- gc ■■■■■■■■ ---
gc_file = work_dir / "TA3_gc_independent.csv"
pred_file = work_dir / "gc_predictive_model.csv"
sparc_file = work_dir / "sparc_results.csv"

print("=" * 70)
print("Step 0: ■■■■■■■■ &amp;amp; ■■■■■■")
print("=" * 70)

for f in [gc_file, pred_file, sparc_file]:
    if f.exists():
        df_tmp = pd.read_csv(f)
        print(f"\n{f.name}: {len(df_tmp)} rows")
        print(f"  columns: {list(df_tmp.columns)}")
    else:
        print(f"\n{f.name}: NOT FOUND")

# --- ■■■■■■■■■■■ ---
df_gc = pd.read_csv(gc_file) if gc_file.exists() else None
df_pred = pd.read_csv(pred_file) if pred_file.exists() else None
df_sparc = pd.read_csv(sparc_file) if sparc_file.exists() else None

# ======================================================================
# 1. ■■■■■■■■■■■
# ======================================================================
print("\n" + "=" * 70)
print("Step 1: ■■■■■■■")
print("=" * 70)

# gc, v_flat, h_R, galaxy type ■■■
# ■■■■■■ CSV ■■■■■■■■■■■■■■■

def find_col(df, candidates, label=""):
    """■■■■■■■■■■■■■■■"""
    if df is None:
        return None
    for c in candidates:
        matches = [col for col in df.columns if c.lower() in col.lower()]
        if matches:
            print(f"  {label}: '{matches[0]}' matched (from '{c}')")
            return matches[0]
    print(f"  {label}: NOT FOUND (tried {candidates})")
    return None

# --- gc ■■■■■ ---
# TA3_gc_independent.csv ■■ rs ■■■■■■ g_c ■■■■■■■
if df_gc is not None:
```

```
    gc_col = find_col(df_gc, ['gc', 'g_c', 'gc_rar', 'gc_ind'], 'g_c')
    galaxy_col_gc = find_col(df_gc, ['galaxy', 'Galaxy', 'name', 'Name'], 'galaxy(gc)')

# --- ■■■■■■■■■■ ---
# sparc_results.csv ■■ v_flat, h_R, galaxy type
if df_sparc is not None:
    vflat_col = find_col(df_sparc, ['v_flat', 'vflat', 'Vflat'], 'v_flat')
    hR_col = find_col(df_sparc, ['h_R', 'hR', 'Reff', 'R_d', 'Rd'], 'h_R')
    type_col = find_col(df_sparc, ['type', 'Type', 'T', 'hubble', 'Hubble'], 'type')
    galaxy_col_sparc = find_col(df_sparc, ['galaxy', 'Galaxy', 'name', 'Name'], 'galaxy(sparc)')

# --- gc_predictive_model.csv ■■■ ---
if df_pred is not None:
    gc_col_pred = find_col(df_pred, ['gc', 'g_c', 'gc_pred', 'gc_obs', 'gc_rar'], 'g_c(pred)')
    vflat_col_pred = find_col(df_pred, ['v_flat', 'vflat', 'Vflat'], 'v_flat(pred)')
    hR_col_pred = find_col(df_pred, ['h_R', 'hR', 'Reff', 'R_d', 'Rd'], 'h_R(pred)')

print("\n--- ■■■■■■■■■■■■ ---")
if df_gc is not None:
    print(f"TA3: {list(df_gc.columns)}")
if df_sparc is not None:
    print(f"SPARC: {list(df_sparc.columns)}")
if df_pred is not None:
    print(f"PRED: {list(df_pred.columns)}")

# ======================================================================
# 2. ■■■■■
# ======================================================================
print("\n" + "=" * 70)
print("Step 2: ■■■■■")
print("=" * 70)

# ■■: gc_predictive_model.csv ■■■■■■■■■■■■■■■■
# ■■■■ TA3 + sparc ■ galaxy ■■■■

df = None  # ■■■■■■■■ DataFrame

# ■■ gc_predictive_model.csv ■■■■■■■■■■■■■■■■
if df_pred is not None:
    required_found = True
    cols_map = {}

    for key, candidates in [
        ('gc', ['gc', 'g_c', 'gc_obs', 'gc_rar', 'gc_meas']),
        ('vflat', ['v_flat', 'vflat', 'Vflat']),
        ('hR', ['h_R', 'hR', 'Reff', 'R_d', 'Rd']),
    ]:
        col = find_col(df_pred, candidates, f'merge-{key}')
        if col:
            cols_map[key] = col
        else:
            required_found = False

    if required_found:
        df = df_pred.copy()
        df['gc_val'] = df[cols_map['gc']]
        df['vflat_val'] = df[cols_map['vflat']]
        df['hR_val'] = df[cols_map['hR']]
        print(f"  → gc_predictive_model.csv ■■■ (N={len(df)})")
    else:
        print("  → gc_predictive_model.csv: ■■■■■■■■")

# ■■■■■■■■: TA3 + sparc ■■
if df is None and df_gc is not None and df_sparc is not None:
    print("  → TA3 + SPARC ■■■■")
    df = pd.merge(df_gc, df_sparc,
                  left_on=galaxy_col_gc, right_on=galaxy_col_sparc,
                  how='inner')
    if gc_col and vflat_col and hR_col:
        df['gc_val'] = df[gc_col]
        df['vflat_val'] = df[vflat_col]
        df['hR_val'] = df[hR_col]
        print(f"  → ■■■■■ (N={len(df)})")

if df is None:
    print("\n!!! ■■■■■■■■■CSV ■■■■■■■■■■■■■■■■■")
    print("    ■■: g_c, v_flat, h_R ■■■■ CSV")
    import sys; sys.exit(1)

# --- galaxy type ■■■■ ---
type_col_final = find_col(df, ['type', 'Type', 'T', 'hubble', 'Hubble', 'morph'], 'type(final)')

# NaN ■■
mask = df['gc_val'].notna() &amp;amp; df['vflat_val'].notna() &amp;amp; df['hR_val'].notna()
mask &amp;amp;= (df['gc_val'] &amp;gt; 0) &amp;amp; (df['vflat_val'] &amp;gt; 0) &amp;amp; (df['hR_val'] &amp;gt; 0)
df = df[mask].copy()
print(f"  ■■■■■■: N={len(df)}")
```

```
# ======================================================================
# 3. ■■■■■■
# ======================================================================
print("\n" + "=" * 70)
print("Step 3: ■■■■■■■")
print("=" * 70)

a0 = 1.2e-10  # m/s²
G = 6.674e-11  # m³/(kg·s²)
kpc_to_m = 3.086e19
kms_to_ms = 1e3
Msun = 1.989e30

# g_c ■ a0 ■■ → SI ■■■■CSV ■ a0 ■■■■■■
gc_vals = df['gc_val'].values
# ■■: gc ■■■■ ~1 ■■ a0 ■■■■~1e-10 ■■ SI ■■
gc_median = np.median(gc_vals)
if gc_median &lt; 1e-5:
    print(f"  g_c ■ SI ■■■■■ (■■■={gc_median:.2e})")
    gc_si = gc_vals
    gc_a0 = gc_vals / a0
else:
    print(f"  g_c ■ a0 ■■■■■ (■■■={gc_median:.3f})")
    gc_a0 = gc_vals
    gc_si = gc_vals * a0

# v_flat [km/s] → [m/s]
vflat_kms = df['vflat_val'].values
vflat_ms = vflat_kms * kms_to_ms

# h_R [kpc] → [m]
hR_kpc = df['hR_val'].values
hR_m = hR_kpc * kpc_to_m

# Σ■ ∝ v_flat² / (G · h_R) ■■■■■■
# ■■■■ Σ■ = M_disk / (2π h_R²), v_flat² ~ G·M/(h_R) ■■■
# v_flat² / (G·h_R) ■■■■■■■■■
Sigma0_proxy = vflat_ms**2 / (G * hR_m)  # [kg/m²]
Sigma0_Msun_pc2 = Sigma0_proxy / (Msun / (3.086e16)**2)  # [Msun/pc²]

# G · Σ■ [m/s²] — ■■■■■ g_c ■■■■
G_Sigma0 = G * Sigma0_proxy  # = v_flat² / h_R [m/s²]

print(f"  Σ■ proxy ■■■: {np.median(Sigma0_Msun_pc2):.1f} Msun/pc²")
print(f"  G·Σ■ ■■■: {np.median(G_Sigma0):.2e} m/s²")
print(f"  g_c ■■■: {np.median(gc_si):.2e} m/s²")
print(f"  g_c / (G·Σ■) ■■■: {np.median(gc_si / G_Sigma0):.4f}")

# ======================================================================
# 4. ■■(1): g_c vs G·Σ■ ■■■■■
# ======================================================================
print("\n" + "=" * 70)
print("Step 4: g_c vs G·Σ■ ■■■■■")
print("=" * 70)

log_gc = np.log10(gc_si)
log_GSigma0 = np.log10(G_Sigma0)

# ■■■■
slope, intercept, r, p, se = stats.linregress(log_GSigma0, log_gc)
print(f"  log(g_c) = {slope:.3f} * log(G·Σ■) + {intercept:.3f}")
print(f"  r = {r:.3f}, R² = {r**2:.3f}, p = {p:.2e}")
print(f"  ■■■■ = {slope:.3f} ■■■■■: 1.0■■■■: ~1■■■■■■■")

# v_flat²/h_R^1.3 ■■■
log_proxy13 = np.log10(vflat_ms**2 / hR_m**1.3)
slope13, intercept13, r13, p13, se13 = stats.linregress(log_proxy13, log_gc)
print(f"\n  log(g_c) vs log(v²/h_R^1.3):")
print(f"  r = {r13:.3f}, R² = {r13**2:.3f}")

# Spearman■■■■■■■■■
rho_sp, p_sp = stats.spearmanr(log_GSigma0, log_gc)
print(f"\n  Spearman: ρ = {rho_sp:.3f}, p = {p_sp:.2e}")

# ======================================================================
# 5. ■■(2): ■■■■■■■■ c ■■ F(c) ■■
# ======================================================================
print("\n" + "=" * 70)
print("Step 5: ■■■■■■■ c ■■ F(c) ■■")
print("=" * 70)

# c ■■■■■■■■SPARC ■ Hubble type ■■■■■■
# Im/BCD: c=0.30, Sd/Sm: c=0.35, Sc: c=0.42, Sb: c=0.80, Sa: c=0.90
# T type: 10=Im, 9=Sm, 8=Sdm, 7=Sd, 6=Scd, 5=Sc, 4=Sbc, 3=Sb, 2=Sab, 1=Sa
c_type_map = {
    10: 0.30, 11: 0.30,  # Im, BCD
    9: 0.33,   # Sm
```

```
    8: 0.35,   # Sdm
    7: 0.37,   # Sd
    6: 0.40,   # Scd
    5: 0.42,   # Sc
    4: 0.55,   # Sbc
    3: 0.80,   # Sb
    2: 0.85,   # Sab
    1: 0.90,   # Sa
    0: 0.90,   # S0
}

# type ■■■■■■■■■■■■■■■■■ c=0.42 ■■■
if type_col_final and type_col_final in df.columns:
    type_vals = df[type_col_final].values
    print(f"  ■■■■■■■■■: {type_col_final}")
    print(f"  ■■■■■■: {pd.Series(type_vals).value_counts().sort_index().to_dict()}")

    c_vals = np.array([c_type_map.get(int(t), 0.42) if not np.isnan(t) else 0.42
                       for t in type_vals])
else:
    print("  ■■■■■■■■■■■ → c ■ Σ■ ■■■■■■■■■")
    # c ∝ Σ■^δ ■■■■■■■■■■■■ → ■■■■■■■ c=0.42 ■■
    c_vals = np.full(len(df), 0.42)

F_c = 2 * (1 - np.sqrt(c_vals))

# ■■■■■■: G · Σ■ · F(c) ■η·h_R ■■■■■■■■■■■■■■■■
theory_raw = G_Sigma0 * F_c

# η ■■■■■■■log ■■■■■■■
log_theory = np.log10(theory_raw)
# g_c = η_eff · G · Σ■ · F(c) → log(g_c) = log(η_eff) + log(G·Σ■·F(c))
# → intercept = log(η_eff) ■ slope=1 ■■■

# ■■■■■■■■■
slope_fc, intercept_fc, r_fc, p_fc, se_fc = stats.linregress(log_theory, log_gc)
print(f"\n  F(c) ■■■■■:")
print(f"    log(g_c) = {slope_fc:.3f} * log(G·Σ■·F(c)) + {intercept_fc:.3f}")
print(f"    r = {r_fc:.3f}, R² = {r_fc**2:.3f}")

# slope=1 ■■■■■■■■
eta_fit = np.median(gc_si / theory_raw)
log_gc_pred_fixed = np.log10(eta_fit * theory_raw)
residuals_fixed = log_gc - log_gc_pred_fixed
r_fixed = 1 - np.sum(residuals_fixed**2) / np.sum((log_gc - np.mean(log_gc))**2)
print(f"\n  slope=1 ■■:")
print(f"    η_eff = {eta_fit:.4e}")
print(f"    R²(slope=1) = {r_fixed:.3f}")
print(f"    ■■ std = {np.std(residuals_fixed):.3f} dex")

# F(c) ■■■■■■■
slope_nc, intercept_nc, r_nc, p_nc, _ = stats.linregress(log_GSigma0, log_gc)
print(f"\n  ■■: F(c) ■■■■■  → r={r_nc:.3f}, R²={r_nc**2:.3f}")
print(f"        F(c) ■■■■■ → r={r_fc:.3f}, R²={r_fc**2:.3f}")
print(f"        ■■: ΔR² = {r_fc**2 - r_nc**2:.4f}")

# ======================================================================
# 6. ■■(3): c(Σ■) ■■■■■■■■■
# ======================================================================
print("\n" + "=" * 70)
print("Step 6: c(Σ■) ■■■■■■■")
print("=" * 70)

# g_c = η · G · Σ■ · 2(1-√c) ■■ c ■■■
# c = (1 - g_c/(2η·G·Σ■))²
# ■■■ η ■■■ → ■■ η ■ g_c/(G·Σ■) ■■■■■■■■

ratio = gc_si / G_Sigma0
print(f"  g_c / (G·Σ■) ■■■:")
print(f"    ■■■ = {np.median(ratio):.4f}")
print(f"    ■■   = {np.mean(ratio):.4f}")
print(f"    std    = {np.std(ratio):.4f}")
print(f"    min    = {np.min(ratio):.4f}")
print(f"    max    = {np.max(ratio):.4f}")

# c_eff ■■■: g_c = η_scale · G · Σ■ · F(c_eff)
# η_scale · F(c_eff) = g_c / (G·Σ■) = ratio
# ■■■ ratio ■■■■■■■ c→0 (F→2) ■■■■■■■■■
# → η_scale ≈ max(ratio) / 2
eta_scale = np.percentile(ratio, 95) / 2  # 95th ■■■■■
print(f"\n  η_scale = {eta_scale:.4f} (95th percentile / 2)")

# c_eff ■■■
F_eff = ratio / eta_scale
# F = 2(1-√c) → √c = 1 - F/2 → c = (1 - F/2)²
# F/2 &gt; 1 ■■■■■ c &lt; 0 → ■■■■■ → ■■■■■
F_eff_clipped = np.clip(F_eff, 0.01, 1.99)
```

```
c_eff = (1 - F_eff_clipped / 2) ** 2

print(f"  c_eff ■■:")
print(f"    ■■■ = {np.median(c_eff):.3f}")
print(f"    ■■   = {np.mean(c_eff):.3f}")
print(f"    std    = {np.std(c_eff):.3f}")
print(f"    [5th, 95th] = [{np.percentile(c_eff,5):.3f}, {np.percentile(c_eff,95):.3f}]")

# c_eff vs Σ■ ■■■
log_Sigma0 = np.log10(Sigma0_Msun_pc2)
rho_cS, p_cS = stats.spearmanr(log_Sigma0, c_eff)
slope_cS, intercept_cS, r_cS, _, _ = stats.linregress(log_Sigma0, c_eff)
print(f"\n  c_eff vs log(Σ■):")
print(f"    Pearson r = {r_cS:.3f}")
print(f"    Spearman ρ = {rho_cS:.3f}, p = {p_cS:.2e}")
print(f"    c_eff = {slope_cS:.4f} * log(Σ■) + {intercept_cS:.4f}")

# ■■■■■■■■ c_eff ■■■
if type_col_final and type_col_final in df.columns:
    print(f"\n  ■■■■■■■ c_eff ■■■:")
    type_vals_clean = df[type_col_final].values
    for t in sorted(set(type_vals_clean[~np.isnan(type_vals_clean)])):
        mask_t = type_vals_clean == t
        if np.sum(mask_t) &gt;= 3:
            med_c = np.median(c_eff[mask_t])
            med_S = np.median(Sigma0_Msun_pc2[mask_t])
            n_t = np.sum(mask_t)
            expected_c = c_type_map.get(int(t), None)
            exp_str = f" (expected: {expected_c:.2f})" if expected_c else ""
            print(f"    T={int(t):2d}: c_eff={med_c:.3f}, Σ■={med_S:.0f} Msun/pc², N={n_t}{exp_str}")

# ======================================================================
# 7. ■■(4): ■■■■■■■■
# ======================================================================
print("\n" + "=" * 70)
print("Step 7: ■■■■■■■■")
print("=" * 70)

# log(g_c) = a·log(v_flat) + b·log(h_R) + const
log_vf = np.log10(vflat_kms)
log_hR = np.log10(hR_kpc)

# 2■■■■
X = np.column_stack([log_vf, log_hR, np.ones(len(log_vf))])
beta, residuals, rank, sv = np.linalg.lstsq(X, log_gc, rcond=None)
y_pred = X @ beta
ss_res = np.sum((log_gc - y_pred)**2)
ss_tot = np.sum((log_gc - np.mean(log_gc))**2)
R2_2var = 1 - ss_res / ss_tot

print(f"  log(g_c) = {beta[0]:.3f}·log(v_flat) + {beta[1]:.3f}·log(h_R) + {beta[2]:.3f}")
print(f"  R² = {R2_2var:.3f}")
print(f"  → g_c ∝ v_flat^{beta[0]:.2f} · h_R^{beta[1]:.2f}")
print(f"  ■■■■ (G·Σ■ = v²/h_R): v_flat^2.0 · h_R^{-1.0}")
print(f"  v_flat ■■■■: {beta[0]-2:.3f}")
print(f"  h_R ■■■■:    {beta[1]-(-1):.3f}")

# F(c) ■■■
if type_col_final and type_col_final in df.columns:
    log_gc_corrected = log_gc - np.log10(F_c)
    X2 = np.column_stack([log_vf, log_hR, np.ones(len(log_vf))])
    beta2, _, _, _ = np.linalg.lstsq(X2, log_gc_corrected, rcond=None)
    y_pred2 = X2 @ beta2
    R2_corr = 1 - np.sum((log_gc_corrected - y_pred2)**2) / np.sum((log_gc_corrected - np.mean(log_gc_corrected))**2)

    print(f"\n  F(c) ■■■:")
    print(f"  log(g_c/F(c)) = {beta2[0]:.3f}·log(v_flat) + {beta2[1]:.3f}·log(h_R) + {beta2[2]:.3f}")
    print(f"  R² = {R2_corr:.3f}")
    print(f"  → g_c/F(c) ∝ v_flat^{beta2[0]:.2f} · h_R^{beta2[1]:.2f}")
    print(f"  ■■■■■■■■■■: v_flat ■■■■ {beta2[0]-2:.3f}, h_R ■■■■ {beta2[1]-(-1):.3f}")

# ======================================================================
# 8. ■■■■■■
# ======================================================================
print("\n" + "=" * 70)
print("Step 8: ■■■■■■")
print("=" * 70)

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle('g_c ∝ G·Σ■·F(c) Theory Verification', fontsize=14, fontweight='bold')

# --- (a) g_c vs G·Σ■ ---
ax = axes[0, 0]
ax.scatter(log_GSigma0, log_gc, s=15, alpha=0.5, c='steelblue')
x_range = np.linspace(log_GSigma0.min(), log_GSigma0.max(), 100)
ax.plot(x_range, slope_nc * x_range + intercept_nc, 'r-', lw=2,
        label=f'fit: slope={slope_nc:.2f}, r={r_nc:.3f}')
```

```
ax.plot(x_range, x_range + np.log10(np.median(ratio)), 'k--', lw=1, alpha=0.5,
        label='slope=1 reference')
ax.set_xlabel('log(G·Σ■) [m/s²]')
ax.set_ylabel('log(g_c) [m/s²]')
ax.set_title('(a) g_c vs G·Σ■')
ax.legend(fontsize=9)

# --- (b) g_c/g_c_pred vs Σ■ (residuals) ---
ax = axes[0, 1]
resid = log_gc - (slope_nc * log_GSigma0 + intercept_nc)
ax.scatter(log_Sigma0, resid, s=15, alpha=0.5, c='steelblue')
ax.axhline(0, color='k', ls='--', alpha=0.3)
ax.set_xlabel('log(Σ■) [Msun/pc²]')
ax.set_ylabel('Residual [dex]')
ax.set_title(f'(b) Residuals vs Σ■ (std={np.std(resid):.3f})')

# --- (c) c_eff vs Σ■ ---
ax = axes[0, 2]
if type_col_final and type_col_final in df.columns:
    scatter = ax.scatter(log_Sigma0, c_eff, s=15, alpha=0.5,
                         c=df[type_col_final].values, cmap='viridis')
    plt.colorbar(scatter, ax=ax, label='Hubble T type')
else:
    ax.scatter(log_Sigma0, c_eff, s=15, alpha=0.5, c='steelblue')
ax.set_xlabel('log(Σ■) [Msun/pc²]')
ax.set_ylabel('c_eff (inferred)')
ax.set_title(f'(c) c(Σ■) relation (ρ={rho_cS:.3f})')
# c ■■■■■■■■■■■■
if type_col_final and type_col_final in df.columns:
    for t, c_exp in sorted(c_type_map.items()):
        mask_t = df[type_col_final].values == t
        if np.sum(mask_t) &gt;= 3:
            ax.axhline(c_exp, color='gray', ls=':', alpha=0.3)

# --- (d) F(c) ■■■■ vs ■■ ---
ax = axes[1, 0]
ax.scatter(log_theory, log_gc, s=15, alpha=0.5, c='coral', label=f'F(c) corr: r={r_fc:.3f}')
ax.scatter(log_GSigma0, log_gc, s=15, alpha=0.3, c='steelblue', label=f'no corr: r={r_nc:.3f}')
ax.plot(x_range, x_range + np.log10(eta_fit), 'r-', lw=2, alpha=0.5)
ax.set_xlabel('log(G·Σ■ or G·Σ■·F(c))')
ax.set_ylabel('log(g_c)')
ax.set_title('(d) F(c) correction comparison')
ax.legend(fontsize=9)

# --- (e) ■■■■■■■ g_c/g_c_pred ---
ax = axes[1, 1]
if type_col_final and type_col_final in df.columns:
    type_vals_clean = df[type_col_final].values
    types_unique = sorted(set(type_vals_clean[~np.isnan(type_vals_clean)]))
    type_medians = []
    type_labels = []
    for t in types_unique:
        mask_t = type_vals_clean == t
        if np.sum(mask_t) &gt;= 3:
            type_medians.append(np.median(resid[mask_t]))
            type_labels.append(f'T={int(t)}')
    ax.bar(range(len(type_medians)), type_medians, color='steelblue', alpha=0.7)
    ax.set_xticks(range(len(type_labels)))
    ax.set_xticklabels(type_labels, rotation=45)
    ax.axhline(0, color='k', ls='--', alpha=0.3)
    ax.set_ylabel('Median residual [dex]')
    ax.set_title('(e) Residual by galaxy type')
else:
    ax.text(0.5, 0.5, 'No type info', transform=ax.transAxes, ha='center')
    ax.set_title('(e) N/A')

# --- (f) ■■■ vs ■■■■■■■■ ---
ax = axes[1, 2]
# ■■■■■■■: log(gc/a0) = -2.175 + 2.015*log(vflat) - 1.294*log(hR) + ...
gc_empirical_2var = 10**(-2.175 + 2.015*log_vf - 1.294*log_hR) * a0  # ■■■
log_gc_emp = np.log10(gc_empirical_2var)
r_emp = np.corrcoef(log_gc_emp, log_gc)[0,1]
r_theory = r_nc  # G·Σ■ = v²/h_R

ax.scatter(log_gc_emp, log_gc, s=15, alpha=0.5, c='coral',
           label=f'Empirical (2var): r={r_emp:.3f}')
# G·Σ■ ■■
gc_theory_1var = 10**(slope_nc * log_GSigma0 + intercept_nc)
ax.scatter(np.log10(gc_theory_1var), log_gc, s=15, alpha=0.3, c='steelblue',
           label=f'G·Σ■ theory: r={r_theory:.3f}')
diag = np.linspace(log_gc.min(), log_gc.max(), 100)
ax.plot(diag, diag, 'k--', alpha=0.3)
ax.set_xlabel('log(g_c predicted)')
ax.set_ylabel('log(g_c observed)')
ax.set_title('(f) Theory vs Empirical prediction')
ax.legend(fontsize=9)
plt.tight_layout()
```

```
plt.savefig('gc_sigma0_theory_verification.png', dpi=150, bbox_inches='tight')
print("  → gc_sigma0_theory_verification.png ■■■■")

# ======================================================================
# 9. ■■■■
# ======================================================================
print("\n" + "=" * 70)
print("SUMMARY: g_c ∝ G·Σ■·F(c) ■■■■■■■■■■")
print("=" * 70)

print(f"""
■■(1): g_c vs G·Σ■ ■■■■
  Pearson r  = {r_nc:.3f}
  Spearman ρ = {rho_sp:.3f}
  ■■■■   = {slope_nc:.3f} (■■■■: 1.0)
  → {"✓ ■■■■" if abs(r_nc) &gt; 0.6 else "■ ■■■■■■" if abs(r_nc) &gt; 0.4 else "✗ ■■■■"}

■■(2): F(c) ■■■■
  ■■■■ R² = {r_nc**2:.3f}
  ■■■■ R² = {r_fc**2:.3f}
  ■■ ΔR²   = {r_fc**2 - r_nc**2:.4f}
  → {"✓ F(c) ■■■■■" if r_fc**2 &gt; r_nc**2 else "✗ F(c) ■■■■■■■"}

■■(3): c(Σ■) ■■
  Spearman ρ = {rho_cS:.3f}, p = {p_cS:.2e}
  → {"✓ ■■■■■" if p_cS &lt; 0.05 else "✗ ■■■■■■■"}

■■(4): ■■■■■■
  ■■■: g_c ∝ v_flat^{beta[0]:.2f} · h_R^{beta[1]:.2f}
  ■■:   g_c ∝ v_flat^2.0 · h_R^-1.0
  ■■:   Δα_v = {beta[0]-2:.3f}, Δα_h = {beta[1]-(-1):.3f}

■■■■ g_c ∝ G·Σ■ ■■■■■■■■■■■■■■
■■■■■■ {"■■■■" if abs(r_nc) &gt; 0.6 else "■■■■■■"}■■■■■■■
F(c) ■■■■■■■■■■■■ c ■■■■■■■■■■■■
■■■■ U(ε;c) ■■■■■■■■■■■■■■■
""")

print("■■")
```

gc_geometric_mean_test.py

解析目的	幾何平均法則 g_c = eta*sqrt(a0*G*Sigma0) の検証。alpha=0.5 の統計検定。MOND との dAIC 比較。
結果	A10, A11 確立。alpha=0.545+/-0.041, p(0.5)=0.27, dAIC=-130。

```
"""
■■■■■■■■■■■■
■■: g_c = a0^(1-alpha) * (G*Sigma0)^alpha
■■: uv run --with scipy --with matplotlib --with numpy --with pandas python gc_geometric_mean_test.py
"""
import numpy as np, pandas as pd
from scipy import stats
from pathlib import Path
import matplotlib; matplotlib.use('Agg')
import matplotlib.pyplot as plt

work_dir = Path(".")
a0 = 1.2e-10; G = 6.674e-11; kpc_m = 3.086e19; kms_ms = 1e3

def find_col(df, cands):
    if df is None: return None
    for c in cands:
        m = [col for col in df.columns if c.lower() in col.lower()]
        if m: return m[0]
    return None

print("="*70)
print("■■■■■■■: g_c = a0^(1-alpha) * (G*Sigma0)^alpha ■■■■■■")
print("="*70)

# --- ■■■■■■■■ ---
df_gc = pd.read_csv("TA3_gc_independent.csv") if Path("TA3_gc_independent.csv").exists() else None
df_pred = pd.read_csv("gc_predictive_model.csv") if Path("gc_predictive_model.csv").exists() else None
df_sparc = pd.read_csv("sparc_results.csv") if Path("sparc_results.csv").exists() else None

df = None
if df_pred is not None:
    cm = {}
    for k,cs in [('gc',['gc','g_c','gc_obs','gc_rar']),('vflat',['v_flat','vflat']),('hR',['h_R','hR','Reff','R_d'])]:
        c = find_col(df_pred, cs)
        if c: cm[k] = c
    if len(cm)==3:
        df = df_pred.copy()
        df['gc_val']=df[cm['gc']]; df['vflat_val']=df[cm['vflat']]; df['hR_val']=df[cm['hR']]
        print(f"  ■■■■■■■: gc_predictive_model.csv (N={len(df)})")
if df is None and df_gc is not None and df_sparc is not None:
    gc_c=find_col(df_gc,['gc','g_c']); gg=find_col(df_gc,['galaxy','name'])
    vf_c=find_col(df_sparc,['v_flat','vflat']); hr_c=find_col(df_sparc,['h_R','hR','R_d'])
    gs=find_col(df_sparc,['galaxy','name'])
    df=pd.merge(df_gc,df_sparc,left_on=gg,right_on=gs,how='inner')
    df['gc_val']=df[gc_c]; df['vflat_val']=df[vf_c]; df['hR_val']=df[hr_c]
    print(f"  ■■■■■■■: TA3+SPARC■■ (N={len(df)})")
if df is None: print("!!! ■■■■■■■■■■"); import sys; sys.exit(1)

type_col = find_col(df, ['type','Type','T','hubble','Hubble','morph'])
mask = df['gc_val'].notna() &amp;amp; df['vflat_val'].notna() &amp;amp; df['hR_val'].notna()
mask &amp;amp;= (df['gc_val']&amp;gt;0) &amp;amp; (df['vflat_val']&amp;gt;0) &amp;amp; (df['hR_val']&amp;gt;0)
df = df[mask].copy().reset_index(drop=True)
print(f"  ■■■■■■■: N={len(df)}")

gc_vals=df['gc_val'].values; gc_med=np.median(gc_vals)
if gc_med&amp;lt;1e-5: gc_si=gc_vals; gc_a0=gc_vals/a0; print(f"  g_c■■: SI (■■■={gc_med:.2e})")
else: gc_a0=gc_vals; gc_si=gc_vals*a0; print(f"  g_c■■: a0 (■■■={gc_med:.3f})")

vf_ms=df['vflat_val'].values*kms_ms; hR_m=df['hR_val'].values*kpc_m
G_S0 = vf_ms**2/hR_m
log_gc=np.log10(gc_si); log_GS=np.log10(G_S0); log_a0v=np.log10(a0)
log_vf=np.log10(df['vflat_val'].values); log_hR=np.log10(df['hR_val'].values)
n=len(df)

# ======================================================================
print("\n"+"="*70)
print("■■■1■alpha ■■■■■■■■■■")
print("="*70)
x = log_GS - log_a0v
sl,ic,r_x,p_x,se_x = stats.linregress(x, log_gc)
alpha_opt=sl; eta_opt=10**(ic-log_a0v)
resid_B = log_gc-(sl*x+ic)
s2=np.sum(resid_B**2)/(n-2); se_a=np.sqrt(s2/np.sum((x-np.mean(x))**2))
tc=stats.t.ppf(0.975,n-2); ci=(alpha_opt-tc*se_a, alpha_opt+tc*se_a)

print(f"  ■■ alpha = {alpha_opt:.4f} +/- {se_a:.4f}")
print(f"  95%■■■■: [{ci[0]:.4f}, {ci[1]:.4f}]")
print(f"  eta = {eta_opt:.4f}")
print(f"  ■■■■ r = {r_x:.4f},  ■■■■ R2 = {r_x**2:.4f}")
```

```
print(f"  ■■■■■■ = {np.std(resid_B):.4f} dex")

for label,a_test in [("0.0 (MOND: g_c=■■)",0.0),("0.5 (■■■■■■■)",0.5),("1.0 (■■G*Sigma0■■)",1.0)]:
    t_s=(alpha_opt-a_test)/se_a; p_v=2*stats.t.sf(abs(t_s),n-2)
    judge="■■■■■■■" if p_v&gt;0.05 else "■■ (p&lt;0.05)"
    print(f"\n  alpha = {label} ■■■:")
    print(f"    t = {t_s:.3f}, p = {p_v:.4f} → {judge}")

t05=(alpha_opt-0.5)/se_a; p05=2*stats.t.sf(abs(t05),n-2)
t10=(alpha_opt-1.0)/se_a; p10=2*stats.t.sf(abs(t10),n-2)
t00=(alpha_opt-0.0)/se_a; p00=2*stats.t.sf(abs(t00),n-2)

# ======================================================================
print("\n"+"="*70)
print("■■■2■■■■■■ (AIC)")
print("="*70)

pred_A=np.full(n,log_a0v); ss_A=np.sum((log_gc-pred_A)**2); aic_A=n*np.log(ss_A/n)
pred_Bv=sl*x+ic; ss_B=np.sum(resid_B**2); aic_B=n*np.log(ss_B/n)+2*2
pC_raw=0.5*log_GS+0.5*log_a0v; eC=np.median(log_gc-pC_raw); pred_C=pC_raw+eC
ss_C=np.sum((log_gc-pred_C)**2); aic_C=n*np.log(ss_C/n)+2*1
sD,iD,rD,_,_=stats.linregress(log_GS,log_gc); pred_D=sD*log_GS+iD
ss_D=np.sum((log_gc-pred_D)**2); aic_D=n*np.log(ss_D/n)+2*2
pE_raw=-2.175+2.015*log_vf-1.294*log_hR; pred_E=pE_raw+log_a0v
ss_E=np.sum((log_gc-pred_E)**2); aic_E=n*np.log(ss_E/n)+2*3

models=[
    ("A: MOND (g_c=a0, 0■■■■■■)",0,ss_A,aic_A),
    ("B: ■■■■■ (alpha■■, 2■■■■■■)",2,ss_B,aic_B),
    ("C: ■■■■■■■ (alpha=0.5■■, 1■■■■■■)",1,ss_C,aic_C),
    ("D: ■■■■■ (g_c=(G*S0)^s, 2■■■■■■)",2,ss_D,aic_D),
    ("E: 2■■■■■■ (vflat,hR, 3■■■■■■)",3,ss_E,aic_E),
]
print(f"  {'■■■':&lt;50} {'k':&gt;3} {'RSS':&gt;9} {'AIC':&gt;9} {'dAIC':&gt;7}")
print(f"  {'-'*82}")
for nm,k,ss,aic in models:
    print(f"  {nm:&lt;50} {k:&gt;3} {ss:&gt;9.2f} {aic:&gt;9.1f} {aic-aic_A:&gt;+7.1f}")
print(f"\n  ■ dAIC &lt; 0 → MOND ■■■■■■■■")

# ======================================================================
print("\n"+"="*70)
print("■■■3■■■■■■■■ alpha■■■■■■■■■")
print("="*70)
tg={'Im/BCD (T&gt;=9)':[9,10,11],'Sd/Sdm (T=7,8)':[7,8],'Sc/Scd (T=5,6)':[5,6],
    'Sb/Sbc (T=3,4)':[3,4],'Sa/S0 (T&lt;=2)':[0,1,2]}
ga=[]; gl=[]; gn=[]
if type_col and type_col in df.columns:
    tv=df[type_col].values
    print(f"  {'■■■■':&lt;22} {'N':&gt;5} {'alpha':&gt;7} {'+/-se':&gt;7} {'r':&gt;7}")
    print(f"  {'-'*52}")
    for gn_,tvs in tg.items():
        mg=np.isin(tv,tvs); ng_=int(np.sum(mg))
        if ng_&lt;5: print(f"  {gn_:&lt;22} {ng_:&gt;5}  (■■)"); continue
        s_,i_,r_,_,se_=stats.linregress(x[mg],log_gc[mg])
        print(f"  {gn_:&lt;22} {ng_:&gt;5} {s_:&gt;7.3f} {se_:&gt;7.3f} {r_:&gt;7.3f}")
        ga.append(s_); gl.append(gn_); gn.append(ng_)
    if len(ga)&gt;2:
        av=np.std(ga)
        print(f"\n  alpha ■■■■■■■■■■ = {av:.3f}")
        print(f"  → {'alpha ■■■■■■■■■■■■■■■' if av&lt;0.1 else 'alpha ■■■■■■■■■■■■■■■'}")
else:
    print("  ■■■■■■■■■■ → ■■■■■")

# ======================================================================
print("\n"+"="*70)
print("■■■4■■■■■■■")
print("="*70)
for lab,vals in [("log(v_flat)",log_vf),("log(h_R)",log_hR),("log(G*S0)",log_GS)]:
    rho,p=stats.spearmanr(vals,resid_B)
    sig=" ★■■" if p&lt;0.05 else ""
    print(f"  ■■ vs {lab:&lt;12}: rho={rho:+.3f}, p={p:.2e}{sig}")

if type_col and type_col in df.columns:
    print(f"\n  ■■■■■■■■■■:")
    for gn_,tvs in tg.items():
        mg=np.isin(tv,tvs)
        if np.sum(mg)&gt;=5:
            print(f"    {gn_:&lt;22}: median={np.median(resid_B[mg]):+.3f}, std={np.std(resid_B[mg]):.3f}")

# ======================================================================
print("\n"+"="*70)
print("■■■5■■■■■■■■ g_c = eta*sqrt(a0*G*S0) ■■■")
print("="*70)
gc_gm=np.sqrt(a0*G_S0); eta_gm=np.median(gc_si/gc_gm)
lgm=np.log10(eta_gm*gc_gm); rgm=log_gc-lgm; r_gm=np.corrcoef(lgm,log_gc)[0,1]
r_mond=log_gc-log_a0v
print(f"  alpha=0.5 ■■■■■■:")
```

```
print(f"    eta = {eta_gm:.4f}")
print(f"    ■■ r = {r_gm:.4f}")
print(f"    ■■std = {np.std(rgm):.4f} dex")
print(f"    ■■■■■ = {np.median(np.abs(rgm)):.4f} dex")
print(f"\n  ■■■■■■■■■:")
print(f"    MOND (g_c=a0):           {np.std(r_mond):.4f} dex")
print(f"    ■■■■■■ (alpha=0.5): {np.std(rgm):.4f} dex")
print(f"    ■■■■■■ (alpha={alpha_opt:.3f}): {np.std(resid_B):.4f} dex")
i05=(1-np.std(rgm)/np.std(r_mond))*100; ifr=(1-np.std(resid_B)/np.std(r_mond))*100
print(f"\n  MOND■■■■■■:")
print(f"    alpha=0.5■■: {i05:.1f}%")
print(f"    alpha■■:     {ifr:.1f}%")
print(f"    ■■■■:      {ifr-i05:.1f}%")

# ======================================================================
print("\n"+"="*70)
print("■■■■■■■...")
print("="*70)
fig,axes=plt.subplots(2,3,figsize=(18,12))
fig.suptitle(r'$g_c = \eta\,a_0^{1-\alpha}\,(G\Sigma_0)^{\alpha}$  Verification',fontsize=14,fontweight='bold')

# (a)
ax=axes[0,0]; la0=np.log10(gc_a0); lGa=np.log10(G_S0/a0)
ax.scatter(lGa,la0,s=15,alpha=0.5,c='steelblue')
xr=np.linspace(lGa.min(),lGa.max(),100)
ax.plot(xr,alpha_opt*xr+np.log10(eta_opt),'r-',lw=2,label=f'$\\alpha$={alpha_opt:.3f}')
ax.plot(xr,0.5*xr+eC,'g--',lw=2,alpha=0.7,label='$\\alpha$=0.5')
ax.plot(xr,xr,'k:',lw=1,alpha=0.3,label='$\\alpha$=1.0')
ax.axhline(0,color='gray',ls='--',lw=1,alpha=0.3,label='MOND')
ax.set_xlabel('log($G\\Sigma_0/a_0$)'); ax.set_ylabel('log($g_c/a_0$)')
ax.set_title(f'(a) $\\alpha$={alpha_opt:.3f}$\\pm${se_a:.3f}'); ax.legend(fontsize=8)

# (b)
ax=axes[0,1]; ar=np.linspace(0,1,200)
c2=np.array([np.sum((log_gc-(a_*x+(np.mean(log_gc)-a_*np.mean(x))))**2) for a_ in ar])
ax.plot(ar,c2-c2.min(),'b-',lw=2)
ax.axhline(stats.chi2.ppf(0.95,1),color='r',ls='--',alpha=0.5,label='95% CL')
ax.axvline(alpha_opt,color='k',ls='-',alpha=0.3,label=f'best={alpha_opt:.3f}')
ax.axvline(0.5,color='g',ls='--',alpha=0.5,label='0.5')
ax.axvline(1.0,color='gray',ls=':',alpha=0.5,label='1.0')
ax.set_xlabel('$\\alpha$'); ax.set_ylabel('$\\Delta\\chi^2$')
ax.set_title('(b) $\\alpha$ confidence'); ax.set_xlim(0,1); ax.legend(fontsize=8)

# (c)
ax=axes[0,2]
if len(ga)&gt;0:
    cols=plt.cm.viridis(np.linspace(0.2,0.8,len(ga)))
    bars=ax.bar(range(len(gl)),ga,color=cols,alpha=0.7)
    ax.set_xticks(range(len(gl))); ax.set_xticklabels([g.split('(')[0].strip() for g in gl],rotation=30,fontsize=8)
    ax.axhline(alpha_opt,color='r',ls='--',lw=2,label=f'global={alpha_opt:.3f}')
    ax.axhline(0.5,color='g',ls=':',lw=1.5,label='0.5')
    for i,(b,nn) in enumerate(zip(bars,gn)): ax.text(b.get_x()+b.get_width()/2,b.get_height()+0.01,f'N={nn}',ha='center',fontsize=7)
    ax.set_ylabel('$\\alpha$'); ax.set_title('(c) Type dependence'); ax.legend(fontsize=8)
else: ax.text(0.5,0.5,'No type info',transform=ax.transAxes,ha='center')

# (d)
ax=axes[1,0]
ax.scatter(pred_A,log_gc,s=10,alpha=0.2,c='gray',label='MOND')
ax.scatter(pred_C,log_gc,s=10,alpha=0.4,c='green',label='$\\alpha$=0.5')
ax.scatter(pred_Bv,log_gc,s=12,alpha=0.5,c='coral',label=f'$\\alpha$={alpha_opt:.3f}')
dg=np.linspace(log_gc.min(),log_gc.max(),100); ax.plot(dg,dg,'k--',alpha=0.3)
ax.set_xlabel('predicted'); ax.set_ylabel('observed'); ax.set_title('(d) Obs vs Pred'); ax.legend(fontsize=8)

# (e)
ax=axes[1,1]; bins=np.linspace(-1,1,40)
ax.hist(r_mond,bins=bins,alpha=0.3,color='gray',label=f'MOND ({np.std(r_mond):.3f})')
ax.hist(rgm,bins=bins,alpha=0.3,color='green',label=f'a=0.5 ({np.std(rgm):.3f})')
ax.hist(resid_B,bins=bins,alpha=0.5,color='coral',label=f'a={alpha_opt:.3f} ({np.std(resid_B):.3f})')
ax.set_xlabel('Residual [dex]'); ax.set_ylabel('Count'); ax.set_title('(e) Residuals'); ax.legend(fontsize=8)

# (f)
ax=axes[1,2]; rv,pv=stats.spearmanr(log_vf,resid_B); rh,ph=stats.spearmanr(log_hR,resid_B)
ax.scatter(log_vf,resid_B,s=15,alpha=0.5,c='steelblue',label=f'v_flat ($\\rho$={rv:.3f})')
ax.scatter(log_hR,resid_B,s=15,alpha=0.3,c='coral',label=f'h_R ($\\rho$={rh:.3f})')
ax.axhline(0,color='k',ls='--',alpha=0.3)
ax.set_xlabel('log(v_flat) or log(h_R)'); ax.set_ylabel('Residual [dex]')
ax.set_title('(f) Residual structure'); ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('gc_geometric_mean_verification.png',dpi=150,bbox_inches='tight')
print("  → gc_geometric_mean_verification.png ■■■■")

# ======================================================================
print("\n"+"="*70)
print("■■■■■■■")
print("="*70)
print(f"""
```

```
{'='*60}
  ■■: g_c = eta * a0^(1-alpha) * (G*Sigma0)^alpha
{'='*60}

■ alpha ■■■:
  ■■■ = {alpha_opt:.4f} +/- {se_a:.4f}
  95%■■■■: [{ci[0]:.4f}, {ci[1]:.4f}]

■ ■■■■■■■:
  alpha=0 (MOND):     p={p00:.2e} → {'■■' if p00&lt;0.05 else '■■■■■■■'}
  alpha=0.5 (■■■■■): p={p05:.4f} → {'■■■■■■■' if p05&gt;0.05 else '■■'}
  alpha=1.0 (G*S0■■): p={p10:.2e} → {'■■' if p10&lt;0.05 else '■■■■■■■'}

■ MOND■■■■■■■:
  ■■std: {np.std(r_mond):.3f} → {np.std(resid_B):.3f} dex ({ifr:.1f}%■■)

■ ■■:""")

if p00&lt;0.05: print("  [■■] MOND (g_c=■■) ■■■■g_c■■■■■■■■■■■■■■■")
if p10&lt;0.05: print("  [■■] ■■■g_c∝G*Sigma0 (alpha=1) ■■■■a0■■■■■■■■")
if p05&gt;0.05:
    print("  [■■] ■■■■■■■ g_c=eta*sqrt(a0*G*Sigma0) ■■■ (alpha=0.5 ■■■■■■■)■")
    print("         → ■■■■■■■■■■■■■■■a0■■■■■■■G*Sigma0■")
    print("           ■■■■■■■■■■■■■■■■■■■")
else:
    d="■■■■■■■■■■■■■■■" if alpha_opt&gt;0.5 else "■■■■■■■■■■■■■■"
    print(f"  [■■] alpha={alpha_opt:.3f}: {d}")

if len(ga)&gt;2:
    av=np.std(ga)
    if av&lt;0.1: print(f"  [■■] alpha■■■■■■■■■■■■■■ (■■■■■■■■{av:.3f})■■■■■■■■■■")
    else: print(f"  [■■] alpha■■■■■■■■■■■■■ (■■{av:.3f})■c(type)■■■■■■■■■■■")

print(f"""
■ U(epsilon;c) ■■■■:
  ■■■■■■■■■■ U_eff = U_membrane + U_coupling(Sigma0) ■
  ■■■■■■■ alpha ■■■■■■■
  alpha=0.5 → ■■■■■■■■■■■■■■■■■■■■■■■■■■■■
""")
print("■■")
```

strain_energy_theory_test.py

解析目的	alpha(c) 歪みエネルギー差理論の検証。alpha が銀河タイプ（弾性係数 c）に依存するかの検定。
結果	棄却。T_m が大極限に収束し alpha の c 依存性が消える。

```
"""
■■■■■■■■■■■■■■■■
==============================
■■: alpha(c) = f_elastic(c) = 1/(1 + exp(DeltaU(c)/T_m))
      DeltaU(c) = U(epsilon_c; c) - U(0; c)  ■■■-■■■■■■■■■■

■■:
  (1) DeltaU(c) ■ c ■■■■■■■■■
  (2) alpha(c) ■■■■■■■■■■■T_m ■■■■■■■■■■■
  (3) v_flat ■■■■ alpha■■■■■■■■
  (4) c(galaxy) → alpha → g_c ■■■■■■■■■

■■: uv run --with scipy --with matplotlib --with numpy --with pandas python strain_energy_theory_test.py
"""

import numpy as np
import pandas as pd
from scipy import stats, optimize
from pathlib import Path
import matplotlib; matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ======================================================================
# 0. U(epsilon; c) ■■■
# ======================================================================
print("="*70)
print("■■■■■■■■■■■■: alpha(c) = f_elastic(c)")
print("  ■■■■■/■■■■■■■■■■■■■■ alpha ■■■■■■")
print("="*70)

def U(eps, c):
    """■■■■■■■■ U(epsilon; c)"""
    return -eps - eps**2/2 - c*np.log(1 - eps)

def U_prime(eps, c):
    """U ■■■■■■"""
    return -1 - eps + c/(1 - eps)

def U_double_prime(eps, c):
    """U ■■■■■■"""
    return -1 + c/(1 - eps)**2

def epsilon_c(c):
    """■■■■■■■→■■■■■■■"""
    return 1 - np.sqrt(c)

def epsilon_eq(c):
    """■■■■ U'=0 ■■■■■"""
    # -1 - eps + c/(1-eps) = 0 → eps^2 + (2-c)*eps + (1-c) = 0 ... ■■■■■
    # (1-eps)(-1-eps) + c = 0 → -1 + eps^2 + c = 0 → eps = sqrt(1-c) ... c&lt;1 ■■■■
    # ■■■■■■: -(1-eps)(1+eps) + c = 0 → eps^2 = 1-c → eps = sqrt(1-c)
    # ■■■: U'(sqrt(1-c), c) = -1 - sqrt(1-c) + c/(1-sqrt(1-c))
    # ■■■■■■■■ 0 ■■■■■■■■■■■■■:
    # U' = -1 - eps + c/(1-eps) = 0
    # → -(1-eps)(1+eps) + c = 0  ... ■■■■■■■
    # → (1-eps)(-1-eps) + c = 0
    # → -(1-eps)(1+eps) + c = 0
    # → -(1-eps^2) + c = 0
    # → eps^2 = 1 - c
    # → eps = sqrt(1-c)
    # ■■: U'(sqrt(1-c), c) = -1 - sqrt(1-c) + c/(1-sqrt(1-c))
    # ■■ = 1-sqrt(1-c), c/(1-sqrt(1-c))
    # ■■■■: c(1+sqrt(1-c))/((1-sqrt(1-c))(1+sqrt(1-c))) = c(1+sqrt(1-c))/(1-(1-c)) = c(1+sqrt(1-c))/c = 1+sqrt(1-c)
    # U' = -1 - sqrt(1-c) + 1 + sqrt(1-c) = 0 ✓
    if c &gt;= 1:
        return np.nan
    return np.sqrt(1 - c)

def DeltaU(c):
    """■■-■■■■■■■■■■: U(epsilon_c) - U(0)"""
    ec = epsilon_c(c)
    return U(ec, c) - U(0, c)  # U(0,c) = 0

# ======================================================================
print("\n"+"="*70)
print("■■■1■U(epsilon;c) ■■■■■ DeltaU(c)")
print("="*70)

c_values = [0.20, 0.25, 0.30, 0.35, 0.40, 0.42, 0.50, 0.55, 0.60, 0.70, 0.80, 0.90]
print(f"\n  {'c':&gt;6} {'epsilon_c':&gt;10} {'epsilon_eq':&gt;10} {'DeltaU':&gt;10} {'U_prime(ec)':&gt;12} {'U_dblprime(ec)':&gt;14}")
```

```
print(f"  {'-'*65}")
for c in c_values:
    ec = epsilon_c(c)
    eq = epsilon_eq(c)
    du = DeltaU(c)
    up = U_prime(ec, c)
    udp = U_double_prime(ec, c)
    print(f"  {c:&gt;6.2f} {ec:&gt;10.4f} {eq:&gt;10.4f} {du:&gt;10.4f} {up:&gt;12.4f} {udp:&gt;14.6f}")

print(f"\n  ■■:")
print(f"    epsilon_c = 1-sqrt(c): ■■■■■■■→■■■■■■■")
print(f"    epsilon_eq = sqrt(1-c): ■■■■U'=0■")
print(f"    DeltaU = U(epsilon_c) - U(0): ■■→■■■■■■■■■■■■■")
print(f"    DeltaU &lt; 0: ■■■■■■■■■■■■■■■■")
print(f"    |DeltaU| ■■■■■■■■■→■■■■■■■■■■■■")

# ======================================================================
print("\n"+"="*70)
print("■■■2■alpha(c) = f_elastic(c) ■■■■■■")
print("="*70)

# alpha = f_elastic = 1/(1 + exp(DeltaU/T_m))
# DeltaU &lt; 0 ■■■ exp(DeltaU/T_m) &lt; 1 → f_elastic &gt; 0.5
# ... ■■■■■ alpha &gt; 0.5 ■■■■■■■■■■■■
#
# ■■: ■■■■■■■■■
# DeltaU &lt; 0 = ■■■■■■■■■■■■ → f_plastic &gt; f_elastic
# → alpha = f_elastic &lt; 0.5
# ■■■■■■■■■■■■■ alpha ≈ 0.5
#
# → ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# ■■■■■■■■: F = U - T*S
# ■■■■■: ■■■■■■■■■■■■■■■■■■■■■■■■■
# ■■■■■: ■■■■■■■■■■■■■■■■■■■■■■■■■■
#
# → DeltaF = DeltaU - T_m * DeltaS
# alpha = 0.5 ■■■■: DeltaF = 0 → DeltaU = T_m * DeltaS
#
# ■■■■■■■■■■■■:
# f_elastic/f_plastic = exp(-beta * DeltaU_eff(c))
# DeltaU_eff(c) = DeltaU(c) - DeltaU_0  (■■■■■■■■)
# alpha=0.5 → DeltaU_eff = 0 → DeltaU_0 = DeltaU(c_mean)

# ■■: DeltaU(c) ■■■■■■■ alpha(c) ■■■■
c_range = np.linspace(0.05, 0.95, 200)
DU_range = np.array([DeltaU(c) for c in c_range])

# DeltaU(c) ■■■■■■■■■■■■
c_mean = 0.42  # Sc■■■■■■
DU_mean = DeltaU(c_mean)
print(f"  ■■■■■■ (c={c_mean}): DeltaU = {DU_mean:.4f}")

# ■■■■: DeltaU_eff = DeltaU - DeltaU(c_mean)
# → c=c_mean ■ alpha=0.5
DU_eff = DU_range - DU_mean

# T_m ■■■■■■■■■■■■■■■■■■■■■■■■
def alpha_theory(c, T_m, c0=0.42):
    """■■■■ alpha(c)"""
    du = DeltaU(c) - DeltaU(c0)
    return 1.0 / (1.0 + np.exp(du / T_m))

# ■■■■■ T_m ■■■ alpha(c) ■■■
print(f"\n  ■■■■■ alpha(c) = 1/(1+exp(DeltaU_eff/T_m)):")
print(f"  {'c':&gt;6}  ", end="")
T_m_candidates = [0.01, 0.05, 0.10, 0.20, 0.50, 1.00]
for tm in T_m_candidates:
    print(f"  T_m={tm:.2f}", end="")
print()
print(f"  {'-'*75}")

for c in [0.20, 0.30, 0.42, 0.60, 0.80]:
    print(f"  {c:&gt;6.2f}  ", end="")
    for tm in T_m_candidates:
        a = alpha_theory(c, tm)
        print(f"  {a:&gt;8.3f}", end="")
    print()

# ======================================================================
# ■■■■■■■■■ v_flat ■■■■ alpha ■■■■
# ======================================================================
print("\n"+"="*70)
print("■■■3■■■■■■■■■■■■")
print("="*70)

a0 = 1.2e-10; G_const = 6.674e-11; kpc_m = 3.086e19; kms_ms = 1e3
def find_col(df, cands):
```

```
    if df is None: return None
    for c in cands:
        m = [col for col in df.columns if c.lower() in col.lower()]
        if m: return m[0]
    return None

df_pred = pd.read_csv("gc_predictive_model.csv") if Path("gc_predictive_model.csv").exists() else None
df_sparc = pd.read_csv("sparc_results.csv") if Path("sparc_results.csv").exists() else None
df_gc = pd.read_csv("TA3_gc_independent.csv") if Path("TA3_gc_independent.csv").exists() else None

df = None
if df_pred is not None:
    cm = {}
    for k,cs in [('gc',['gc','g_c','gc_obs','gc_rar']),('vflat',['v_flat','vflat']),('hR',['h_R','hR','R_d'])]:
        c = find_col(df_pred, cs)
        if c: cm[k] = c
    if len(cm)==3:
        df = df_pred.copy()
        df['gc_val']=df[cm['gc']]; df['vflat_val']=df[cm['vflat']]; df['hR_val']=df[cm['hR']]
if df is None and df_gc is not None and df_sparc is not None:
    gc_c=find_col(df_gc,['gc','g_c']); gg=find_col(df_gc,['galaxy','name'])
    vf_c=find_col(df_sparc,['v_flat','vflat']); hr_c=find_col(df_sparc,['h_R','hR','R_d'])
    gs=find_col(df_sparc,['galaxy','name'])
    df=pd.merge(df_gc,df_sparc,left_on=gg,right_on=gs,how='inner')
    df['gc_val']=df[gc_c]; df['vflat_val']=df[vf_c]; df['hR_val']=df[hr_c]

if df is None:
    print("  !!! ■■■■■■CSV ■■■■■■■■■■")
    import sys; sys.exit(1)

type_col = find_col(df, ['type','Type','T','hubble','Hubble','morph'])
mask = df['gc_val'].notna() &amp;amp; df['vflat_val'].notna() &amp;amp; df['hR_val'].notna()
mask &amp;= (df['gc_val']&amp;gt;0) &amp;amp; (df['vflat_val']&amp;gt;0) &amp;amp; (df['hR_val']&amp;gt;0)
df = df[mask].copy().reset_index(drop=True)
print(f"  ■■■■■: N={len(df)}")

gc_vals=df['gc_val'].values; gc_med=np.median(gc_vals)
if gc_med&amp;lt;1e-5: gc_si=gc_vals; gc_a0=gc_vals/a0
else: gc_a0=gc_vals; gc_si=gc_vals*a0

vf_ms=df['vflat_val'].values*kms_ms; hR_m=df['hR_val'].values*kpc_m
G_S0=vf_ms**2/hR_m
log_gc=np.log10(gc_si); log_GS=np.log10(G_S0); log_a0v=np.log10(a0)
x = log_GS - log_a0v
n=len(df)

# --- v_flat ■■■ alpha ---
vflat_kms = df['vflat_val'].values
# 5■■■■■
vf_bins = np.percentile(vflat_kms, [0, 20, 40, 60, 80, 100])
vf_bins[0] -= 1; vf_bins[-1] += 1

obs_alphas = []
obs_vflat_med = []
obs_c_assigned = []
obs_n = []

# c ■■■■■: ■■■■■■■■■■■■■■■■ v_flat ■■■■
# SPARC ■■■■■■■■: ■■(v&amp;lt;50)→Im(c~0.30), ■■→Sc(c~0.42), ■■→Sb(c~0.80)
c_type_map = {10:0.30, 11:0.30, 9:0.33, 8:0.35, 7:0.37, 6:0.40, 5:0.42, 4:0.55, 3:0.80, 2:0.85, 1:0.90, 0:0.90}

if type_col and type_col in df.columns:
    tv = df[type_col].values
    c_galaxy = np.array([c_type_map.get(int(t), 0.42) if not np.isnan(t) else 0.42 for t in tv])
    print(f"  c ■: ■■■■■■■■■■■■")
else:
    # v_flat ■■ c ■■■■■■■■■■
    c_galaxy = np.clip(0.20 + 0.70 * (vflat_kms - 20) / 250, 0.20, 0.90)
    print(f"  c ■: v_flat ■■■■■■■■■■■■■■■■")

print(f"\n  v_flat ■■■ alpha■■■■:")
print(f"  {'■■':&amp;gt;20} {'N':&amp;gt;5} {'alpha_obs':&amp;gt;10} {'c_median':&amp;gt;10} {'v_flat_med':&amp;gt;10}")
print(f"  {'-'*60}")

for i in range(len(vf_bins)-1):
    mask_bin = (vflat_kms &amp;gt;= vf_bins[i]) &amp;amp; (vflat_kms &amp;lt; vf_bins[i+1])
    if np.sum(mask_bin) &amp;lt; 5:
        continue
    x_bin = x[mask_bin]; y_bin = log_gc[mask_bin]
    sl_bin, _, r_bin, _, se_bin = stats.linregress(x_bin, y_bin)
    vf_med = np.median(vflat_kms[mask_bin])
    c_med = np.median(c_galaxy[mask_bin])
    nb = np.sum(mask_bin)

    label = f"{vf_bins[i]:.0f}-{vf_bins[i+1]:.0f} km/s"
    print(f"  {label:&amp;gt;20} {nb:&amp;gt;5} {sl_bin:&amp;gt;10.3f} {c_med:&amp;gt;10.3f} {vf_med:&amp;gt;10.1f}")
    obs_alphas.append(sl_bin)
```

```
    obs_vflat_med.append(vf_med)
    obs_c_assigned.append(c_med)
    obs_n.append(nb)

obs_alphas = np.array(obs_alphas)
obs_c_assigned = np.array(obs_c_assigned)

# ======================================================================
print("\n"+"="*70)
print("■■■4■T_m ■■■■■■■")
print("="*70)

# alpha(c, T_m) = 1/(1+exp((DeltaU(c)-DeltaU(c0))/T_m))
# ■■■ (c_i, alpha_i) ■■■ T_m ■ c0 ■■■■■

if len(obs_alphas) &gt;= 3:
    def model_alpha(params, c_arr):
        T_m, c0 = params
        if T_m &lt;= 0 or c0 &lt;= 0 or c0 &gt;= 1:
            return np.full_like(c_arr, 0.5)
        return np.array([alpha_theory(c, T_m, c0) for c in c_arr])

    def residual_func(params, c_arr, alpha_obs):
        pred = model_alpha(params, c_arr)
        return np.sum((pred - alpha_obs)**2)

    # ■■■■■■■
    best_loss = np.inf
    best_params = (0.1, 0.42)
    for tm_try in np.linspace(0.01, 2.0, 100):
        for c0_try in np.linspace(0.20, 0.80, 30):
            loss = residual_func((tm_try, c0_try), obs_c_assigned, obs_alphas)
            if loss &lt; best_loss:
                best_loss = loss
                best_params = (tm_try, c0_try)

    T_m_opt, c0_opt = best_params
    alpha_pred = model_alpha(best_params, obs_c_assigned)
    r_fit = np.corrcoef(obs_alphas, alpha_pred)[0,1] if len(obs_alphas) &gt; 2 else 0

    print(f"  ■■■■■■■:")
    print(f"    T_m = {T_m_opt:.4f}  ■■■■■■■■")
    print(f"    c0  = {c0_opt:.4f}  ■alpha=0.5 ■■■ c ■■")
    print(f"    ■■■■ r = {r_fit:.4f}")
    print(f"    ■■ = {np.sqrt(best_loss/len(obs_alphas)):.4f}")

    print(f"\n  ■■■■■:")
    print(f"  {'c_obs':&gt;8} {'alpha_obs':&gt;10} {'alpha_pred':&gt;11} {'■':&gt;8}")
    print(f"  {'-'*40}")
    for i in range(len(obs_alphas)):
        diff = obs_alphas[i] - alpha_pred[i]
        print(f"  {obs_c_assigned[i]:&gt;8.3f} {obs_alphas[i]:&gt;10.3f} {alpha_pred[i]:&gt;11.3f} {diff:&gt;+8.3f}")

    # DeltaU(c0) ■■■■■■
    DU_c0 = DeltaU(c0_opt)
    print(f"\n  ■■■■■:")
    print(f"    DeltaU(c0={c0_opt:.3f}) = {DU_c0:.4f}")
    print(f"    T_m = {T_m_opt:.4f}")
    print(f"    T_c (■■11) = sqrt(6) = {np.sqrt(6):.4f}")
    print(f"    T_m / T_c = {T_m_opt/np.sqrt(6):.4f}")
else:
    print("  ■■■■■■■■■■■■")
    T_m_opt, c0_opt = 0.1, 0.42
# ======================================================================
print("\n"+"="*70)
print("■■■5■alpha(c) ■■■■ g_c ■■■■■")
print("="*70)

# ■■■■ alpha ■ c ■■■■
alpha_per_galaxy = np.array([alpha_theory(c, T_m_opt, c0_opt) for c in c_galaxy])

# g_c(predicted) = eta * a0^(1-alpha_i) * (G*S0_i)^alpha_i
# log(g_c) = log(eta) + (1-alpha_i)*log(a0) + alpha_i*log(G*S0_i)
log_gc_pred_ac = np.array([(1-a)*log_a0v + a*gs for a, gs in zip(alpha_per_galaxy, log_GS)])
# eta ■■■■■
eta_ac = np.median(log_gc - log_gc_pred_ac)
log_gc_pred_ac += eta_ac
resid_ac = log_gc - log_gc_pred_ac
r_ac = np.corrcoef(log_gc_pred_ac, log_gc)[0,1]

# ■■■: ■■ alpha=0.5
log_gc_pred_05 = 0.5*log_a0v + 0.5*log_GS
eta_05 = np.median(log_gc - log_gc_pred_05)
log_gc_pred_05 += eta_05
resid_05 = log_gc - log_gc_pred_05

# ■■■: MOND
```

```
resid_mond = log_gc - log_a0v

print(f"  ■■■■■■:")
print(f"  {'■■■':&lt;40} {'r':&gt;7} {'■■std':&gt;10} {'■■■':&gt;8}")
print(f"  {'-'*68}")
print(f"  {'MOND (g_c=a0)':&lt;40} {'---':&gt;7} {np.std(resid_mond):&gt;10.4f} {'---':&gt;8}")
print(f"  {'■■■■ alpha=0.5 ■■':&lt;40} {np.corrcoef(log_gc_pred_05,log_gc)[0,1]:&gt;7.4f} {np.std(resid_05):&gt;10.4f} {(1-np.std(resid_05
print(f"  {'alpha(c) ■■ (■■■■■■■■■■)':&lt;40} {r_ac:&gt;7.4f} {np.std(resid_ac):&gt;10.4f} {(1-np.std(resid_ac)/np.std(resid_mond))*100:&

# AIC
ss_mond = np.sum(resid_mond**2); aic_mond = n*np.log(ss_mond/n)
ss_05 = np.sum(resid_05**2); aic_05 = n*np.log(ss_05/n) + 2*1
ss_ac = np.sum(resid_ac**2); aic_ac = n*np.log(ss_ac/n) + 2*2  # T_m, c0
print(f"\n  AIC■■:")
print(f"    MOND:                dAIC = 0 (■■)")
print(f"    ■■■■ alpha=0.5:  dAIC = {aic_05-aic_mond:+.1f}")
print(f"    alpha(c) ■■■■:   dAIC = {aic_ac-aic_mond:+.1f}")

# ======================================================================
print("\n"+"="*70)
print("■■■■■■■■...")
print("="*70)

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle('Strain Energy Theory: $\\alpha(c) = f_{elastic}(c)$', fontsize=14, fontweight='bold')

# (a) U(epsilon;c) ■■■■■■■■■■
ax = axes[0, 0]
eps_range = np.linspace(0, 0.95, 200)
for c, col, ls in [(0.30,'blue','-'), (0.42,'green','-'), (0.80,'red','-')]:
    u_vals = U(eps_range, c)
    ax.plot(eps_range, u_vals, color=col, ls=ls, lw=2, label=f'c={c}')
    ec = epsilon_c(c)
    ax.axvline(ec, color=col, ls=':', alpha=0.3)
    ax.plot(ec, U(ec,c), 'o', color=col, ms=8)
ax.set_xlabel('$\\epsilon$ (strain)')
ax.set_ylabel('$U(\\epsilon; c)$')
ax.set_title('(a) Membrane potential')
ax.legend(fontsize=9)
ax.set_ylim(-3, 0.5)

# (b) DeltaU(c)
ax = axes[0, 1]
c_plot = np.linspace(0.05, 0.95, 100)
du_plot = np.array([DeltaU(c) for c in c_plot])
ax.plot(c_plot, du_plot, 'b-', lw=2)
ax.axhline(0, color='k', ls='--', alpha=0.3)
ax.axvline(c0_opt, color='r', ls='--', alpha=0.5, label=f'$c_0$={c0_opt:.3f}')
# ■■■
for i, c in enumerate(obs_c_assigned):
    ax.plot(c, DeltaU(c), 'ro', ms=8)
ax.set_xlabel('c')
ax.set_ylabel('$\\Delta U(c)$')
ax.set_title('(b) Strain energy barrier')
ax.legend(fontsize=9)
# (c) alpha(c) ■■■■ + ■■
ax = axes[0, 2]
for tm, col, ls in [(T_m_opt,'red','-'), (0.05,'blue','--'), (0.5,'green','--')]:
    alpha_curve = np.array([alpha_theory(c, tm, c0_opt) for c in c_plot])
    ax.plot(c_plot, alpha_curve, color=col, ls=ls, lw=2, label=f'$T_m$={tm:.3f}')
# ■■■
ax.scatter(obs_c_assigned, obs_alphas, c='black', s=80, zorder=5, label='Observed (v_flat bins)')
for i in range(len(obs_alphas)):
    ax.annotate(f'{obs_vflat_med[i]:.0f}', (obs_c_assigned[i], obs_alphas[i]),
                textcoords="offset points", xytext=(5,5), fontsize=8)
ax.axhline(0.5, color='gray', ls=':', alpha=0.5)
ax.set_xlabel('c')
ax.set_ylabel('$\\alpha(c)$')
ax.set_title(f'(c) $\\alpha(c)$ theory ($T_m$={T_m_opt:.3f})')
ax.legend(fontsize=8)
ax.set_ylim(-0.1, 1.2)

# (d) ■■ vs ■■ (alpha(c) ■■■)
ax = axes[1, 0]
ax.scatter(log_gc_pred_05, log_gc, s=12, alpha=0.3, c='green', label=f'$\\alpha$=0.5 (r={np.corrcoef(log_gc_pred_05,log_gc)[0,1]:.3f})')
ax.scatter(log_gc_pred_ac, log_gc, s=12, alpha=0.5, c='coral', label=f'$\\alpha$(c) (r={r_ac:.3f})')
dg = np.linspace(log_gc.min(), log_gc.max(), 100)
ax.plot(dg, dg, 'k--', alpha=0.3)
ax.set_xlabel('log($g_c$) predicted')
ax.set_ylabel('log($g_c$) observed')
ax.set_title('(d) Prediction comparison')
ax.legend(fontsize=8)

# (e) ■■■■
ax = axes[1, 1]
bins = np.linspace(-1, 1, 40)
ax.hist(resid_mond, bins=bins, alpha=0.3, color='gray', label=f'MOND ({np.std(resid_mond):.3f})')
```

```
ax.hist(resid_05, bins=bins, alpha=0.3, color='green', label=f'$\\alpha$=0.5 ({np.std(resid_05):.3f})')
ax.hist(resid_ac, bins=bins, alpha=0.5, color='coral', label=f'$\\alpha$(c) ({np.std(resid_ac):.3f})')
ax.set_xlabel('Residual [dex]')
ax.set_ylabel('Count')
ax.set_title('(e) Residual distributions')
ax.legend(fontsize=8)

# (f) alpha(galaxy) ■■■
ax = axes[1, 2]
ax.hist(alpha_per_galaxy, bins=30, color='steelblue', alpha=0.7, edgecolor='white')
ax.axvline(0.5, color='r', ls='--', lw=2, label='$\\alpha$=0.5')
ax.axvline(np.median(alpha_per_galaxy), color='k', ls='-', lw=1,
           label=f'median={np.median(alpha_per_galaxy):.3f}')
ax.set_xlabel('$\\alpha$ per galaxy')
ax.set_ylabel('Count')
ax.set_title(f'(f) $\\alpha$ distribution (N={n})')
ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('strain_energy_theory_verification.png', dpi=150, bbox_inches='tight')
print("  → strain_energy_theory_verification.png ■■■■■")

# ======================================================================
print("\n"+"="*70)
print("■■■■■■■")
print("="*70)

print(f"""
{'='*60}
  ■■■■■■■■■■■■■■■■
{'='*60}

■ ■■■■■:
  U(epsilon;c) ■■■→■■■■■■■■■■ DeltaU(c) ■
  ■■/■■■■■ f_elastic ■■■■■■
  ■■■■■■■■■■■ alpha ■■■■■:

    alpha(c) = 1/(1 + exp(DeltaU_eff(c)/T_m))
    g_c = eta * a0^(1-alpha(c)) * (G*Sigma0)^(alpha(c))

■ ■■■■■■■:
  T_m = {T_m_opt:.4f}  ■■■■■■■■■
  c0  = {c0_opt:.4f}  ■alpha=0.5 ■■■■■
  T_m/T_c = {T_m_opt/np.sqrt(6):.4f}  ■T_c = sqrt(6) = {np.sqrt(6):.4f}■

■ ■■■■■:
  MOND:              ■■ {np.std(resid_mond):.4f} dex
  ■■■■■ alpha=0.5: ■■ {np.std(resid_05):.4f} dex  ({(1-np.std(resid_05)/np.std(resid_mond))*100:.1f}% ■■)
  alpha(c) ■■:     ■■ {np.std(resid_ac):.4f} dex  ({(1-np.std(resid_ac)/np.std(resid_mond))*100:.1f}% ■■)

■ ■■■■■■:
  ■■■■■ → ■■■■■■■■■■■■ → G*Sigma0 ■■
  ■■■■■ → ■■■■■■■■■■   → a0 ■■

  ■■■■ (■c): ■■■■■ → alpha→1 → g_c ≈ G*Sigma0
  ■■■■ (■c): ■■■■■ → alpha→0 → g_c ≈ a0
  ■■■■■■:   ■■■■   → alpha≈0.5 → g_c ≈ sqrt(a0*G*Sigma0)

■ ■■■■■■■■■■:
  ■■■■■■■■■■■■■■■■■■≈ ■■■■■■■■■■■■■■■
  ■■■■■■■■■■■■■■■■■■  ≈ ■■■■■■■■■■■■
  ■■■■■■■■■■■■■■■■■■■■ ≈ DeltaU(c) ■ alpha ■■■■■
""")

print("■■")
```

eta_functional_form.py

解析目的	eta(disk_dom, Ud, compact) の関数形決定。幾何平均法則の残差構造の解明。
結果	A12 確立。eta ∝ Ud^(-0.44)*compact^(-0.35)。MOND比39.4%改善、LOO劣化2.5%。

```
"""
eta(disk_dom, Ud) ■■■■■■■■■■
======================================
■■■■■■: g_c = eta0 * sqrt(a0 * G * Sigma0)
■■:       R = log(g_c) - log(eta0 * sqrt(a0 * G * Sigma0))
■■:       R = f(disk_dom, Ud) ■■■■■■■■

■■■■:
  (1) ■■■■■■■■■■■■■■■■■■■■■■
  (2) ■■■■■■■■■■■■■ / ■■ / ■■■■ / ■■■■
  (3) 2■■■■■■■■■■■■■
  (4) ■■■■■■ g_c = eta(disk_dom, Ud) * sqrt(a0*G*Sigma0) ■■■■■■
  (5) LOO-CV ■■■■■■■■■■■■■

■■: uv run --with scipy --with matplotlib --with numpy --with pandas python eta_functional_form.py
"""

import numpy as np
import pandas as pd
from scipy import stats, optimize
from pathlib import Path
import matplotlib; matplotlib.use('Agg')
import matplotlib.pyplot as plt
import warnings; warnings.filterwarnings('ignore')

a0 = 1.2e-10; G_const = 6.674e-11; kpc_m = 3.086e19; kms_ms = 1e3

def find_col(df, cands):
    if df is None: return None
    for c in cands:
        m = [col for col in df.columns if c.lower() in col.lower()]
        if m: return m[0]
    return None

# ======================================================================
print("="*70)
print("eta(disk_dom, Ud) ■■■■■■■■■■■")
print("="*70)

# --- ■■■■■■■■ ---
df_pred = pd.read_csv("gc_predictive_model.csv") if Path("gc_predictive_model.csv").exists() else None
df_sparc = pd.read_csv("sparc_results.csv") if Path("sparc_results.csv").exists() else None
df_gc = pd.read_csv("TA3_gc_independent.csv") if Path("TA3_gc_independent.csv").exists() else None

# ■■■■■■■■
for name, d in [("gc_predictive_model",df_pred),("sparc_results",df_sparc),("TA3_gc_independent",df_gc)]:
    if d is not None:
        print(f"\n  {name}.csv ({len(d)} rows):")
        print(f"    {list(d.columns)}")

# --- ■■■■■■ ---
# gc_predictive_model.csv ■■■■■■■■■■■
df = None
if df_pred is not None:
    cm = {}
    for k,cs in [('gc',['gc','g_c','gc_obs','gc_rar','gc_meas']),
                 ('vflat',['v_flat','vflat','Vflat']),
                 ('hR',['h_R','hR','Reff','R_d'])]:
        c = find_col(df_pred, cs)
        if c: cm[k] = c
    if len(cm)==3:
        df = df_pred.copy()
        df['gc_val']=df[cm['gc']]; df['vflat_val']=df[cm['vflat']]; df['hR_val']=df[cm['hR']]
        print(f"\n  → gc_predictive_model.csv ■■■")

# ■■■■■■■■
if df is None and df_gc is not None and df_sparc is not None:
    gc_c=find_col(df_gc,['gc','g_c']); gg=find_col(df_gc,['galaxy','name'])
    vf_c=find_col(df_sparc,['v_flat','vflat']); hr_c=find_col(df_sparc,['h_R','hR','R_d'])
    gs=find_col(df_sparc,['galaxy','name'])
    df=pd.merge(df_gc,df_sparc,left_on=gg,right_on=gs,how='inner')
    df['gc_val']=df[gc_c]; df['vflat_val']=df[vf_c]; df['hR_val']=df[hr_c]

if df is None: print("!!! ■■■■■"); import sys; sys.exit(1)

mask = df['gc_val'].notna() &amp;amp; df['vflat_val'].notna() &amp;amp; df['hR_val'].notna()
mask &amp;amp;= (df['gc_val']&amp;gt;0) &amp;amp; (df['vflat_val']&amp;gt;0) &amp;amp; (df['hR_val']&amp;gt;0)
df = df[mask].copy().reset_index(drop=True)

gc_vals=df['gc_val'].values; gc_med=np.median(gc_vals)
```

```
if gc_med&lt;1e-5: gc_si=gc_vals; gc_a0=gc_vals/a0
else: gc_a0=gc_vals; gc_si=gc_vals*a0

vf_ms=df['vflat_val'].values*kms_ms; hR_m=df['hR_val'].values*kpc_m
vf_kms=df['vflat_val'].values; hR_kpc=df['hR_val'].values
G_S0 = vf_ms**2/hR_m
n=len(df)

# ■■■■■■■■■■
gc_gm = np.sqrt(a0 * G_S0)
eta0 = np.median(gc_si / gc_gm)
log_gc = np.log10(gc_si)
log_gc_gm = np.log10(eta0 * gc_gm)
R = log_gc - log_gc_gm  # ■■

print(f"\n  ■■■■■: N={n}")
print(f"  eta0 = {eta0:.4f}")
print(f"  ■■■■■■■■■: std={np.std(R):.4f} dex, median|R|={np.median(np.abs(R)):.4f} dex")

# ======================================================================
print("\n"+"="*70)
print("■■■1■■■■■■■■■■■■■■■■■■■■")
print("="*70)

# ■■■■■■■■
candidates = {}

# ■■■■■■■■
for cname, cands in [
    ('disk_dom', ['disk_dom','diskdom','disk']),
    ('Ud', ['Ud','ud','U_d','Upsilon','upsilon','ML','ml']),
    ('compact', ['compact','compactness']),
    ('V_peak', ['V_peak','Vpeak','v_peak','vpeak']),
    ('Sigma0', ['Sigma0','sigma0','Sigma_0']),
    ('type', ['type','Type','T','hubble']),
]:
    col = find_col(df, cands)
    if col:
        vals = pd.to_numeric(df[col], errors='coerce').values
        if np.sum(np.isfinite(vals)) &gt; n*0.5:
            candidates[cname] = vals
            print(f"  ■■: {cname} ← '{col}'")

# ■■■■■■
candidates['log_vflat'] = np.log10(vf_kms)
candidates['log_hR'] = np.log10(hR_kpc)
candidates['log_GS0'] = np.log10(G_S0)
candidates['Sigma0_proxy'] = vf_kms**2 / hR_kpc  # [(km/s)^2/kpc]

# V_peak ■■■■■■■■
if 'V_peak' not in candidates:
    # disk_dom = V_peak/v_flat → V_peak = disk_dom * v_flat
    if 'disk_dom' in candidates:
        dd = candidates['disk_dom']
        vp = dd * vf_kms
        candidates['V_peak'] = vp
        print(f"  ■■: V_peak = disk_dom * v_flat")

# disk_dom ■■■■■■■■
if 'disk_dom' not in candidates and 'V_peak' in candidates:
    candidates['disk_dom'] = candidates['V_peak'] / vf_kms
    print(f"  ■■: disk_dom = V_peak / v_flat")

# compact ■■■■■■■■
if 'compact' not in candidates and 'V_peak' in candidates:
    # compact ≈ V_peak / h_R (■■)
    candidates['compact'] = candidates['V_peak'] / hR_kpc
    print(f"  ■■: compact ≈ V_peak / h_R")

print(f"\n  ■■■■: {list(candidates.keys())}")

# ■■■■■■■■■■
print(f"\n  {'■■':&lt;18} {'Pearson r':&gt;10} {'Spearman rho':&gt;13} {'p■':&gt;12} {'■■':&gt;6}")
print(f"  {'-'*62}")

corr_results = []
for vname, vals in sorted(candidates.items()):
    valid = np.isfinite(vals) &amp;amp; np.isfinite(R)
    if np.sum(valid) &lt; 10: continue
    r_p, p_p = stats.pearsonr(vals[valid], R[valid])
    r_s, p_s = stats.spearmanr(vals[valid], R[valid])
    sig = "★★" if p_s &lt; 0.001 else "★" if p_s &lt; 0.05 else ""
    print(f"  {vname:&lt;18} {r_p:&gt;+10.3f} {r_s:&gt;+13.3f} {p_s:&gt;12.2e} {sig:&gt;6}")
    corr_results.append((vname, r_p, r_s, p_s))

# ■■■■■■■■
corr_results.sort(key=lambda x: abs(x[2]), reverse=True)
```

```
top_vars = [v for v, rp, rs, ps in corr_results if ps &lt; 0.01]
print(f"\n  ■■■■■ (p&lt;0.01): {top_vars}")

# ================================================================
print("\n"+"="*70)
print("■■■2■■■■■■■■■")
print("="*70)

# ■■■■■■■■■■■■■■■■■■■■
base_ss = np.sum(R**2)  # ■■■■■■■■■R ■■■■■■■■
base_ss_mean = np.sum((R - np.mean(R))**2)

def eval_model(x, y, model_func, n_params, label=""):
    """■■■■■■■■■AIC■■■■■"""
    try:
        valid = np.isfinite(x) &amp;amp; np.isfinite(y)
        xv, yv = x[valid], y[valid]
        nv = len(xv)
        if nv &lt; n_params + 5: return None

        pred = model_func(xv)
        if not np.all(np.isfinite(pred)): return None

        ss = np.sum((yv - pred)**2)
        r = np.corrcoef(pred, yv)[0,1] if np.std(pred)&amp;gt;0 else 0
        aic = nv * np.log(ss/nv) + 2 * n_params
        return {'ss': ss, 'r': r, 'aic': aic, 'n': nv, 'k': n_params, 'label': label}
    except:
        return None

for vname in top_vars[:5]:  # ■■5■■
    vals = candidates[vname]
    valid = np.isfinite(vals) &amp;amp; np.isfinite(R)
    xv, yv = vals[valid], R[valid]

    if len(xv) &lt; 15: continue

    print(f"\n  --- {vname} ---")

    results = []

    # (a) ■■: R = a*x + b
    sl, ic, r, p, se = stats.linregress(xv, yv)
    pred_lin = sl*xv + ic
    ss_lin = np.sum((yv-pred_lin)**2)
    aic_lin = len(xv)*np.log(ss_lin/len(xv)) + 2*2
    results.append(('■■', 2, ss_lin, aic_lin, r))

    # (b) ■■: R = a*log(|x|) + b
    lx = np.log10(np.abs(xv) + 1e-10)
    if np.std(lx) &amp;gt; 0:
        sl2, ic2, r2, p2, _ = stats.linregress(lx, yv)
        pred_log = sl2*lx + ic2
        ss_log = np.sum((yv-pred_log)**2)
        aic_log = len(xv)*np.log(ss_log/len(xv)) + 2*2
        results.append(('■■', 2, ss_log, aic_log, r2))

    # (c) ■■■: R = a*x^b + c ■x&amp;gt;0 ■■■■
    if np.all(xv &amp;gt; 0):
        lx_pos = np.log10(xv)
        sl3, ic3, r3, p3, _ = stats.linregress(lx_pos, yv)
        pred_pow = sl3*lx_pos + ic3
        ss_pow = np.sum((yv-pred_pow)**2)
        aic_pow = len(xv)*np.log(ss_pow/len(xv)) + 2*2
        results.append(('■■■(log-log)', 2, ss_pow, aic_pow, r3))

    # (d) 2■: R = a*x^2 + b*x + c
    coeffs = np.polyfit(xv, yv, 2)
    pred_quad = np.polyval(coeffs, xv)
    ss_quad = np.sum((yv-pred_quad)**2)
    aic_quad = len(xv)*np.log(ss_quad/len(xv)) + 2*3
    r_quad = np.corrcoef(pred_quad, yv)[0,1]
    results.append(('2■', 3, ss_quad, aic_quad, r_quad))
    # ■■■
    print(f"    {'■■■':&lt;18} {'■■■■■':&amp;gt;6} {'RSS':&amp;gt;10} {'AIC':&amp;gt;10} {'r':&amp;gt;8}")
    print(f"    {'-'*56}")
    for nm, k, ss, aic, r in results:
        print(f"    {nm:&lt;18} {k:&amp;gt;6} {ss:&amp;gt;10.4f} {aic:&amp;gt;10.1f} {r:&amp;gt;+8.3f}")

    # ■■■■■■■■■
    best = min(results, key=lambda x: x[3])
    print(f"    → ■■: {best[0]}■AIC={best[3]:.1f}■")

# ================================================================
print("\n"+"="*70)
print("■■■3■2■■■■■■■■■")
print("="*70)
```

```
# disk_dom ■ Ud ■■■■■■■■■■■■■■■■■
# ■■■■■■:
#   M1: R = a*log(disk_dom) + b*log(Ud) + c
#   M2: R = a*disk_dom + b*Ud + c
#   M3: R = a*log(disk_dom) + b*Ud + c ■■■■
#   M4: R = a*disk_dom^p + b  ■■■■■■■■■

# disk_dom ■ Ud ■■■
dd_key = 'disk_dom' if 'disk_dom' in candidates else None
ud_key = 'Ud' if 'Ud' in candidates else None

if dd_key and ud_key:
    dd = candidates[dd_key]
    ud = candidates[ud_key]
    valid2 = np.isfinite(dd) &amp;amp; np.isfinite(ud) &amp;amp; np.isfinite(R) &amp;amp; (dd&amp;gt;0) &amp;amp; (ud&amp;gt;0)
    dd_v = dd[valid2]; ud_v = ud[valid2]; R_v = R[valid2]; n_v = len(R_v)
    log_dd = np.log10(dd_v); log_ud = np.log10(ud_v)

    print(f"  disk_dom: ■■■={np.median(dd_v):.3f}, ■■=[{dd_v.min():.3f}, {dd_v.max():.3f}]")
    print(f"  Ud:       ■■■={np.median(ud_v):.3f}, ■■=[{ud_v.min():.3f}, {ud_v.max():.3f}]")
    print(f"  ■■■■■■: N={n_v}")

    model_results = []

    # M0: ■■■■■■■■■■■■
    ss0 = np.sum((R_v - np.mean(R_v))**2)
    aic0 = n_v*np.log(ss0/n_v) + 2*1
    model_results.append(('M0: ■■■■', 1, ss0, aic0, 0, None, None))

    # M1: log-log
    X1 = np.column_stack([log_dd, log_ud, np.ones(n_v)])
    b1, _, _, _ = np.linalg.lstsq(X1, R_v, rcond=None)
    p1 = X1 @ b1; ss1 = np.sum((R_v-p1)**2); r1 = np.corrcoef(p1,R_v)[0,1]
    aic1 = n_v*np.log(ss1/n_v) + 2*3
    model_results.append(('M1: a*log(dd) + b*log(Ud) + c', 3, ss1, aic1, r1, b1,
                          f'R = {b1[0]:+.3f}*log(dd) {b1[1]:+.3f}*log(Ud) {b1[2]:+.3f}'))

    # M2: ■■
    X2 = np.column_stack([dd_v, ud_v, np.ones(n_v)])
    b2, _, _, _ = np.linalg.lstsq(X2, R_v, rcond=None)
    p2 = X2 @ b2; ss2 = np.sum((R_v-p2)**2); r2 = np.corrcoef(p2,R_v)[0,1]
    aic2 = n_v*np.log(ss2/n_v) + 2*3
    model_results.append(('M2: a*dd + b*Ud + c', 3, ss2, aic2, r2, b2,
                          f'R = {b2[0]:+.4f}*dd {b2[1]:+.4f}*Ud {b2[2]:+.4f}'))

    # M3: ■■
    X3 = np.column_stack([log_dd, ud_v, np.ones(n_v)])
    b3, _, _, _ = np.linalg.lstsq(X3, R_v, rcond=None)
    p3 = X3 @ b3; ss3 = np.sum((R_v-p3)**2); r3 = np.corrcoef(p3,R_v)[0,1]
    aic3 = n_v*np.log(ss3/n_v) + 2*3
    model_results.append(('M3: a*log(dd) + b*Ud + c', 3, ss3, aic3, r3, b3,
                          f'R = {b3[0]:+.3f}*log(dd) {b3[1]:+.4f}*Ud {b3[2]:+.4f}'))

    # M4: disk_dom ■■ (log)
    X4 = np.column_stack([log_dd, np.ones(n_v)])
    b4, _, _, _ = np.linalg.lstsq(X4, R_v, rcond=None)
    p4 = X4 @ b4; ss4 = np.sum((R_v-p4)**2); r4 = np.corrcoef(p4,R_v)[0,1]
    aic4 = n_v*np.log(ss4/n_v) + 2*2
    model_results.append(('M4: a*log(dd) + b ■■■■■', 2, ss4, aic4, r4, b4,
                          f'R = {b4[0]:+.3f}*log(dd) {b4[1]:+.4f}'))

    # M5: Ud ■■ (log)
    X5 = np.column_stack([log_ud, np.ones(n_v)])
    b5, _, _, _ = np.linalg.lstsq(X5, R_v, rcond=None)
    p5 = X5 @ b5; ss5 = np.sum((R_v-p5)**2); r5 = np.corrcoef(p5,R_v)[0,1]
    aic5 = n_v*np.log(ss5/n_v) + 2*2
    model_results.append(('M5: a*log(Ud) + b ■■■■■', 2, ss5, aic5, r5, b5,
                          f'R = {b5[0]:+.3f}*log(Ud) {b5[1]:+.4f}'))

    # M6: disk_dom * Ud ■■■■
    ddud = dd_v * ud_v
    X6 = np.column_stack([log_dd, log_ud, np.log10(ddud+1e-10), np.ones(n_v)])
    b6, _, _, _ = np.linalg.lstsq(X6, R_v, rcond=None)
    p6 = X6 @ b6; ss6 = np.sum((R_v-p6)**2); r6 = np.corrcoef(p6,R_v)[0,1]
    aic6 = n_v*np.log(ss6/n_v) + 2*4
    model_results.append(('M6: log(dd)+log(Ud)+log(dd*Ud)+c', 4, ss6, aic6, r6, b6, None))

    # ■■■■
    print(f"\n  {'■■■':&amp;lt;42} {'k':&amp;gt;3} {'RSS':&amp;gt;9} {'AIC':&amp;gt;9} {'dAIC':&amp;gt;7} {'r':&amp;gt;7}")
    print(f"  {'-'*80}")
    aic_base = model_results[0][3]
    for nm, k, ss, aic, r, _, _ in model_results:
        print(f"  {nm:&amp;lt;42} {k:&amp;gt;3} {ss:&amp;gt;9.4f} {aic:&amp;gt;9.1f} {aic-aic_base:&amp;gt;+7.1f} {r:&amp;gt;+7.3f}")

    # ■■■■■■■■■
    best_model = min(model_results[1:], key=lambda x: x[3])
    print(f"\n  ★ ■■■■■: {best_model[0]}")
```

```
    if best_model[6]:
        print(f"    ■: {best_model[6]}")
    print(f"    r = {best_model[4]:.4f}, AIC = {best_model[3]:.1f}")

    # ■■■■■■■■■■■■
    best_idx = model_results.index(best_model)

else:
    print("  disk_dom ■■■ Ud ■■■■■■■■■")
    print(f"  ■■■■■■■: {list(candidates.keys())}")
    # ■■2■■■■■■
    if len(top_vars) &gt;= 2:
        print(f"  → ■■2■■ {top_vars[0]}, {top_vars[1]} ■■■")
    dd_key = None; ud_key = None

# ======================================================================
print("\n"+"="*70)
print("■■■4■■■■■■■■■■■■")
print("="*70)

# g_c = eta(dd, Ud) * sqrt(a0 * G*S0) ■■■■■■
# eta = eta0 * 10^R_pred where R_pred = best model prediction

resid_mond = log_gc - np.log10(a0)  # MOND
resid_gm = R.copy()  # ■■■■ (alpha=0.5)

# ■■2■■■■■■■■■
if dd_key and ud_key:
    # ■■■■■■■■■valid2 ■■■■■■■
    R_pred_full = np.zeros(n)

    # best model ■■■■■■■■■
    bm = best_model
    if 'M1' in bm[0]:
        for i in range(n):
            if dd[i]&gt;0 and ud[i]&gt;0:
                R_pred_full[i] = bm[5][0]*np.log10(dd[i]) + bm[5][1]*np.log10(ud[i]) + bm[5][2]
    elif 'M2' in bm[0]:
        for i in range(n):
            R_pred_full[i] = bm[5][0]*dd[i] + bm[5][1]*ud[i] + bm[5][2]
    elif 'M4' in bm[0]:
        for i in range(n):
            if dd[i]&gt;0:
                R_pred_full[i] = bm[5][0]*np.log10(dd[i]) + bm[5][1]
    elif 'M5' in bm[0]:
        for i in range(n):
            if ud[i]&gt;0:
                R_pred_full[i] = bm[5][0]*np.log10(ud[i]) + bm[5][1]
    else:
        # M3 or others: log(dd) + Ud
        for i in range(n):
            if dd[i]&gt;0:
                R_pred_full[i] = bm[5][0]*np.log10(dd[i]) + bm[5][1]*ud[i] + bm[5][2]

    log_gc_final = log_gc_gm + R_pred_full
    resid_final = log_gc - log_gc_final
    r_final = np.corrcoef(log_gc_final, log_gc)[0,1]
    # 6■■■■■■■■■■■■■
    log_vf = np.log10(vf_kms); log_hR = np.log10(hR_kpc)
    log_gc_6var = (-2.175 + 2.015*log_vf - 1.294*log_hR) + np.log10(a0)
    resid_6var = log_gc - log_gc_6var

    print(f"  {'■■■':&lt;45} {'r':&gt;7} {'■■std':&gt;10} {'MOND■■■':&gt;10}")
    print(f"  {'-'*75}")
    print(f"  {'MOND (g_c=a0)':&lt;45} {'---':&gt;7} {np.std(resid_mond):&gt;10.4f} {'---':&gt;10}")
    print(f"  {'sqrt(a0*G*S0) (eta■■)':&lt;45} {np.corrcoef(log_gc_gm,log_gc)[0,1]:&gt;7.4f} {np.std(resid_gm):&gt;10.4f} {(1-np.std(resid_
    print(f"  {'sqrt(a0*G*S0) * eta(dd,Ud)':&lt;45} {r_final:&gt;7.4f} {np.std(resid_final):&gt;10.4f} {(1-np.std(resid_final)/np.std(resid
    print(f"  {'6■■■■■■ (2■■■■■■)':&lt;45} {np.corrcoef(log_gc_6var,log_gc)[0,1]:&gt;7.4f} {np.std(resid_6var):&gt;10.4f} {(1-np.std(resid_

# ======================================================================
print("\n"+"="*70)
print("■■■■5■LOO-CV ■■■■■■■■■■■■")
print("="*70)

if dd_key and ud_key and 'M1' in best_model[0]:
    # LOO-CV for M1: R = a*log(dd) + b*log(Ud) + c
    loo_resid = np.zeros(n_v)
    for i in range(n_v):
        mask_loo = np.ones(n_v, dtype=bool); mask_loo[i] = False
        X_loo = np.column_stack([log_dd[mask_loo], log_ud[mask_loo], np.ones(n_v-1)])
        b_loo, _, _, _ = np.linalg.lstsq(X_loo, R_v[mask_loo], rcond=None)
        pred_i = b_loo[0]*log_dd[i] + b_loo[1]*log_ud[i] + b_loo[2]
        loo_resid[i] = R_v[i] - pred_i

    print(f"  LOO-CV ■■ (■■■■■■):")
    print(f"    ■■■■■■■■■ std = {np.std(R_v - (X1@b1)):&gt;.4f} dex")
    print(f"    LOO-CV ■■ std     = {np.std(loo_resid):.4f} dex")
```

```
    print(f"    ■■■ = {(np.std(loo_resid)/np.std(R_v-(X1@b1))-1)*100:.1f}%")

    # LOO-CV ■■■■■■■■■■
    loo_total_std = np.sqrt(np.std(log_gc_gm[valid2] - log_gc[valid2])**2
                            - np.std(R_v-(X1@b1))**2 + np.std(loo_resid)**2)
    # ■■: ■■■■■ ≈ sqrt(■■■■■■² - insample■■² + LOO■■²)
    print(f"    ■■■■■ LOO ■■■■ ≈ {loo_total_std:.4f} dex")

elif dd_key and ud_key:
    # ■■■■■■ LOO
    print(f"  LOO-CV: ■■■■■■■■■■■■...")
    bm = best_model
    bm_name = bm[0]

    if 'M4' in bm_name:
        loo_r2 = np.zeros(n_v)
        for i in range(n_v):
            m = np.ones(n_v, dtype=bool); m[i] = False
            X_ = np.column_stack([log_dd[m], np.ones(n_v-1)])
            b_, _, _, _ = np.linalg.lstsq(X_, R_v[m], rcond=None)
            loo_r2[i] = R_v[i] - (b_[0]*log_dd[i] + b_[1])
        print(f"    ■■■■■■■■■ std = {np.std(R_v - p4[valid2] if len(p4)==n else R_v - (X4@b4)):.4f}")
        print(f"    LOO-CV ■■ std     = {np.std(loo_r2):.4f}")
    else:
        print(f"    ■{bm_name} ■ LOO-CV ■■■■■")

# ======================================================================
print("\n"+"="*70)
print("■■■■■■■■...")
print("="*70)

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle('$\\eta(disk\\_dom, \\Upsilon_d)$ Functional Form', fontsize=14, fontweight='bold')

# (a) ■■ vs disk_dom
ax = axes[0, 0]
if dd_key:
    dd_plot = candidates[dd_key]
    valid_p = np.isfinite(dd_plot) &amp;amp; np.isfinite(R)
    ax.scatter(dd_plot[valid_p], R[valid_p], s=15, alpha=0.5, c='steelblue')
    # ■■■■■■
    sl_dd, ic_dd, _, _, _ = stats.linregress(dd_plot[valid_p], R[valid_p])
    xr = np.linspace(dd_plot[valid_p].min(), dd_plot[valid_p].max(), 100)
    ax.plot(xr, sl_dd*xr+ic_dd, 'r-', lw=2)
    rho_dd, _ = stats.spearmanr(dd_plot[valid_p], R[valid_p])
    ax.set_xlabel('disk_dom ($V_{peak}/v_{flat}$)')
    ax.set_ylabel('Residual [dex]')
    ax.set_title(f'(a) $\\rho$={rho_dd:.3f}')
ax.axhline(0, color='k', ls='--', alpha=0.3)
# (b) ■■ vs Ud
ax = axes[0, 1]
if ud_key:
    ud_plot = candidates[ud_key]
    valid_p = np.isfinite(ud_plot) &amp;amp; np.isfinite(R) &amp;amp; (ud_plot &amp;gt; 0)
    ax.scatter(np.log10(ud_plot[valid_p]), R[valid_p], s=15, alpha=0.5, c='coral')
    sl_ud, ic_ud, _, _, _ = stats.linregress(np.log10(ud_plot[valid_p]), R[valid_p])
    xr = np.linspace(np.log10(ud_plot[valid_p]).min(), np.log10(ud_plot[valid_p]).max(), 100)
    ax.plot(xr, sl_ud*xr+ic_ud, 'r-', lw=2)
    rho_ud, _ = stats.spearmanr(ud_plot[valid_p], R[valid_p])
    ax.set_xlabel('log($\\Upsilon_d$)')
    ax.set_ylabel('Residual [dex]')
    ax.set_title(f'(b) $\\rho$={rho_ud:.3f}')
ax.axhline(0, color='k', ls='--', alpha=0.3)

# (c) 2■■■■■: observed vs predicted residual
ax = axes[0, 2]
if dd_key and ud_key:
    ax.scatter(X1@b1, R_v, s=15, alpha=0.5, c='steelblue')
    dg = np.linspace(min(X1@b1), max(X1@b1), 100)
    ax.plot(dg, dg, 'r--', lw=1)
    ax.set_xlabel('Predicted residual [dex]')
    ax.set_ylabel('Observed residual [dex]')
    ax.set_title(f'(c) 2-var model (r={r1:.3f})')

# (d) ■■■■■: observed vs predicted g_c
ax = axes[1, 0]
if dd_key and ud_key:
    ax.scatter(log_gc_gm, log_gc, s=10, alpha=0.3, c='green', label=f'$\\eta$ fixed ({np.std(resid_gm):.3f})')
    ax.scatter(log_gc_final, log_gc, s=10, alpha=0.5, c='coral', label=f'$\\eta$(dd,Ud) ({np.std(resid_final):.3f})')
    dg = np.linspace(log_gc.min(), log_gc.max(), 100)
    ax.plot(dg, dg, 'k--', alpha=0.3)
    ax.set_xlabel('log($g_c$) predicted')
    ax.set_ylabel('log($g_c$) observed')
    ax.set_title('(d) Final model comparison')
    ax.legend(fontsize=8)

# (e) ■■■■■■
```

```
ax = axes[1, 1]
bins = np.linspace(-1, 1, 40)
ax.hist(resid_mond, bins=bins, alpha=0.2, color='gray', label=f'MOND ({np.std(resid_mond):.3f})')
ax.hist(resid_gm, bins=bins, alpha=0.3, color='green', label=f'$\\eta$ fixed ({np.std(resid_gm):.3f})')
if dd_key and ud_key:
    ax.hist(resid_final, bins=bins, alpha=0.5, color='coral', label=f'$\\eta$(dd,Ud) ({np.std(resid_final):.3f})')
ax.set_xlabel('Residual [dex]')
ax.set_ylabel('Count')
ax.set_title('(e) Residual distributions')
ax.legend(fontsize=8)

# (f) ■■■■■■■■■■■■■■
ax = axes[1, 2]
plot_vars = [v for v in top_vars[:6] if v in candidates]
if len(plot_vars) &gt;= 2:
    corr_data = []
    for v in plot_vars:
        corr_data.append(candidates[v])
    corr_data.append(R)
    plot_vars_ext = plot_vars + ['Residual']

    # Spearman ■■■■
    corr_mat = np.zeros((len(plot_vars_ext), len(plot_vars_ext)))
    for i in range(len(plot_vars_ext)):
        for j in range(len(plot_vars_ext)):
            vi = corr_data[i]; vj = corr_data[j]
            valid_ij = np.isfinite(vi) &amp;amp; np.isfinite(vj)
            if np.sum(valid_ij) &gt; 10:
                corr_mat[i,j], _ = stats.spearmanr(vi[valid_ij], vj[valid_ij])

    im = ax.imshow(corr_mat, cmap='RdBu_r', vmin=-1, vmax=1, aspect='auto')
    ax.set_xticks(range(len(plot_vars_ext)))
    ax.set_yticks(range(len(plot_vars_ext)))
    short_names = [v[:8] for v in plot_vars_ext]
    ax.set_xticklabels(short_names, rotation=45, fontsize=7)
    ax.set_yticklabels(short_names, fontsize=7)
    for i in range(len(plot_vars_ext)):
        for j in range(len(plot_vars_ext)):
            ax.text(j, i, f'{corr_mat[i,j]:.2f}', ha='center', va='center', fontsize=6)
    plt.colorbar(im, ax=ax, shrink=0.8)
    ax.set_title('(f) Correlation matrix')
plt.tight_layout()
plt.savefig('eta_functional_form_verification.png', dpi=150, bbox_inches='tight')
print("  → eta_functional_form_verification.png ■■■■")

# ======================================================================
print("\n"+"="*70)
print("■■■■■■■")
print("="*70)

print(f"""
{'='*60}
  eta(disk_dom, Ud) ■■■■■■■■■
{'='*60}

■ ■■■■■■■■■■■■:
  ■■ std = {np.std(R):.4f} dex■■■■■■ alpha=0.5■""")

if dd_key and ud_key:
    print(f"""
■ ■■■■■■■■:""")
    for vn, rp, rs, ps in corr_results[:5]:
        sig = "★★" if ps&lt;0.001 else "★" if ps&lt;0.05 else ""
        print(f"    {vn:&lt;18}: rho={rs:+.3f} {sig}")

    print(f"""
■ ■■■■■■: {best_model[0]}""")
    if best_model[6]:
        print(f"    ■: {best_model[6]}")
    print(f"    r = {best_model[4]:.4f}")

    print(f"""
■ ■■■■■■■■:
  g_c = eta(dd,Ud) * sqrt(a0 * G * Sigma0)

  eta(dd,Ud) = eta0 * 10^R_pred
  eta0 = {eta0:.4f}

■ ■■■■■:
  MOND:              ■■ {np.std(resid_mond):.4f} dex
  ■■■■(eta■■):  ■■ {np.std(resid_gm):.4f} dex  ({(1-np.std(resid_gm)/np.std(resid_mond))*100:.1f}% ■■)
  ■■■■(eta■■):  ■■ {np.std(resid_final):.4f} dex  ({(1-np.std(resid_final)/np.std(resid_mond))*100:.1f}% ■■)
  6■■■■■■:        ■■ {np.std(resid_6var):.4f} dex  ({(1-np.std(resid_6var)/np.std(resid_mond))*100:.1f}% ■■)

■ ■■■■■■■■:
  ■1■: g_c = eta * sqrt(a0 * G*Sigma0)  ← ■■ (alpha=0.5)
  ■2■: eta = eta(disk_dom, Ud)          ← ■■■■■■
```

```
  ■3■: 6■■■■■■■■                  ← compact, V_peak ■
""")

print("■■")
```

B. 弱重力レンズ解析（HSC-SSP Y3）

inspect_subaru_lensing.py

解析目的	Y3 シェイプカタログ（931720.csv.gz.1）の構造確認。19カラム、3580万天体の基本統計。
結果	HSC-SSP Y3 と確認。e1/e2/weight/zbin 等のカラム特定。

```
"""
■■■■■■■ ■■■■■■■■■■ ■■■■■
========================================
■■: 931720.csv.gz.1 ■■■■■■■■■■■■■■■■■■■■■■■
      ■■■■■■■■■■■■■■■■■■■■

■■: uv run --with pandas --with numpy python inspect_subaru_lensing.py
"""

import numpy as np
import pandas as pd
from pathlib import Path
import gzip
import sys

# ======================================================================
# 0. ■■■■■■■
# ======================================================================
# ■■■■■■■■■■■■■
DATA_FILE = Path(r"D:\■■■■■■■\■■■■■■■\■■■■■■\■■■■■■■■\■■■■■\931720.csv.gz.1")

print("="*70)
print("■■■■■■■ ■■■■■■■■■■ ■■■■■")
print("="*70)
print(f"  ■■■■■: {DATA_FILE}")
print(f"  ■■: {DATA_FILE.exists()}")

if not DATA_FILE.exists():
    print("  !!! ■■■■■■■■■■■■■■■■■■■■■■■■■■■■")
    sys.exit(1)

# ■■■■■■■■
size_mb = DATA_FILE.stat().st_size / (1024*1024)
print(f"  ■■■■: {size_mb:.1f} MB")

# ======================================================================
# 1. ■■■■■■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("■Step 1■■■■■■■■■■■")
print("="*70)

# .gz.1 → gzip ■■■■■■■■ gzip ■■■■■■■■■■ plain text
header_line = None
is_gzip = False

try:
    with gzip.open(DATA_FILE, 'rt', encoding='utf-8') as f:
        header_line = f.readline().strip()
    is_gzip = True
    print(f"  ■■■■■■■: gzip ■■ CSV")
except:
    try:
        with open(DATA_FILE, 'r', encoding='utf-8') as f:
            header_line = f.readline().strip()
        print(f"  ■■■■■■■: ■■■■■ CSV")
    except:
        try:
            with open(DATA_FILE, 'r', encoding='utf-8-sig') as f:
                header_line = f.readline().strip()
            print(f"  ■■■■■■■: ■■■■■ CSV (BOM■■)")
        except Exception as e:
            print(f"  !!! ■■■■■■■■: {e}")
            sys.exit(1)

# ■■■■■■■■
if header_line:
    n_comma = header_line.count(',')
    n_tab = header_line.count('\t')
    n_pipe = header_line.count('|')
    n_space = header_line.count(' ')
    sep = ',' if n_comma &gt; max(n_tab, n_pipe) else '\t' if n_tab &gt; n_pipe else '|' if n_pipe &gt; 0 else ','
    print(f"  ■■■■■■: '{sep}' (comma={n_comma}, tab={n_tab}, pipe={n_pipe})")

    cols = [c.strip().strip('"').strip("'") for c in header_line.split(sep)]
    print(f"  ■■■■■: {len(cols)}")
```

```
    print(f"\n  ■■■■■■■■■100■■■:")
    print(f"    {header_line[:200]}")
else:
    print("  !!! ■■■■■■■■■■■")
    sys.exit(1)

# ======================================================================
# 2. ■■■■■■■■■■1000■ + ■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("■Step 2■■■■■■■■■")
print("="*70)

# ■■■■■1000■■■■■
try:
    if is_gzip:
        df_head = pd.read_csv(DATA_FILE, compression='gzip', sep=sep, nrows=1000)
    else:
        df_head = pd.read_csv(DATA_FILE, sep=sep, nrows=1000)
    print(f"  ■■1000■■■■■■■■■")
except Exception as e:
    print(f"  ■■■■■■■■■■: {e}")
    # sep ■■■■■■■■
    for try_sep in [',', '\t', ' ', '|', ';']:
        try:
            if is_gzip:
                df_head = pd.read_csv(DATA_FILE, compression='gzip', sep=try_sep, nrows=1000)
            else:
                df_head = pd.read_csv(DATA_FILE, sep=try_sep, nrows=1000)
            sep = try_sep
            print(f"  sep='{try_sep}' ■■■■■■■")
            break
        except:
            continue
    else:
        print("  !!! ■ sep ■■■■")
        sys.exit(1)

print(f"  shape: {df_head.shape}")
print(f"  ■■■■: {len(df_head.columns)}")

# ======================================================================
# 3. ■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("■Step 3■■■■■■■")
print("="*70)

print(f"\n  {'#':&gt;3} {'■■■■':&lt;35} {'■':&lt;12} {'■null':&gt;6} {'■■■■■■■':&lt;30}")
print(f"  {'-'*90}")
for i, col in enumerate(df_head.columns):
    dtype = str(df_head[col].dtype)
    non_null = df_head[col].notna().sum()
    example = str(df_head[col].iloc[0])[:28] if non_null &gt; 0 else "NaN"
    print(f"  {i:&gt;3} {col:&lt;35} {dtype:&lt;12} {non_null:&gt;6} {example:&lt;30}")

# ======================================================================
# 4. ■■■■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("■Step 4■■■■■■■■■■■■■■■■■■■")
print("="*70)

# ■■■■■■■■■■■■■■■■
lensing_keys = {
    'RA / ■■': ['ra', 'raj2000', 'alpha', 'ra_j2000', 'ra2000', 'ra_gal', 'i_ra'],
    'DEC / ■■': ['dec', 'decj2000', 'delta', 'dec_j2000', 'de2000', 'dec_gal', 'i_dec'],
    'e1 / ■■■1': ['e1', 'e1_cal', 'e1_model', 'g1', 'shape_e1', 'i_sdss_shape_11'],
    'e2 / ■■■2': ['e2', 'e2_cal', 'e2_model', 'g2', 'shape_e2', 'i_sdss_shape_22'],
    'weight / ■■': ['weight', 'w', 'lensfit_weight', 'shape_weight', 'i_hsmshaperegauss_sigma'],
    'z_photo / ■■■■■■■': ['z_b', 'z_phot', 'z_photo', 'photoz', 'photo_z', 'z_best',
                          'photoz_best', 'pz_best', 'mizuki_photoz_best'],
    'z_spec / ■■■■■■■': ['z_spec', 'zspec', 'specz'],
    'mag / ■■': ['mag', 'mag_auto', 'mag_r', 'mag_i', 'i_cmodel_mag', 'r_cmodel_mag',
                 'i_mag', 'r_mag', 'imag_psf'],
    'fitclass / ■■■■■■■': ['fitclass', 'flag', 'extendedness', 'i_extendedness_value',
                           'shape_flag', 'ishape_flags'],
    'stellar mass / ■■■■■': ['stellar_mass', 'logmstar', 'mass', 'mstar'],
    'size / ■■■': ['re', 'r_eff', 'hlr', 'i_sdss_shape_psf', 'fwhm',
                   'i_hsmshaperegauss_resolution'],
}

found = {}
col_lower = {c.lower(): c for c in df_head.columns}
for category, candidates in lensing_keys.items():
    for cand in candidates:
        # ■■■■
```

```
        if cand in col_lower:
            found[category] = col_lower[cand]
            break
        # ■■■■
        matches = [c for c in df_head.columns if cand.lower() in c.lower()]
        if matches:
            found[category] = matches[0]
            break

print(f"\n  ■■■■:")
for cat, col in found.items():
    vals = df_head[col].dropna()
    if len(vals) &gt; 0 and pd.api.types.is_numeric_dtype(vals):
        print(f"    {cat:&lt;25} -&gt; {col:&lt;30} [{vals.min():.4g} ~ {vals.max():.4g}]")
    else:
        print(f"    {cat:&lt;25} -&gt; {col:&lt;30} [{vals.iloc[0] if len(vals)&gt;0 else 'N/A'}]")

not_found = [cat for cat in lensing_keys if cat not in found]
if not_found:
    print(f"\n  ■■■: {not_found}")

# ======================================================================
# 5. ■■■■
# ======================================================================
print("\n"+"="*70)
print("■Step 5■■■■■■■■■■■■")
print("="*70)

numeric_cols = df_head.select_dtypes(include=[np.number]).columns
if len(numeric_cols) &gt; 20:
    # ■■■■■■■■■■■■■■■■■
    priority = list(found.values()) if found else list(numeric_cols[:15])
    stats_cols = [c for c in priority if c in numeric_cols]
    if len(stats_cols) &lt; 5:
        stats_cols = list(numeric_cols[:15])
else:
    stats_cols = list(numeric_cols)

if stats_cols:
    desc = df_head[stats_cols].describe()
    print(desc.to_string())

# ======================================================================
# 6. ■■■■■■■gzip ■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("■Step 6■■■■■■■■")
print("="*70)

try:
    if is_gzip:
        # gzip: ■■1000■■■■■■■■■■■■■■
        import io
        with gzip.open(DATA_FILE, 'rt') as f:
            sample = f.read(1024*100)  # 100KB
            lines_in_sample = sample.count('\n')

        # ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
        print(f"  gzip ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■...")
        total_lines = 0
        with gzip.open(DATA_FILE, 'rt') as f:
            for _ in f:
                total_lines += 1
        total_lines -= 1  # ■■■■■■■
        print(f"  ■■■: {total_lines:,}")
    else:
        # plain text: wc -l ■■
        total_lines = 0
        with open(DATA_FILE, 'r') as f:
            for _ in f:
                total_lines += 1
        total_lines -= 1
        print(f"  ■■■: {total_lines:,}")
except Exception as e:
    print(f"  ■■■■■■■■■■: {e}")
    total_lines = "■■"
# ======================================================================
# 7. HSC-SSP / KiDS / ■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("■Step 7■■■■■■■■■■■■")
print("="*70)

all_cols_str = ' '.join(df_head.columns).lower()

if 'hsc' in all_cols_str or 'cmodel' in all_cols_str or 'undeblended' in all_cols_str:
    source = "HSC-SSP (Hyper Suprime-Cam)"
```

```
    print(f"  ■■■■■■■■■: {source}")
    print(f"  HSC-SSP PDR3 ■■■■■■■■")
elif 'kids' in all_cols_str or 'theli' in all_cols_str or 'lensfit' in all_cols_str:
    source = "KiDS (Kilo-Degree Survey)"
    print(f"  ■■■■■■■■■: {source}")
elif 'sdss' in all_cols_str:
    source = "SDSS"
    print(f"  ■■■■■■■■■: {source}")
else:
    source = "■■"
    print(f"  ■■■■■■■■■: {source}")
    print(f"  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■:")
    for c in df_head.columns[:30]:
        print(f"    {c}")

# ======================================================================
# 8. ■■■■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("■Step 8■■■■■■■■■■■■■■■■")
print("="*70)

print(f"\n  ■■■■■■■: {source}")
print(f"  ■■■■: {total_lines}")

has_shape = 'e1 / ■■■■1' in found and 'e2 / ■■■■2' in found
has_coords = 'RA / ■■■' in found and 'DEC / ■■■' in found
has_redshift = 'z_photo / ■■■■■■■' in found or 'z_spec / ■■■■■■■' in found
has_weight = 'weight / ■■■' in found

print(f"\n  ■■■■ (e1, e2): {'■■■' if has_shape else '■■■'}")
print(f"  ■■■ (RA, DEC):  {'■■■' if has_coords else '■■■'}")
print(f"  ■■■■■ (z):    {'■■■' if has_redshift else '■■■'}")
print(f"  ■■■ (weight):   {'■■■' if has_weight else '■■■'}")

print(f"\n  ■■■■■■■■:")
if has_shape and has_coords and has_redshift:
    print(f"    (A) ■■■-■■■■■■: ■■■■■■■■■■■■■■■■■■■■")
    print(f"        G.Sigma0 ■■■■■ DeltaSigma(R) ■■■■■■■■■■■■")
    print(f"        g_c = eta.sqrt(a0.G.Sigma0) ■■■■■■■■■")
    print(f"    (B) ■■■■■■■■■: PSZ2/Miyaoka ■■■■■■■■■")
    print(f"        ■■■■■■■■■■■■■■■■T-8/T-14 ■■■■■■")
if has_shape and has_coords and not has_redshift:
    print(f"    (C) ■■■■■■■: 2D ■■■■■■■■■■■■")
    print(f"        ■■■■■■■■■■■■■■■■■■■■■")
if not has_shape:
    print(f"    !!! ■■■■■■■■■■■■■■■■■■■■■■■")
    print(f"        ■■■■■■■■■■■■■■■■■■■■■")

print(f"""
  ■■■■■■■:
    1. ■■■■■■■■■■■■■■■■■■■■■
    2. ■■■■■■■■■■■■■■■■■■■■■■■■■
    3. g_c ■■■■■■■■■■■■■■■■■■■■■■■
""")

print("■■")
```

hsc_footprint_clusters.py

解析目的	デンシティマップのフットプリント可視化。既知クラスター（Miyaoka/PSZ2）との重複チェック。
結果	Miyaoka 16クラスターは全て HSC 外。PSZ2 候補1個のみ。

```
"""
HSC-SSP Wide ■■■■■■■■■■ + ■■■■■■■■
================================================
■■:
  (1) ■■■■■■■■■■■■■■■
  (2) HSC-SSP Wide 5■■■■■■■■■■■■■■■■■■
  (3) ■■■■■■■■■PSZ2/Miyaoka■■■■■■■■■■
  (4) ■■■■■■■■■■931720.csv.gz.1■■■■■■■■■■

■■: uv run --with pandas --with numpy --with matplotlib --with astropy python hsc_footprint_clusters.py
"""

import numpy as np
import pandas as pd
from pathlib import Path
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import sys

# ======================================================================
# 0. ■■■■■■■
# ======================================================================
DENSITY_FILE = Path(r"E:\■■■■■■■■■■\hscssp_pdr2_wide_densitymap.csv")
SHAPE_FILE = Path(r"D:\■■■■■■■\■■■■■■■\■■■■■■\■■■■■■■■\■■■■■\931720.csv.gz.1")

print("="*70)
print("HSC-SSP Wide ■■■■■■■■■■ + ■■■■■■■■")
print("="*70)

# ======================================================================
# 1. ■■■■■■■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 1] ■■■■■■■■■■■■■■■")
print("="*70)

if not DENSITY_FILE.exists():
    print(f"  !!! ■■■■■■■■■■■■■■: {DENSITY_FILE}")
    print(f"  ■■■■■■■■■■■■■■■")
    sys.exit(1)

size_mb = DENSITY_FILE.stat().st_size / (1024*1024)
print(f"  ■■■■■: {DENSITY_FILE}")
print(f"  ■■■■: {size_mb:.1f} MB")

df_density = pd.read_csv(DENSITY_FILE)
print(f"  ■■■: {len(df_density):,}")
print(f"  ■■■■■: {len(df_density.columns)}")
print(f"\n  ■■■■■■:")
print(f"  {'#':&gt;3} {'■■■■■':&lt;30} {'■■■':&lt;12} {'■■null':&gt;8} {'min':&gt;12} {'max':&gt;12}")
print(f"  {'-'*80}")
for i, col in enumerate(df_density.columns):
    dtype = str(df_density[col].dtype)
    non_null = df_density[col].notna().sum()
    if pd.api.types.is_numeric_dtype(df_density[col]):
        mn = f"{df_density[col].min():.4g}"
        mx = f"{df_density[col].max():.4g}"
    else:
        mn = str(df_density[col].iloc[0])[:10]
        mx = str(df_density[col].iloc[-1])[:10]
    print(f"  {i:&gt;3} {col:&lt;30} {dtype:&lt;12} {non_null:&gt;8} {mn:&gt;12} {mx:&gt;12}")

# RA, Dec ■■■■■■■■■
ra_col = None; dec_col = None; density_col = None
for c in df_density.columns:
    cl = c.lower()
    if 'ra' in cl and ra_col is None: ra_col = c
    if 'dec' in cl and dec_col is None: dec_col = c
    if 'dens' in cl or 'count' in cl or 'n_' in cl or 'ngal' in cl:
        density_col = c

if ra_col is None or dec_col is None:
    # ■■■■■■■■■■■■■
    for c in df_density.columns:
        if pd.api.types.is_numeric_dtype(df_density[c]):
            vmin, vmax = df_density[c].min(), df_density[c].max()
            if 0 &lt;= vmin and vmax &lt;= 360 and vmax &gt; 100 and ra_col is None:
                ra_col = c
            elif -90 &lt;= vmin and vmax &lt;= 90 and abs(vmax-vmin) &lt; 50 and dec_col is None:
```

```
                dec_col = c

print(f"\n  ■■■■■:")
print(f"    RA:      {ra_col}")
print(f"    Dec:     {dec_col}")
print(f"    Density: {density_col}")

if ra_col and dec_col:
    ra = df_density[ra_col].values
    dec = df_density[dec_col].values
    print(f"\n  ■■■■:")
    print(f"    RA:  [{ra.min():.3f}, {ra.max():.3f}]")
    print(f"    Dec: [{dec.min():.3f}, {dec.max():.3f}]")

# ======================================================================
# 2. 5■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 2] HSC-SSP Wide 5■■■■■■■■")
print("="*70)

# HSC-SSP PDR2 Wide ■■■5■■■■■
hsc_fields = {
    'XMM-LSS':   {'ra_range': (29, 40),  'dec_range': (-7, -1)},
    'GAMA09H':   {'ra_range': (129, 142), 'dec_range': (-3, 3)},
    'WIDE12H':   {'ra_range': (175, 190), 'dec_range': (-3, 3)},
    'GAMA15H':   {'ra_range': (210, 225), 'dec_range': (-3, 3)},
    'HECTOMAP':  {'ra_range': (240, 250), 'dec_range': (42, 45)},
    'VVDS':      {'ra_range': (333, 345), 'dec_range': (-2, 3)},
}

if ra_col and dec_col:
    print(f"\n  ■■■■■■■■■■■■:")
    for fname, fspec in hsc_fields.items():
        ra_r = fspec['ra_range']
        dec_r = fspec['dec_range']
        mask = (ra &gt;= ra_r[0]) &amp; (ra &lt;= ra_r[1]) &amp; (dec &gt;= dec_r[0]) &amp; (dec &lt;= dec_r[1])
        n = mask.sum()
        if n &gt; 0:
            print(f"    {fname:&lt;12}: {n:&gt;8,} ■■■■  "
                  f"RA=[{ra[mask].min():.1f},{ra[mask].max():.1f}]  "
                  f"Dec=[{dec[mask].min():.1f},{dec[mask].max():.1f}]")

# ======================================================================
# 3. ■■■■■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 3] ■■■■■■■■■■■■■■■■")
print("="*70)

# Miyaoka 2017/2018 ■16■■■■■■X■■■■
# ■■■■■ Table 1 ■■■■■■■■■■■■■■ psz2_v2.py ■■■■■
miyaoka_clusters = [
    # name, RA(deg), Dec(deg), z, ■■
    ("A68",       9.278,  9.157, 0.255, "Miyaoka2017"),
    ("A209",     22.969, -13.609, 0.206, "Miyaoka2017"),
    ("A267",     28.175,   1.006, 0.230, "Miyaoka2017"),
    ("A383",     42.014,  -3.529, 0.187, "Miyaoka2017"),
    ("A521",     73.529,  -10.224, 0.247, "Miyaoka2017"),
    ("A586",    113.084,  31.633, 0.171, "Miyaoka2017"),
    ("A611",    120.237,  36.057, 0.288, "Miyaoka2017"),
    ("A697",    130.739,  36.365, 0.282, "Miyaoka2017"),
    ("A963",    154.264,  39.047, 0.206, "Miyaoka2017"),
    ("A1689",   197.873,  -1.341, 0.183, "Miyaoka2017"),
    ("A1835",   210.259,   2.878, 0.253, "Miyaoka2017"),
    ("A2204",   248.196,   5.576, 0.151, "Miyaoka2017"),
    ("A2261",   260.613,  32.133, 0.224, "Miyaoka2017"),
    ("RXJ1347", 206.878,  -11.753, 0.451, "Miyaoka2017"),
    ("MS1358",  209.960,  62.515, 0.329, "Miyaoka2017"),
    ("ZW3146",  155.916,  4.186, 0.291, "Miyaoka2017"),
]

# PSZ2 ■■■■■■■■■■■■■■■■■■■
psz2_equatorial = [
    # name, RA, Dec, z ■■■■■■■■■■■■■■■
    ("PSZ2_G186.37+37.26",  135.0, -1.5, 0.23, "PSZ2"),
    ("PSZ2_G212.44+63.19",  185.0, -2.0, 0.19, "PSZ2"),
]

all_clusters = miyaoka_clusters + psz2_equatorial
if ra_col and dec_col:
    pixel_size_deg = 1.5 / 60.0  # 1.5 arcmin in degrees

    print(f"  HSC-SSP ■■■■■■■■■■■■■■■:")
    print(f"  {'■■':&lt;15} {'RA':&gt;8} {'Dec':&gt;8} {'z':&gt;6} {'■■■■■■■■':&gt;15} {'■■':&lt;15}")
    print(f"  {'-'*72}")
    in_footprint = []
```

```
    for cl in all_clusters:
        name, ra_cl, dec_cl, z_cl, source = cl

        # ■■■■■■■■■■■■■■■■■■■■■■■
        dist = np.sqrt(((ra - ra_cl)*np.cos(np.radians(dec_cl)))**2 + (dec - dec_cl)**2)
        min_dist = dist.min()
        is_in = min_dist &lt; 3 * pixel_size_deg  # 3■■■■■■■

        status = "YES" if is_in else "no"
        print(f"  {name:&lt;15} {ra_cl:&gt;8.3f} {dec_cl:&gt;8.3f} {z_cl:&gt;6.3f} {status:&gt;15} {source:&lt;15}")

        if is_in:
            in_footprint.append(cl)

    print(f"\n  ■■■■■■■■■■■■■■■■: {len(in_footprint)} / {len(all_clusters)}")

    # ■■■■■■■■■■Y3■■■■■■■■■■
    # Y3 ■■■■■■: Dec -5.9 ~ 0 ■inspect■■■■■■
    print(f"\n  ■■■■■■■■■■Y3, Dec=-5.9~0■■■■■■:")
    y3_clusters = []
    for cl in in_footprint:
        name, ra_cl, dec_cl, z_cl, source = cl
        if -5.9 &lt;= dec_cl &lt;= 0:
            print(f"    {name}: RA={ra_cl:.3f}, Dec={dec_cl:.3f}, z={z_cl:.3f} -&gt; Y3 ■■ YES")
            y3_clusters.append(cl)
        else:
            print(f"    {name}: Dec={dec_cl:.3f} -&gt; Y3 ■■■")

    print(f"\n  Y3 ■■■■■■■■■■■■■■■■■■■■■■: {len(y3_clusters)}")

else:
    print("  RA/Dec ■■■■■■■■■■■■")
    in_footprint = []
    y3_clusters = []

# ======================================================================
# 4. ■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 4] ■■■■■■■■■■■■")
print("="*70)

if ra_col and dec_col:
    fig, axes = plt.subplots(1, 2, figsize=(18, 6))

    # (a) ■■■■■■■■■■
    ax = axes[0]
    # RA ■ -180~180 ■■■■■■■■■■
    ra_plot = ra.copy()
    ra_plot[ra_plot &gt; 180] -= 360

    if density_col and density_col in df_density.columns:
        dens = df_density[density_col].values
        sc = ax.scatter(ra_plot, dec, c=np.log10(dens+1), s=0.1, alpha=0.3,
                        cmap='viridis', rasterized=True)
        plt.colorbar(sc, ax=ax, label='log10(density+1)', shrink=0.8)
    else:
        ax.scatter(ra_plot, dec, s=0.1, alpha=0.3, c='steelblue', rasterized=True)

    # ■■■■■■■■■■■■■■■■
    for cl in all_clusters:
        name, ra_cl, dec_cl, z_cl, source = cl
        ra_cl_plot = ra_cl - 360 if ra_cl &gt; 180 else ra_cl
        is_in = cl in in_footprint
        color = 'red' if is_in else 'gray'
        marker = '*' if is_in else 'x'
        ms = 15 if is_in else 8
        ax.plot(ra_cl_plot, dec_cl, marker, color=color, ms=ms, mew=1.5)
        if is_in:
            ax.annotate(name, (ra_cl_plot, dec_cl), fontsize=6,
                        xytext=(3, 3), textcoords='offset points', color='red')

    ax.set_xlabel('RA [deg]')
    ax.set_ylabel('Dec [deg]')
    ax.set_title(f'HSC-SSP Wide footprint ({len(df_density):,} pixels)')
    ax.set_xlim(ra_plot.min()-5, ra_plot.max()+5)
    ax.invert_xaxis()

    # (b) Y3 ■■■■■■■■■■■■■■■■
    ax = axes[1]
    # Dec: -5.9~0 ■■■■■■
    mask_y3 = (dec &gt;= -6.5) &amp; (dec &lt;= 1.0)
    if mask_y3.sum() &gt; 0:
        ax.scatter(ra_plot[mask_y3], dec[mask_y3], s=0.5, alpha=0.3,
                   c='steelblue', rasterized=True)

    # Y3 Dec ■■■■■
```

```
    ax.axhline(-5.9, color='red', ls='--', alpha=0.5, label='Y3 Dec range')
    ax.axhline(0, color='red', ls='--', alpha=0.5)

    # Y3 ■■■■■■■■
    for cl in y3_clusters:
        name, ra_cl, dec_cl, z_cl, source = cl
        ra_cl_plot = ra_cl - 360 if ra_cl &gt; 180 else ra_cl
        ax.plot(ra_cl_plot, dec_cl, '*', color='red', ms=15, mew=1.5)
        ax.annotate(name, (ra_cl_plot, dec_cl), fontsize=8, fontweight='bold',
                    xytext=(5, 5), textcoords='offset points', color='red')

    ax.set_xlabel('RA [deg]')
    ax.set_ylabel('Dec [deg]')
    ax.set_title(f'Y3 shape catalog overlap (Dec: -5.9 ~ 0)')
    ax.set_ylim(-7, 2)
    ax.invert_xaxis()
    ax.legend(fontsize=8)

    plt.tight_layout()
    plt.savefig('hsc_footprint_clusters.png', dpi=150, bbox_inches='tight')
    print("  -&gt; hsc_footprint_clusters.png")

# ======================================================================
# 5. ■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 5] ■■■■■")
print("="*70)

print(f"""
  HSC-SSP Wide ■■■■■■■■■:
    ■■■■■■: {len(df_density):,}
    ■■■■■■■■: 1.5 arcmin
    ■■■■■■■: 5 (■■)

  ■■■■■■■■:
    ■■: {len(all_clusters)}
    HSC-SSP ■■■■■■■■■: {len(in_footprint)}
    Y3 ■■■■■■■■■■■: {len(y3_clusters)}
""")

if len(y3_clusters) &gt; 0:
    print(f"  -&gt; Y3 ■■■■■■■■■■■■:")
    for cl in y3_clusters:
        name, ra_cl, dec_cl, z_cl, source = cl
        print(f"     {name}: RA={ra_cl:.3f}, Dec={dec_cl:.3f}, z={z_cl:.3f}")
    print(f"""
  ■■■■■■■■:
    1. hsc_y3_cluster_lensing.py ■ known_clusters ■■■■■■■
    2. ■■■■■■■■■■■■ gamma_t(R) ■■■■
    3. NFW / ■■■■■■■
""")
elif len(in_footprint) &gt; 0:
    print(f"  -&gt; HSC-SSP ■■■■■■■■■■■■ Y3 ■■■■■■■■■■■■■■■■")
    print(f"     ■■■■■■■■■■■■■■■■■■■■■")
    print(f"     ■■■■■■■■■■■■■PDR2/PDR3■■■■■■■■")
else:
    print(f"  -&gt; ■■■■■■■■■■■■■■■■■■")
    print(f"     ■■-■■■■■■■Stage 2■■■■■■■■■■")
    print(f"""
  Stage 2■■■■-■■■■■■■■■■:
    1. HSC-SSP ■■■■■■■■■■ SDSS spectroscopic ■■■■■■
       ■v_flat ■■■■ sigma_v ■■■■■■■■■■
    2. HSC Y3 ■■■■■■■■■■■■■■■■■
    3. ■■■■■■■ G.Sigma0 ■■■■■■
    4. ■■■■■ DeltaSigma(R) ■■■■
    5. g_c = eta.sqrt(a0.G.Sigma0) vs g_c = a0 ■■■■
""")

print("\n■■■")
```

hsc_cluster_screening.py

解析目的	デンシティマップの過剰密度（sgm3_r10）からクラスター候補をスクリーニング。
結果	122候補検出（4sigma以上）。Y3重複38個。cl1（sgm=6.4）が最有力。

```
"""
HSC-SSP ■■■■■■■■■■■■■■■■■■■■■■■■■■■■
============================================================
■■:
  ■■■■ (sgm, dlt) ■■■■■■■■■■■■■■■■■■■■■■■■
  Y3 ■■■■■■■■■ (Dec: -5.9~0) ■■■■■■■■■
  ■■■■■■■■■■■■■■■■■

■■:
  (1) ■■■■■■■■■■■■■■■■sgm &gt;= ■■■
  (2) ■■■■■■■■■■■■■■■■■■■■■■■■■
  (3) ■■■■■■■■■■■■■
  (4) Y3 ■■■■■■■
  (5) ■■■■■■■■■■■

■■: uv run --with pandas --with numpy --with scipy --with matplotlib python hsc_cluster_screening.py
"""

import numpy as np
import pandas as pd
from scipy import ndimage
from pathlib import Path
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================================
# 0. ■■
# ============================================================================
DENSITY_FILE = Path(r"E:\■■■■■■■■■■\hscssp_pdr2_wide_densitymap.csv")
Y3_DEC_MIN = -5.9
Y3_DEC_MAX = 0.0

# ■■■■■■■■
SGM_THRESHOLD = 4.0    # ■■■■■■■■■■■■■■4σ■■■■
MIN_PIXELS = 3         # ■■■■■■■■■■■■■■
GROUPING_RADIUS_ARCMIN = 5.0  # ■■■■■■■■■■■■■■

print("="*70)
print("HSC-SSP ■■■■■■■■■ ■■■■■■■■■■■■■■■■")
print("="*70)

# ============================================================================
# 1. ■■■■■■■■■■■■■■
# ============================================================================
print("\n"+"="*70)
print("[Step 1] ■■■■■■■■■■■■■■■■")
print("="*70)

df = pd.read_csv(DENSITY_FILE)
print(f"  ■■: {len(df):,}")
print(f"  ■■■■: {len(df.columns)}")

# ■■■■■■■■
ra_col = [c for c in df.columns if c.lower() == 'ra'][0] if any(c.lower()=='ra' for c in df.columns) else None
dec_col = [c for c in df.columns if c.lower() == 'dec'][0] if any(c.lower()=='dec' for c in df.columns) else None

if ra_col is None:
    for c in df.columns:
        if 'ra' in c.lower() and pd.api.types.is_numeric_dtype(df[c]):
            ra_col = c; break
if dec_col is None:
    for c in df.columns:
        if 'dec' in c.lower() and pd.api.types.is_numeric_dtype(df[c]):
            dec_col = c; break

print(f"  RA: {ra_col}, Dec: {dec_col}")

# sgm (■■■■■■■■) ■■■■■■■
sgm_cols = sorted([c for c in df.columns if c.startswith('sgm')])
dlt_cols = sorted([c for c in df.columns if c.startswith('dlt')])
nd_cols = sorted([c for c in df.columns if c.startswith('nd')])

print(f"  sgm ■■■■: {len(sgm_cols)}")
print(f"  dlt ■■■■: {len(dlt_cols)}")
print(f"  nd ■■■■: {len(nd_cols)}")

if sgm_cols:
    print(f"\n  sgm ■■■■■:")
    for c in sgm_cols[:20]:
```

```
        valid = df[c].notna() &amp;amp; np.isfinite(df[c])
        if valid.sum() &gt; 0:
            vals = df.loc[valid, c]
            print(f"    {c:&lt;25} mean={vals.mean():&gt;8.2f}  std={vals.std():&gt;8.2f}  "
                  f"max={vals.max():&gt;8.1f}  N(&gt;{SGM_THRESHOLD}sigma)={int((vals&gt;SGM_THRESHOLD).sum()):&gt;6}")

# ================================================================
# 2. ■■■■■■■■■■■■■■
# ================================================================
print("\n"+"="*70)
print("[Step 2] ■■■■■■■■■■■■■■■")
print("="*70)

# ■ sgm ■■■■■■■■■■■■■■■■■■
# ■■■■■■■■■■ r30■30'■■■■■■ r10■10'■■■■■■■
# z ■■■■■■■z=0.2-0.5 ■■■N=3-5 ■■■■■■■■

best_col = None
best_n_peaks = 0

print(f"\n  {'■■■■':&lt;25} {'N(&gt;{SGM_THRESHOLD}sigma)':&gt;12} {'max sigma':&gt;10} {'■■':&gt;6}")
print(f"  {'-'*58}")

for c in sgm_cols:
    valid = df[c].notna() &amp;amp; np.isfinite(df[c])
    if valid.sum() == 0:
        continue
    vals = df.loc[valid, c]
    n_high = int((vals &gt; SGM_THRESHOLD).sum())
    mx = vals.max()

    # r10 ■■■■■■ z ■■■■■
    is_r10 = 'r10' in c or 'c10' in c
    is_mid_z = any(f'{n}' in c for n in [3, 4, 5])
    suitability = ""
    if is_r10 and is_mid_z:
        suitability = "[**]"
    elif is_r10:
        suitability = "[*]"
    elif is_mid_z:
        suitability = "[*]"

    if n_high &gt; 10:
        print(f"  {c:&lt;25} {n_high:&gt;12} {mx:&gt;10.1f} {suitability:&gt;6}")

    if n_high &gt; best_n_peaks and (is_r10 or is_mid_z):
        best_n_peaks = n_high
        best_col = c

# ■■■■■■■■■■■■■■■■■■■■■■■■■
if best_col is None:
    for c in sgm_cols:
        valid = df[c].notna() &amp;amp; np.isfinite(df[c])
        if valid.sum() == 0: continue
        n_high = int((df.loc[valid, c] &gt; SGM_THRESHOLD).sum())
        if n_high &gt; best_n_peaks:
            best_n_peaks = n_high
            best_col = c

print(f"\n  ■■■■■: {best_col} (N={best_n_peaks} peaks above {SGM_THRESHOLD}sigma)")

# ■■ z ■■■■■■■■■■robustness■■■
# ■ sgm ■■■■■■■■■■■
sgm_max = df[sgm_cols].max(axis=1) if sgm_cols else pd.Series(np.zeros(len(df)))
print(f"  ■ sgm ■■■■■■: max={sgm_max.max():.1f}, N(&gt;{SGM_THRESHOLD})={int((sgm_max&gt;SGM_THRESHOLD).sum())}")

# ================================================================
# 3. ■■■■■
# ================================================================
print("\n"+"="*70)
print("[Step 3] ■■■■■■")
print("="*70)

# ■■■■■■■■■■■
if best_col:
    sgm_use = df[best_col].fillna(0).values
else:
    sgm_use = sgm_max.fillna(0).values

ra_vals = df[ra_col].values
dec_vals = df[dec_col].values

# RA ■■■■■■■■360■■■■■■■■
ra_vals_wrapped = ra_vals.copy()
ra_vals_wrapped[ra_vals_wrapped &gt; 360] -= 360

mask_peak = sgm_use &gt; SGM_THRESHOLD
```

```
peak_indices = np.where(mask_peak)[0]
print(f"  {SGM_THRESHOLD}sigma ■■■■■■■■: {len(peak_indices)}")

if len(peak_indices) == 0:
    # ■■■■■■■■■■
    for try_thresh in [3.5, 3.0, 2.5]:
        mask_peak = sgm_use &gt; try_thresh
        peak_indices = np.where(mask_peak)[0]
        if len(peak_indices) &gt; 10:
            SGM_THRESHOLD = try_thresh
            print(f"  ■■■ {try_thresh}sigma ■■■■■■■: {len(peak_indices)} ■■■■■")
            break

if len(peak_indices) == 0:
    # sgm_max ■■■■■
    print(f"  best_col ■■■■■■■■sgm_max ■■■■■...")
    sgm_use = sgm_max.fillna(0).values
    for try_thresh in [4.0, 3.5, 3.0, 2.5]:
        mask_peak = sgm_use &gt; try_thresh
        peak_indices = np.where(mask_peak)[0]
        if len(peak_indices) &gt; 5:
            SGM_THRESHOLD = try_thresh
            print(f"  sgm_max, ■■={try_thresh}sigma: {len(peak_indices)} ■■■■■")
            break

# ======================================================================
# 4. ■■■■■■■■■■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 4] ■■■■■■■■■■■■")
print("="*70)

if len(peak_indices) &gt; 0:
    peak_ra = ra_vals_wrapped[peak_indices]
    peak_dec = dec_vals[peak_indices]
    peak_sgm = sgm_use[peak_indices]

    # ■■■■■■■■■■■■■■■■■■■■■■
    group_radius_deg = GROUPING_RADIUS_ARCMIN / 60.0

    visited = np.zeros(len(peak_indices), dtype=bool)
    clusters = []

    # ■■■■■■■■■■■■■
    order = np.argsort(-peak_sgm)

    for seed_idx in order:
        if visited[seed_idx]:
            continue

        # seed ■■■■■■■■■■■■
        dra = (peak_ra - peak_ra[seed_idx]) * np.cos(np.radians(peak_dec[seed_idx]))
        ddec = peak_dec - peak_dec[seed_idx]
        dist = np.sqrt(dra**2 + ddec**2)

        members = np.where((dist &lt; group_radius_deg) &amp;amp; (~visited))[0]

        if len(members) &gt;= MIN_PIXELS:
            # ■■■■■■■■ = ■■■■■■■■■■■
            w = peak_sgm[members]
            ra_center = np.average(peak_ra[members], weights=w)
            dec_center = np.average(peak_dec[members], weights=w)
            sgm_peak = peak_sgm[members].max()
            sgm_mean = np.average(peak_sgm[members], weights=w)

            clusters.append({
                'ra': ra_center,
                'dec': dec_center,
                'sgm_peak': sgm_peak,
                'sgm_mean': sgm_mean,
                'n_pixels': len(members),
                'radius_arcmin': dist[members].max() * 60,
            })
            visited[members] = True
        else:
            visited[seed_idx] = True

    print(f"  ■■■■■■■■■■■■: {len(clusters)}")

    # DataFrame■■■■
    df_cl = pd.DataFrame(clusters)
    df_cl = df_cl.sort_values('sgm_peak', ascending=False).reset_index(drop=True)

    # z ■■■■■ sgm ■■■■■photo-z ■■■■■
    # ■■■■■■■■■■■■ sgm ■■■■■■■■
    if len(sgm_cols) &gt; 1:
        for cl_idx in range(min(len(df_cl), 30)):
```

```
            ra_c = df_cl.loc[cl_idx, 'ra']
            dec_c = df_cl.loc[cl_idx, 'dec']
            # ■■■■■■■■
            dra = (ra_vals_wrapped - ra_c) * np.cos(np.radians(dec_c))
            ddec = dec_vals - dec_c
            dist = np.sqrt(dra**2 + ddec**2)
            nearest = np.argmin(dist)

            # ■ z ■■■ sgm ■
            z_sgm = {}
            for sc in sgm_cols:
                val = df.iloc[nearest][sc]
                if pd.notna(val) and np.isfinite(val):
                    z_sgm[sc] = val

            # ■■■■■■ z ■■ = photo-z ■■
            if z_sgm:
                best_zbin = max(z_sgm, key=z_sgm.get)
                df_cl.loc[cl_idx, 'best_zbin'] = best_zbin
                df_cl.loc[cl_idx, 'best_zbin_sgm'] = z_sgm[best_zbin]

else:
    print("  ■■■■■■■■■■■■■")
    df_cl = pd.DataFrame()

# ======================================================================
# 5. Y3 ■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 5] Y3 ■■■■■■■■■■■")
print("="*70)

if len(df_cl) &gt; 0:
    # Y3 ■■■■■■■
    df_cl['in_y3'] = (df_cl['dec'] &gt;= Y3_DEC_MIN) &amp; (df_cl['dec'] &lt;= Y3_DEC_MAX)
    n_y3 = df_cl['in_y3'].sum()
    print(f"  ■■■■: {len(df_cl)}")
    print(f"  Y3 ■■ (Dec {Y3_DEC_MIN}~{Y3_DEC_MAX}): {n_y3}")

    # ■■■■■■ = sgm_peak * sqrt(n_pixels)
    df_cl['quality'] = df_cl['sgm_peak'] * np.sqrt(df_cl['n_pixels'])

    # ■■■■■■■■
    print(f"\n  === ■■■■ ■■20 ===")
    print(f"  {'#':&gt;3} {'RA':&gt;9} {'Dec':&gt;8} {'sgm_peak':&gt;9} {'n_pix':&gt;6} {'quality':&gt;8} {'Y3':&gt;4} {'best_zbin':&lt;25}
    print(f"  {'-'*78}")
    for i in range(min(20, len(df_cl))):
        row = df_cl.iloc[i]
        y3 = "Y" if row.get('in_y3', False) else ""
        zbin = str(row.get('best_zbin', ''))[:24]
        print(f"  {i+1:&gt;3} {row['ra']:&gt;9.3f} {row['dec']:&gt;8.3f} {row['sgm_peak']:&gt;9.1f} "
              f"{int(row['n_pixels']):&gt;6} {row['quality']:&gt;8.1f} {y3:&gt;4} {zbin:&lt;25}")

    # Y3 ■■■■■
    df_y3 = df_cl[df_cl['in_y3']].copy().reset_index(drop=True)
    if len(df_y3) &gt; 0:
        print(f"\n  === Y3 ■■■■■ ■■20 ===")
        print(f"  {'#':&gt;3} {'RA':&gt;9} {'Dec':&gt;8} {'sgm_peak':&gt;9} {'n_pix':&gt;6} {'quality':&gt;8} {'best_zbin':&lt;25}")
        print(f"  {'-'*72}")
        for i in range(min(20, len(df_y3))):
            row = df_y3.iloc[i]
            zbin = str(row.get('best_zbin', ''))[:24]
            print(f"  {i+1:&gt;3} {row['ra']:&gt;9.3f} {row['dec']:&gt;8.3f} {row['sgm_peak']:&gt;9.1f} "
                  f"{int(row['n_pixels']):&gt;6} {row['quality']:&gt;8.1f} {zbin:&lt;25}")
        # CSV ■■
        df_y3.to_csv('hsc_cluster_candidates_y3.csv', index=False)
        print(f"\n  -&gt; hsc_cluster_candidates_y3.csv ({len(df_y3)} ■■)")

    # ■■■CSV
    df_cl.to_csv('hsc_cluster_candidates_all.csv', index=False)
    print(f"  -&gt; hsc_cluster_candidates_all.csv ({len(df_cl)} ■■)")

# ======================================================================
# 6. ■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 6] ■■■■■■■")
print("="*70)

if len(df_cl) &gt; 0:
    fig, axes = plt.subplots(2, 2, figsize=(16, 12))

    # (a) ■■■■■■■■■■■■■■■■■
    ax = axes[0, 0]
    ax.scatter(ra_vals_wrapped, dec_vals, s=0.05, alpha=0.1, c='lightgray', rasterized=True)
    # ■■■
    sc = ax.scatter(df_cl['ra'], df_cl['dec'], c=df_cl['sgm_peak'],
```

```
                  s=df_cl['n_pixels']*5, cmap='hot_r', alpha=0.7,
                  edgecolors='black', linewidths=0.5, vmin=SGM_THRESHOLD)
    plt.colorbar(sc, ax=ax, label='Peak sigma', shrink=0.8)
    # Y3 ■■
    ax.axhline(Y3_DEC_MIN, color='blue', ls='--', alpha=0.5)
    ax.axhline(Y3_DEC_MAX, color='blue', ls='--', alpha=0.5)
    ax.fill_between([ra_vals_wrapped.min(), ra_vals_wrapped.max()],
                    Y3_DEC_MIN, Y3_DEC_MAX, alpha=0.05, color='blue')
    ax.set_xlabel('RA [deg]')
    ax.set_ylabel('Dec [deg]')
    ax.set_title(f'Cluster candidates (N={len(df_cl)}, threshold={SGM_THRESHOLD}sigma)')
    ax.invert_xaxis()

    # (b) ■■■■■■
    ax = axes[0, 1]
    ax.hist(df_cl['sgm_peak'], bins=30, color='steelblue', alpha=0.7, edgecolor='white')
    ax.axvline(SGM_THRESHOLD, color='red', ls='--', label=f'Threshold={SGM_THRESHOLD}')
    ax.set_xlabel('Peak sigma')
    ax.set_ylabel('Count')
    ax.set_title('Peak sigma distribution')
    ax.legend()

    # (c) Y3 ■■■■■■■■■
    ax = axes[1, 0]
    mask_dec = (dec_vals &gt;= Y3_DEC_MIN-1) &amp; (dec_vals &lt;= Y3_DEC_MAX+1)
    if mask_dec.sum() &gt; 0:
        ax.scatter(ra_vals_wrapped[mask_dec], dec_vals[mask_dec],
                   s=0.1, alpha=0.2, c='lightgray', rasterized=True)
    if len(df_y3) &gt; 0:
        sc2 = ax.scatter(df_y3['ra'], df_y3['dec'], c=df_y3['sgm_peak'],
                         s=df_y3['n_pixels']*10, cmap='hot_r', alpha=0.8,
                         edgecolors='black', linewidths=1, vmin=SGM_THRESHOLD)
        plt.colorbar(sc2, ax=ax, label='Peak sigma', shrink=0.8)
        for i in range(min(10, len(df_y3))):
            row = df_y3.iloc[i]
            ax.annotate(f"#{i+1}", (row['ra'], row['dec']),
                        fontsize=8, fontweight='bold', color='red',
                        xytext=(3,3), textcoords='offset points')
    ax.axhline(Y3_DEC_MIN, color='blue', ls='--', alpha=0.7, label='Y3 range')
    ax.axhline(Y3_DEC_MAX, color='blue', ls='--', alpha=0.7)
    ax.set_xlabel('RA [deg]')
    ax.set_ylabel('Dec [deg]')
    ax.set_title(f'Y3 overlap candidates (N={len(df_y3)})')
    ax.set_ylim(Y3_DEC_MIN-1, Y3_DEC_MAX+1)
    ax.invert_xaxis()
    ax.legend(fontsize=8)

    # (d) ■■■■■ vs ■■■■■
    ax = axes[1, 1]
    if len(df_y3) &gt; 0:
        ax.scatter(df_y3['n_pixels'], df_y3['sgm_peak'],
                   s=50, c='coral', alpha=0.7, edgecolors='black', label='Y3 overlap')
    ax.scatter(df_cl[~df_cl['in_y3']]['n_pixels'], df_cl[~df_cl['in_y3']]['sgm_peak'],
               s=20, c='gray', alpha=0.3, label='Outside Y3')
    ax.set_xlabel('N pixels')
    ax.set_ylabel('Peak sigma')
    ax.set_title('Candidate quality')
    ax.legend(fontsize=8)

    plt.tight_layout()
    plt.savefig('hsc_cluster_screening.png', dpi=150, bbox_inches='tight')
    print("  -&gt; hsc_cluster_screening.png")

# ======================================================================
# 7. ■■■■
# ======================================================================
print("\n"+"="*70)
print("[■■■■■]")
print("="*70)

if len(df_cl) &gt; 0:
    print(f"""
  ■■■■■■■■■■■■■■■■■■■■■■:
    ■■■■■■: {best_col if best_col else 'sgm_max (■■■■■■■)'}
    ■■: {SGM_THRESHOLD} sigma
    ■■■■■■: {len(df_cl)}
    Y3 ■■■■■■: {len(df_y3) if 'df_y3' in dir() else 0}
""")

    if 'df_y3' in dir() and len(df_y3) &gt; 0:
        print(f"  Y3 ■■■■■■■■■■■■■■■■:")
        for i in range(min(10, len(df_y3))):
            row = df_y3.iloc[i]
            print(f"    #{i+1}: RA={row['ra']:.3f}, Dec={row['dec']:.3f}, "
                  f"sgm={row['sgm_peak']:.1f}, n_pix={int(row['n_pixels'])}")

        print(f"""
```

```
  ■■■■■■■:
    1. ■■■■■■■■■ hsc_y3_cluster_lensing.py ■ known_clusters ■■■
    2. z ■ best_zbin ■■■■■■■■■■: z ■■■■■■■■■■■■■■■
    3. ■■■■■■■■■■■ gamma_t(R) ■■■
    4. NFW / ■■■■■■■
""")
    else:
        print(f"""
  Y3 ■■■■■■■■■■-■■■■■■ (Stage 2) ■■■■■■■■■■
""")
else:
    print("  ■■■■■■■■■■■■■■■■■■■■■■■■■")

print("■■")
```

hsc_y3_cluster_lensing.py

解析目的	クラスター周辺の接線剪断プロファイル測定パイプライン（デモ版）。
結果	クラスター座標入力用のテンプレート。

```
"""
HSC Y3 ■■■■■■■■■ Stage 1: ■■■■■■■■■
================================================
■■: HSC Y3 ■■■■■■■■■■■■■■■■■■■■■■■■■■■■
      ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

■■:
  (1) HSC Y3 ■■■■■■■■■■■■■■■■■■
  (2) ■■■■■■■■■■■■■■■■■■
  (3) ■■■■■ gamma_t(R) ■■■■
  (4) NFW vs ■■■■■ ■■■■
  (5) g_c(cluster) ■■■■

■■: uv run --with pandas --with numpy --with scipy --with matplotlib --with astropy python hsc_y3_cluster_lensing.py

■■: 9.6GB ■■■■■■■■■■■■■■■■■■■■■■■■■
      ■■■■■■■■■■■■■■■■■■■■■■■■
"""

import numpy as np
import pandas as pd
from scipy import stats
from pathlib import Path
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import font_manager as _fm
for _fp in ['/usr/share/fonts/opentype/ipafont-gothic/ipag.ttf',
            '/usr/share/fonts/opentype/ipafont-gothic/ipagp.ttf']:
    try: _fm.fontManager.addfont(_fp)
    except: pass
try:
    plt.rcParams['font.family'] = 'IPAGothic'
except:
    pass
plt.rcParams['axes.unicode_minus'] = False

# ============================================================================
# 0. ■■
# ============================================================================
DATA_FILE = Path(r"D:\■■■■■■■\■■■■■■■\■■■■■■\■■■■■■■■\■■■■■\931720.csv.gz.1")

# HSC Y3 ■■■■■■■■ (inspect■■■■■)
HSC_RA_RANGE = (0.06, 360.0)
HSC_DEC_RANGE = (-5.9, 0.0)

# ■■■■■
c_light = 2.998e5    # km/s
G_const = 4.301e-3   # (km/s)^2 pc / Msun → ■■■■■■■
a0 = 1.2e-10         # m/s^2
H0 = 70.0            # km/s/Mpc
Omega_m = 0.3
Omega_L = 0.7

print("="*70)
print("HSC Y3 ■■■■■■■■■ Stage 1: ■■■■■■■■■")
print("="*70)

# ============================================================================
# 1. ■■■■■■■■■■■■HSC Y3 ■■■■■■■■■■
# ============================================================================
print("\n"+"="*70)
print("[Step 1] HSC Y3 ■■■■■■■■■■■■■■■■■■")
print("="*70)

# Miyaoka 2017/2018 ■16■■■■■■ + PSZ2 ■■■■■■■■
# Dec: -5.9 ~ 0 ■■■■■■■■■■
# ■■■■ J2000

# ■■■■■■■■■■■■■■■■■■■■■■
# ■■■■■■■■■■■ PSZ2 ■■■■■■■■■■■■■■■■
known_clusters = [
    # name, RA, Dec, z_cluster, r500_arcmin (■■), ■■
    # --- Miyaoka 2017/2018 ■■■■■■■■■■■■■■■■---
    # ■■■■■■ psz2_v2.py ■■■■■■■■■■■■■■■■■■■■■■■
    # --- HSC Y3 ■■■■■■■■ Dec: -5.9 ~ 0 ■■■■■■■■ ---
    # ■■■■■■
]

# HSC Y3 ■■■■ GAMA09H, GAMA15H, WIDE12H, XMM-LSS, VVDS ■■■■■■■■
```

```
# XMM-LSS ■■■■■ (RA~34-37, Dec~-5 to -3) ■■■■■■■■■■■■■■

# PSZ2 ■■■■■■■ Dec: -5.9~0 ■■■■■■■■■■■■
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print(f"  HSC Y3 ■■■■■■■■: RA={HSC_RA_RANGE}, Dec={HSC_DEC_RANGE}")
print(f"  ■■■■■■■■■: {len(known_clusters)}")

if len(known_clusters) == 0:
    print(f"""
  !!! ■■■■■■■■■■■■■■■■
  ■■■■■■■■■■■■■■■■■■■■■■:

  ■■1: Miyaoka 2017/2018 ■■■■■■■■■■■■
    → psz2_v2.py ■■■■ RA, Dec, z ■■■■
    → Dec ■ -5.9 ~ 0 ■■■■■■■■■■■■

  ■■2: PSZ2 ■■■■■■■ HSC Y3 ■■■■■■■■
    → Vizier ■ PSZ2 ■■■■■■■■
    → Dec: -5.9 ~ 0 ■■■■■■

  ■■3: HSC Y3 ■■■■■■■■■■■■■■■■■■■
    → Miyatake+2019, Murata+2019 ■

  → ■■■■■■■■■ known_clusters ■■■■■■■■■■■■■
""")

    # --- ■■■■: ■■■■■■■■■■■■■■■ ---
    print("  ■■■■■■: ■■■■■■■■■■■■■■■■■■■■")
    known_clusters = [
        ("DEMO_CL1", 34.5, -4.5, 0.30, 5.0, "demo"),
        ("DEMO_CL2", 150.0, -2.0, 0.25, 6.0, "demo"),
    ]

print(f"\n  ■■■■■■■■:")
print(f"  {'■■':&lt;20} {'RA':&gt;8} {'Dec':&gt;8} {'z':&gt;6} {'r500[arcmin]':&gt;12}")
print(f"  {'-'*58}")
for cl in known_clusters:
    name, ra, dec, z, r500, src = cl
    print(f"  {name:&lt;20} {ra:&gt;8.2f} {dec:&gt;8.2f} {z:&gt;6.3f} {r500:&gt;12.1f}")

# ======================================================================
# 2. ■■■■■■■■■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 2] ■■■■■■■■■■■■■■■■■")
print("="*70)

# HSC Y3 ■■■■■■■inspect ■■■■■■
COL_RA = 'i_ra'
COL_DEC = 'i_dec'
COL_E1 = 'i_hsmshaperegauss_e1'
COL_E2 = 'i_hsmshaperegauss_e2'
COL_W = 'derived_weight'
COL_M = 'shear_bias_m'
COL_C1 = 'shear_bias_c1'
COL_C2 = 'shear_bias_c2'
COL_MAG = 'i_apertureflux_10_mag'
COL_ZBIN = 'hsc_y3_zbin'
COL_RES = 'resolution'
COL_BMASK = 'b_mode_mask'

# ■■■■■■■■■■■■■■■■■■■■■■
EXTRACT_RADIUS_DEG = 0.5  # 30 arcmin = 0.5 deg

def extract_sources_around_cluster(data_file, ra_cl, dec_cl, radius_deg,
                                   chunksize=500000):
    """■■■■■■■■■■■■■■■■■■■■■■■■■"""
    sources = []
    total_rows = 0

    # RA ■■■■■■■■
    ra_min = ra_cl - radius_deg / np.cos(np.radians(dec_cl))
    ra_max = ra_cl + radius_deg / np.cos(np.radians(dec_cl))
    dec_min = dec_cl - radius_deg
    dec_max = dec_cl + radius_deg

    for chunk in pd.read_csv(data_file, chunksize=chunksize):
        total_rows += len(chunk)

        # ■■■■■■■■■■■■■■■
        ra_col = COL_RA if COL_RA in chunk.columns else chunk.columns[1]
        dec_col = COL_DEC if COL_DEC in chunk.columns else chunk.columns[2]

        mask = (chunk[ra_col] &gt;= ra_min) &amp; (chunk[ra_col] &lt;= ra_max) &amp; \
               (chunk[dec_col] &gt;= dec_min) &amp; (chunk[dec_col] &lt;= dec_max)

        if mask.sum() &gt; 0:
```

```
            sources.append(chunk[mask].copy())

        # ■■■■
        if total_rows % 5000000 == 0:
            n_found = sum(len(s) for s in sources)
            print(f"    ■■■■: {total_rows/1e6:.1f}M■, ■■: {n_found} ■■")

    if sources:
        df = pd.concat(sources, ignore_index=True)

        # ■■■■■■■■■■■■■■■■■■
        dra = (df[ra_col] - ra_cl) * np.cos(np.radians(dec_cl))
        ddec = df[dec_col] - dec_cl
        dist_deg = np.sqrt(dra**2 + ddec**2)
        df = df[dist_deg &lt;= radius_deg].copy()
        df['_dist_deg'] = dist_deg[dist_deg &lt;= radius_deg].values
        df['_phi'] = np.arctan2(ddec[dist_deg &lt;= radius_deg].values,
                                dra[dist_deg &lt;= radius_deg].values)

        return df
    else:
        return pd.DataFrame()

# ■■■■■■■■■■■■
cluster_sources = {}
for cl in known_clusters:
    name, ra, dec, z, r500, src = cl
    print(f"\n  ■■■: {name} (RA={ra:.2f}, Dec={dec:.2f}, ■■={EXTRACT_RADIUS_DEG:.1f}deg)")

    if not DATA_FILE.exists():
        print(f"    !!! ■■■■■■■■■■■■■■■■: {DATA_FILE}")
        continue

    df_src = extract_sources_around_cluster(DATA_FILE, ra, dec, EXTRACT_RADIUS_DEG)

    if len(df_src) &gt; 0:
        print(f"    ■■■■: {len(df_src)} ■■")
        cluster_sources[name] = (cl, df_src)
    else:
        print(f"    ■■■■■■■■■■■■■■■■■■■■■■■■■")

# ======================================================================
# 3. ■■■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 3] ■■■■■■■■■■■ gamma_t(R) ■■■")
print("="*70)

def measure_tangential_shear(df, ra_cl, dec_cl, z_cl,
                             r_min_arcmin=0.5, r_max_arcmin=30.0, n_bins=15):
    """■■■■■■■■■■■■■■■"""

    # ■■■■■■■■■
    e1_col = COL_E1 if COL_E1 in df.columns else [c for c in df.columns if 'e1' in c.lower()][0]
    e2_col = COL_E2 if COL_E2 in df.columns else [c for c in df.columns if 'e2' in c.lower()][0]
    w_col = COL_W if COL_W in df.columns else [c for c in df.columns if 'weight' in c.lower()][0]
    m_col = COL_M if COL_M in df.columns else None
    c1_col = COL_C1 if COL_C1 in df.columns else None
    c2_col = COL_C2 if COL_C2 in df.columns else None

    e1 = df[e1_col].values
    e2 = df[e2_col].values
    w = df[w_col].values

    # ■■■■■■■■■
    if c1_col and c1_col in df.columns:
        e1 = e1 - df[c1_col].values
    if c2_col and c2_col in df.columns:
        e2 = e2 - df[c2_col].values

    # ■■■■■ [arcmin]
    phi = df['_phi'].values
    dist_arcmin = df['_dist_deg'].values * 60.0
    # ■■■■: gamma_t = -(e1*cos(2phi) + e2*sin(2phi))
    gamma_t = -(e1 * np.cos(2*phi) + e2 * np.sin(2*phi))
    # ■■■■: gamma_x = +(e1*sin(2phi) - e2*cos(2phi))  ← ■■■■■■■■■■
    gamma_x = +(e1 * np.sin(2*phi) - e2 * np.cos(2*phi))

    # ■■■■■■■■ (log■■■)
    r_bins = np.logspace(np.log10(r_min_arcmin), np.log10(r_max_arcmin), n_bins+1)
    r_centers = np.sqrt(r_bins[:-1] * r_bins[1:])  # ■■■■

    gamma_t_profile = np.zeros(n_bins)
    gamma_x_profile = np.zeros(n_bins)
    gamma_t_err = np.zeros(n_bins)
    n_sources = np.zeros(n_bins, dtype=int)
    for i in range(n_bins):
```

```
        mask = (dist_arcmin &gt;= r_bins[i]) &amp; (dist_arcmin &lt; r_bins[i+1])
        n_in_bin = np.sum(mask)
        n_sources[i] = n_in_bin

        if n_in_bin &lt; 5:
            gamma_t_profile[i] = np.nan
            gamma_x_profile[i] = np.nan
            gamma_t_err[i] = np.nan
            continue

        w_bin = w[mask]
        gt_bin = gamma_t[mask]
        gx_bin = gamma_x[mask]

        # ■■■■■■■■■■
        if m_col and m_col in df.columns:
            m_bin = df[m_col].values[mask]
            m_mean = np.average(m_bin, weights=w_bin)
        else:
            m_mean = 0.0

        # ■■■■■■■
        w_sum = np.sum(w_bin)
        gamma_t_profile[i] = np.sum(w_bin * gt_bin) / w_sum / (1 + m_mean)
        gamma_x_profile[i] = np.sum(w_bin * gx_bin) / w_sum / (1 + m_mean)

        # ■■■■■■■■■■
        sigma_e = np.sqrt(np.sum(w_bin * gt_bin**2) / w_sum - gamma_t_profile[i]**2)
        gamma_t_err[i] = sigma_e / np.sqrt(n_in_bin)

    # ■■■■■■■■■■ [Mpc]
    # ■■■■ D_A(z)
    from scipy.integrate import quad
    def E(z): return np.sqrt(Omega_m*(1+z)**3 + Omega_L)
    D_c, _ = quad(lambda z: c_light/(H0*E(z)), 0, z_cl)  # [Mpc]
    D_A = D_c / (1 + z_cl)  # [Mpc]

    arcmin_to_Mpc = D_A * np.pi / (180 * 60)  # 1 arcmin = ? Mpc
    r_Mpc = r_centers * arcmin_to_Mpc

    return {
        'r_arcmin': r_centers,
        'r_Mpc': r_Mpc,
        'gamma_t': gamma_t_profile,
        'gamma_x': gamma_x_profile,
        'gamma_t_err': gamma_t_err,
        'n_sources': n_sources,
        'D_A_Mpc': D_A,
        'z_cl': z_cl,
    }

# ■■■■■■■■■■■■■■■■■■
profiles = {}
for name, (cl, df_src) in cluster_sources.items():
    _, ra, dec, z, r500, src = cl
    print(f"\n  ■■■: {name} (N={len(df_src)}■■, z={z:.3f})")

    prof = measure_tangential_shear(df_src, ra, dec, z)
    profiles[name] = prof

    # ■■■■
    valid = ~np.isnan(prof['gamma_t'])
    if valid.sum() &gt; 0:
        gt_max = np.nanmax(prof['gamma_t'])
        sn = np.nanmax(prof['gamma_t'] / prof['gamma_t_err'])
        print(f"    gamma_t ■■■: {gt_max:.4f}")
        print(f"    S/N ■■■: {sn:.1f}")
        print(f"    ■■■■: {valid.sum()}/{len(prof['gamma_t'])}")
    else:
        print(f"    ■■■■■■■")

# ======================================================================
# 4. ■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 4] ■■■■■■■")
print("="*70)

n_cl = len(profiles)
if n_cl &gt; 0:
    fig, axes = plt.subplots(1, min(n_cl, 4), figsize=(6*min(n_cl, 4), 5),
                             squeeze=False)

    for idx, (name, prof) in enumerate(profiles.items()):
        if idx &gt;= 4: break
        ax = axes[0, idx]
        valid = ~np.isnan(prof['gamma_t'])
```

```
        if valid.sum() &gt; 0:
            ax.errorbar(prof['r_arcmin'][valid], prof['gamma_t'][valid],
                        yerr=prof['gamma_t_err'][valid],
                        fmt='o', color='steelblue', ms=5, capsize=3,
                        label='$\\gamma_t$ (tangential)')
            ax.errorbar(prof['r_arcmin'][valid], prof['gamma_x'][valid],
                        yerr=prof['gamma_t_err'][valid],
                        fmt='s', color='coral', ms=4, capsize=2, alpha=0.5,
                        label='$\\gamma_\\times$ (cross, null test)')
            ax.axhline(0, color='k', ls='--', alpha=0.3)

        ax.set_xscale('log')
        ax.set_xlabel('R [arcmin]')
        ax.set_ylabel('$\\gamma$')
        ax.set_title(f'{name} (z={prof["z_cl"]:.2f})')
        ax.legend(fontsize=8)

    plt.tight_layout()
    plt.savefig('hsc_y3_cluster_shear.png', dpi=150, bbox_inches='tight')
    print("  -&gt; hsc_y3_cluster_shear.png")

# ======================================================================
# 5. ■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 5] ■■■■■")
print("="*70)

print(f"""
  ■■■■■■■■■: {len(profiles)}

  ■■■■■■■■■■:""")

for name, prof in profiles.items():
    valid = ~np.isnan(prof['gamma_t'])
    n_valid = valid.sum()
    if n_valid &gt; 0:
        gt_max = np.nanmax(prof['gamma_t'])
        sn_max = np.nanmax(np.abs(prof['gamma_t']) / prof['gamma_t_err'])
        print(f"    {name}: ■■■■■={n_valid}, gamma_t_max={gt_max:.4f}, S/N_max={sn_max:.1f}")
    else:
        print(f"    {name}: ■■■■■■■")

print(f"""
  ■■■■■■■:

  (1) ■■■■■■■■■■■■:
      known_clusters ■■■■■ PSZ2/Miyaoka ■■■■■■■
      ■■■■■■■RA, Dec, z, r500■■■■■■■■■■■■

  (2) NFW ■■■■■■■■:
      gamma_t(R) ■ NFW ■■■■■■■■■■■■■■■
      M_500, c_500 ■■■■■

  (3) ■■■■■■■:
      g_c = eta.sqrt(a0.G.Sigma0) ■■■■■■■■
      gamma_t(R) ■■■■■■NFW/MOND ■■■■■

  (4) ■■-■■■■■■■Stage 2■:
      ■■■■■■■■■■■SDSS spectroscopic ■■■■
      ■■■■■■■■■Sigma0 ■■■■■ DeltaSigma(R) ■■■■■
""")

print("■■")
```

hsc_y3_shear_measure.py

解析目的	17クラスター候補のスタック接線剪断プロファイル測定。9.6GB を1パスでチャンク処理。
結果	S/N=14.1。637,014天体抽出。42秒で完了。

```
"""
HSC Y3 ■■■■■■■■■■■■■
==============================
■■:
  (1) hsc_cluster_candidates_y3.csv — ■■■■■■■■■■■■■■■■■■■38■■
  (2) 931720.csv.gz.1 — HSC Y3 ■■■■■■■■■■3580■■■■■

■■:
  (1) Y3 ■■■■■■■■■■■■■■■■■■
  (2) ■■■■■■■■■■■■■■■■■■
  (3) ■■■■■ gamma_t(R) ■■■
  (4) ■■■■■■■■■■■
  (5) NFW ■■■■■ + ■■■■■■■

■■: uv run --with pandas --with numpy --with scipy --with matplotlib python hsc_y3_shear_measure.py

■■: 9.6GB ■1■■■■■■■■■■■■■■■■■■■■■■■■■■■■
      ■■■■■■■■■■■■■■■ x ■■■■■■■■■■
"""

import numpy as np
import pandas as pd
from scipy import stats, optimize, integrate
from pathlib import Path
import time
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ======================================================================
# 0. ■■
# ======================================================================
SHAPE_FILE = Path(r"D:\■■■■■■■\■■■■■■■\■■■■■■\■■■■■■■■\■■■■■\931720.csv.gz.1")
CLUSTER_FILE = Path("hsc_cluster_candidates_y3.csv")

# ■■■■■■■■
EXTRACT_RADIUS_ARCMIN = 30.0   # ■■■■■ [arcmin]
R_MIN_ARCMIN = 0.5             # ■■■■■■■■■■■■
R_MAX_ARCMIN = 25.0            # ■■■■■■■■■■■■
N_BINS = 12                    # ■■■■■■
CHUNKSIZE = 1_000_000          # ■■■■■■■■■■■■■

# ■■■■■■■■■■
H0 = 70.0; Om = 0.3; OL = 0.7; c_light = 2.998e5  # km/s

# z-bin → ■■■■■■■■■HSC-SSP Y3 ■■■■
# bin 0: z=0.3-0.6, bin 1: z=0.6-0.9, bin 2: z=0.9-1.2, bin 3: z=1.2-1.5
# ■■■■ hsc_y3_zbin ■■■■■■■■■■■■■■■■■■■■■■■■■
ZBIN_CENTERS = {0: 0.45, 1: 0.75, 2: 1.05, 3: 1.35}  # ■■■■
# ■: ■■■■■■■■ zbin=3 ■■■■■■■ → ■■■■■■■■■■

# ■■■■■■■■■■■■■■
# sgm3_r10 ■■■■■ → z-bin 3 ■■■■■■■■■
# ■■■■■■■■■■■ z-bin 3 ■■■■■ photo-z ■■
# ■■■■■■■■■ z ■ z-bin 3 ■■■■■■■■■■
Z_CLUSTER_ASSUMED = 0.35  # ■■■■■■■■■

print("="*70)
print("HSC Y3 ■■■■■■■■■■■■■■")
print("="*70)

# ======================================================================
# 1. ■■■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 1] ■■■■■■■■■■■■■■")
print("="*70)

if CLUSTER_FILE.exists():
    df_cl = pd.read_csv(CLUSTER_FILE)
    print(f"  ■■■: {len(df_cl)}")
else:
    print(f"  !!! {CLUSTER_FILE} ■■■■■■■■■")
    print(f"  hsc_cluster_screening.py ■■■■■■■■■■■■■")
    # ■■■■■■■■: ■■10■■■■■■■■■■
    df_cl = pd.DataFrame([
        {'ra': 140.450, 'dec': -0.251, 'sgm_peak': 6.4, 'n_pixels': 37},
        {'ra': 335.403, 'dec': -0.947, 'sgm_peak': 5.9, 'n_pixels': 22},
        {'ra': 140.509, 'dec': -0.349, 'sgm_peak': 5.8, 'n_pixels': 21},
        {'ra': 140.503, 'dec': -0.441, 'sgm_peak': 5.5, 'n_pixels': 17},
```

```
        {'ra': 140.337, 'dec': -0.249, 'sgm_peak': 5.4, 'n_pixels': 23},
        {'ra': 342.433, 'dec': -0.572, 'sgm_peak': 5.2, 'n_pixels': 26},
        {'ra': 220.153, 'dec': -1.691, 'sgm_peak': 5.2, 'n_pixels': 33},
        {'ra': 138.943, 'dec': -0.413, 'sgm_peak': 5.1, 'n_pixels': 26},
        {'ra': 139.109, 'dec': -0.444, 'sgm_peak': 5.1, 'n_pixels': 27},
        {'ra': 182.638, 'dec': -0.304, 'sgm_peak': 5.1, 'n_pixels': 30},
    ])
    print(f"  ████████: ██10██████████")

# GAMA09H ████████████10'████████████
print(f"\n  ██████████10'███:")
merge_radius_deg = 10.0 / 60.0
merged = []
used = set()
for i, row_i in df_cl.iterrows():
    if i in used: continue
    group = [i]
    for j, row_j in df_cl.iterrows():
        if j &lt;= i or j in used: continue
        dra = (row_i['ra'] - row_j['ra']) * np.cos(np.radians(row_i['dec']))
        ddec = row_i['dec'] - row_j['dec']
        dist = np.sqrt(dra**2 + ddec**2)
        if dist &lt; merge_radius_deg:
            group.append(j)
            used.add(j)
    used.add(i)
    # ███████████
    sub = df_cl.loc[group]
    w = sub['sgm_peak'].values
    merged.append({
        'ra': np.average(sub['ra'].values, weights=w),
        'dec': np.average(sub['dec'].values, weights=w),
        'sgm_peak': sub['sgm_peak'].max(),
        'n_pixels': sub['n_pixels'].sum(),
        'n_merged': len(group),
    })

df_merged = pd.DataFrame(merged).sort_values('sgm_peak', ascending=False).reset_index(drop=True)
print(f"  ████: {len(df_cl)} ██ -&gt; ████: {len(df_merged)} ████")

# ============================================================================
# 2. █████
# ============================================================================
def comoving_dist(z):
    """████ [Mpc]"""
    f = lambda zz: 1.0 / np.sqrt(Om*(1+zz)**3 + OL)
    d, _ = integrate.quad(f, 0, z)
    return c_light / H0 * d

def angular_diameter_dist(z):
    """████ [Mpc]"""
    return comoving_dist(z) / (1 + z)

def sigma_cr(z_l, z_s):
    """█████ [Msun/pc^2]"""
    D_l = angular_diameter_dist(z_l)
    D_s = angular_diameter_dist(z_s)
    D_ls_comoving = comoving_dist(z_s) - comoving_dist(z_l)
    D_ls = D_ls_comoving / (1 + z_s)
    if D_ls &lt;= 0: return np.inf
    # Sigma_cr = c^2/(4*pi*G) * D_s/(D_l*D_ls)
    # in Msun/pc^2: G = 4.301e-3 (km/s)^2 pc/Msun
    G_pc = 4.301e-3  # (km/s)^2 pc / Msun
    Mpc_to_pc = 1e6
    sigma = c_light**2 / (4 * np.pi * G_pc) * (D_s * Mpc_to_pc) / (D_l * Mpc_to_pc * D_ls * Mpc_to_pc)
    return sigma  # [Msun/pc^2]

# ============================================================================
# 3. ████████████████
# ============================================================================
print("\n"+"="*70)
print("[Step 2] Y3 ████████████████")
print("="*70)

extract_radius_deg = EXTRACT_RADIUS_ARCMIN / 60.0
n_clusters = len(df_merged)

# ███████████
cl_ra = df_merged['ra'].values
cl_dec = df_merged['dec'].values
cos_dec = np.cos(np.radians(cl_dec))

# ██████████████████
sources_per_cluster = [[] for _ in range(n_clusters)]

print(f"  ██████: {n_clusters}")
print(f"  ████: {EXTRACT_RADIUS_ARCMIN} arcmin = {extract_radius_deg:.4f} deg")
```

```
print(f"  ■■■■■■■■: {SHAPE_FILE}")
print(f"  ■■■■■■■: {CHUNKSIZE:,}")

if not SHAPE_FILE.exists():
    print(f"  !!! ■■■■■■■: {SHAPE_FILE}")
    print(f"  ■■■■■■■■■■■■■■■■■■■■■■■")
    # ■■: ■■■■■■■■■■■■
else:
    t0 = time.time()
    total_rows = 0
    total_extracted = 0

    for chunk in pd.read_csv(SHAPE_FILE, chunksize=CHUNKSIZE):
        total_rows += len(chunk)

        # ■■■■■■■■■■■■■■■
        if total_rows == len(chunk):
            cols = list(chunk.columns)
            # RA, Dec ■■■
            ra_col = [c for c in cols if c.lower() in ['i_ra', 'ra']]
            dec_col = [c for c in cols if c.lower() in ['i_dec', 'dec']]
            e1_col = [c for c in cols if 'e1' in c.lower() and 'hsmshape' in c.lower()]
            e2_col = [c for c in cols if 'e2' in c.lower() and 'hsmshape' in c.lower()]
            w_col = [c for c in cols if c.lower() == 'derived_weight']
            m_col = [c for c in cols if c.lower() == 'shear_bias_m']
            c1_col = [c for c in cols if c.lower() == 'shear_bias_c1']
            c2_col = [c for c in cols if c.lower() == 'shear_bias_c2']
            zbin_col = [c for c in cols if 'zbin' in c.lower()]

            ra_col = ra_col[0] if ra_col else cols[1]
            dec_col = dec_col[0] if dec_col else cols[2]
            e1_col = e1_col[0] if e1_col else cols[3]
            e2_col = e2_col[0] if e2_col else cols[4]
            w_col = w_col[0] if w_col else cols[7]
            m_col = m_col[0] if m_col else None
            c1_col = c1_col[0] if c1_col else None
            c2_col = c2_col[0] if c2_col else None
            zbin_col = zbin_col[0] if zbin_col else None

            print(f"  ■■■: RA={ra_col}, Dec={dec_col}, e1={e1_col}, e2={e2_col}")
            print(f"         w={w_col}, m={m_col}, c1={c1_col}, c2={c2_col}, zbin={zbin_col}")

        src_ra = chunk[ra_col].values
        src_dec = chunk[dec_col].values

        # ■■■■■■■■■■■■■■■■■■■
        for ic in range(n_clusters):
            dra = (src_ra - cl_ra[ic]) * cos_dec[ic]
            ddec = src_dec - cl_dec[ic]
            dist2 = dra**2 + ddec**2
            mask = dist2 &lt; extract_radius_deg**2

            if mask.sum() &gt; 0:
                sub = chunk[mask].copy()
                sub['_dist_deg'] = np.sqrt(dist2[mask])
                sub['_phi'] = np.arctan2(ddec[mask], dra[mask])
                sources_per_cluster[ic].append(sub)
                total_extracted += mask.sum()

        # ■■
        if total_rows % 5_000_000 == 0:
            elapsed = time.time() - t0
            pct = total_rows / 35_800_000 * 100
            print(f"    {total_rows/1e6:.1f}M■ ({pct:.0f}%), ■■: {total_extracted:,}, "
                  f"■■: {elapsed:.0f}■")

    elapsed = time.time() - t0
    print(f"\n  ■■: {total_rows:,}■■■, {total_extracted:,}■■■■, {elapsed:.0f}■")

# ■■■■■■■■■■■■■■
for ic in range(n_clusters):
    if sources_per_cluster[ic]:
        sources_per_cluster[ic] = pd.concat(sources_per_cluster[ic], ignore_index=True)
    else:
        sources_per_cluster[ic] = pd.DataFrame()

n_per_cl = [len(s) for s in sources_per_cluster]
print(f"\n  ■■■■■■■■■■■:")
for ic in range(n_clusters):
    print(f"    #{ic+1} (RA={cl_ra[ic]:.2f}, Dec={cl_dec[ic]:.2f}): "
          f"{n_per_cl[ic]:,} ■■, sgm={df_merged.iloc[ic]['sgm_peak']:.1f}")

# ======================================================================
# 4. ■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 3] ■■■■■■■■■■■■■")
```

```
print("="*70)

r_bins = np.logspace(np.log10(R_MIN_ARCMIN), np.log10(R_MAX_ARCMIN), N_BINS+1)
r_centers = np.sqrt(r_bins[:-1] * r_bins[1:])

all_profiles = []

for ic in range(n_clusters):
    df_src = sources_per_cluster[ic]
    if len(df_src) &lt; 50:
        all_profiles.append(None)
        continue

    e1 = df_src[e1_col].values
    e2 = df_src[e2_col].values
    w = df_src[w_col].values
    phi = df_src['_phi'].values
    dist_arcmin = df_src['_dist_deg'].values * 60.0

    # ■■■■■■■■■■
    if c1_col and c1_col in df_src.columns:
        e1 = e1 - df_src[c1_col].values
    if c2_col and c2_col in df_src.columns:
        e2 = e2 - df_src[c2_col].values

    # ■■■■■■■■
    m_vals = df_src[m_col].values if (m_col and m_col in df_src.columns) else np.zeros(len(df_src))

    # ■■■■■■■■
    gamma_t = -(e1 * np.cos(2*phi) + e2 * np.sin(2*phi))
    gamma_x = +(e1 * np.sin(2*phi) - e2 * np.cos(2*phi))

    gt_prof = np.full(N_BINS, np.nan)
    gx_prof = np.full(N_BINS, np.nan)
    gt_err = np.full(N_BINS, np.nan)
    n_src = np.zeros(N_BINS, dtype=int)

    for ib in range(N_BINS):
        mask = (dist_arcmin &gt;= r_bins[ib]) &amp;amp; (dist_arcmin &lt; r_bins[ib+1])
        n_src[ib] = mask.sum()
        if n_src[ib] &lt; 5: continue

        w_b = w[mask]
        w_sum = np.sum(w_b)
        m_mean = np.average(m_vals[mask], weights=w_b)

        gt_prof[ib] = np.sum(w_b * gamma_t[mask]) / w_sum / (1 + m_mean)
        gx_prof[ib] = np.sum(w_b * gamma_x[mask]) / w_sum / (1 + m_mean)

        sigma_shape = np.sqrt(np.sum(w_b * gamma_t[mask]**2) / w_sum - gt_prof[ib]**2)
        gt_err[ib] = sigma_shape / np.sqrt(n_src[ib])

    # ■■■■■■■■■
    D_A = angular_diameter_dist(Z_CLUSTER_ASSUMED)
    arcmin_to_Mpc = D_A * np.pi / (180 * 60)
    r_Mpc = r_centers * arcmin_to_Mpc

    # S/N
    valid = ~np.isnan(gt_prof)
    sn_total = 0
    if valid.sum() &gt; 0 and np.any(gt_err[valid] &gt; 0):
        sn_total = np.sqrt(np.sum((gt_prof[valid] / gt_err[valid])**2))

    prof = {
        'r_arcmin': r_centers, 'r_Mpc': r_Mpc,
        'gamma_t': gt_prof, 'gamma_x': gx_prof,
        'gamma_t_err': gt_err, 'n_sources': n_src,
        'sn_total': sn_total,
    }
    all_profiles.append(prof)

    if sn_total &gt; 1:
        print(f"  #{ic+1}: S/N={sn_total:.1f}, N_src={n_src.sum():,}, "
              f"gamma_t_max={np.nanmax(gt_prof):.4f}")

# ======================================================================
# 5. ■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 4] ■■■■■■■")
print("="*70)

# ■■■■■■■■■■■■■■■■■■
gt_stack = np.zeros(N_BINS)
gx_stack = np.zeros(N_BINS)
gt_stack_err = np.zeros(N_BINS)
w_stack = np.zeros(N_BINS)
```

```
n_stack = np.zeros(N_BINS, dtype=int)
n_used = 0

for ic, prof in enumerate(all_profiles):
    if prof is None: continue
    valid = ~np.isnan(prof['gamma_t']) &amp;amp; (prof['gamma_t_err'] &amp;gt; 0)
    if valid.sum() == 0: continue

    n_used += 1
    for ib in range(N_BINS):
        if not valid[ib]: continue
        ivar = 1.0 / prof['gamma_t_err'][ib]**2
        gt_stack[ib] += prof['gamma_t'][ib] * ivar
        gx_stack[ib] += prof['gamma_x'][ib] * ivar
        w_stack[ib] += ivar
        n_stack[ib] += 1

for ib in range(N_BINS):
    if w_stack[ib] &amp;gt; 0:
        gt_stack[ib] /= w_stack[ib]
        gx_stack[ib] /= w_stack[ib]
        gt_stack_err[ib] = 1.0 / np.sqrt(w_stack[ib])
    else:
        gt_stack[ib] = np.nan
        gx_stack[ib] = np.nan
        gt_stack_err[ib] = np.nan

D_A_stack = angular_diameter_dist(Z_CLUSTER_ASSUMED)
r_Mpc_stack = r_centers * D_A_stack * np.pi / (180*60)

valid_s = ~np.isnan(gt_stack) &amp;amp; (gt_stack_err &amp;gt; 0)
sn_stack = np.sqrt(np.sum((gt_stack[valid_s]/gt_stack_err[valid_s])**2)) if valid_s.sum()&amp;gt;0 else 0

print(f"  ■■■■■■■■■■■■■: {n_used}")
print(f"  ■■■■■ S/N: {sn_stack:.1f}")
if valid_s.sum() &amp;gt; 0:
    print(f"  gamma_t(stack) ■■■■: {np.nanmax(gt_stack):.5f}")
    print(f"  gamma_t(stack) ■■■■■: {gt_stack[valid_s][0]:.5f} +/- {gt_stack_err[valid_s][0]:.5f}")

# ======================================================================
# 6. NFW ■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 5] NFW ■■■■■")
print("="*70)

def nfw_shear(r_Mpc, M200, c200, z_l):
    """NFW ■■■■■■■■■■■■■■■■■■"""
    # r200 from M200
    rho_cr = 3 * H0**2 / (8 * np.pi * 4.301e-3) * 1e-6  # Msun/Mpc^3 ■■
    r200 = (3 * M200 / (4 * np.pi * 200 * rho_cr))**(1/3)  # Mpc
    rs = r200 / c200
    x = r_Mpc / rs

    # NFW convergence profile (simplified)
    kappa = np.zeros_like(x)
    for i, xi in enumerate(x):
        if xi &amp;lt; 1:
            f = 1.0 / (xi**2 - 1) * (1 - np.log((1+np.sqrt(1-xi**2))/xi) / np.sqrt(1-xi**2))
        elif xi &amp;gt; 1:
            f = 1.0 / (xi**2 - 1) * (1 - np.arctan(np.sqrt(xi**2-1)/1) / np.sqrt(xi**2-1))
        else:
            f = 1.0 / 3.0
        kappa[i] = f

    # ■■: gamma_t ≈ kappa * scaling
    rho_s = M200 / (4 * np.pi * rs**3 * (np.log(1+c200) - c200/(1+c200)))
    # ■■■ shear ■ kappa_bar - kappa ■■■■■■■■■■
    scaling = 2 * rho_s * rs / sigma_cr(z_l, 0.75)  # z_s=0.75 ■■
    gamma_nfw = kappa * scaling * 1e-6  # ■■■■■■■■■

    return gamma_nfw

# ■■■■■■■■■■■■■■■■■■■■
if sn_stack &amp;gt; 3:
    print(f"  ■■■■ S/N={sn_stack:.1f} &amp;gt; 3: NFW ■■■■■■■")
    # ■■■■■■■■■■■■■■NFW■■■■■■■■■■■■■■■■■■
    print(f"  ■■■■ NFW ■■■■■■■■■■■■■■■■")
else:
    print(f"  ■■■■ S/N={sn_stack:.1f} &amp;lt; 3: NFW ■■■■■■■■")

# ======================================================================
# 7. ■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 6] ■■■■")
print("="*70)
```

```
fig, axes = plt.subplots(2, 2, figsize=(14, 12))

# (a) ■■■■■■■■■■■■
ax = axes[0, 0]
valid_p = ~np.isnan(gt_stack)
if valid_p.sum() &gt; 0:
    ax.errorbar(r_centers[valid_p], gt_stack[valid_p]*1e3,
                yerr=gt_stack_err[valid_p]*1e3,
                fmt='o', color='steelblue', ms=6, capsize=3, label='$\\gamma_t$ (stacked)')
    ax.errorbar(r_centers[valid_p], gx_stack[valid_p]*1e3,
                yerr=gt_stack_err[valid_p]*1e3,
                fmt='s', color='coral', ms=4, capsize=2, alpha=0.5, label='$\\gamma_\\times$ (null)')
ax.axhline(0, color='k', ls='--', alpha=0.3)
ax.set_xscale('log')
ax.set_xlabel('R [arcmin]')
ax.set_ylabel('$\\gamma \\times 10^3$')
ax.set_title(f'Stacked shear (N={n_used}, S/N={sn_stack:.1f})')
ax.legend(fontsize=9)

# (b) ■■■■■■■■ S/N
ax = axes[0, 1]
sn_vals = [p['sn_total'] if p else 0 for p in all_profiles]
ax.barh(range(len(sn_vals)), sn_vals, color='steelblue', alpha=0.7)
ax.axvline(3, color='red', ls='--', alpha=0.5, label='S/N=3')
ax.set_xlabel('Total S/N')
ax.set_ylabel('Cluster #')
ax.set_title('Individual S/N')
ax.legend(fontsize=8)

# (c) ■■3■■■■■■■■■■■■■■
ax = axes[1, 0]
colors_list = ['steelblue', 'coral', 'green', 'purple', 'orange']
plotted = 0
for ic, prof in enumerate(all_profiles):
    if prof is None or prof['sn_total'] &lt; 1: continue
    if plotted &gt;= 5: break
    valid_i = ~np.isnan(prof['gamma_t'])
    if valid_i.sum() &lt; 3: continue
    ax.errorbar(prof['r_arcmin'][valid_i], prof['gamma_t'][valid_i]*1e3,
                yerr=prof['gamma_t_err'][valid_i]*1e3,
                fmt='o-', ms=4, capsize=2, color=colors_list[plotted % 5],
                label=f'#{ic+1} (S/N={prof["sn_total"]:.1f})', alpha=0.7)
    plotted += 1
ax.axhline(0, color='k', ls='--', alpha=0.3)
ax.set_xscale('log')
ax.set_xlabel('R [arcmin]')
ax.set_ylabel('$\\gamma_t \\times 10^3$')
ax.set_title('Top individual profiles')
ax.legend(fontsize=7)

# (d) ■■■■■■■
ax = axes[1, 1]
if valid_p.sum() &gt; 0:
    ax.bar(range(N_BINS), n_stack, color='steelblue', alpha=0.7)
    ax.set_xticks(range(N_BINS))
    ax.set_xticklabels([f'{r:.1f}' for r in r_centers], rotation=45, fontsize=7)
    ax.set_xlabel('R [arcmin]')
    ax.set_ylabel('N clusters contributing')
    ax.set_title('Bin participation')

plt.tight_layout()
plt.savefig('hsc_y3_shear_profiles.png', dpi=150, bbox_inches='tight')
print("  -&gt; hsc_y3_shear_profiles.png")

# ======================================================================
# 8. ■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 7] ■■■■■■")
print("="*70)

# ■■■■■■■■■
result = pd.DataFrame({
    'r_arcmin': r_centers,
    'r_Mpc': r_Mpc_stack,
    'gamma_t': gt_stack,
    'gamma_x': gx_stack,
    'gamma_t_err': gt_stack_err,
    'n_clusters': n_stack,
})
result.to_csv('hsc_y3_stacked_shear.csv', index=False)
print(f"  -&gt; hsc_y3_stacked_shear.csv")

# ■■■■■■■■■■■■■
summary = []
for ic in range(n_clusters):
    prof = all_profiles[ic]
```

```
    row = df_merged.iloc[ic].to_dict()
    row['sn_total'] = prof['sn_total'] if prof else 0
    row['n_sources'] = n_per_cl[ic]
    summary.append(row)
pd.DataFrame(summary).to_csv('hsc_y3_cluster_shear_summary.csv', index=False)
print(f"  -&gt; hsc_y3_cluster_shear_summary.csv")

# ======================================================================
# 9. ■■■■
# ======================================================================
print("\n"+"="*70)
print("[■■■■]")
print("="*70)

print(f"""
  ■■■■■■■■■■■■■■■:
    ■■■■■■■■■: {n_clusters} (■■■■)
    ■■■■■■■■■: {n_used} (■■■■■■■■)
    ■■■■■ S/N: {sn_stack:.1f}
    ■■ z_cluster: {Z_CLUSTER_ASSUMED}

  ■■:""")

if sn_stack &gt; 5:
    print(f"    S/N &gt; 5: ■■■■■■■■■■■■■■NFW ■■■■■ + ■■■■■■■■■■■■")
    print(f"    ■■■■■■■■:")
    print(f"      (1) ■■■■ NFW ■■■■■■■■■■■■■■")
    print(f"      (2) g_c = eta.sqrt(a0.G.Sigma0) ■■■■■■■■■■■")
    print(f"      (3) NFW vs ■■■■■■ chi2 ■■")
elif sn_stack &gt; 3:
    print(f"    S/N &gt; 3: ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■")
elif sn_stack &gt; 0:
    print(f"    S/N &lt; 3: ■■■■■■■■■■■■■■■■■■■■■■■■■■■■")
    print(f"    ■■■■: ■■-■■■■■■■Stage 2■■■■■■■■")
else:
    print(f"    ■■■■■■■■■■■■■■■■■■■")

print("\n■■")
```

hsc_nfw_membrane_compare.py

解析目的	NFW/MOND/膜モデルの chi2 比較。NFWスケーリング近似を使用。
結果	★撤回。MOND>>NFW(dAIC=-32) は近似のアーティファクト（教訓13）。

```
"""
NFW ■■■■ + ■■■■■■
==========================
■■: hsc_y3_stacked_shear.csv■■■■■■■■■■■■■■■■
■■:
  (1) NFW ■■■■■■■■■■■■M_200, c_200■
  (2) MOND■g_c=a0■■■■■■■■■
  (3) ■■■■■■g_c=eta*sqrt(a0*G*Sigma0)■■■■■■■■■
  (4) 3■■■■■ chi2 ■■
  (5) g_c ■■■■■■

■■: uv run --with pandas --with numpy --with scipy --with matplotlib python hsc_nfw_membrane_compare.py
"""

import numpy as np
import pandas as pd
from scipy import optimize, integrate
from pathlib import Path
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ======================================================================
# 0. ■■■■■■
# ======================================================================
H0 = 70.0; Om = 0.3; OL = 0.7; c_light = 2.998e5
G_N = 4.301e-3  # (km/s)^2 pc / Msun
a0 = 1.2e-10     # m/s^2
a0_pc = a0 / 3.086e-17  # pc/s^2 ... ■■■■■■■■ Mpc ■■■■

# ■■■: ■■ Mpc, ■■ Msun, ■■ km/s
# G = 4.301e-3 (km/s)^2 pc/Msun = 4.301e-9 (km/s)^2 Mpc/Msun
G_Mpc = 4.301e-9  # (km/s)^2 Mpc / Msun

Z_L = 0.35   # ■■■■■■■■■■■■
Z_S = 0.75   # ■■■■■■■■■z-bin ■■■

print("="*70)
print("NFW ■■■■ + ■■■■■■■")
print("="*70)
print(f"  z_lens = {Z_L}, z_source = {Z_S}")

# ======================================================================
# 1. ■■■■■
# ======================================================================
def E(z): return np.sqrt(Om*(1+z)**3 + OL)
def comoving(z):
    d, _ = integrate.quad(lambda zz: c_light/(H0*E(zz)), 0, z)
    return d
def D_A(z): return comoving(z) / (1+z)
def D_A2(z1, z2):
    return (comoving(z2) - comoving(z1)) / (1+z2)

def rho_cr(z):
    """■■■■■ [Msun/Mpc^3]"""
    return 3*(H0*E(z))**2 / (8*np.pi*G_Mpc)

def Sigma_cr(z_l, z_s):
    """■■■■■■ [Msun/Mpc^2]"""
    Dl = D_A(z_l)
    Ds = D_A(z_s)
    Dls = D_A2(z_l, z_s)
    if Dls &lt;= 0: return np.inf
    return c_light**2 / (4*np.pi*G_Mpc) * Ds / (Dl * Dls)

S_cr = Sigma_cr(Z_L, Z_S)
D_l = D_A(Z_L)
print(f"  D_A(z_l) = {D_l:.1f} Mpc")
print(f"  Sigma_cr = {S_cr:.3e} Msun/Mpc^2")
print(f"  Sigma_cr = {S_cr * 1e-12:.1f} Msun/pc^2")

# ======================================================================
# 2. NFW ■■■■■■■■Wright &amp;amp; Brainerd 2000■
# ======================================================================
def nfw_params(M200, c200, z_l):
    """NFW ■■■■■■■■■"""
    rc = rho_cr(z_l)
    r200 = (3*M200 / (4*np.pi*200*rc))**(1./3.)  # Mpc
    rs = r200 / c200
    delta_c = 200./3. * c200**3 / (np.log(1+c200) - c200/(1+c200))
```

```
    rho_s = delta_c * rc  # Msun/Mpc^3
    return rs, rho_s, r200

def nfw_sigma(R, rs, rho_s):
    """NFW ■■■ Sigma(R) [Msun/Mpc^2]"""
    x = R / rs
    result = np.zeros_like(x, dtype=float)
    for i, xi in enumerate(x):
        if xi &lt; 1e-6:
            result[i] = 0
        elif abs(xi - 1) &lt; 1e-6:
            result[i] = 2 * rs * rho_s / 3.0
        elif xi &lt; 1:
            sq = np.sqrt(1 - xi**2)
            result[i] = 2*rs*rho_s / (xi**2-1) * (1 - np.log((1+sq)/xi)/sq)
        else:
            sq = np.sqrt(xi**2 - 1)
            result[i] = 2*rs*rho_s / (xi**2-1) * (1 - np.arctan(sq)/sq)
    return result

def nfw_sigma_mean(R, rs, rho_s):
    """NFW ■■■■■ Sigma_mean(&lt;R) [Msun/Mpc^2]"""
    x = R / rs
    result = np.zeros_like(x, dtype=float)
    for i, xi in enumerate(x):
        if xi &lt; 1e-6:
            result[i] = 0
        elif abs(xi - 1) &lt; 1e-6:
            result[i] = 4*rs*rho_s * (1 + np.log(0.5))
        elif xi &lt; 1:
            sq = np.sqrt(1 - xi**2)
            g = np.log(xi/2) + np.log((1+sq)/xi) / sq
            result[i] = 4*rs*rho_s * g / xi**2
        else:
            sq = np.sqrt(xi**2 - 1)
            g = np.log(xi/2) + np.arctan(sq) / sq
            result[i] = 4*rs*rho_s * g / xi**2
    return result

def nfw_delta_sigma(R, M200, c200, z_l):
    """NFW ■ DeltaSigma = Sigma_mean(&lt;R) - Sigma(R)"""
    rs, rho_s, r200 = nfw_params(M200, c200, z_l)
    sig = nfw_sigma(R, rs, rho_s)
    sig_mean = nfw_sigma_mean(R, rs, rho_s)
    return sig_mean - sig

def nfw_gamma_t(R_arcmin, M200, c200, z_l, z_s):
    """NFW ■■■■■ gamma_t"""
    Dl = D_A(z_l)
    R_Mpc = R_arcmin * np.pi / (180*60) * Dl
    ds = nfw_delta_sigma(R_Mpc, M200, c200, z_l)
    Scr = Sigma_cr(z_l, z_s)
    return ds / Scr

# ======================================================================
# 3. ■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 1] ■■■■■■■")
print("="*70)

data_file = Path("hsc_y3_stacked_shear.csv")
if data_file.exists():
    df = pd.read_csv(data_file)
    print(f"  ■■■■: {data_file} ({len(df)} ■■)")
else:
    print(f"  !!! {data_file} ■■■■■■■■■■■■■■■")
    df = pd.DataFrame({
        'r_arcmin': [0.59, 0.82, 1.14, 1.59, 2.22, 3.09, 4.30, 5.99, 8.34, 11.61, 16.17, 22.51],
        'gamma_t': [0.038, 0.053, 0.032, 0.023, 0.018, 0.015, 0.012, 0.010, 0.008, 0.007, 0.006, 0.006],
        'gamma_t_err': [0.025, 0.019, 0.010, 0.006, 0.004, 0.003, 0.002, 0.002, 0.001, 0.001, 0.001, 0.001],
        'gamma_x': [0.005, -0.003, 0.002, -0.001, 0.001, 0.000, -0.001, 0.001, 0.000, 0.000, 0.000, 0.000],
    })

r_data = df['r_arcmin'].values
gt_data = df['gamma_t'].values
gt_err = df['gamma_t_err'].values
gx_data = df['gamma_x'].values if 'gamma_x' in df.columns else np.zeros_like(gt_data)

valid = np.isfinite(gt_data) &amp;amp; np.isfinite(gt_err) &amp;amp; (gt_err &amp;gt; 0)
r_fit = r_data[valid]
gt_fit = gt_data[valid]
err_fit = gt_err[valid]

print(f"  ■■■■■: {valid.sum()}")
print(f"  gamma_t ■■: [{gt_fit.min():.4f}, {gt_fit.max():.4f}]")
# ======================================================================
```

```
# 4. NFW ■■■■
# ================================================================
print("\n"+"="*70)
print("[Step 2] NFW ■■■■■")
print("="*70)

def chi2_nfw(params):
    log_M200, log_c200 = params
    M200 = 10**log_M200
    c200 = 10**log_c200
    if c200 &lt; 1 or c200 &gt; 20 or M200 &lt; 1e12 or M200 &gt; 1e16:
        return 1e20
    try:
        gt_model = nfw_gamma_t(r_fit, M200, c200, Z_L, Z_S)
        return np.sum(((gt_fit - gt_model) / err_fit)**2)
    except:
        return 1e20

# ■■■■■■■■ + ■■■
best_chi2 = np.inf
best_params = (14.0, 0.5)

for lm in np.linspace(13.0, 15.5, 20):
    for lc in np.linspace(0.0, 1.2, 15):
        c2 = chi2_nfw((lm, lc))
        if c2 &lt; best_chi2:
            best_chi2 = c2
            best_params = (lm, lc)

# ■■■
try:
    res = optimize.minimize(chi2_nfw, best_params, method='Nelder-Mead',
                            options={'xatol': 0.001, 'fatol': 0.01})
    if res.fun &lt; best_chi2:
        best_params = res.x
        best_chi2 = res.fun
except:
    pass

M200_fit = 10**best_params[0]
c200_fit = 10**best_params[1]
dof_nfw = len(r_fit) - 2
chi2_nfw_val = best_chi2

rs_fit, rho_s_fit, r200_fit = nfw_params(M200_fit, c200_fit, Z_L)

print(f"  M_200 = {M200_fit:.2e} Msun")
print(f"  c_200 = {c200_fit:.2f}")
print(f"  r_200 = {r200_fit:.2f} Mpc")
print(f"  r_s   = {rs_fit:.3f} Mpc")
print(f"  chi2/dof = {chi2_nfw_val:.1f}/{dof_nfw} = {chi2_nfw_val/dof_nfw:.2f}")

gt_nfw = nfw_gamma_t(r_fit, M200_fit, c200_fit, Z_L, Z_S)

# ================================================================
# 5. MOND ■■■■g_c = a0■
# ================================================================
print("\n"+"="*70)
print("[Step 3] MOND / ■■■■■■■")
print("="*70)

# ■■ MOND ■■■■■:
# g_obs = (1/2)(g_N + sqrt(g_N^2 + 4*g_c*g_N))
# g_N = G*M_bar(&lt;R) / R^2 ■■■■■■■■
# gamma_t = DeltaSigma_eff / Sigma_cr

# NFW ■■■■■■■■■■■■■■■■■■■■■■■■■■■■
f_bar = 0.17  # Omega_b / Omega_m
M_bar_200 = f_bar * M200_fit
print(f"  ■■■■■■■■■: M_bar = {f_bar} x M_200 = {M_bar_200:.2e} Msun")

# ■■■■■■■■■■■■■■NFW ■■■■■■■■■■ f_bar ■■
def baryon_mass_enclosed(R_Mpc, M_bar, c200, z_l):
    """■■■■■■■■■■■■■■NFW■■■■■■■"""
    rc = rho_cr(z_l)
    r200 = (M_bar / f_bar / (4./3.*np.pi*200*rc))**(1./3.)
    rs = r200 / c200
    x = R_Mpc / rs
    # NFW enclosed mass: M(&lt;r) = 4*pi*rho_s*rs^3 * [ln(1+x) - x/(1+x)]
    delta_c = 200./3. * c200**3 / (np.log(1+c200) - c200/(1+c200))
    rho_s = delta_c * rc
    M_enclosed = 4 * np.pi * rho_s * rs**3 * (np.log(1+x) - x/(1+x))
    return M_enclosed * f_bar

def mond_gamma_t(R_arcmin, M_bar, c200, z_l, z_s, g_c_value):
    """MOND/■■■■■■■■■■"""
    Dl = D_A(z_l)
```

```
    Scr = Sigma_cr(z_l, z_s)
    R_Mpc = R_arcmin * np.pi / (180*60) * Dl

    # ■■■■■ g_N
    M_enc = baryon_mass_enclosed(R_Mpc, M_bar, c200, z_l)
    R_m = R_Mpc * 3.086e22  # Mpc -&gt; m
    M_kg = M_enc * 1.989e30  # Msun -&gt; kg
    G_SI = 6.674e-11
    g_N = G_SI * M_kg / R_m**2  # m/s^2

    # MOND ■■
    g_obs = 0.5 * (g_N + np.sqrt(g_N**2 + 4*g_c_value*g_N))

    # g_obs ■■■■■■■■■■
    M_eff = g_obs * R_m**2 / G_SI / 1.989e30  # Msun

    # DeltaSigma ■■■■■■■■■■
    # Sigma_eff(R) = M_eff / (pi * R^2) ■■■■■
    # ■■: gamma_t ~ (M_eff - M_enc) / (pi * R_Mpc^2) / Sigma_cr
    # ■■■■■■■■■■M_eff(&lt;R)■■■■■■■■

    # ■■■■: DeltaSigma = Sigma_mean(&lt;R) - Sigma(R) ■ g_obs ■■■■
    # ■■■■■■■■■■■■NFW ■■■ M_eff/M_NFW ■■■■■
    gt_nfw_here = nfw_gamma_t(R_arcmin, M200_fit, c200_fit, z_l, z_s)
    scaling = M_eff / (baryon_mass_enclosed(R_Mpc, M_bar, c200, z_l) / f_bar)
    # ■■■■■■■■■■■■
    scaling = np.clip(scaling, 0.01, 100)

    return gt_nfw_here * scaling

# MOND■g_c = a0■
gt_mond = mond_gamma_t(r_fit, M_bar_200, c200_fit, Z_L, Z_S, a0)
chi2_mond = np.sum(((gt_fit - gt_mond) / err_fit)**2)
dof_mond = len(r_fit) - 1  # M_bar ■■■■

# g_c ■■■■■■■■■■■■■■■■
def chi2_gc(log_gc):
    gc_val = 10**log_gc
    try:
        gt_m = mond_gamma_t(r_fit, M_bar_200, c200_fit, Z_L, Z_S, gc_val)
        return np.sum(((gt_fit - gt_m) / err_fit)**2)
    except:
        return 1e20

# g_c ■■■■■■■■■
gc_grid = np.logspace(-12, -8, 100)
chi2_gc_grid = np.array([chi2_gc(np.log10(gc)) for gc in gc_grid])
best_gc_idx = np.argmin(chi2_gc_grid)
gc_best = gc_grid[best_gc_idx]
chi2_gc_best = chi2_gc_grid[best_gc_idx]

# ■■■
try:
    res_gc = optimize.minimize_scalar(chi2_gc, bounds=(np.log10(gc_best)-1, np.log10(gc_best)+1),
                                      method='bounded')
    if res_gc.fun &lt; chi2_gc_best:
        gc_best = 10**res_gc.x
        chi2_gc_best = res_gc.fun
except:
    pass

gt_membrane = mond_gamma_t(r_fit, M_bar_200, c200_fit, Z_L, Z_S, gc_best)
dof_gc = len(r_fit) - 2  # M_bar + g_c

print(f"\n  MOND (g_c = a0 = {a0:.1e} m/s^2):")
print(f"    chi2/dof = {chi2_mond:.1f}/{dof_mond} = {chi2_mond/dof_mond:.2f}")
print(f"\n  ■■■■ (g_c free):")
print(f"    g_c_best = {gc_best:.2e} m/s^2 = {gc_best/a0:.3f} a0")
print(f"    chi2/dof = {chi2_gc_best:.1f}/{dof_gc} = {chi2_gc_best/dof_gc:.2f}")

# ======================================================================
# 6. ■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 4] ■■■■■■")
print("="*70)

n_data = len(r_fit)
aic_nfw = chi2_nfw_val + 2*2
aic_mond = chi2_mond + 2*1
aic_membrane = chi2_gc_best + 2*2

print(f"  {'■■■':&lt;35} {'chi2':&gt;8} {'dof':&gt;5} {'chi2/dof':&gt;9} {'AIC':&gt;8} {'dAIC':&gt;8}")
print(f"  {'-'*78}")
print(f"  {'NFW (M200, c200)':&lt;35} {chi2_nfw_val:&gt;8.1f} {dof_nfw:&gt;5} {chi2_nfw_val/dof_nfw:&gt;9.2f} {aic_nfw:&gt;8.1f} {aic_nfw-a
print(f"  {'MOND (g_c=a0, M_bar free)':&lt;35} {chi2_mond:&gt;8.1f} {dof_mond:&gt;5} {chi2_mond/dof_mond:&gt;9.2f} {aic_mond:&gt;8.1f} {aic
print(f"  {'■■■■ (g_c free, M_bar free)':&lt;35} {chi2_gc_best:&gt;8.1f} {dof_gc:&gt;5} {chi2_gc_best/dof_gc:&gt;9.2f} {aic_membrane:&gt;8
```

```
# g_c ■■■■■■■■■■■■
# ■■■■■■■ Sigma0 ■■■NFW ■■■■■■■■
# Sigma0 ~ M_bar / (pi * rs^2)
Sigma0_cluster = M_bar_200 / (np.pi * (rs_fit * 1e6)**2)  # Msun/pc^2
G_SI = 6.674e-11; Msun = 1.989e30; pc_m = 3.086e16
G_Sigma0_cluster = G_SI * Sigma0_cluster * Msun / pc_m**2  # m/s^2

gc_geomean = np.sqrt(a0 * G_Sigma0_cluster)
print(f"\n  ■■■■■■■ Sigma0 ■■: {Sigma0_cluster:.1f} Msun/pc^2")
print(f"  G.Sigma0 = {G_Sigma0_cluster:.2e} m/s^2")
print(f"  ■■■■■■■: g_c = sqrt(a0 . G.Sigma0) = {gc_geomean:.2e} m/s^2 = {gc_geomean/a0:.2f} a0")
print(f"  ■■■■■■:   g_c = {gc_best:.2e} m/s^2 = {gc_best/a0:.3f} a0")
print(f"  ■: g_c(fit) / g_c(geomean) = {gc_best/gc_geomean:.3f}")

# ======================================================================
# 7. g_c ■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 5] g_c ■■■■■■")
print("="*70)

# Delta chi2 = 1 ■ 68% CI, = 4 ■ 95% CI
gc_scan = np.logspace(np.log10(gc_best)-2, np.log10(gc_best)+2, 500)
chi2_scan = np.array([chi2_gc(np.log10(g)) for g in gc_scan])
dchi2 = chi2_scan - chi2_gc_best

# 68% CI
mask_68 = dchi2 &lt; 1.0
if mask_68.sum() &gt; 0:
    gc_lo_68 = gc_scan[mask_68].min()
    gc_hi_68 = gc_scan[mask_68].max()
    print(f"  68% CI: [{gc_lo_68:.2e}, {gc_hi_68:.2e}] m/s^2")
    print(f"          [{gc_lo_68/a0:.3f}, {gc_hi_68/a0:.3f}] a0")

# 95% CI
mask_95 = dchi2 &lt; 4.0
if mask_95.sum() &gt; 0:
    gc_lo_95 = gc_scan[mask_95].min()
    gc_hi_95 = gc_scan[mask_95].max()
    print(f"  95% CI: [{gc_lo_95:.2e}, {gc_hi_95:.2e}] m/s^2")
    print(f"          [{gc_lo_95/a0:.3f}, {gc_hi_95/a0:.3f}] a0")

# a0 ■CI■■■■
a0_in_68 = gc_lo_68 &lt;= a0 &lt;= gc_hi_68 if mask_68.sum() &gt; 0 else False
a0_in_95 = gc_lo_95 &lt;= a0 &lt;= gc_hi_95 if mask_95.sum() &gt; 0 else False
print(f"\n  a0 = {a0:.1e} m/s^2 ■:")
print(f"    68% CI ■: {'YES' if a0_in_68 else 'NO'}")
print(f"    95% CI ■: {'YES' if a0_in_95 else 'NO'}")

# ======================================================================
# 8. ■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 6] ■■■■")
print("="*70)

fig, axes = plt.subplots(2, 2, figsize=(14, 12))
# (a) ■■■■■■■■ + ■■■■■■
ax = axes[0, 0]
ax.errorbar(r_fit, gt_fit*1e3, yerr=err_fit*1e3,
           fmt='o', color='black', ms=6, capsize=3, label='HSC Y3 stacked', zorder=5)
ax.plot(r_fit, gt_nfw*1e3, 'r-', lw=2,
       label=f'NFW (M={M200_fit:.1e}, c={c200_fit:.1f})')
ax.plot(r_fit, gt_mond*1e3, 'b--', lw=2,
       label=f'MOND ($g_c$=$a_0$)')
ax.plot(r_fit, gt_membrane*1e3, 'g-.', lw=2,
       label=f'Membrane ($g_c$={gc_best/a0:.2f}$a_0$)')
ax.set_xscale('log')
ax.set_xlabel('R [arcmin]')
ax.set_ylabel('$\\gamma_t \\times 10^3$')
ax.set_title('Shear profile: NFW vs MOND vs Membrane')
ax.legend(fontsize=8)

# (b) ■■
ax = axes[0, 1]
ax.errorbar(r_fit, (gt_fit-gt_nfw)/err_fit, fmt='o', color='red', ms=5, label='NFW')
ax.errorbar(r_fit*1.05, (gt_fit-gt_mond)/err_fit, fmt='s', color='blue', ms=5, label='MOND')
ax.errorbar(r_fit*1.1, (gt_fit-gt_membrane)/err_fit, fmt='^', color='green', ms=5, label='Membrane')
ax.axhline(0, color='k', ls='--', alpha=0.3)
ax.axhline(2, color='gray', ls=':', alpha=0.3)
ax.axhline(-2, color='gray', ls=':', alpha=0.3)
ax.set_xscale('log')
ax.set_xlabel('R [arcmin]')
ax.set_ylabel('Residual / $\\sigma$')
ax.set_title('Residuals')
ax.legend(fontsize=8)
# (c) g_c ■ chi2 ■■■■■■
```

```
ax = axes[1, 0]
ax.plot(gc_scan/a0, dchi2, 'b-', lw=2)
ax.axhline(1, color='orange', ls='--', alpha=0.5, label='68% CL')
ax.axhline(4, color='red', ls='--', alpha=0.5, label='95% CL')
ax.axvline(1.0, color='gray', ls=':', alpha=0.5, label='$g_c=a_0$ (MOND)')
ax.axvline(gc_best/a0, color='green', ls='-', alpha=0.7, label=f'best={gc_best/a0:.2f}$a_0$')
if gc_geomean &gt; 0:
    ax.axvline(gc_geomean/a0, color='purple', ls='--', alpha=0.5,
               label=f'geomean={gc_geomean/a0:.2f}$a_0$')
ax.set_xlabel('$g_c / a_0$')
ax.set_ylabel('$\\Delta\\chi^2$')
ax.set_title('$g_c$ constraint')
ax.set_xscale('log')
ax.set_ylim(0, min(20, dchi2.max()))
ax.legend(fontsize=7)

# (d) ■■■■■■■■■■
ax = axes[1, 1]
models = ['NFW', 'MOND\n($g_c=a_0$)', 'Membrane\n($g_c$ free)']
chi2_vals = [chi2_nfw_val/dof_nfw, chi2_mond/dof_mond, chi2_gc_best/dof_gc]
colors = ['red', 'blue', 'green']
bars = ax.bar(models, chi2_vals, color=colors, alpha=0.7, edgecolor='black')
ax.axhline(1, color='k', ls='--', alpha=0.3)
ax.set_ylabel('$\\chi^2$/dof')
ax.set_title('Model comparison')
for bar, val in zip(bars, chi2_vals):
    ax.text(bar.get_x()+bar.get_width()/2, bar.get_height()+0.02,
            f'{val:.2f}', ha='center', fontsize=10)

plt.tight_layout()
plt.savefig('hsc_nfw_membrane_comparison.png', dpi=150, bbox_inches='tight')
print("  -&gt; hsc_nfw_membrane_comparison.png")

# ======================================================================
# 9. ■■■■
# ======================================================================
print("\n"+"="*70)
print("[■■■■■]")
print("="*70)

print(f"""
  HSC Y3 ■■■■■■■■■■■■■■■:
    S/N = 14.1
    z_lens = {Z_L} (■■), z_source = {Z_S}

  NFW ■■■■■:
    M_200 = {M200_fit:.2e} Msun
    c_200 = {c200_fit:.2f}
    chi2/dof = {chi2_nfw_val:.1f}/{dof_nfw} = {chi2_nfw_val/dof_nfw:.2f}

  MOND (g_c = a0):
    chi2/dof = {chi2_mond:.1f}/{dof_mond} = {chi2_mond/dof_mond:.2f}
    dAIC vs NFW = {aic_mond-aic_nfw:+.1f}

  ■■■■■ (g_c free):
    g_c = {gc_best:.2e} m/s^2 = {gc_best/a0:.3f} a0
    chi2/dof = {chi2_gc_best:.1f}/{dof_gc} = {chi2_gc_best/dof_gc:.2f}
    dAIC vs NFW = {aic_membrane-aic_nfw:+.1f}

  ■■■■■■■■■■■:
    g_c(geomean) = sqrt(a0 . G.Sigma0) = {gc_geomean:.2e} m/s^2 = {gc_geomean/a0:.2f} a0
    g_c(fit) / g_c(geomean) = {gc_best/gc_geomean:.3f}

  ■■:""")

if chi2_gc_best/dof_gc &lt; chi2_nfw_val/dof_nfw:
    print(f"    ■■■■■■ NFW ■■■■■■■■■■■■■■")
elif abs(chi2_gc_best/dof_gc - chi2_nfw_val/dof_nfw) &lt; 0.1:
    print(f"    ■■■■■■ NFW ■■■■■■■■■■■■")
else:
    print(f"    NFW ■■■■■■■■■■■■■■■■■■■")

if not a0_in_95:
    print(f"    g_c = a0■MOND■■ 95% CI ■■■■■")
elif not a0_in_68:
    print(f"    g_c = a0■MOND■■ 68% CI ■■■■■■■■ 95% CI ■■■■■■■■■■")
else:
    print(f"    g_c = a0■MOND■■■■■■■■■■")

print(f"""
  ■■■■:
    * z_lens = {Z_L} ■■■■■■■■■■■■■■■■■■■■■■■■
    * ■■■■■■■■NFW■■■■■■ f_bar = {f_bar} ■■■■■■■■■■■■■■■■
    * ■■ MOND ■■■■■■■■■■■■■■■■■■■■■■■■
    * ■■■■■■■■ z_lens, f_bar ■■■■■■■■■■
""")
print("■■")
```

hsc_download_photoz.py

解析目的	HSC-SSP PDR3 API からクラスター周辺の photo-z とレンズ銀河カタログをダウンロード。
結果	5クラスターの photo-z 取得成功。z=0.37-0.79。

```
"""
HSC-SSP PDR3 ■■■■■■■■■■■■■■■■■■■
================================================
Claude Code■VS Code■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■

■■: HSC-SSP ■■■■■■■■■■■■■■■■■■

■■: python hsc_download_photoz.py --user YOUR_USERNAME --password YOUR_PASSWORD

■■: https://hsc-release.mtk.nao.ac.jp/doc/
"""

import argparse
import json
import time
import sys
import os
from pathlib import Path
from urllib.parse import urlencode
from getpass import getpass

# requests ■■■■■■■■■■■■■■
try:
    import requests
except ImportError:
    print("requests ■■■■■■:")
    print("  pip install requests")
    print("  ■■■■: uv run --with requests python hsc_download_photoz.py")
    sys.exit(1)

# ======================================================================
# 0. ■■
# ======================================================================
HSC_API_URL = "https://hsc-release.mtk.nao.ac.jp/datasearch/api/catalog_jobs/"
OUTPUT_DIR = Path(r"D:\■■■■■■■\■■■■■■■\■■■■■■\■■■■■■■■\■■■■■")

# ■■■■■■■■■■■■■■■■■■■■■■■■■
CLUSTERS = [
    {"name": "cl1", "ra": 140.45, "dec": -0.25, "label": "GAMA09H #1 (sgm=6.4)"},
    {"name": "cl2", "ra": 142.21, "dec": -0.12, "label": "GAMA09H #2 (S/N=8.0)"},
    {"name": "cl3", "ra": 140.30, "dec": -0.35, "label": "GAMA09H #3 (S/N=7.2)"},
    {"name": "cl4", "ra": 182.68, "dec": -0.28, "label": "WIDE12H (S/N=6.8)"},
    {"name": "cl5", "ra": 216.75, "dec": -0.20, "label": "GAMA15H (S/N=6.4)"},
    {"name": "cl6", "ra": 335.40, "dec": -0.95, "label": "VVDS (sgm=5.9)"},
    {"name": "cl7", "ra": 342.43, "dec": -0.57, "label": "VVDS #2 (sgm=5.2)"},
    {"name": "cl8", "ra": 139.00, "dec": -0.41, "label": "GAMA09H #4 (S/N=6.2)"},
]

# ======================================================================
# 1. HSC-SSP API ■■■
# ======================================================================
class HSC_SSP_API:
    def __init__(self, user, password):
        self.session = requests.Session()
        self.user = user
        self.password = password
        self.api_url = HSC_API_URL

    def _auth(self):
        """Basic■■■■■■■■■■"""
        return (self.user, self.password)

    def submit_query(self, sql, release_version="pdr3"):
        """SQL■■■■■■■"""
        payload = {
            "sql": sql,
            "out_format": "csv",
            "include_metainfo": "no",
            "release_version": release_version,
        }

        print(f"    ■■■■■■■...")
        try:
            r = self.session.post(
                self.api_url,
                data=payload,
                auth=self._auth(),
                timeout=60
            )
```

```
        except requests.exceptions.ConnectionError as e:
            print(f"    !!! ■■■■■: {e}")
            return None

        if r.status_code == 401:
            print(f"    !!! ■■■■■■■■■■■/■■■■■■■■■■■■■■■■")
            return None

        if r.status_code != 200:
            print(f"    !!! HTTP■■■: {r.status_code}")
            print(f"    {r.text[:500]}")
            return None

        try:
            result = r.json()
        except:
            print(f"    !!! JSON■■■■■■■")
            print(f"    {r.text[:500]}")
            return None

        job_id = result.get("id")
        if job_id:
            print(f"    ■■■ID: {job_id}")
        return job_id

    def wait_for_job(self, job_id, max_wait=300, interval=5):
        """■■■■■■■■■"""
        url = f"{self.api_url}{job_id}/"
        elapsed = 0

        while elapsed &lt; max_wait:
            r = self.session.get(url, auth=self._auth())
            if r.status_code != 200:
                print(f"    !!! ■■■■■■■■■■: {r.status_code}")
                return None

            status = r.json()
            state = status.get("status", "unknown")

            if state == "done":
                print(f"    ■■ ({elapsed}■)")
                return status
            elif state == "error":
                print(f"    !!! ■■■: {status.get('error', 'unknown')}")
                return None

            time.sleep(interval)
            elapsed += interval
            if elapsed % 30 == 0:
                print(f"    ■■■... ({elapsed}■)")

        print(f"    !!! ■■■■■■ ({max_wait}■)")
        return None

    def download_result(self, job_id, output_path):
        """■■CSV■■■■■■■■"""
        url = f"{self.api_url}{job_id}/result/"

        print(f"    ■■■■■■■■...")
        r = self.session.get(url, auth=self._auth(), stream=True)

        if r.status_code != 200:
            print(f"    !!! ■■■■■■■■■■: {r.status_code}")
            return False

        with open(output_path, 'wb') as f:
            for chunk in r.iter_content(chunk_size=8192):
                f.write(chunk)

        size_mb = os.path.getsize(output_path) / (1024*1024)
        print(f"    ■■: {output_path} ({size_mb:.1f} MB)")
        return True

    def query_and_download(self, sql, output_path, release="pdr3"):
        """■■■■■→■■→■■■■■■■■■■■■"""
        job_id = self.submit_query(sql, release)
        if job_id is None:
            return False

        status = self.wait_for_job(job_id)
        if status is None:
            return False

        return self.download_result(job_id, output_path)


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

```
# 2. ■■■■■
# ================================================================
def photoz_query(ra, dec, radius_arcsec=300):
    """■■■■■■■■ photo-z ■■■"""
    return f"""
SELECT
    object_id,
    ra, dec,
    photoz_best,
    photoz_err68_min, photoz_err68_max,
    i_cmodel_mag,
    i_extendedness_value
FROM
    pdr3_wide.forced
    LEFT JOIN pdr3_wide.photoz_mizuki USING (object_id)
WHERE
    cone_search(coord, {ra}, {dec}, {radius_arcsec})
    AND i_cmodel_mag &lt; 24.5
    AND photoz_best IS NOT NULL
    AND photoz_best BETWEEN 0.01 AND 2.0
    AND i_extendedness_value = 1
"""

def lens_galaxy_query():
    """■■■■■■■■■■■■■■■■-■■■■■■■■"""
    return """
SELECT
    object_id,
    ra, dec,
    photoz_best,
    photoz_err68_min, photoz_err68_max,
    i_cmodel_mag,
    r_cmodel_mag,
    g_cmodel_mag,
    i_extendedness_value,
    i_sdss_shape_shape11,
    i_sdss_shape_shape22
FROM
    pdr3_wide.forced
    LEFT JOIN pdr3_wide.photoz_mizuki USING (object_id)
WHERE
    dec BETWEEN -5.9 AND 0.0
    AND i_cmodel_mag &lt; 21.5
    AND i_extendedness_value = 1
    AND photoz_best BETWEEN 0.05 AND 0.45
    AND photoz_best IS NOT NULL

LIMIT 100000
"""


# ================================================================
# 3. ■■■■■
# ================================================================
def main():
    parser = argparse.ArgumentParser(description="HSC-SSP ■■■■■■■■■■")
    parser.add_argument("--user", "-u", help="HSC-SSP ■■■■■■")
    parser.add_argument("--password", "-p", help="HSC-SSP ■■■■■■")
    parser.add_argument("--skip-clusters", action="store_true", help="■■■■■■photo-z■■■■■■")
    parser.add_argument("--skip-lens", action="store_true", help="■■■■■■■■■■■■■")
    args = parser.parse_args()

    print("="*70)
    print("HSC-SSP PDR3 ■■■■■■■■■■")
    print("="*70)

    # ■■■■■
    user = args.user or input("HSC-SSP ■■■■■■: ")
    password = args.password or getpass("HSC-SSP ■■■■■■: ")

    api = HSC_SSP_API(user, password)

    # ■■■■■■■■■■■
    OUTPUT_DIR.mkdir(parents=True, exist_ok=True)
    print(f"  ■■■■: {OUTPUT_DIR}")

    # --- ■■■■■■■■ photo-z ---
    if not args.skip_clusters:
        print(f"\n{'='*70}")
        print(f"[Phase 1] ■■■■■■■■■■■■ photo-z ■■■■■■■")
        print(f"{'='*70}")

        for cl in CLUSTERS:
            name = cl['name']
            ra = cl['ra']
            dec = cl['dec']
            label = cl['label']
```

```
            outpath = OUTPUT_DIR / f"hsc_photoz_{name}.csv"

            print(f"\n  --- {label} (RA={ra}, Dec={dec}) ---")

            if outpath.exists():
                print(f"    ■■■■: {outpath} → ■■■■")
                continue

            sql = photoz_query(ra, dec)
            success = api.query_and_download(sql, outpath)

            if not success:
                print(f"    !!! ■■■■■■■■■■■■■")

            time.sleep(2)  # API ■■■■

    # --- ■■■■■■■■■■ ---
    if not args.skip_lens:
        print(f"\n{'='*70}")
        print(f"[Phase 2] ■■■■■■■■■■■■■■■■■■")
        print(f"{'='*70}")

        outpath = OUTPUT_DIR / "hsc_lens_galaxies.csv"

        if outpath.exists():
            print(f"  ■■■■: {outpath} → ■■■■")
        else:
            sql = lens_galaxy_query()
            success = api.query_and_download(sql, outpath)

            if not success:
                print(f"  !!! ■■■■■■■■■■■■■■■■")

    # --- ■■■■ ---
    print(f"\n{'='*70}")
    print(f"[■■■■]")
    print(f"{'='*70}")

    for f in sorted(OUTPUT_DIR.glob("hsc_photoz_*.csv")):
        size_kb = f.stat().st_size / 1024
        try:
            import pandas as pd
            df = pd.read_csv(f, nrows=5)
            n_cols = len(df.columns)
            # ■■■■■■
            n_lines = sum(1 for _ in open(f)) - 1
            print(f"  {f.name}: {n_lines:,} ■, {n_cols} ■■■, {size_kb:.0f} KB")
        except:
            print(f"  {f.name}: {size_kb:.0f} KB")

    lens_file = OUTPUT_DIR / "hsc_lens_galaxies.csv"
    if lens_file.exists():
        size_mb = lens_file.stat().st_size / (1024*1024)
        print(f"  {lens_file.name}: {size_mb:.1f} MB")

    print(f"""
  ■■■■■■■:
    1. hsc_photoz_cl*.csv ■■■■■■■■■■■■■
       ■■■■■■■■■■■■■■■
    2. photo-z ■■■■■■■■■■■■■■■■■■
    3. z_lens ■■■■■■ NFW/■■■■■■■■■■■■
""")

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

hsc_refit_with_photoz.py

解析目的	photo-z 更新（z=0.56）での NFW/膜モデル再比較。z スキャンで g_c(z) の安定性確認。
結果	g_c=0.777a0 (95%CI=[0.537,1.026])。a0 棄却できない。

```
"""
NFW / ■■■■■■■■z_lens ■■■■■
==================================
photo-z ■■■■■■■■■■■■■■■■■■■■■■■■
cl1 (z=0.79 &gt; z_source) ■■■■■

■■: uv run --with pandas --with numpy --with scipy --with matplotlib python hsc_refit_with_photoz.py
"""

import numpy as np
import pandas as pd
from scipy import optimize, integrate
from pathlib import Path
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ============================================================================
# 0. ■■■■■■■
# ============================================================================
H0 = 70.0; Om = 0.3; OL = 0.7; c_light = 2.998e5
G_Mpc = 4.301e-9  # (km/s)^2 Mpc / Msun
a0 = 1.2e-10       # m/s^2

# Y3 ■■■■■■■■■■
# zbin=3: z~1.2-1.5 (■■ ~1.35)
# ■■■■■■■■■■■■■■■■ z_s ~ 0.85-1.0 ■■■■
Z_S_EFF = 1.0  # ■■■■■ z_s=1.0

# photo-z ■■■■■■■■■■■■■■■■
CLUSTERS_PHOTOZ = [
    # name, ra, dec, z_cluster, z_SN, ■■
    ("cl2", 142.21, -0.12, 0.670, 5.5, "■■■"),
    ("cl3", 140.30, -0.35, 0.370, 4.2, "■■■"),
    ("cl4", 182.68, -0.28, 0.510, 5.2, "■■■"),
    ("cl5", 216.75, -0.20, 0.610, 7.4, "■■■"),
    # cl1 ■ z=0.789 ■ z_s ■■■■■■■■■■
    # ("cl1", 140.45, -0.25, 0.789, 4.0, "■■: z_l ~ z_s"),
]

print("="*70)
print("NFW / ■■■■■■■■photo-z ■■■■■")
print("="*70)
print(f"  z_source (■■) = {Z_S_EFF}")
print(f"  ■■■■■■■■■: {len(CLUSTERS_PHOTOZ)}")
print(f"  ■■: cl1 (z=0.789 &gt; z_s ■■■■)")

for name, ra, dec, z, sn, status in CLUSTERS_PHOTOZ:
    print(f"    {name}: z={z:.3f}, S/N={sn:.1f} [{status}]")

# ============================================================================
# 1. ■■■■■■
# ============================================================================
def E(z): return np.sqrt(Om*(1+z)**3 + OL)
def comoving(z):
    d, _ = integrate.quad(lambda zz: c_light/(H0*E(zz)), 0, z)
    return d
def D_A(z): return comoving(z) / (1+z)
def D_A2(z1, z2): return (comoving(z2) - comoving(z1)) / (1+z2)
def rho_cr(z): return 3*(H0*E(z))**2 / (8*np.pi*G_Mpc)

def Sigma_cr(z_l, z_s):
    Dl = D_A(z_l); Ds = D_A(z_s); Dls = D_A2(z_l, z_s)
    if Dls &lt;= 0: return np.inf
    return c_light**2 / (4*np.pi*G_Mpc) * Ds / (Dl * Dls)

# ■■■■■■■■ Sigma_cr ■■■■
print(f"\n  Sigma_cr ■■:")
for name, ra, dec, z_l, sn, st in CLUSTERS_PHOTOZ:
    Scr = Sigma_cr(z_l, Z_S_EFF)
    Dl = D_A(z_l)
    arcmin_to_Mpc = Dl * np.pi / (180*60)
    print(f"    {name}: z={z_l:.3f}, D_A={Dl:.0f} Mpc, "
          f"Sigma_cr={Scr:.2e} Msun/Mpc^2, 1'={arcmin_to_Mpc:.4f} Mpc")

# ============================================================================
# 2. NFW ■■■■■■■■Wright &amp;amp; Brainerd 2000■
# ============================================================================
def nfw_params(M200, c200, z_l):
    rc = rho_cr(z_l)
```

```
    r200 = (3*M200 / (4*np.pi*200*rc))**(1./3.)
    rs = r200 / c200
    delta_c = 200./3. * c200**3 / (np.log(1+c200) - c200/(1+c200))
    rho_s = delta_c * rc
    return rs, rho_s, r200

def nfw_sigma(R, rs, rho_s):
    x = R / rs
    result = np.zeros_like(x, dtype=float)
    for i, xi in enumerate(x):
        if xi &lt; 1e-6: result[i] = 0
        elif abs(xi-1) &lt; 1e-6: result[i] = 2*rs*rho_s/3.0
        elif xi &lt; 1:
            sq = np.sqrt(1-xi**2)
            result[i] = 2*rs*rho_s/(xi**2-1) * (1 - np.log((1+sq)/xi)/sq)
        else:
            sq = np.sqrt(xi**2-1)
            result[i] = 2*rs*rho_s/(xi**2-1) * (1 - np.arctan(sq)/sq)
    return result

def nfw_sigma_mean(R, rs, rho_s):
    x = R / rs
    result = np.zeros_like(x, dtype=float)
    for i, xi in enumerate(x):
        if xi &lt; 1e-6: result[i] = 0
        elif abs(xi-1) &lt; 1e-6: result[i] = 4*rs*rho_s*(1+np.log(0.5))
        elif xi &lt; 1:
            sq = np.sqrt(1-xi**2)
            g = np.log(xi/2) + np.log((1+sq)/xi)/sq
            result[i] = 4*rs*rho_s*g/xi**2
        else:
            sq = np.sqrt(xi**2-1)
            g = np.log(xi/2) + np.arctan(sq)/sq
            result[i] = 4*rs*rho_s*g/xi**2
    return result

def nfw_gamma_t(R_arcmin, M200, c200, z_l, z_s):
    rs, rho_s, r200 = nfw_params(M200, c200, z_l)
    Dl = D_A(z_l)
    R_Mpc = R_arcmin * np.pi/(180*60) * Dl
    R_Mpc = np.maximum(R_Mpc, 1e-6)
    sig = nfw_sigma(R_Mpc, rs, rho_s)
    sig_mean = nfw_sigma_mean(R_Mpc, rs, rho_s)
    ds = sig_mean - sig
    Scr = Sigma_cr(z_l, z_s)
    return ds / Scr

# ============================================================================
# 3. MOND/■ ■■■■■
# ============================================================================
f_bar = 0.17

def baryon_enclosed(R_Mpc, M200_total, c200, z_l):
    rs, rho_s, r200 = nfw_params(M200_total, c200, z_l)
    x = R_Mpc / rs
    x = np.maximum(x, 1e-6)
    M_enc = 4*np.pi*rho_s*rs**3 * (np.log(1+x) - x/(1+x))
    return M_enc * f_bar

def mond_gamma_t(R_arcmin, M200, c200, z_l, z_s, gc_val):
    Dl = D_A(z_l)
    Scr = Sigma_cr(z_l, z_s)
    R_Mpc = R_arcmin * np.pi/(180*60) * Dl
    R_Mpc = np.maximum(R_Mpc, 1e-6)

    M_bar = baryon_enclosed(R_Mpc, M200, c200, z_l)
    R_m = R_Mpc * 3.086e22
    M_kg = M_bar * 1.989e30
    G_SI = 6.674e-11
    g_N = G_SI * M_kg / R_m**2

    g_obs = 0.5 * (g_N + np.sqrt(g_N**2 + 4*gc_val*g_N))

    M_eff_kg = g_obs * R_m**2 / G_SI
    M_eff = M_eff_kg / 1.989e30

    # NFW ■■■■■■■
    gt_nfw = nfw_gamma_t(R_arcmin, M200, c200, z_l, z_s)
    M_nfw = baryon_enclosed(R_Mpc, M200, c200, z_l) / f_bar
    scaling = np.where(M_nfw &gt; 0, M_eff / M_nfw, 1.0)
    scaling = np.clip(scaling, 0.01, 100)

    return gt_nfw * scaling

# ============================================================================
# 4. ■■■■■■■
# ============================================================================
```

```
print("\n"+"="*70)
print("[Step 1] ■■■■■■■■■■■■■■■■■■■■")
print("="*70)

# hsc_y3_stacked_shear.csv ■■■■■■■■■■■■■■■■
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
stacked_file = Path("hsc_y3_stacked_shear.csv")
summary_file = Path("hsc_y3_cluster_shear_summary.csv")

if stacked_file.exists():
    df_stack = pd.read_csv(stacked_file)
    print(f"  ■■■■■■■: {stacked_file} ({len(df_stack)} ■■)")
else:
    print(f"  ■■■■■■■■■■■■■■■■■■■■■■■■■")
    df_stack = pd.DataFrame({
        'r_arcmin': [0.59, 0.82, 1.14, 1.59, 2.22, 3.09, 4.30, 5.99, 8.34, 11.61, 16.17, 22.51],
        'gamma_t': [0.038, 0.053, 0.032, 0.023, 0.018, 0.015, 0.012, 0.010, 0.008, 0.007, 0.006, 0.006],
        'gamma_t_err': [0.025, 0.019, 0.010, 0.006, 0.004, 0.003, 0.002, 0.002, 0.001, 0.001, 0.001, 0.001],
        'gamma_x': [0.005, -0.003, 0.002, -0.001, 0.001, 0.000, -0.001, 0.001, 0.000, 0.000, 0.000, 0.000],
    })

r_data = df_stack['r_arcmin'].values
gt_data = df_stack['gamma_t'].values
gt_err = df_stack['gamma_t_err'].values

valid = np.isfinite(gt_data) &amp;amp; np.isfinite(gt_err) &amp;amp; (gt_err &amp;gt; 0)
r_fit = r_data[valid]
gt_fit = gt_data[valid]
err_fit = gt_err[valid]

# ═══════════════════════════════════════════════════════════════════
# 5. z_lens ■■■ NFW/■■■■■■■■
# ═══════════════════════════════════════════════════════════════════
print("\n"+"="*70)
print("[Step 2] z_lens ■■■■ + ■■ z ■■■■")
print("="*70)

# (A) z_lens ■■■■■■■■■■
z_scan = np.arange(0.20, 0.85, 0.05)
gc_vs_z = []

print(f"\n  z_lens ■■■■■■■:")
print(f"  {'z_l':&amp;gt;6} {'chi2_NFW':&amp;gt;10} {'chi2_MOND':&amp;gt;10} {'chi2_membrane':&amp;gt;14} {'g_c/a0':&amp;gt;8} {'M200':&amp;gt;12}")
print(f"  {'-'*65}")

for z_l in z_scan:
    if z_l &amp;gt;= Z_S_EFF - 0.05:
        continue

    # NFW ■■■■
    def chi2_nfw_z(params):
        lm, lc = params
        M, c = 10**lm, 10**lc
        if c&amp;lt;1 or c&amp;gt;20 or M&amp;lt;1e12 or M&amp;gt;1e16: return 1e20
        try: return np.sum(((gt_fit - nfw_gamma_t(r_fit, M, c, z_l, Z_S_EFF))/err_fit)**2)
        except: return 1e20

    best_c2n = np.inf; best_pn = (14, 0.5)
    for lm in np.linspace(13, 15.5, 15):
        for lc in np.linspace(0, 1.2, 10):
            c2 = chi2_nfw_z((lm, lc))
            if c2 &amp;lt; best_c2n: best_c2n = c2; best_pn = (lm, lc)
    try:
        res = optimize.minimize(chi2_nfw_z, best_pn, method='Nelder-Mead')
        if res.fun &amp;lt; best_c2n: best_c2n = res.fun; best_pn = res.x
    except: pass
    M200_z = 10**best_pn[0]; c200_z = 10**best_pn[1]

    # MOND
    gt_mond_z = mond_gamma_t(r_fit, M200_z, c200_z, z_l, Z_S_EFF, a0)
    c2_mond = np.sum(((gt_fit - gt_mond_z)/err_fit)**2)

    # ■■■■■g_c ■■■
    def chi2_gc_z(lgc):
        try: return np.sum(((gt_fit - mond_gamma_t(r_fit, M200_z, c200_z, z_l, Z_S_EFF, 10**lgc))/err_fit)**2)
        except: return 1e20

    gc_grid = np.logspace(-12, -8, 80)
    c2_grid = [chi2_gc_z(np.log10(g)) for g in gc_grid]
    gc_best_z = gc_grid[np.argmin(c2_grid)]
    c2_membrane = min(c2_grid)

    gc_vs_z.append({'z_l': z_l, 'gc': gc_best_z, 'gc_a0': gc_best_z/a0,
                    'chi2_nfw': best_c2n, 'chi2_mond': c2_mond, 'chi2_membrane': c2_membrane,
                    'M200': M200_z})

    print(f"  {z_l:&amp;gt;6.2f} {best_c2n:&amp;gt;10.1f} {c2_mond:&amp;gt;10.1f} {c2_membrane:&amp;gt;14.1f} "
```

```
          f"{gc_best_z/a0:&gt;8.3f} {M200_z:&gt;12.2e}")

df_zscan = pd.DataFrame(gc_vs_z)

# (B) ■■■■ z ■■■■■■■■■■■■
z_weights = np.array([sn for _, _, _, _, sn, _ in CLUSTERS_PHOTOZ])
z_values = np.array([z for _, _, _, z, _, _ in CLUSTERS_PHOTOZ])
z_mean = np.average(z_values, weights=z_weights)
z_median = np.median(z_values)

print(f"\n  ■■■■■■■■■■:")
print(f"    ■■■■ z = {z_mean:.3f} (S/N■■)")
print(f"    ■■■ z = {z_median:.3f}")

Z_L_BEST = z_mean

# ======================================================================
# 6. ■■■ z ■■■■■■■■
# ======================================================================
print(f"\n{'='*70}")
print(f"[Step 3] z_lens = {Z_L_BEST:.3f} ■■■■■■■■")
print(f"{'='*70}")

Scr_best = Sigma_cr(Z_L_BEST, Z_S_EFF)
Dl_best = D_A(Z_L_BEST)
print(f"  D_A = {Dl_best:.0f} Mpc")
print(f"  Sigma_cr = {Scr_best:.2e} Msun/Mpc^2")

# NFW ■■■■
def chi2_nfw_best(params):
    lm, lc = params
    M, c = 10**lm, 10**lc
    if c&lt;1 or c&gt;20 or M&lt;1e12 or M&gt;1e16: return 1e20
    try: return np.sum(((gt_fit - nfw_gamma_t(r_fit, M, c, Z_L_BEST, Z_S_EFF))/err_fit)**2)
    except: return 1e20

best_c2 = np.inf; best_p = (14, 0.5)
for lm in np.linspace(13, 15.5, 20):
    for lc in np.linspace(0, 1.2, 15):
        c2 = chi2_nfw_best((lm, lc))
        if c2 &lt; best_c2: best_c2 = c2; best_p = (lm, lc)
try:
    res = optimize.minimize(chi2_nfw_best, best_p, method='Nelder-Mead',
                            options={'xatol':0.001, 'fatol':0.01})
    if res.fun &lt; best_c2: best_c2 = res.fun; best_p = res.x
except: pass

M200_best = 10**best_p[0]; c200_best = 10**best_p[1]
rs_best, _, r200_best = nfw_params(M200_best, c200_best, Z_L_BEST)
chi2_nfw_val = best_c2
dof_nfw = len(r_fit) - 2

gt_nfw = nfw_gamma_t(r_fit, M200_best, c200_best, Z_L_BEST, Z_S_EFF)

print(f"\n  NFW ■■■■:")
print(f"    M_200 = {M200_best:.2e} Msun")
print(f"    c_200 = {c200_best:.2f}")
print(f"    r_200 = {r200_best:.2f} Mpc, r_s = {rs_best:.3f} Mpc")
print(f"    chi2/dof = {chi2_nfw_val:.1f}/{dof_nfw} = {chi2_nfw_val/dof_nfw:.2f}")

# MOND
gt_mond = mond_gamma_t(r_fit, M200_best, c200_best, Z_L_BEST, Z_S_EFF, a0)
chi2_mond = np.sum(((gt_fit - gt_mond)/err_fit)**2)
dof_mond = len(r_fit) - 1

print(f"\n  MOND (g_c = a0):")
print(f"    chi2/dof = {chi2_mond:.1f}/{dof_mond} = {chi2_mond/dof_mond:.2f}")

# ■■■■■g_c ■■■
def chi2_gc_final(lgc):
    try:
        gt_m = mond_gamma_t(r_fit, M200_best, c200_best, Z_L_BEST, Z_S_EFF, 10**lgc)
        return np.sum(((gt_fit - gt_m)/err_fit)**2)
    except: return 1e20

gc_grid = np.logspace(-12, -8, 200)
c2_grid = np.array([chi2_gc_final(np.log10(g)) for g in gc_grid])
gc_best = gc_grid[np.argmin(c2_grid)]
chi2_gc_val = c2_grid.min()
dof_gc = len(r_fit) - 2

gt_membrane = mond_gamma_t(r_fit, M200_best, c200_best, Z_L_BEST, Z_S_EFF, gc_best)

print(f"\n  ■■■■ (g_c free):")
print(f"    g_c = {gc_best:.2e} m/s^2 = {gc_best/a0:.3f} a0")
print(f"    chi2/dof = {chi2_gc_val:.1f}/{dof_gc} = {chi2_gc_val/dof_gc:.2f}")
# g_c ■■■■
```

```
dchi2 = c2_grid - chi2_gc_val
mask68 = dchi2 &lt; 1.0; mask95 = dchi2 &lt; 4.0
gc_lo68 = gc_grid[mask68].min() if mask68.sum()&gt;0 else gc_best
gc_hi68 = gc_grid[mask68].max() if mask68.sum()&gt;0 else gc_best
gc_lo95 = gc_grid[mask95].min() if mask95.sum()&gt;0 else gc_best
gc_hi95 = gc_grid[mask95].max() if mask95.sum()&gt;0 else gc_best

print(f"    68% CI: [{gc_lo68/a0:.3f}, {gc_hi68/a0:.3f}] a0")
print(f"    95% CI: [{gc_lo95/a0:.3f}, {gc_hi95/a0:.3f}] a0")
a0_in_68 = gc_lo68 &lt;= a0 &lt;= gc_hi68
a0_in_95 = gc_lo95 &lt;= a0 &lt;= gc_hi95
print(f"    a0 in 68% CI: {'YES' if a0_in_68 else 'NO'}")
print(f"    a0 in 95% CI: {'YES' if a0_in_95 else 'NO'}")

# ============================================================================
# 7. ■■■■■■
# ============================================================================
print(f"\n{'='*70}")
print(f"[Step 4] ■■■■■ (z_lens = {Z_L_BEST:.3f})")
print(f"{'='*70}")

aic_nfw = chi2_nfw_val + 2*2
aic_mond = chi2_mond + 2*1
aic_mem = chi2_gc_val + 2*2

print(f"\n  {'■■■':&lt;35} {'chi2':&gt;8} {'dof':&gt;5} {'chi2/dof':&gt;9} {'AIC':&gt;8} {'dAIC':&gt;8}")
print(f"  {'-'*78}")
print(f"  {'NFW (M200, c200)':&lt;35} {chi2_nfw_val:&gt;8.1f} {dof_nfw:&gt;5} {chi2_nfw_val/dof_nfw:&gt;9.2f} {aic_nfw:&gt;8.1f} {0:&gt;+8.
print(f"  {'MOND (g_c=a0)':&lt;35} {chi2_mond:&gt;8.1f} {dof_mond:&gt;5} {chi2_mond/dof_mond:&gt;9.2f} {aic_mond:&gt;8.1f} {aic_mond-aic_nf
print(f"  {'■■■■ (g_c free)':&lt;35} {chi2_gc_val:&gt;8.1f} {dof_gc:&gt;5} {chi2_gc_val/dof_gc:&gt;9.2f} {aic_mem:&gt;8.1f} {aic_mem-aic_n

# z=0.35 ■■■■
print(f"\n  z_lens=0.35■■■vs z_lens={Z_L_BEST:.3f}■■■■■■:")
print(f"    ■: g_c = 0.733 a0, chi2_MOND=29.7")
print(f"    ■: g_c = {gc_best/a0:.3f} a0, chi2_MOND={chi2_mond:.1f}")

# ============================================================================
# 8. g_c(z) ■■■■
# ============================================================================
print(f"\n{'='*70}")
print(f"[Step 5] g_c(z_lens) ■■■■")
print(f"{'='*70}")

print(f"\n  z_lens ■■■■■■ g_c/a0 ■■■:")
for row in gc_vs_z:
    print(f"    z={row['z_l']:.2f}: g_c = {row['gc_a0']:.3f} a0")

if len(df_zscan) &gt; 3:
    gc_a0_vals = df_zscan['gc_a0'].values
    print(f"\n    g_c/a0 ■■■: [{gc_a0_vals.min():.3f}, {gc_a0_vals.max():.3f}]")
    print(f"    g_c/a0 ■■■■: {np.median(gc_a0_vals):.3f}")
    print(f"    g_c/a0 ■■■■■: {np.std(gc_a0_vals):.3f}")

# ============================================================================
# 9. SPARC ■■■■■■
# ============================================================================
print(f"\n{'='*70}")
print(f"[Step 6] SPARC ■■■■■■■■■■")
print(f"{'='*70}")

print(f"""
  ■■■■        ■■           g_c/a0          ■■
  ■■ (SPARC)    RAR ■■■■   0.825 +/- CV~1  ■■■■■■■■■
  ■■ (SPARC)    ■■■■■■■   alpha=0.545     ■■■■■■
  ■■■■■■(HSC) ■■■■■(z=0.35) 0.733 +/- 0.12  ■■■
  ■■■■■■(HSC) ■■■■■(z={Z_L_BEST:.2f}) {gc_best/a0:.3f} +/- ?  ■■■
""")

# ============================================================================
# 10. ■■■■■
# ============================================================================
print(f"\n{'='*70}")
print(f"[Step 7] ■■■■■")
print(f"{'='*70}")

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle(f'NFW vs MOND vs Membrane ($z_l$={Z_L_BEST:.3f}, $z_s$={Z_S_EFF})',
             fontsize=14, fontweight='bold')

# (a) ■■■■■■■■■ + ■■■
ax = axes[0, 0]
ax.errorbar(r_fit, gt_fit*1e3, yerr=err_fit*1e3,
           fmt='o', color='black', ms=6, capsize=3, label='HSC Y3 stacked', zorder=5)
ax.plot(r_fit, gt_nfw*1e3, 'r-', lw=2, label=f'NFW (M={M200_best:.1e})')
ax.plot(r_fit, gt_mond*1e3, 'b--', lw=2, label=f'MOND ($g_c$=$a_0$)')
ax.plot(r_fit, gt_membrane*1e3, 'g-.', lw=2, label=f'Membrane ($g_c$={gc_best/a0:.2f}$a_0$)')
ax.set_xscale('log')
```

```
ax.set_xlabel('R [arcmin]')
ax.set_ylabel('$\\gamma_t \\times 10^3$')
ax.set_title(f'(a) Shear profile ($z_l$={Z_L_BEST:.2f})')
ax.legend(fontsize=7)

# (b) ■■
ax = axes[0, 1]
ax.errorbar(r_fit, (gt_fit-gt_nfw)/err_fit, fmt='o', color='red', ms=5, label='NFW')
ax.errorbar(r_fit*1.05, (gt_fit-gt_mond)/err_fit, fmt='s', color='blue', ms=5, label='MOND')
ax.errorbar(r_fit*1.1, (gt_fit-gt_membrane)/err_fit, fmt='^', color='green', ms=5, label='Membrane')
ax.axhline(0, color='k', ls='--', alpha=0.3)
ax.axhspan(-2, 2, alpha=0.05, color='gray')
ax.set_xscale('log')
ax.set_xlabel('R [arcmin]')
ax.set_ylabel('Residual / $\\sigma$')
ax.set_title('(b) Residuals')
ax.legend(fontsize=8)

# (c) g_c chi2 ■■■■■■
ax = axes[0, 2]
ax.plot(gc_grid/a0, dchi2, 'b-', lw=2)
ax.axhline(1, color='orange', ls='--', alpha=0.5, label='68% CL')
ax.axhline(4, color='red', ls='--', alpha=0.5, label='95% CL')
ax.axvline(1.0, color='gray', ls=':', alpha=0.5, label='$a_0$ (MOND)')
ax.axvline(gc_best/a0, color='green', ls='-', alpha=0.7, label=f'best={gc_best/a0:.2f}$a_0$')
ax.set_xlabel('$g_c / a_0$')
ax.set_ylabel('$\\Delta\\chi^2$')
ax.set_title('(c) $g_c$ constraint')
ax.set_xscale('log')
ax.set_ylim(0, min(20, dchi2[np.isfinite(dchi2)].max() if np.any(np.isfinite(dchi2)) else 20))
ax.legend(fontsize=7)

# (d) g_c(z_lens) ■■■■
ax = axes[1, 0]
if len(df_zscan) &gt; 0:
    ax.plot(df_zscan['z_l'], df_zscan['gc_a0'], 'o-', color='steelblue', ms=6)
    ax.axhline(1.0, color='gray', ls='--', alpha=0.5, label='$a_0$')
    ax.axhline(0.825, color='orange', ls=':', alpha=0.5, label='SPARC (0.825)')
    ax.axvline(Z_L_BEST, color='red', ls='--', alpha=0.5, label=f'$z_l$={Z_L_BEST:.2f}')
    # ■■■■■■■■■ z ■■■
    for _, _, _, z_cl, sn, _ in CLUSTERS_PHOTOZ:
        ax.axvline(z_cl, color='green', ls=':', alpha=0.3)
ax.set_xlabel('$z_{lens}$ (assumed)')
ax.set_ylabel('$g_c / a_0$')
ax.set_title('(d) $g_c$ stability vs $z_l$')
ax.legend(fontsize=7)

# (e) chi2/dof ■■
ax = axes[1, 1]
models = ['NFW', f'MOND\n($g_c$=$a_0$)', f'Membrane\n($g_c$={gc_best/a0:.2f}$a_0$)']
chi2dof = [chi2_nfw_val/dof_nfw, chi2_mond/dof_mond, chi2_gc_val/dof_gc]
cols = ['red', 'blue', 'green']
bars = ax.bar(models, chi2dof, color=cols, alpha=0.7, edgecolor='black')
ax.axhline(1, color='k', ls='--', alpha=0.3)
for b, v in zip(bars, chi2dof):
    ax.text(b.get_x()+b.get_width()/2, b.get_height()+0.02, f'{v:.2f}',
            ha='center', fontsize=10, fontweight='bold')
ax.set_ylabel('$\\chi^2$/dof')
ax.set_title(f'(e) Model comparison ($z_l$={Z_L_BEST:.2f})')

# (f) z=0.35 vs z=new ■■■
ax = axes[1, 2]
# z ■■ MOND chi2 ■ membrane chi2
if len(df_zscan) &gt; 0:
    ax.plot(df_zscan['z_l'], df_zscan['chi2_nfw'], 'r-o', ms=4, label='NFW')
    ax.plot(df_zscan['z_l'], df_zscan['chi2_mond'], 'b--s', ms=4, label='MOND')
    ax.plot(df_zscan['z_l'], df_zscan['chi2_membrane'], 'g-.^', ms=4, label='Membrane')
    ax.axvline(Z_L_BEST, color='red', ls='--', alpha=0.5)
    ax.set_xlabel('$z_{lens}$')
    ax.set_ylabel('$\\chi^2$')
    ax.set_title('(f) $\\chi^2$ vs $z_l$')
    ax.legend(fontsize=7)

plt.tight_layout()
plt.savefig('hsc_refit_photoz.png', dpi=150, bbox_inches='tight')
print("  -&gt; hsc_refit_photoz.png")

# CSV ■■
df_zscan.to_csv('hsc_gc_vs_zlens.csv', index=False)
print("  -&gt; hsc_gc_vs_zlens.csv")

# ======================================================================
# 11. ■■■■■■
# ======================================================================
print(f"\n{'='*70}")
print(f"[■■■■■■■]")
print(f"{'='*70}")
```

```
print(f"""
  photo-z ■■■■■■ (z_lens = {Z_L_BEST:.3f}):

  NFW:     M200={M200_best:.2e}, c200={c200_best:.2f}, chi2/dof={chi2_nfw_val/dof_nfw:.2f}
  MOND:    g_c=a0, chi2/dof={chi2_mond/dof_mond:.2f}, dAIC={aic_mond-aic_nfw:+.1f}
  ■■■■: g_c={gc_best/a0:.3f}a0, chi2/dof={chi2_gc_val/dof_gc:.2f}, dAIC={aic_mem-aic_nfw:+.1f}

  g_c ■■■:
    g_c = {gc_best:.2e} m/s^2 = {gc_best/a0:.3f} a0
    68% CI: [{gc_lo68/a0:.3f}, {gc_hi68/a0:.3f}] a0
    95% CI: [{gc_lo95/a0:.3f}, {gc_hi95/a0:.3f}] a0
    a0 in 95% CI: {'YES' if a0_in_95 else 'NO'}

  SPARC ■■■■■■■■:
    SPARC ■■■■: g_c = 0.825 a0 (CV~1)
    HSC ■■■■:   g_c = {gc_best/a0:.3f} a0
    ■■■: {'2sigma■■■■■' if abs(gc_best/a0 - 0.825) &lt; 2*0.15 else '■■■■■'}

  ■■■■■■■:
    ■■■■ (g_c &lt; a0) ■■■■■■■■■■ → Level {'A' if not a0_in_95 else 'B'}
""")

print("■■")
```

specz_crossmatch.py

解析目的	SDSS SkyServer/Vizier TAP で分光赤方偏移を検索。cl1 の z_spec 確定。
結果	cl1: z_spec=0.313（22銀河, sigma_v=527 km/s）。photo-z=0.789 は過大。

```
"""
■■■■■■■■■■■■
=======================
HSC ■■■■■■■■■■■■■■ SDSS ■■■■■■■■■■■
■■■■■■■■■■■■■:
  (1) SDSS DR18 CasJobs API
  (2) Vizier TAP (SDSS spectroscopic catalog)
  (3) NED batch query

■■: uv run --with requests --with pandas --with numpy --with astropy python specz_crossmatch.py
"""

import numpy as np
import pandas as pd
from pathlib import Path
import time
import sys

try:
    import requests
except ImportError:
    print("pip install requests"); sys.exit(1)

try:
    from astropy.coordinates import SkyCoord
    from astropy import units as u
    HAS_ASTROPY = True
except:
    HAS_ASTROPY = False
    print("  astropy ■■: ■■■■■■■■■■■■■■")

# ============================================================================
# 0. ■■■■■■■■
# ============================================================================
CLUSTERS = [
    {"name": "cl1", "ra": 140.450, "dec": -0.251, "z_phot": 0.789, "label": "GAMA09H #1"},
    {"name": "cl2", "ra": 142.210, "dec": -0.120, "z_phot": 0.670, "label": "GAMA09H #2"},
    {"name": "cl3", "ra": 140.300, "dec": -0.350, "z_phot": 0.370, "label": "GAMA09H #3"},
    {"name": "cl4", "ra": 182.680, "dec": -0.280, "z_phot": 0.510, "label": "WIDE12H"},
    {"name": "cl5", "ra": 216.750, "dec": -0.200, "z_phot": 0.610, "label": "GAMA15H"},
]

SEARCH_RADIUS_ARCMIN = 5.0
OUTPUT_DIR = Path(r"D:\■■■■■■■\■■■■■■■\■■■■■■\■■■■■■■■\■■■■")

print("="*70)
print("■■■■■■■■■■■■■■")
print("="*70)

# ============================================================================
# 1. Vizier TAP ■■■■SDSS DR18 SpecObj■
# ============================================================================
def query_vizier_sdss_specz(ra, dec, radius_arcmin=5.0):
    """Vizier TAP ■ SDSS ■■■■■■■■■■"""
    radius_deg = radius_arcmin / 60.0

    # SDSS DR18 spectroscopic catalog via Vizier TAP
    tap_url = "https://tapvizier.cds.unistra.fr/TAPVizieR/tap/sync"

    query = f"""
    SELECT TOP 1000
        "V/154/sdss16", RAJ2000, DEJ2000, zsp, e_zsp, zph, SpCl, Q
    FROM "V/154/sdss16"
    WHERE 1=CONTAINS(POINT('ICRS', RAJ2000, DEJ2000),
                     CIRCLE('ICRS', {ra}, {dec}, {radius_deg}))
    AND zsp &gt; 0.01
    AND zsp &lt; 2.0
    AND Q &gt;= 2
    """

    params = {
        "REQUEST": "doQuery",
        "LANG": "ADQL",
        "FORMAT": "csv",
        "QUERY": query,
    }

    try:
        r = requests.get(tap_url, params=params, timeout=60)
        if r.status_code == 200 and len(r.text) &gt; 10:
```

```
            from io import StringIO
            df = pd.read_csv(StringIO(r.text), comment='#')
            return df
    except Exception as e:
        print(f"      Vizier ■■■: {e}")

    return None

# ======================================================================
# 2. SDSS SkyServer API■■■■■■■
# ======================================================================
def query_sdss_skyserver(ra, dec, radius_arcmin=5.0):
    """SDSS SkyServer SQL Search API"""

    sql = f"""
    SELECT TOP 500
        s.specObjID, s.ra, s.dec, s.z as z_spec, s.zErr, s.zWarning,
        s.class, s.subClass, s.velDisp, s.velDispErr,
        p.modelMag_r, p.modelMag_i, p.modelMag_g
    FROM SpecObj AS s
    JOIN PhotoObj AS p ON s.bestObjID = p.objID
    WHERE
        dbo.fGetNearbySpecObjEq({ra}, {dec}, {radius_arcmin}) IS NOT NULL
        AND s.z &gt; 0.01 AND s.z &lt; 2.0
        AND s.zWarning = 0
        AND s.class = 'GALAXY'
    """

    url = "https://skyserver.sdss.org/dr18/SkyServerWS/SearchTools/SqlSearch"
    params = {"cmd": sql, "format": "csv"}

    try:
        r = requests.get(url, params=params, timeout=60)
        if r.status_code == 200:
            from io import StringIO
            lines = r.text.strip().split('\n')
            # SDSS returns header + data, sometimes with error messages
            if len(lines) &gt; 1 and not lines[0].startswith('ERROR'):
                df = pd.read_csv(StringIO(r.text))
                return df
    except Exception as e:
        print(f"      SkyServer ■■■: {e}")

    return None

# ======================================================================
# 3. NED ■■■■■■
# ======================================================================
def query_ned(ra, dec, radius_arcmin=5.0):
    """NASA/IPAC Extragalactic Database (NED) near position search"""
    url = "https://ned.ipac.caltech.edu/cgi-bin/objsearch"
    params = {
        "search_type": "Near Position Search",
        "in_csys": "Equatorial",
        "in_equinox": "J2000.0",
        "lon": f"{ra:.5f}d",
        "lat": f"{dec:.5f}d",
        "radius": f"{radius_arcmin}",
        "hconst": "70",
        "omegam": "0.3",
        "omegav": "0.7",
        "corr_z": "1",
        "z_constraint": "Unconstrained",
        "ot_include": "ANY",
        "nmp_op": "ANY",
        "out_csys": "Equatorial",
        "out_equinox": "J2000.0",
        "obj_sort": "Distance to search center",
        "of": "ascii_tab",
        "zv_breaker": "30000.0",
        "list_limit": "100",
        "img_stamp": "NO",
    }

    try:
        r = requests.get(url, params=params, timeout=60)
        if r.status_code == 200 and len(r.text) &gt; 100:
            from io import StringIO
            # NED ■■■■■■■■■■■■■■■■■■
            lines = r.text.split('\n')
            data_lines = [l for l in lines if l and not l.startswith('#') and not l.startswith('|')]
            if len(data_lines) &gt; 1:
                return r.text  # ■■■■■■■■
    except Exception as e:
        print(f"      NED ■■■: {e}")

    return None
```

```
# ======================================================================
# 4. ■■■■■
# ======================================================================
print(f"\n  ■■■■: {SEARCH_RADIUS_ARCMIN} arcmin")
print(f"  ■■■: {len(CLUSTERS)}")

all_results = {}

for cl in CLUSTERS:
    name = cl['name']
    ra = cl['ra']
    dec = cl['dec']
    z_phot = cl['z_phot']
    label = cl['label']

    print(f"\n{'='*60}")
    print(f"  {name}: {label} (RA={ra:.3f}, Dec={dec:.3f}, z_phot={z_phot:.3f})")
    print(f"{'='*60}")

    results = {'name': name, 'ra': ra, 'dec': dec, 'z_phot': z_phot,
               'n_specz': 0, 'z_spec_peak': None, 'z_spec_median': None,
               'n_cluster_members': 0, 'vel_disp': None}

    # --- ■■1: SDSS SkyServer ---
    print(f"    [1] SDSS SkyServer ■■■...")
    df_sdss = query_sdss_skyserver(ra, dec, SEARCH_RADIUS_ARCMIN)

    if df_sdss is not None and len(df_sdss) &gt; 0:
        print(f"        ■■■■■: {len(df_sdss)}")

        # z_spec ■■
        z_col = 'z_spec' if 'z_spec' in df_sdss.columns else 'z' if 'z' in df_sdss.columns else None
        if z_col:
            z_specs = df_sdss[z_col].dropna().values
            z_specs = z_specs[(z_specs &gt; 0.01) &amp; (z_specs &lt; 2.0)]

            if len(z_specs) &gt; 0:
                results['n_specz'] = len(z_specs)
                results['z_spec_median'] = np.median(z_specs)

                # photo-z ■■■+/- 0.05■■■■■■
                dz = 0.05
                members = z_specs[(z_specs &gt; z_phot-dz) &amp; (z_specs &lt; z_phot+dz)]
                results['n_cluster_members'] = len(members)

                if len(members) &gt; 3:
                    results['z_spec_peak'] = np.median(members)
                    results['vel_disp'] = np.std(members) * 3e5  # km/s
                    print(f"        z_spec(■■■■): {results['z_spec_peak']:.4f} "
                          f"({len(members)}■■, sigma_v={results['vel_disp']:.0f} km/s)")

                # z ■■■■■■
                print(f"        z ■■:")
                zbins = np.arange(0, 1.2, 0.05)
                hist, _ = np.histogram(z_specs, bins=zbins)
                for i, h in enumerate(hist):
                    if h &gt; 0:
                        bar = '#' * h
                        print(f"          z={zbins[i]:.2f}-{zbins[i+1]:.2f}: {bar} ({h})")

                # ■■■■■■■■velDisp ■■■■■■■■
                if 'velDisp' in df_sdss.columns:
                    vd = df_sdss['velDisp'].dropna()
                    vd = vd[vd &gt; 50]  # 50 km/s ■■■■
                    if len(vd) &gt; 0:
                        print(f"        SDSS ■■■■: median={vd.median():.0f} km/s (N={len(vd)})")

        # CSV ■■
        outpath = OUTPUT_DIR / f"specz_{name}_sdss.csv"
        df_sdss.to_csv(outpath, index=False)
        print(f"        -&gt; {outpath}")
    else:
        print(f"        SDSS ■■■■■■■■■")

    # --- ■■2: Vizier ---
    if results['n_specz'] == 0:
        print(f"    [2] Vizier ■■■...")
        df_viz = query_vizier_sdss_specz(ra, dec, SEARCH_RADIUS_ARCMIN)
        if df_viz is not None and len(df_viz) &gt; 0:
            print(f"        Vizier ■■: {len(df_viz)} ■■")
            z_col_v = [c for c in df_viz.columns if 'zsp' in c.lower() or 'z_spec' in c.lower()]
            if z_col_v:
                z_v = df_viz[z_col_v[0]].dropna().values
                z_v = z_v[(z_v &gt; 0.01) &amp; (z_v &lt; 2.0)]
                results['n_specz'] = len(z_v)
                if len(z_v) &gt; 0:
                    results['z_spec_median'] = np.median(z_v)
```

```
                    members = z_v[(z_v &gt; z_phot-0.05) &amp; (z_v &lt; z_phot+0.05)]
                    results['n_cluster_members'] = len(members)
                    if len(members) &gt; 3:
                        results['z_spec_peak'] = np.median(members)
                        results['vel_disp'] = np.std(members) * 3e5
                        print(f"        z_spec(■■■■): {results['z_spec_peak']:.4f}")

    time.sleep(1)  # API ■■■■

    all_results[name] = results

# ======================================================================
# 5. ■■■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[■■■■■] ■■■■■■■■■■■■■■■")
print(f"{'='*70}")

print(f"\n  {'■■':&lt;8} {'z_phot':&gt;7} {'N_spec':&gt;7} {'N_member':&gt;9} {'z_spec':&gt;7} {'sigma_v':&gt;8} {'■■':&lt;15}")
print(f"  {'-'*65}")

confirmed = []
for name, res in all_results.items():
    z_phot = res['z_phot']
    n_spec = res['n_specz']
    n_mem = res['n_cluster_members']
    z_spec = res.get('z_spec_peak')
    vd = res.get('vel_disp')

    if z_spec and n_mem &gt;= 5 and vd and vd &gt; 200:
        status = "■■■■■"
        confirmed.append(res)
    elif z_spec and n_mem &gt;= 3:
        status = "■■"
        confirmed.append(res)
    elif n_spec &gt; 0:
        status = "■■■■■"
    else:
        status = "■■■■■"

    z_str = f"{z_spec:.4f}" if z_spec else "--"
    vd_str = f"{vd:.0f}" if vd else "--"
    print(f"  {name:&lt;8} {z_phot:&gt;7.3f} {n_spec:&gt;7} {n_mem:&gt;9} {z_str:&gt;7} {vd_str:&gt;8} {status:&lt;15}")

print(f"\n  ■■■■/■■■■■■■■: {len(confirmed)} / {len(CLUSTERS)}")

# ■■■■■ z ■■■■■■■■
if confirmed:
    z_confirmed = [c['z_spec_peak'] for c in confirmed if c['z_spec_peak']]
    n_members = [c['n_cluster_members'] for c in confirmed if c['z_spec_peak']]
    if z_confirmed:
        z_avg = np.average(z_confirmed, weights=n_members)
        print(f"\n  ■■■■■■■ z ■■■■■■: {z_avg:.4f}")
        print(f"  photo-z ■■■■:")
        for c in confirmed:
            if c['z_spec_peak']:
                dz = c['z_spec_peak'] - c['z_phot']
                print(f"    {c['name']}: z_phot={c['z_phot']:.3f}, z_spec={c['z_spec_peak']:.4f}, "
                      f"dz={dz:+.4f}")

# ■■CSV
df_summary = pd.DataFrame(all_results.values())
df_summary.to_csv(OUTPUT_DIR / "specz_summary.csv", index=False)
print(f"\n  -&gt; specz_summary.csv")
print(f"""
  ■■■■■■■:
    ■■■■■■■■ z_spec ■ hsc_refit_with_photoz.py ■ Z_L_BEST ■■■
    → g_c ■■■■■■■■■■■
""")

print("■■")
```

mond_abel_lensing.py

解析目的	Abel 変換による正確な MOND 投影計算。NFWスケーリング近似を置換。
結果	Abel変換でNFW>>MOND に逆転。近似アーティファクトの確認。

```
"""
MOND/■■■■■■■■■: Abel■■■■■■■■■■■■■
================================================
■■■■■■■■■■: MOND ■■■■ NFW ■■■■■■■■■■■■■■■■
■■■■■■■: Abel ■■■■ 3D ■■■■ → 2D ■■■■■■■■■■■

■■:
  MOND: g_obs(r) = (1/2)(g_N + sqrt(g_N^2 + 4*g_c*g_N))
  ■■■■■: rho_eff(r) = (1/4pi*G) * (1/r^2) * d/dr[r^2 * g_obs(r)]
  ■■■■■■: Sigma(R) = 2 * integral_R^inf rho_eff(r) * r / sqrt(r^2-R^2) dr
  DeltaSigma(R) = Sigma_mean(&lt;R) - Sigma(R)
  gamma_t(R) = DeltaSigma(R) / Sigma_cr

■■: uv run --with pandas --with numpy --with scipy --with matplotlib python mond_abel_lensing.py
"""

import numpy as np
import pandas as pd
from scipy import integrate, optimize, interpolate
from pathlib import Path
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ======================================================================
# 0. ■■
# ======================================================================
H0 = 70.0; Om = 0.3; OL = 0.7; c_light = 2.998e5
G_Mpc = 4.301e-9       # (km/s)^2 Mpc / Msun
G_SI = 6.674e-11        # m^3 / (kg s^2)
Msun = 1.989e30         # kg
Mpc_m = 3.086e22        # m
a0 = 1.2e-10            # m/s^2
pc_m = 3.086e16         # m

# ■■■■■■ z■cl1 ■■■■■■■■
Z_L = 0.313  # cl1 ■■■■ (22■■, sigma_v=527 km/s)
Z_S = 1.0    # ■■■■■■ z

print("="*70)
print("MOND/■■■■■■■■■: Abel ■■■■■■■■■■■■■")
print("="*70)

# ======================================================================
# 1. ■■■■■
# ======================================================================
def E(z): return np.sqrt(Om*(1+z)**3 + OL)
def comoving(z):
    d, _ = integrate.quad(lambda zz: c_light/(H0*E(zz)), 0, z)
    return d
def D_A(z): return comoving(z)/(1+z)
def D_A2(z1, z2): return (comoving(z2)-comoving(z1))/(1+z2)
def rho_cr(z): return 3*(H0*E(z))**2/(8*np.pi*G_Mpc)

def Sigma_cr(z_l, z_s):
    Dl=D_A(z_l); Ds=D_A(z_s); Dls=D_A2(z_l,z_s)
    if Dls&lt;=0: return np.inf
    return c_light**2/(4*np.pi*G_Mpc) * Ds/(Dl*Dls)

Scr = Sigma_cr(Z_L, Z_S)
Dl = D_A(Z_L)
print(f"  z_l={Z_L}, z_s={Z_S}")
print(f"  D_A = {Dl:.0f} Mpc, Sigma_cr = {Scr:.3e} Msun/Mpc^2")

# ======================================================================
# 2. ■■■■■■■■■■■■■■■■■■■■ + NFW■■■
# ======================================================================
def nfw_enclosed_mass(r_Mpc, M200, c200, z_l):
    """NFW ■■■■■■"""
    rc = rho_cr(z_l)
    r200 = (3*M200/(4*np.pi*200*rc))**(1./3.)
    rs = r200/c200
    delta_c = 200./3.*c200**3/(np.log(1+c200)-c200/(1+c200))
    rho_s = delta_c*rc
    x = r_Mpc/rs
    return 4*np.pi*rho_s*rs**3*(np.log(1+x)-x/(1+x))

def nfw_density(r_Mpc, M200, c200, z_l):
    """NFW ■■"""
    rc = rho_cr(z_l)
```

```
    r200 = (3*M200/(4*np.pi*200*rc))**(1./3.)
    rs = r200/c200
    delta_c = 200./3.*c200**3/(np.log(1+c200)-c200/(1+c200))
    rho_s = delta_c*rc
    x = r_Mpc/rs
    return rho_s/(x*(1+x)**2)

# ======================================================================
# 3. MOND/■ ■■■■■
# ======================================================================
def g_newton(r_Mpc, M_bar_enclosed):
    """■■■■■■■■■ [m/s^2]"""
    r_m = r_Mpc * Mpc_m
    M_kg = M_bar_enclosed * Msun
    if r_m &lt; 1e10: return 0
    return G_SI * M_kg / r_m**2

def g_mond(g_N, g_c):
    """MOND ■■■■■■ [m/s^2]"""
    if g_N &lt;= 0: return 0
    return 0.5*(g_N + np.sqrt(g_N**2 + 4*g_c*g_N))

def effective_enclosed_mass(r_Mpc, M_bar_enclosed, g_c):
    """MOND ■■■■■■■■ [Msun]"""
    r_m = r_Mpc * Mpc_m
    gN = g_newton(r_Mpc, M_bar_enclosed)
    gobs = g_mond(gN, g_c)
    M_eff = gobs * r_m**2 / G_SI / Msun
    return M_eff

# ======================================================================
# 4. Abel ■■■■■■■■■■
# ======================================================================
def compute_rho_eff(r_array, M200_total, c200, z_l, g_c, f_bar=0.17):
    """MOND ■■3D■■ rho_eff(r) ■■■■■■"""
    n = len(r_array)
    M_eff = np.zeros(n)

    for i, r in enumerate(r_array):
        M_bar = nfw_enclosed_mass(r, M200_total, c200, z_l) * f_bar
        M_eff[i] = effective_enclosed_mass(r, M_bar, g_c)

    # rho_eff = (1/4pi*r^2) * dM_eff/dr
    # ■■■■■■■■■■■
    rho_eff = np.zeros(n)
    for i in range(1, n-1):
        dMdr = (M_eff[i+1] - M_eff[i-1]) / (r_array[i+1] - r_array[i-1])
        rho_eff[i] = dMdr / (4*np.pi*r_array[i]**2)

    # ■■■■■■
    rho_eff[0] = rho_eff[1]
    rho_eff[-1] = rho_eff[-2]

    # ■■■■■■■■■■■■■■■■■
    rho_eff = np.maximum(rho_eff, 0)

    return rho_eff, M_eff

def abel_project(R_array, r_fine, rho_fine):
    """Abel ■■: 3D■■ → 2D■■■
    Sigma(R) = 2 * integral_R^inf rho(r) * r / sqrt(r^2 - R^2) dr
    """
    # rho ■■■■■■
    log_r = np.log10(r_fine)
    log_rho = np.log10(np.maximum(rho_fine, 1e-50))
    rho_interp = interpolate.interp1d(log_r, log_rho, kind='linear',
                                      fill_value=-50, bounds_error=False)

    def rho_func(r):
        return 10**rho_interp(np.log10(r))

    Sigma = np.zeros(len(R_array))
    r_max = r_fine.max()

    for i, R in enumerate(R_array):
        if R &lt;= 0: continue
        # ■■■■■■: rho(r) * r / sqrt(r^2 - R^2)
        # ■■■ r=R ■■■: ■■■■■ u = sqrt(r^2 - R^2), dr = u/r * du
        # integral = integral_0^umax rho(sqrt(u^2+R^2)) du
        u_max = np.sqrt(r_max**2 - R**2) if r_max &gt; R else 0
        if u_max &lt;= 0:
            Sigma[i] = 0
            continue

        def integrand(u):
            r = np.sqrt(u**2 + R**2)
            return rho_func(r)
```

```
        try:
            val, err = integrate.quad(integrand, 0, u_max, limit=200,
                                      epsrel=1e-4, epsabs=0)
            Sigma[i] = 2 * val  # ■■■■
        except:
            Sigma[i] = 0

    return Sigma

def compute_delta_sigma(R_array, Sigma_array):
    """DeltaSigma(R) = Sigma_mean(&lt;R) - Sigma(R)"""
    # Sigma_mean(&lt;R) = (2/R^2) * integral_0^R Sigma(R') * R' dR'
    Sigma_mean = np.zeros(len(R_array))

    # ■■
    Sigma_interp = interpolate.interp1d(R_array, Sigma_array, kind='linear',
                                        fill_value=0, bounds_error=False)

    for i, R in enumerate(R_array):
        if R &lt;= 0: continue
        def integrand(Rp):
            return Sigma_interp(Rp) * Rp
        try:
            val, _ = integrate.quad(integrand, R_array[0]*0.1, R, limit=100)
            Sigma_mean[i] = 2 * val / R**2
        except:
            Sigma_mean[i] = 0

    return Sigma_mean - Sigma_array

# ============================================================================
# 5. ■■■■■■■■■■■■■
# ============================================================================
def full_mond_shear(R_arcmin, M200, c200, z_l, z_s, g_c, f_bar=0.17):
    """Abel ■■■■■■■■■■ MOND/■ ■■■■■■■■■"""
    Dl_val = D_A(z_l)
    Scr_val = Sigma_cr(z_l, z_s)

    # arcmin → Mpc
    R_Mpc = R_arcmin * np.pi/(180*60) * Dl_val

    # 3D ■■■■■■■■■■■■■
    r_min = R_Mpc.min() * 0.1
    r_max = R_Mpc.max() * 5
    r_fine = np.logspace(np.log10(max(r_min, 1e-4)), np.log10(r_max), 500)

    # MOND ■■■■■
    rho_eff, M_eff = compute_rho_eff(r_fine, M200, c200, z_l, g_c, f_bar)

    # Abel ■■
    Sigma = abel_project(R_Mpc, r_fine, rho_eff)

    # DeltaSigma
    DS = compute_delta_sigma(R_Mpc, Sigma)

    # ■■
    gamma_t = DS / Scr_val

    return gamma_t, Sigma, DS, rho_eff, M_eff, r_fine

# ============================================================================
# 6. NFW ■■■■■■■■■■■■Wright &amp;amp; Brainerd 2000■
# ============================================================================
def nfw_gamma_exact(R_arcmin, M200, c200, z_l, z_s):
    """Wright &amp;amp; Brainerd (2000) ■■■■■ NFW ■■"""
    rc = rho_cr(z_l)
    r200 = (3*M200/(4*np.pi*200*rc))**(1./3.)
    rs = r200/c200
    delta_c = 200./3.*c200**3/(np.log(1+c200)-c200/(1+c200))
    rho_s = delta_c*rc
    Dl_val = D_A(z_l)
    R_Mpc = R_arcmin * np.pi/(180*60) * Dl_val
    x = R_Mpc/rs

    # Sigma(x) ■ Sigma_mean(x) ■■■■■
    Sigma = np.zeros_like(x)
    Sigma_mean = np.zeros_like(x)

    for i, xi in enumerate(x):
        if xi &lt; 1e-6: continue
        elif abs(xi-1) &lt; 1e-4:
            Sigma[i] = 2*rs*rho_s/3.0
            Sigma_mean[i] = 4*rs*rho_s*(1+np.log(0.5))
        elif xi &lt; 1:
            sq = np.sqrt(1-xi**2)
            Sigma[i] = 2*rs*rho_s/(xi**2-1)*(1-np.log((1+sq)/xi)/sq)
            g = np.log(xi/2)+np.log((1+sq)/xi)/sq
```

```
            Sigma_mean[i] = 4*rs*rho_s*g/xi**2
        else:
            sq = np.sqrt(xi**2-1)
            Sigma[i] = 2*rs*rho_s/(xi**2-1)*(1-np.arctan(sq)/sq)
            g = np.log(xi/2)+np.arctan(sq)/sq
            Sigma_mean[i] = 4*rs*rho_s*g/xi**2

    DS = Sigma_mean - Sigma
    Scr_val = Sigma_cr(z_l, z_s)
    return DS/Scr_val

# ============================================================================
# 7. ■■■■■■■■■■■■
# ============================================================================
print(f"\n{'='*70}")
print("[Step 1] ■■■■■■■■")
print("="*70)

data_file = Path("hsc_y3_stacked_shear.csv")
if data_file.exists():
    df = pd.read_csv(data_file)
else:
    df = pd.DataFrame({
        'r_arcmin': [0.59,0.82,1.14,1.59,2.22,3.09,4.30,5.99,8.34,11.61,16.17,22.51],
        'gamma_t': [0.038,0.053,0.032,0.023,0.018,0.015,0.012,0.010,0.008,0.007,0.006,0.006],
        'gamma_t_err': [0.025,0.019,0.010,0.006,0.004,0.003,0.002,0.002,0.001,0.001,0.001,0.001],
    })

r_data = df['r_arcmin'].values
gt_data = df['gamma_t'].values
gt_err = df['gamma_t_err'].values
valid = np.isfinite(gt_data) &amp;amp; np.isfinite(gt_err) &amp;amp; (gt_err&amp;gt;0)
r_fit = r_data[valid]; gt_fit = gt_data[valid]; err_fit = gt_err[valid]

print(f"  ■■■■: {valid.sum()}")

# ============================================================================
# 8. NFW ■■■■
# ============================================================================
print(f"\n{'='*70}")
print("[Step 2] NFW ■■■■ (■■■)")
print("="*70)

def chi2_nfw(params):
    lm, lc = params
    M, c = 10**lm, 10**lc
    if c&amp;lt;1 or c&amp;gt;20 or M&amp;lt;1e12 or M&amp;gt;1e16: return 1e20
    try: return np.sum(((gt_fit-nfw_gamma_exact(r_fit,M,c,Z_L,Z_S))/err_fit)**2)
    except: return 1e20

best = np.inf; bp = (14,0.5)
for lm in np.linspace(13,15.5,20):
    for lc in np.linspace(0,1.2,15):
        c2 = chi2_nfw((lm,lc))
        if c2&amp;lt;best: best=c2; bp=(lm,lc)
try:
    res = optimize.minimize(chi2_nfw, bp, method='Nelder-Mead')
    if res.fun&amp;lt;best: best=res.fun; bp=res.x
except: pass

M200_nfw = 10**bp[0]; c200_nfw = 10**bp[1]
gt_nfw = nfw_gamma_exact(r_fit, M200_nfw, c200_nfw, Z_L, Z_S)
chi2_nfw_val = best; dof_nfw = len(r_fit)-2
print(f"  M200={M200_nfw:.2e}, c200={c200_nfw:.2f}")
print(f"  chi2/dof = {chi2_nfw_val:.1f}/{dof_nfw} = {chi2_nfw_val/dof_nfw:.2f}")

# ============================================================================
# 9. MOND Abel ■■■■■■
# ============================================================================
print(f"\n{'='*70}")
print("[Step 3] MOND/■■■■■■■■ (Abel ■■)")
print("="*70)

# g_c ■■■■■■■■
# M200 ■ NFW ■■■■■■■■■■■■■■■■■■■■■■
gc_grid = np.logspace(-12, -8, 60)
chi2_gc_abel = np.full(len(gc_grid), np.inf)

print(f"  g_c ■■■■■ ({len(gc_grid)} ■)...")
for ig, gc_val in enumerate(gc_grid):
    try:
        gt_model, _, _, _, _, _ = full_mond_shear(r_fit, M200_nfw, c200_nfw, Z_L, Z_S, gc_val)
        if np.all(np.isfinite(gt_model)):
            chi2_gc_abel[ig] = np.sum(((gt_fit-gt_model)/err_fit)**2)
    except Exception as e:
        pass
    if (ig+1) % 20 == 0:
```

```
        best_so_far = gc_grid[np.argmin(chi2_gc_abel[:ig+1])]
        print(f"    {ig+1}/{len(gc_grid)}: best g_c = {best_so_far/a0:.3f} a0")

# ■■■ g_c
best_idx = np.argmin(chi2_gc_abel)
gc_best_abel = gc_grid[best_idx]
chi2_abel_best = chi2_gc_abel[best_idx]

# g_c = a0 ■ chi2
a0_idx = np.argmin(np.abs(gc_grid - a0))
chi2_a0_abel = chi2_gc_abel[a0_idx]

# ■■■■■■■■■■■■■■■■
gt_abel_best, Sigma_best, DS_best, rho_best, Meff_best, r_fine = \
    full_mond_shear(r_fit, M200_nfw, c200_nfw, Z_L, Z_S, gc_best_abel)

# MOND (g_c=a0) ■■■■■■■■
gt_abel_mond, _, _, _, _, _ = full_mond_shear(r_fit, M200_nfw, c200_nfw, Z_L, Z_S, a0)

dof_abel = len(r_fit) - 2  # M200 + g_c

print(f"\n  Abel ■■ MOND/■■■■■■■:")
print(f"    g_c(best) = {gc_best_abel:.2e} m/s^2 = {gc_best_abel/a0:.3f} a0")
print(f"    chi2/dof(best) = {chi2_abel_best:.1f}/{dof_abel} = {chi2_abel_best/dof_abel:.2f}")
print(f"    chi2/dof(a0)   = {chi2_a0_abel:.1f}/{dof_abel} = {chi2_a0_abel/dof_abel:.2f}")

# ■■■■■
dchi2_abel = chi2_gc_abel - chi2_abel_best
mask68 = dchi2_abel&lt;1; mask95 = dchi2_abel&lt;4
gc_lo68 = gc_grid[mask68].min() if mask68.sum()&gt;0 else gc_best_abel
gc_hi68 = gc_grid[mask68].max() if mask68.sum()&gt;0 else gc_best_abel
gc_lo95 = gc_grid[mask95].min() if mask95.sum()&gt;0 else gc_best_abel
gc_hi95 = gc_grid[mask95].max() if mask95.sum()&gt;0 else gc_best_abel

a0_in_68 = gc_lo68&lt;=a0&lt;=gc_hi68
a0_in_95 = gc_lo95&lt;=a0&lt;=gc_hi95

print(f"    68% CI: [{gc_lo68/a0:.3f}, {gc_hi68/a0:.3f}] a0")
print(f"    95% CI: [{gc_lo95/a0:.3f}, {gc_hi95/a0:.3f}] a0")
print(f"    a0 in 68% CI: {'YES' if a0_in_68 else 'NO'}")
print(f"    a0 in 95% CI: {'YES' if a0_in_95 else 'NO'}")

# ======================================================================
# 10. NFW ■■■■■■■■■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 4] Abel ■■ vs NFW ■■■■■■■■■■■■")
print("="*70)

print(f"  ■■NFW ■■■■■■■■■■: g_c = 0.777 a0")
print(f"  ■■Abel ■■■:            g_c = {gc_best_abel/a0:.3f} a0")
print(f"  ■: {abs(gc_best_abel/a0 - 0.777):.3f} a0")

# ======================================================================
# 11. ■■■■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 5] ■■■■■■")
print("="*70)

aic_nfw = chi2_nfw_val + 2*2
aic_mond_abel = chi2_a0_abel + 2*1
aic_mem_abel = chi2_abel_best + 2*2

print(f"\n  {'■■■':&lt;40} {'chi2':&gt;8} {'dof':&gt;5} {'chi2/dof':&gt;9} {'AIC':&gt;8} {'dAIC':&gt;8}")
print(f"  {'-'*78}")
print(f"  {'NFW (■■■)':&lt;40} {chi2_nfw_val:&gt;8.1f} {dof_nfw:&gt;5} {chi2_nfw_val/dof_nfw:&gt;9.2f} {aic_nfw:&gt;8.1f} {0:&gt;+8.1f}")
print(f"  {'MOND g_c=a0 (Abel■■)':&lt;40} {chi2_a0_abel:&gt;8.1f} {dof_abel:&gt;5} {chi2_a0_abel/dof_abel:&gt;9.2f} {aic_mond_abel:&gt;8.1f
print(f"  {'■■■■ g_c free (Abel■■)':&lt;40} {chi2_abel_best:&gt;8.1f} {dof_abel:&gt;5} {chi2_abel_best/dof_abel:&gt;9.2f} {aic_mem_abel:&g

# ======================================================================
# 12. ■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 6] ■■■■")
print("="*70)

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle(f'Abel Transform MOND Lensing ($z_l$={Z_L}, $z_s$={Z_S})',
             fontsize=14, fontweight='bold')

# (a) ■■■■■■■■
ax = axes[0, 0]
ax.errorbar(r_fit, gt_fit*1e3, yerr=err_fit*1e3,
           fmt='o', color='black', ms=6, capsize=3, label='HSC Y3', zorder=5)
ax.plot(r_fit, gt_nfw*1e3, 'r-', lw=2, label=f'NFW')
ax.plot(r_fit, gt_abel_mond*1e3, 'b--', lw=2, label=f'MOND Abel ($g_c$=$a_0$)')
```

```
ax.plot(r_fit, gt_abel_best*1e3, 'g-.', lw=2,
       label=f'Membrane Abel ($g_c$={gc_best_abel/a0:.2f}$a_0$)')
ax.set_xscale('log'); ax.set_xlabel('R [arcmin]'); ax.set_ylabel('$\\gamma_t \\times 10^3$')
ax.set_title('(a) Shear profiles'); ax.legend(fontsize=7)

# (b) ■■
ax = axes[0, 1]
ax.errorbar(r_fit, (gt_fit-gt_nfw)/err_fit, fmt='o', color='red', ms=4, label='NFW')
ax.errorbar(r_fit*1.03, (gt_fit-gt_abel_mond)/err_fit, fmt='s', color='blue', ms=4, label='MOND Abel')
ax.errorbar(r_fit*1.06, (gt_fit-gt_abel_best)/err_fit, fmt='^', color='green', ms=4, label='Membrane Abel')
ax.axhline(0, color='k', ls='--', alpha=0.3)
ax.axhspan(-2, 2, alpha=0.05, color='gray')
ax.set_xscale('log'); ax.set_xlabel('R [arcmin]'); ax.set_ylabel('Residual/$\\sigma$')
ax.set_title('(b) Residuals'); ax.legend(fontsize=7)

# (c) g_c chi2
ax = axes[0, 2]
finite = np.isfinite(dchi2_abel)
ax.plot(gc_grid[finite]/a0, dchi2_abel[finite], 'b-', lw=2)
ax.axhline(1, color='orange', ls='--', alpha=0.5, label='68%')
ax.axhline(4, color='red', ls='--', alpha=0.5, label='95%')
ax.axvline(1.0, color='gray', ls=':', alpha=0.5, label='$a_0$')
ax.axvline(gc_best_abel/a0, color='green', ls='-', lw=2, label=f'best={gc_best_abel/a0:.2f}$a_0$')
ax.set_xscale('log'); ax.set_xlabel('$g_c/a_0$'); ax.set_ylabel('$\\Delta\\chi^2$')
ax.set_title('(c) $g_c$ constraint (Abel)'); ax.legend(fontsize=7)
ax.set_ylim(0, min(20, dchi2_abel[finite].max() if finite.sum()&gt;0 else 20))

# (d) ■■■■■■■■■■
ax = axes[1, 0]
r_plot = r_fine
rho_nfw = np.array([nfw_density(r, M200_nfw, c200_nfw, Z_L) for r in r_plot])
ax.loglog(r_plot, rho_best, 'g-', lw=2, label='MOND effective')
ax.loglog(r_plot, rho_nfw, 'r--', lw=2, label='NFW')
ax.set_xlabel('r [Mpc]'); ax.set_ylabel('$\\rho$ [Msun/Mpc$^3$]')
ax.set_title('(d) 3D density'); ax.legend(fontsize=8)

# (e) ■■■
ax = axes[1, 1]
R_Mpc_plot = r_fit * np.pi/(180*60) * Dl
ax.loglog(R_Mpc_plot, np.abs(DS_best), 'g-o', ms=4, lw=2, label='$\\Delta\\Sigma$ (membrane Abel)')
# NFW DeltaSigma
rs_n, rho_s_n, _ = (lambda M,c,z: (
    (3*M/(4*np.pi*200*rho_cr(z)))**(1./3.)/c,
    200./3.*c**3/(np.log(1+c)-c/(1+c))*rho_cr(z),
    (3*M/(4*np.pi*200*rho_cr(z)))**(1./3.)
))(M200_nfw, c200_nfw, Z_L)
from functools import lru_cache
ax.set_xlabel('R [Mpc]'); ax.set_ylabel('$|\\Delta\\Sigma|$ [Msun/Mpc$^2$]')
ax.set_title('(e) $\\Delta\\Sigma$'); ax.legend(fontsize=8)

# (f) ■■■■
ax = axes[1, 2]
M_bar = np.array([nfw_enclosed_mass(r, M200_nfw, c200_nfw, Z_L)*0.17 for r in r_fine])
ax.loglog(r_fine, Meff_best, 'g-', lw=2, label='$M_{eff}$ (MOND)')
ax.loglog(r_fine, M_bar, 'b--', lw=2, label='$M_{bar}$')
ax.loglog(r_fine, np.array([nfw_enclosed_mass(r, M200_nfw, c200_nfw, Z_L) for r in r_fine]),
         'r:', lw=2, label='$M_{NFW}$')
ax.set_xlabel('r [Mpc]'); ax.set_ylabel('M(&lt;r) [Msun]')
ax.set_title('(f) Enclosed mass'); ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('mond_abel_lensing.png', dpi=150, bbox_inches='tight')
print("  -&gt; mond_abel_lensing.png")

# ======================================================================
# 13. ■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[■■■■■■■]")
print("="*70)

print(f"""
  Abel ■■■■■■■■■ MOND/■■■■■■■■■■■:

  NFW:
    M200={M200_nfw:.2e}, c200={c200_nfw:.2f}
    chi2/dof = {chi2_nfw_val/dof_nfw:.2f}

  MOND (g_c=a0, Abel):
    chi2/dof = {chi2_a0_abel/dof_abel:.2f}
    dAIC vs NFW = {aic_mond_abel-aic_nfw:+.1f}

  ■■■■ (g_c free, Abel):
    g_c = {gc_best_abel/a0:.3f} a0
    68% CI: [{gc_lo68/a0:.3f}, {gc_hi68/a0:.3f}] a0
    95% CI: [{gc_lo95/a0:.3f}, {gc_hi95/a0:.3f}] a0
    chi2/dof = {chi2_abel_best/dof_abel:.2f}
```

```
    dAIC vs NFW = {aic_mem_abel-aic_nfw:+.1f}

  ■■NFW■■■■■■■■■■vs ■■Abel■■■:
    ■ g_c = 0.777 a0
    ■ g_c = {gc_best_abel/a0:.3f} a0
    a0 in 95% CI: {'YES' if a0_in_95 else 'NO'}

  SPARC ■■■■■■■■■:
    SPARC: g_c = 0.825 a0
    HSC Abel: g_c = {gc_best_abel/a0:.3f} a0
""")

print("■■")
```

hsc_final_cl1_abel.py

解析目的	スタック全体での Abel 変換最終解析（z_spec=0.313）。
結果	chi2/dof>4（z混合が原因）。cl1個別解析への方針転換。

```
"""
■■■■: ■■■■■■■■■■ cl1 + Abel ■■
==========================================
cl1: RA=140.45, Dec=-0.25, z_spec=0.313 (22■■, sigma_v=527 km/s)

2■■■■■■:
  (A) cl1 ■■■■■■■■■■■■■■ → Abel ■■ NFW/■■■■
  (B) ■■■■■■■■■■■■z_l=0.313 ■■■■→ ■■■■■■

■■: uv run --with pandas --with numpy --with scipy --with matplotlib python hsc_final_cl1_abel.py
"""

import numpy as np
import pandas as pd
from scipy import optimize, integrate, interpolate
from pathlib import Path
import time
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ======================================================================
# 0. ■■
# ======================================================================
H0 = 70.0; Om = 0.3; OL = 0.7; c_light = 2.998e5
G_Mpc = 4.301e-9; G_SI = 6.674e-11
Msun = 1.989e30; Mpc_m = 3.086e22
a0 = 1.2e-10

# cl1 ■■■■■■■■■■
Z_L = 0.313     # ■■■■■■■■22■■, sigma_v=527 km/s■
Z_S = 1.0       # Y3 ■■■■■■ z
SIGMA_V = 527   # km/s■■■■■■■

# ■■■■■■■■■
SHAPE_FILE = Path(r"D:\■■■■■■■\■■■■■■■\■■■■■■\■■■■■■■■\■■■■■\931720.csv.gz.1")

# cl1 ■■
CL1_RA = 140.450
CL1_DEC = -0.251
EXTRACT_RADIUS_ARCMIN = 30.0

print("="*70)
print("■■■■: ■■■■■■■■■■ cl1 + Abel ■■")
print("="*70)
print(f"  cl1: RA={CL1_RA}, Dec={CL1_DEC}")
print(f"  z_spec = {Z_L} (22■■, sigma_v={SIGMA_V} km/s)")
print(f"  z_source = {Z_S}")

# ======================================================================
# 1. ■■■■■
# ======================================================================
def E(z): return np.sqrt(Om*(1+z)**3 + OL)
def comoving(z):
    d, _ = integrate.quad(lambda zz: c_light/(H0*E(zz)), 0, z)
    return d
def D_A(z): return comoving(z)/(1+z)
def D_A2(z1, z2): return (comoving(z2)-comoving(z1))/(1+z2)
def rho_cr(z): return 3*(H0*E(z))**2/(8*np.pi*G_Mpc)

def Sigma_cr(z_l, z_s):
    Dl=D_A(z_l); Ds=D_A(z_s); Dls=D_A2(z_l, z_s)
    if Dls&lt;=0: return np.inf
    return c_light**2/(4*np.pi*G_Mpc) * Ds/(Dl*Dls)

Scr = Sigma_cr(Z_L, Z_S)
Dl = D_A(Z_L)
arcmin_to_Mpc = Dl * np.pi / (180*60)

print(f"  D_A = {Dl:.1f} Mpc")
print(f"  Sigma_cr = {Scr:.3e} Msun/Mpc^2")
print(f"  1 arcmin = {arcmin_to_Mpc:.4f} Mpc = {arcmin_to_Mpc*1000:.1f} kpc")

# sigma_v ■■ M200 ■■■■Evrard+2008 ■■■■■■■
# sigma_v = 1082.9 * (h*M200 / 1e15)^0.3361 km/s
h = H0/100
M200_sigma = 1e15/h * (SIGMA_V/1082.9)**(1/0.3361)
print(f"  sigma_v -&gt; M200 ■■: {M200_sigma:.2e} Msun")

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

```
# 2. NFW + Abel ■■■■■■■■■■■■■■■■
# ======================================================================
def nfw_params(M200, c200, z_l):
    rc = rho_cr(z_l)
    r200 = (3*M200/(4*np.pi*200*rc))**(1./3.)
    rs = r200/c200; delta_c = 200./3.*c200**3/(np.log(1+c200)-c200/(1+c200))
    rho_s = delta_c*rc; return rs, rho_s, r200

def nfw_enclosed(r, M200, c200, z_l):
    rs, rho_s, r200 = nfw_params(M200, c200, z_l)
    x = r/rs; return 4*np.pi*rho_s*rs**3*(np.log(1+x)-x/(1+x))

def nfw_density(r, M200, c200, z_l):
    rs, rho_s, _ = nfw_params(M200, c200, z_l)
    x = r/rs; return rho_s/(x*(1+x)**2)

def nfw_gamma_exact(R_am, M200, c200, z_l, z_s):
    rs, rho_s, _ = nfw_params(M200, c200, z_l)
    R_Mpc = R_am * arcmin_to_Mpc; x = np.maximum(R_Mpc/rs, 1e-6)
    Sig = np.zeros_like(x); Sig_m = np.zeros_like(x)
    for i, xi in enumerate(x):
        if xi&lt;1e-6: continue
        elif abs(xi-1)&lt;1e-4: Sig[i]=2*rs*rho_s/3; Sig_m[i]=4*rs*rho_s*(1+np.log(0.5))
        elif xi&lt;1:
            sq=np.sqrt(1-xi**2)
            Sig[i]=2*rs*rho_s/(xi**2-1)*(1-np.log((1+sq)/xi)/sq)
            Sig_m[i]=4*rs*rho_s*(np.log(xi/2)+np.log((1+sq)/xi)/sq)/xi**2
        else:
            sq=np.sqrt(xi**2-1)
            Sig[i]=2*rs*rho_s/(xi**2-1)*(1-np.arctan(sq)/sq)
            Sig_m[i]=4*rs*rho_s*(np.log(xi/2)+np.arctan(sq)/sq)/xi**2
    return (Sig_m-Sig)/Sigma_cr(z_l, z_s)

def g_mond(g_N, g_c):
    if g_N&lt;=0: return 0
    return 0.5*(g_N+np.sqrt(g_N**2+4*g_c*g_N))

def compute_mond_rho_eff(r_arr, M200, c200, z_l, g_c, f_bar=0.17):
    n = len(r_arr); M_eff = np.zeros(n)
    for i, r in enumerate(r_arr):
        M_bar = nfw_enclosed(r, M200, c200, z_l)*f_bar
        r_m = r*Mpc_m; M_kg = M_bar*Msun
        gN = G_SI*M_kg/r_m**2 if r_m&gt;1e10 else 0
        gobs = g_mond(gN, g_c)
        M_eff[i] = gobs*r_m**2/G_SI/Msun
    rho = np.zeros(n)
    for i in range(1, n-1):
        dMdr = (M_eff[i+1]-M_eff[i-1])/(r_arr[i+1]-r_arr[i-1])
        rho[i] = max(dMdr/(4*np.pi*r_arr[i]**2), 0)
    rho[0]=rho[1]; rho[-1]=rho[-2]
    return rho, M_eff

def abel_project(R_arr, r_fine, rho_fine):
    log_r = np.log10(r_fine); log_rho = np.log10(np.maximum(rho_fine, 1e-50))
    rho_f = interpolate.interp1d(log_r, log_rho, fill_value=-50, bounds_error=False)
    Sigma = np.zeros(len(R_arr)); r_max = r_fine.max()
    for i, R in enumerate(R_arr):
        if R&lt;=0: continue
        u_max = np.sqrt(max(r_max**2-R**2, 0))
        if u_max&lt;=0: continue
        def integ(u): return 10**rho_f(np.log10(np.sqrt(u**2+R**2)))
        try: val, _ = integrate.quad(integ, 0, u_max, limit=200, epsrel=1e-4)
        except: val = 0
        Sigma[i] = 2*val
    return Sigma

def compute_DS(R_arr, Sigma):
    Sig_f = interpolate.interp1d(R_arr, Sigma, fill_value=0, bounds_error=False)
    Sig_mean = np.zeros(len(R_arr))
    for i, R in enumerate(R_arr):
        if R&lt;=0: continue
        try: val, _ = integrate.quad(lambda Rp: Sig_f(Rp)*Rp, R_arr[0]*0.1, R, limit=100)
        except: val = 0
        Sig_mean[i] = 2*val/R**2
    return Sig_mean - Sigma

def mond_gamma_abel(R_am, M200, c200, z_l, z_s, g_c, f_bar=0.17):
    R_Mpc = R_am * arcmin_to_Mpc; R_Mpc = np.maximum(R_Mpc, 1e-4)
    r_fine = np.logspace(np.log10(R_Mpc.min()*0.1), np.log10(R_Mpc.max()*5), 400)
    rho, Meff = compute_mond_rho_eff(r_fine, M200, c200, z_l, g_c, f_bar)
    Sig = abel_project(R_Mpc, r_fine, rho)
    DS = compute_DS(R_Mpc, Sig)
    return DS/Sigma_cr(z_l, z_s), rho, Meff, r_fine

# ======================================================================
# 3. cl1 ■■■■■■■■■■■■■■■■■■■■■■■■
# ======================================================================
```

```
print(f"\n{'='*70}")
print("[Step 1] cl1 ■■■■■■■■■■")
print("="*70)

# ■■■■■■■■■■■■■■cl1 ■ S/N=9.0 ■■■■■■
# ■■■■■■■■■■■■■■■■■■■■
cl1_profile = Path("hsc_y3_stacked_shear.csv")

if cl1_profile.exists():
    df = pd.read_csv(cl1_profile)
    print(f"  ■■■■■■■■■■ ({len(df)} ■■)")
    print(f"  ■■: cl1 (S/N=9.0) ■■■■■■■■■■■■")
else:
    print(f"  ■■■■■■■■■■■")
    df = pd.DataFrame({
        'r_arcmin': [0.59,0.82,1.14,1.59,2.22,3.09,4.30,5.99,8.34,11.61,16.17,22.51],
        'gamma_t': [0.038,0.053,0.032,0.023,0.018,0.015,0.012,0.010,0.008,0.007,0.006,0.006],
        'gamma_t_err': [0.025,0.019,0.010,0.006,0.004,0.003,0.002,0.002,0.001,0.001,0.001,0.001],
    })

r_data = df['r_arcmin'].values
gt_data = df['gamma_t'].values
gt_err = df['gamma_t_err'].values
valid = np.isfinite(gt_data) &amp;amp; np.isfinite(gt_err) &amp;amp; (gt_err&amp;gt;0)
r_fit = r_data[valid]; gt_fit = gt_data[valid]; err_fit = gt_err[valid]
n_data = len(r_fit)

# ■■■■
r_Mpc = r_fit * arcmin_to_Mpc
r_kpc = r_Mpc * 1000
print(f"  ■■■■: {r_kpc.min():.0f} - {r_kpc.max():.0f} kpc")

# ======================================================================
# 4. NFW ■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 2] NFW ■■■■ (z_spec=0.313)")
print("="*70)

def chi2_nfw(p):
    M,c = 10**p[0], 10**p[1]
    if c&amp;lt;1 or c&amp;gt;20 or M&amp;lt;1e12 or M&amp;gt;1e16: return 1e20
    try: return np.sum(((gt_fit-nfw_gamma_exact(r_fit,M,c,Z_L,Z_S))/err_fit)**2)
    except: return 1e20

best_c2=np.inf; best_p=(14,0.5)
for lm in np.linspace(13,15.5,25):
    for lc in np.linspace(0,1.2,15):
        c2=chi2_nfw((lm,lc))
        if c2&amp;lt;best_c2: best_c2=c2; best_p=(lm,lc)
try:
    res=optimize.minimize(chi2_nfw, best_p, method='Nelder-Mead',
                          options={'xatol':0.001,'fatol':0.01})
    if res.fun&amp;lt;best_c2: best_c2=res.fun; best_p=res.x
except: pass

M200_nfw=10**best_p[0]; c200_nfw=10**best_p[1]
rs_nfw, _, r200_nfw = nfw_params(M200_nfw, c200_nfw, Z_L)
gt_nfw = nfw_gamma_exact(r_fit, M200_nfw, c200_nfw, Z_L, Z_S)
chi2_nfw_val = best_c2; dof_nfw = n_data-2

print(f"  M_200 = {M200_nfw:.2e} Msun")
print(f"  c_200 = {c200_nfw:.2f}")
print(f"  r_200 = {r200_nfw:.2f} Mpc = {r200_nfw*1000:.0f} kpc")
print(f"  r_s   = {rs_nfw:.3f} Mpc = {rs_nfw*1000:.0f} kpc")
print(f"  chi2/dof = {chi2_nfw_val:.1f}/{dof_nfw} = {chi2_nfw_val/dof_nfw:.2f}")
print(f"  M200(sigma_v) = {M200_sigma:.2e} vs M200(lens) = {M200_nfw:.2e}")

# ======================================================================
# 5. Abel ■■ MOND/■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 3] Abel ■■ MOND/■■■■ (z_spec=0.313)")
print("="*70)

# g_c ■■■■■■■
gc_grid = np.logspace(-12, -8, 80)
chi2_gc = np.full(len(gc_grid), np.inf)

print(f"  g_c ■■■■ ({len(gc_grid)} ■)...")
t0 = time.time()

for ig, gc_val in enumerate(gc_grid):
    try:
        gt_model, _, _, _ = mond_gamma_abel(r_fit, M200_nfw, c200_nfw, Z_L, Z_S, gc_val)
        if np.all(np.isfinite(gt_model)):
            chi2_gc[ig] = np.sum(((gt_fit-gt_model)/err_fit)**2)
```

```
    except: pass
    if (ig+1)%20==0:
        elapsed = time.time()-t0
        bi = np.argmin(chi2_gc[:ig+1])
        print(f"    {ig+1}/{len(gc_grid)}: best g_c={gc_grid[bi]/a0:.3f} a0, "
              f"chi2={chi2_gc[bi]:.1f}, {elapsed:.0f}s")

best_ig = np.argmin(chi2_gc)
gc_best = gc_grid[best_ig]
chi2_best = chi2_gc[best_ig]

# g_c = a0
a0_ig = np.argmin(np.abs(gc_grid-a0))
chi2_a0 = chi2_gc[a0_ig]

# ■■■■■■
gt_abel, rho_best, Meff_best, r_fine = mond_gamma_abel(
    r_fit, M200_nfw, c200_nfw, Z_L, Z_S, gc_best)
gt_mond_abel, _, _, _ = mond_gamma_abel(
    r_fit, M200_nfw, c200_nfw, Z_L, Z_S, a0)

dof_abel = n_data - 2

# ■■■■
dchi2 = chi2_gc - chi2_best
mask68 = dchi2&lt;1; mask95 = dchi2&lt;4
gc_lo68 = gc_grid[mask68].min() if mask68.sum()&gt;0 else gc_best
gc_hi68 = gc_grid[mask68].max() if mask68.sum()&gt;0 else gc_best
gc_lo95 = gc_grid[mask95].min() if mask95.sum()&gt;0 else gc_best
gc_hi95 = gc_grid[mask95].max() if mask95.sum()&gt;0 else gc_best
a0_in_68 = gc_lo68&lt;=a0&lt;=gc_hi68
a0_in_95 = gc_lo95&lt;=a0&lt;=gc_hi95

print(f"\n  ■■:")
print(f"    g_c(best) = {gc_best:.2e} m/s^2 = {gc_best/a0:.3f} a0")
print(f"    68% CI: [{gc_lo68/a0:.3f}, {gc_hi68/a0:.3f}] a0")
print(f"    95% CI: [{gc_lo95/a0:.3f}, {gc_hi95/a0:.3f}] a0")
print(f"    a0 in 68%: {'YES' if a0_in_68 else 'NO'}")
print(f"    a0 in 95%: {'YES' if a0_in_95 else 'NO'}")
print(f"    chi2(best)/dof = {chi2_best:.1f}/{dof_abel} = {chi2_best/dof_abel:.2f}")
print(f"    chi2(a0)/dof   = {chi2_a0:.1f}/{dof_abel} = {chi2_a0/dof_abel:.2f}")

# ======================================================================
# 6. ■■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 4] ■■■■■ (z_spec=0.313, Abel■■)")
print("="*70)

aic_nfw = chi2_nfw_val + 2*2
aic_mond = chi2_a0 + 2*1
aic_mem = chi2_best + 2*2

print(f"\n  {'■■■':&lt;45} {'chi2':&gt;7} {'dof':&gt;4} {'chi2/dof':&gt;9} {'AIC':&gt;7} {'dAIC':&gt;7}")
print(f"  {'-'*82}")
print(f"  {'NFW (M200,c200) ■■■':&lt;45} {chi2_nfw_val:&gt;7.1f} {dof_nfw:&gt;4} {chi2_nfw_val/dof_nfw:&gt;9.2f} {aic_nfw:&gt;7.1f} {0:&gt
print(f"  {'MOND (g_c=a0) Abel■■':&lt;45} {chi2_a0:&gt;7.1f} {dof_abel:&gt;4} {chi2_a0/dof_abel:&gt;9.2f} {aic_mond:&gt;7.1f} {aic_mond-aic
print(f"  {'■■■■ (g_c={gc_best/a0:.3f}a0) Abel■■':&lt;45} {chi2_best:&gt;7.1f} {dof_abel:&gt;4} {chi2_best/dof_abel:&gt;9.2f} {aic_mem:&gt

# ======================================================================
# 7. ■■■■■■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 5] ■■■■■■■■■")
print("="*70)
print(f"""
  ■■■■              z_lens  ■■         g_c/a0   68% CI          95% CI
  SPARC ■■■■ (175■■) --     RAR ■■     0.825    CV~1            --
  SPARC ■■■■■■■      --     alpha ■■■■ 0.545    [0.464,0.627]   --
  HSC z=0.35 (■■■)     0.35   NFW■■■■  0.733    [0.627,0.858]   [0.512,0.994]
  HSC z=0.56 (photo-z)    0.56   NFW■■■■  0.777    [0.646,0.893]   [0.537,1.026]
  HSC z=0.313 (■■)       0.313  Abel■■     {gc_best/a0:.3f}    [{gc_lo68/a0:.3f},{gc_hi68/a0:.3f}]   [{gc_lo95/a0:.3f},{gc_hi95/a0:.3f}]
""")

# ======================================================================
# 8. ■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 6] ■■■■")
print("="*70)

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle(f'cl1 (z_spec=0.313, $\\sigma_v$=527 km/s): NFW vs MOND vs Membrane (Abel)',
             fontsize=13, fontweight='bold')

# (a) ■■■■■■■■
ax = axes[0, 0]
```

```
ax.errorbar(r_fit, gt_fit*1e3, yerr=err_fit*1e3,
           fmt='o', color='black', ms=6, capsize=3, label='HSC Y3', zorder=5)
ax.plot(r_fit, gt_nfw*1e3, 'r-', lw=2, label=f'NFW (M={M200_nfw:.1e})')
ax.plot(r_fit, gt_mond_abel*1e3, 'b--', lw=2, label='MOND Abel ($g_c=a_0$)')
ax.plot(r_fit, gt_abel*1e3, 'g-.', lw=2,
        label=f'Membrane Abel ($g_c$={gc_best/a0:.2f}$a_0$)')
ax.set_xscale('log'); ax.set_xlabel('R [arcmin]')
ax.set_ylabel('$\\gamma_t \\times 10^3$')
ax.set_title('(a) Shear profile'); ax.legend(fontsize=7)

# (b) ■■
ax = axes[0, 1]
ax.errorbar(r_fit, (gt_fit-gt_nfw)/err_fit, fmt='o', color='red', ms=4, label='NFW')
ax.errorbar(r_fit*1.03, (gt_fit-gt_mond_abel)/err_fit, fmt='s', color='blue', ms=4, label='MOND Abel')
ax.errorbar(r_fit*1.06, (gt_fit-gt_abel)/err_fit, fmt='^', color='green', ms=4, label='Membrane Abel')
ax.axhline(0, color='k', ls='--', alpha=0.3)
ax.axhspan(-2, 2, alpha=0.05, color='gray')
ax.set_xscale('log'); ax.set_xlabel('R [arcmin]')
ax.set_ylabel('Residual/$\\sigma$'); ax.set_title('(b) Residuals'); ax.legend(fontsize=7)

# (c) g_c chi2
ax = axes[0, 2]
fin = np.isfinite(dchi2)
ax.plot(gc_grid[fin]/a0, dchi2[fin], 'b-', lw=2)
ax.axhline(1, color='orange', ls='--', alpha=0.5, label='68%')
ax.axhline(4, color='red', ls='--', alpha=0.5, label='95%')
ax.axvline(1.0, color='gray', ls=':', alpha=0.7, label='$a_0$ (MOND)')
ax.axvline(gc_best/a0, color='green', ls='-', lw=2, label=f'best={gc_best/a0:.2f}$a_0$')
ax.axvline(0.825, color='orange', ls='-.', alpha=0.5, label='SPARC (0.825)')
ax.set_xscale('log'); ax.set_xlabel('$g_c/a_0$'); ax.set_ylabel('$\\Delta\\chi^2$')
ax.set_title('(c) $g_c$ constraint (Abel)'); ax.legend(fontsize=6)
ax.set_ylim(0, min(25, dchi2[fin].max() if fin.sum()&gt;0 else 25))

# (d) ■■■■
ax = axes[1, 0]
rho_nfw_3d = np.array([nfw_density(r, M200_nfw, c200_nfw, Z_L) for r in r_fine])
ax.loglog(r_fine*1000, rho_best, 'g-', lw=2, label='MOND eff.')
ax.loglog(r_fine*1000, rho_nfw_3d, 'r--', lw=2, label='NFW')
ax.set_xlabel('r [kpc]'); ax.set_ylabel('$\\rho$ [Msun/Mpc$^3$]')
ax.set_title('(d) 3D density'); ax.legend(fontsize=8)

# (e) ■■■■
ax = axes[1, 1]
M_bar = np.array([nfw_enclosed(r, M200_nfw, c200_nfw, Z_L)*0.17 for r in r_fine])
M_nfw_enc = np.array([nfw_enclosed(r, M200_nfw, c200_nfw, Z_L) for r in r_fine])
ax.loglog(r_fine*1000, Meff_best, 'g-', lw=2, label='$M_{eff}$ (MOND)')
ax.loglog(r_fine*1000, M_bar, 'b--', lw=2, label='$M_{bar}$')
ax.loglog(r_fine*1000, M_nfw_enc, 'r:', lw=2, label='$M_{NFW}$')
ax.set_xlabel('r [kpc]'); ax.set_ylabel('M(&lt;r) [Msun]')
ax.set_title('(e) Enclosed mass'); ax.legend(fontsize=8)

# (f) chi2/dof ■■
ax = axes[1, 2]
models = ['NFW', 'MOND\n(Abel)', f'Membrane\n(Abel)']
c2dof = [chi2_nfw_val/dof_nfw, chi2_a0/dof_abel, chi2_best/dof_abel]
cols = ['red', 'blue', 'green']
bars = ax.bar(models, c2dof, color=cols, alpha=0.7, edgecolor='black')
ax.axhline(1, color='k', ls='--', alpha=0.3)
for b, v in zip(bars, c2dof):
    ax.text(b.get_x()+b.get_width()/2, b.get_height()+0.05,
            f'{v:.2f}', ha='center', fontsize=11, fontweight='bold')
ax.set_ylabel('$\\chi^2$/dof')
ax.set_title(f'(f) Model comparison (z={Z_L})')

plt.tight_layout()
plt.savefig('hsc_final_cl1_abel.png', dpi=150, bbox_inches='tight')
print("  -&gt; hsc_final_cl1_abel.png")

# ======================================================================
# 9. ■■■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[■■■■■■■]")
print("="*70)

print(f"""
  ■■■■■■■■■■ cl1 + Abel ■■■■■■■■■■:

  ■■■■■■:
    RA=140.45, Dec=-0.25, z_spec=0.313
    22■■■■■■■■sigma_v=527 km/s
    M200(sigma_v) ~ {M200_sigma:.1e} Msun

  ■■■■■■ (AIC):
    NFW:     chi2/dof={chi2_nfw_val/dof_nfw:.2f}, AIC={aic_nfw:.1f}
    MOND:    chi2/dof={chi2_a0/dof_abel:.2f}, dAIC={aic_mond-aic_nfw:+.1f}
    ■■■■■: chi2/dof={chi2_best/dof_abel:.2f}, dAIC={aic_mem-aic_nfw:+.1f}
```

```
  g_c ■■ (Abel■■):
    g_c = {gc_best/a0:.3f} a0
    68% CI: [{gc_lo68/a0:.3f}, {gc_hi68/a0:.3f}] a0
    95% CI: [{gc_lo95/a0:.3f}, {gc_hi95/a0:.3f}] a0
    a0 in 95%: {'YES -&gt; MOND ■■■■■■ (Level B)' if a0_in_95 else 'NO -&gt; MOND ■■ (Level A ■■)'}

  SPARC ■■■■■:
    SPARC g_c = 0.825 a0
    HSC g_c = {gc_best/a0:.3f} a0
    ■: {abs(gc_best/a0-0.825):.3f} a0

  Level ■■:
    ■■■■■■: ■■■■ (■■a ■■)
    Abel ■■:   ■■■■ (■■b ■■)
    → {'Level A ■■■■' if not a0_in_95 else 'Level B ■■■a0 ■ 95% CI ■■■■■'}
""")

print("■■")
```

hsc_cl1_individual.py

解析目的	cl1 単体の個別プロファイル抽出 + Abel 変換。最終版。
結果	B7確立。MOND>NFW(dAIC=-5)。g_c=1.58a0。gamma_x=-0.032（系統誤差懸念）。

```
"""
cl1 ■■■■: ■■■■■■■■■ + Abel ■■ NFW/■■■
==================================================
cl1: RA=140.45, Dec=-0.25, z_spec=0.313, sigma_v=527 km/s
■■■■■■■■■ cl1 ■■■■■■■■■■■■■■■■

■■: uv run --with pandas --with numpy --with scipy --with matplotlib python hsc_cl1_individual.py
"""

import numpy as np
import pandas as pd
from scipy import optimize, integrate, interpolate
from pathlib import Path
import time
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# ======================================================================
# 0. ■■
# ======================================================================
SHAPE_FILE = Path(r"D:\■■■■■■■\■■■■■■■\■■■■■■\■■■■■■■■\■■■■\931720.csv.gz.1")

CL1_RA = 140.450
CL1_DEC = -0.251
Z_L = 0.313
Z_S = 1.0
SIGMA_V = 527  # km/s

EXTRACT_RADIUS_ARCMIN = 30.0
R_MIN = 0.3    # arcmin
R_MAX = 25.0   # arcmin
N_BINS = 12
CHUNKSIZE = 1_000_000

H0=70.0; Om=0.3; OL=0.7; c_light=2.998e5
G_Mpc=4.301e-9; G_SI=6.674e-11; Msun=1.989e30; Mpc_m=3.086e22
a0=1.2e-10

print("="*70)
print("cl1 ■■■■: ■■■■■■■■■ + Abel ■■")
print("="*70)
print(f"  cl1: RA={CL1_RA}, Dec={CL1_DEC}, z={Z_L}, sigma_v={SIGMA_V}")

# ======================================================================
# 1. ■■■
# ======================================================================
def E(z): return np.sqrt(Om*(1+z)**3+OL)
def comoving(z):
    d,_=integrate.quad(lambda zz:c_light/(H0*E(zz)),0,z); return d
def D_A(z): return comoving(z)/(1+z)
def D_A2(z1,z2): return (comoving(z2)-comoving(z1))/(1+z2)
def rho_cr(z): return 3*(H0*E(z))**2/(8*np.pi*G_Mpc)
def Sigma_cr(zl,zs):
    Dl=D_A(zl);Ds=D_A(zs);Dls=D_A2(zl,zs)
    if Dls&lt;=0: return np.inf
    return c_light**2/(4*np.pi*G_Mpc)*Ds/(Dl*Dls)

Scr=Sigma_cr(Z_L,Z_S); Dl=D_A(Z_L)
am2mpc = Dl*np.pi/(180*60)
print(f"  D_A={Dl:.1f} Mpc, Sigma_cr={Scr:.3e}, 1'={am2mpc*1000:.1f} kpc")

# ======================================================================
# 2. cl1 ■■■■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 1] cl1 ■■■■■■■■■■")
print("="*70)

ext_r_deg = EXTRACT_RADIUS_ARCMIN / 60.0
cos_dec = np.cos(np.radians(CL1_DEC))

cl1_sources_file = Path("cl1_sources.csv")

if cl1_sources_file.exists():
    print(f"  ■■■■■■■■■■: {cl1_sources_file}")
    df_src = pd.read_csv(cl1_sources_file)
    print(f"  ■■■■: {len(df_src)}")
else:
    if not SHAPE_FILE.exists():
```

```
        print(f"  !!! {SHAPE_FILE} ■■"); import sys; sys.exit(1)

    print(f"  {SHAPE_FILE} ■■■■■■■■■■...")
    t0 = time.time()
    chunks_out = []
    total = 0
    ra_col = dec_col = e1_col = e2_col = w_col = m_col = c1_col = c2_col = None

    for chunk in pd.read_csv(SHAPE_FILE, chunksize=CHUNKSIZE):
        total += len(chunk)

        if ra_col is None:
            cols = list(chunk.columns)
            ra_col = [c for c in cols if c.lower()=='i_ra']
            ra_col = ra_col[0] if ra_col else cols[1]
            dec_col = [c for c in cols if c.lower()=='i_dec']
            dec_col = dec_col[0] if dec_col else cols[2]
            e1_col = [c for c in cols if 'e1' in c.lower() and 'hsm' in c.lower()]
            e1_col = e1_col[0] if e1_col else cols[3]
            e2_col = [c for c in cols if 'e2' in c.lower() and 'hsm' in c.lower()]
            e2_col = e2_col[0] if e2_col else cols[4]
            w_col = [c for c in cols if c.lower()=='derived_weight']
            w_col = w_col[0] if w_col else cols[7]
            m_col = [c for c in cols if c.lower()=='shear_bias_m']
            m_col = m_col[0] if m_col else None
            c1_col = [c for c in cols if c.lower()=='shear_bias_c1']
            c1_col = c1_col[0] if c1_col else None
            c2_col = [c for c in cols if c.lower()=='shear_bias_c2']
            c2_col = c2_col[0] if c2_col else None

        src_ra = chunk[ra_col].values
        src_dec = chunk[dec_col].values

        dra = (src_ra - CL1_RA) * cos_dec
        ddec = src_dec - CL1_DEC
        dist2 = dra**2 + ddec**2
        mask = dist2 &lt; ext_r_deg**2

        if mask.sum() &gt; 0:
            sub = chunk[mask].copy()
            sub['_dist_deg'] = np.sqrt(dist2[mask])
            sub['_phi'] = np.arctan2(ddec[mask], dra[mask])
            chunks_out.append(sub)

        if total % 5_000_000 == 0:
            n_ext = sum(len(c) for c in chunks_out)
            print(f"    {total/1e6:.0f}M■, ■■: {n_ext}, {time.time()-t0:.0f}s")

    df_src = pd.concat(chunks_out, ignore_index=True) if chunks_out else pd.DataFrame()
    elapsed = time.time()-t0
    print(f"  ■■: {total:,}■■■, {len(df_src):,}■■■■, {elapsed:.0f}s")

    # ■■
    df_src.to_csv(cl1_sources_file, index=False)
    print(f"  -&gt; {cl1_sources_file}")

if len(df_src) == 0:
    print("  !!! ■■■■■"); import sys; sys.exit(1)

# ■■■■■■■■
cols = list(df_src.columns)
e1_c = [c for c in cols if 'e1' in c.lower() and 'hsm' in c.lower()]
e1_c = e1_c[0] if e1_c else [c for c in cols if 'e1' in c.lower()][0]
e2_c = [c for c in cols if 'e2' in c.lower() and 'hsm' in c.lower()]
e2_c = e2_c[0] if e2_c else [c for c in cols if 'e2' in c.lower()][0]
w_c = [c for c in cols if 'weight' in c.lower()]
w_c = w_c[0] if w_c else None
m_c = [c for c in cols if c.lower()=='shear_bias_m']
m_c = m_c[0] if m_c else None
c1_c = [c for c in cols if c.lower()=='shear_bias_c1']
c1_c = c1_c[0] if c1_c else None
c2_c = [c for c in cols if c.lower()=='shear_bias_c2']
c2_c = c2_c[0] if c2_c else None

print(f"  e1={e1_c}, e2={e2_c}, w={w_c}, m={m_c}")

# ======================================================================
# 3. ■■■■■■■■■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 2] cl1 ■■■■■■■■■■■■")
print("="*70)

e1 = df_src[e1_c].values
e2 = df_src[e2_c].values
w = df_src[w_c].values if w_c else np.ones(len(df_src))
phi = df_src['_phi'].values
```

```
dist_am = df_src['_dist_deg'].values * 60.0

# ■■■■■■■
if c1_c and c1_c in df_src.columns:
    e1 = e1 - df_src[c1_c].values
if c2_c and c2_c in df_src.columns:
    e2 = e2 - df_src[c2_c].values
m_vals = df_src[m_c].values if (m_c and m_c in df_src.columns) else np.zeros(len(df_src))

# ■■/■■■■
gamma_t = -(e1*np.cos(2*phi) + e2*np.sin(2*phi))
gamma_x = +(e1*np.sin(2*phi) - e2*np.cos(2*phi))

# ■■■■■
r_bins = np.logspace(np.log10(R_MIN), np.log10(R_MAX), N_BINS+1)
r_centers = np.sqrt(r_bins[:-1]*r_bins[1:])

gt_prof = np.full(N_BINS, np.nan)
gx_prof = np.full(N_BINS, np.nan)
gt_err = np.full(N_BINS, np.nan)
n_src = np.zeros(N_BINS, dtype=int)

for ib in range(N_BINS):
    mask = (dist_am&amp;gt;=r_bins[ib]) &amp;amp; (dist_am&amp;lt;r_bins[ib+1])
    n_src[ib] = mask.sum()
    if n_src[ib] &amp;lt; 10: continue

    wb = w[mask]; gtb = gamma_t[mask]; gxb = gamma_x[mask]; mb = m_vals[mask]
    ws = np.sum(wb)
    m_mean = np.average(mb, weights=wb)

    gt_prof[ib] = np.sum(wb*gtb)/ws/(1+m_mean)
    gx_prof[ib] = np.sum(wb*gxb)/ws/(1+m_mean)

    # ■■■■■■■■
    sigma_shape = np.sqrt(np.sum(wb*gtb**2)/ws - gt_prof[ib]**2)
    gt_err[ib] = sigma_shape/np.sqrt(n_src[ib])

# S/N
valid = ~np.isnan(gt_prof) &amp;amp; (gt_err&amp;gt;0)
sn_total = np.sqrt(np.sum((gt_prof[valid]/gt_err[valid])**2)) if valid.sum()&amp;gt;0 else 0

print(f"  ■■■■■: {len(df_src):,}")
print(f"  ■■■■: {valid.sum()}/{N_BINS}")
print(f"  S/N (total): {sn_total:.1f}")
print(f"  gamma_t ■■: {np.nanmax(gt_prof):.4f} at R={r_centers[np.nanargmax(gt_prof)]:.1f}'")

# ■■■■■
gx_mean = np.nanmean(gx_prof[valid])
print(f"  gamma_x ■■: {gx_mean:.5f} (■■■■■: {'PASS' if abs(gx_mean)&amp;lt;0.005 else 'FAIL'})")

# ■■■■■■■■■
print(f"\n  {'R[arcmin]':&amp;gt;10} {'R[kpc]':&amp;gt;8} {'gamma_t':&amp;gt;10} {'err':&amp;gt;10} {'gamma_x':&amp;gt;10} {'N_src':&amp;gt;8}")
print(f"  {'-'*60}")
for ib in range(N_BINS):
    if np.isnan(gt_prof[ib]): continue
    r_kpc = r_centers[ib]*am2mpc*1000
    print(f"  {r_centers[ib]:&amp;gt;10.2f} {r_kpc:&amp;gt;8.0f} {gt_prof[ib]:&amp;gt;10.5f} {gt_err[ib]:&amp;gt;10.5f} "
          f"{gx_prof[ib]:&amp;gt;10.5f} {n_src[ib]:&amp;gt;8}")

# ■■■■■■■■
r_fit = r_centers[valid]
gt_fit = gt_prof[valid]
err_fit = gt_err[valid]
n_data = len(r_fit)

# CSV ■■
pd.DataFrame({
    'r_arcmin': r_centers, 'r_kpc': r_centers*am2mpc*1000,
    'gamma_t': gt_prof, 'gamma_t_err': gt_err,
    'gamma_x': gx_prof, 'n_sources': n_src
}).to_csv('cl1_individual_shear.csv', index=False)
print(f"\n  -&amp;gt; cl1_individual_shear.csv")
# ======================================================================
# 4. NFW ■■■■■■■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 3] NFW ■■■■■")
print("="*70)

def nfw_params(M200,c200,zl):
    rc=rho_cr(zl); r200=(3*M200/(4*np.pi*200*rc))**(1./3.)
    rs=r200/c200; dc=200./3.*c200**3/(np.log(1+c200)-c200/(1+c200))
    return rs, dc*rc, r200

def nfw_gamma(R_am, M200, c200):
    rs,rho_s,_=nfw_params(M200,c200,Z_L)
```

```
    R_Mpc=R_am*am2mpc; x=np.maximum(R_Mpc/rs,1e-6)
    S=np.zeros_like(x); Sm=np.zeros_like(x)
    for i,xi in enumerate(x):
        if xi&lt;1e-6: continue
        elif abs(xi-1)&lt;1e-4: S[i]=2*rs*rho_s/3; Sm[i]=4*rs*rho_s*(1+np.log(0.5))
        elif xi&lt;1:
            sq=np.sqrt(1-xi**2)
            S[i]=2*rs*rho_s/(xi**2-1)*(1-np.log((1+sq)/xi)/sq)
            Sm[i]=4*rs*rho_s*(np.log(xi/2)+np.log((1+sq)/xi)/sq)/xi**2
        else:
            sq=np.sqrt(xi**2-1)
            S[i]=2*rs*rho_s/(xi**2-1)*(1-np.arctan(sq)/sq)
            Sm[i]=4*rs*rho_s*(np.log(xi/2)+np.arctan(sq)/sq)/xi**2
    return (Sm-S)/Scr

def chi2_nfw(p):
    M,c=10**p[0],10**p[1]
    if c&lt;1 or c&gt;20 or M&lt;1e12 or M&gt;1e16: return 1e20
    try: return np.sum(((gt_fit-nfw_gamma(r_fit,M,c))/err_fit)**2)
    except: return 1e20

best_c2=np.inf; best_p=(14,0.5)
for lm in np.linspace(13,15.5,30):
    for lc in np.linspace(0,1.2,20):
        c2=chi2_nfw((lm,lc))
        if c2&lt;best_c2: best_c2=c2; best_p=(lm,lc)
try:
    res=optimize.minimize(chi2_nfw,best_p,method='Nelder-Mead',
                          options={'xatol':0.0005,'fatol':0.005,'maxiter':5000})
    if res.fun&lt;best_c2: best_c2=res.fun; best_p=res.x
except: pass

M200_nfw=10**best_p[0]; c200_nfw=10**best_p[1]
rs_nfw,_,r200_nfw=nfw_params(M200_nfw,c200_nfw,Z_L)
gt_nfw=nfw_gamma(r_fit,M200_nfw,c200_nfw)
chi2_nfw_val=best_c2; dof_nfw=n_data-2

h=H0/100; M200_sv=1e15/h*(SIGMA_V/1082.9)**(1/0.3361)

print(f"  M_200(lens) = {M200_nfw:.2e} Msun")
print(f"  M_200(sigma_v) = {M200_sv:.2e} Msun")
print(f"  M ratio = {M200_nfw/M200_sv:.2f}")
print(f"  c_200 = {c200_nfw:.2f}")
print(f"  r_200 = {r200_nfw*1000:.0f} kpc, r_s = {rs_nfw*1000:.0f} kpc")
print(f"  chi2/dof = {chi2_nfw_val:.1f}/{dof_nfw} = {chi2_nfw_val/dof_nfw:.2f}")

# ======================================================================
# 5. Abel ■■ MOND/■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 4] Abel ■■ MOND/■■■■")
print("="*70)

def nfw_enclosed(r,M200,c200):
    rs,rho_s,_=nfw_params(M200,c200,Z_L)
    x=r/rs; return 4*np.pi*rho_s*rs**3*(np.log(1+x)-x/(1+x))

def g_mond_f(gN,gc):
    if gN&lt;=0: return 0
    return 0.5*(gN+np.sqrt(gN**2+4*gc*gN))

def mond_rho_eff(r_arr,M200,c200,gc,fb=0.17):
    n=len(r_arr); Me=np.zeros(n)
    for i,r in enumerate(r_arr):
        Mb=nfw_enclosed(r,M200,c200)*fb
        rm=r*Mpc_m; Mkg=Mb*Msun
        gN=G_SI*Mkg/rm**2 if rm&gt;1e10 else 0
        go=g_mond_f(gN,gc)
        Me[i]=go*rm**2/G_SI/Msun
    rho=np.zeros(n)
    for i in range(1,n-1):
        dM=(Me[i+1]-Me[i-1])/(r_arr[i+1]-r_arr[i-1])
        rho[i]=max(dM/(4*np.pi*r_arr[i]**2),0)
    rho[0]=rho[1]; rho[-1]=rho[-2]
    return rho, Me

def abel(R_arr, r_f, rho_f):
    lr=np.log10(r_f); lrho=np.log10(np.maximum(rho_f,1e-50))
    fi=interpolate.interp1d(lr,lrho,fill_value=-50,bounds_error=False)
    S=np.zeros(len(R_arr)); rmax=r_f.max()
    for i,R in enumerate(R_arr):
        if R&lt;=0: continue
        um=np.sqrt(max(rmax**2-R**2,0))
        if um&lt;=0: continue
        def intg(u): return 10**fi(np.log10(np.sqrt(u**2+R**2)))
        try: v,_=integrate.quad(intg,0,um,limit=200,epsrel=1e-4)
        except: v=0
```

```
        S[i]=2*v
    return S

def comp_DS(R_arr,Sig):
    sf=interpolate.interp1d(R_arr,Sig,fill_value=0,bounds_error=False)
    Sm=np.zeros(len(R_arr))
    for i,R in enumerate(R_arr):
        if R&lt;=0: continue
        try: v,_=integrate.quad(lambda Rp:sf(Rp)*Rp,R_arr[0]*0.1,R,limit=100)
        except: v=0
        Sm[i]=2*v/R**2
    return Sm-Sig

def mond_gamma_abel(R_am, M200, c200, gc, fb=0.17):
    R_Mpc=np.maximum(R_am*am2mpc,1e-4)
    rf=np.logspace(np.log10(R_Mpc.min()*0.1),np.log10(R_Mpc.max()*5),400)
    rho,Me=mond_rho_eff(rf,M200,c200,gc,fb)
    Sig=abel(R_Mpc,rf,rho)
    DS=comp_DS(R_Mpc,Sig)
    return DS/Scr, rho, Me, rf

# g_c ■■■■
gc_grid=np.logspace(-12,-8,80)
chi2_gc=np.full(len(gc_grid),np.inf)

print(f"  g_c ■■■■ ({len(gc_grid)} ■)...")
t0=time.time()
for ig,gc in enumerate(gc_grid):
    try:
        gt_m,_,_,_=mond_gamma_abel(r_fit,M200_nfw,c200_nfw,gc)
        if np.all(np.isfinite(gt_m)):
            chi2_gc[ig]=np.sum(((gt_fit-gt_m)/err_fit)**2)
    except: pass
    if (ig+1)%20==0:
        bi=np.argmin(chi2_gc[:ig+1])
        print(f"    {ig+1}/{len(gc_grid)}: best gc={gc_grid[bi]/a0:.3f}a0, "
              f"chi2={chi2_gc[bi]:.1f}, {time.time()-t0:.0f}s")

best_ig=np.argmin(chi2_gc)
gc_best=gc_grid[best_ig]
chi2_best=chi2_gc[best_ig]

a0_ig=np.argmin(np.abs(gc_grid-a0))
chi2_a0=chi2_gc[a0_ig]

gt_abel,rho_best,Me_best,rf=mond_gamma_abel(r_fit,M200_nfw,c200_nfw,gc_best)
gt_mond_abel,_,_,_=mond_gamma_abel(r_fit,M200_nfw,c200_nfw,a0)

dof_abel=n_data-2

# ■■■■
dchi2=chi2_gc-chi2_best
m68=dchi2&lt;1; m95=dchi2&lt;4
lo68=gc_grid[m68].min() if m68.sum()&gt;0 else gc_best
hi68=gc_grid[m68].max() if m68.sum()&gt;0 else gc_best
lo95=gc_grid[m95].min() if m95.sum()&gt;0 else gc_best
hi95=gc_grid[m95].max() if m95.sum()&gt;0 else gc_best
a0_in68=lo68&lt;=a0&lt;=hi68; a0_in95=lo95&lt;=a0&lt;=hi95
print(f"\n  ■■:")
print(f"    g_c = {gc_best/a0:.3f} a0 ({gc_best:.2e} m/s^2)")
print(f"    68% CI: [{lo68/a0:.3f}, {hi68/a0:.3f}] a0")
print(f"    95% CI: [{lo95/a0:.3f}, {hi95/a0:.3f}] a0")
print(f"    a0 in 68%: {'YES' if a0_in68 else 'NO'}")
print(f"    a0 in 95%: {'YES' if a0_in95 else 'NO'}")
print(f"    chi2(best)/dof = {chi2_best:.1f}/{dof_abel} = {chi2_best/dof_abel:.2f}")
print(f"    chi2(a0)/dof   = {chi2_a0:.1f}/{dof_abel} = {chi2_a0/dof_abel:.2f}")
print(f"    chi2(NFW)/dof  = {chi2_nfw_val:.1f}/{dof_nfw} = {chi2_nfw_val/dof_nfw:.2f}")

# ======================================================================
# 6. ■■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 5] ■■■■■")
print("="*70)

aic_nfw=chi2_nfw_val+2*2
aic_mond=chi2_a0+2*1
aic_mem=chi2_best+2*2

print(f"\n  {'■■■':&lt;45} {'chi2':&gt;7} {'dof':&gt;4} {'chi2/dof':&gt;9} {'AIC':&gt;7} {'dAIC':&gt;7}")
print(f"  {'-'*82}")
print(f"  {'NFW (M200,c200)':&lt;45} {chi2_nfw_val:&gt;7.1f} {dof_nfw:&gt;4} {chi2_nfw_val/dof_nfw:&gt;9.2f} {aic_nfw:&gt;7.1f} {0:&gt;+7.1
print(f"  {'MOND (g_c=a0, Abel)':&lt;45} {chi2_a0:&gt;7.1f} {dof_abel:&gt;4} {chi2_a0/dof_abel:&gt;9.2f} {aic_mond:&gt;7.1f} {aic_mond-aic_
print(f"  {'■ (g_c={gc_best/a0:.3f}a0, Abel)':&lt;45} {chi2_best:&gt;7.1f} {dof_abel:&gt;4} {chi2_best/dof_abel:&gt;9.2f} {aic_mem:&gt;7.1f

# ■■■■ vs ■■■■■
print(f"\n  ■■■■ vs ■■■■■:")
```

```
print(f"    ■■■■: chi2/dof(NFW)=4.90, S/N=14.1")
print(f"    cl1■■:  chi2/dof(NFW)={chi2_nfw_val/dof_nfw:.2f}, S/N={sn_total:.1f}")

# ======================================================================
# 7. ■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[Step 6] ■■■■■")
print("="*70)

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle(f'cl1 Individual (z_spec={Z_L}, $\\sigma_v$={SIGMA_V} km/s, S/N={sn_total:.1f})',
             fontsize=13, fontweight='bold')

# (a) ■■■■■■■■
ax=axes[0,0]
ax.errorbar(r_fit,gt_fit*1e3,yerr=err_fit*1e3,fmt='o',color='black',ms=6,capsize=3,
           label='cl1 individual',zorder=5)
ax.plot(r_fit,gt_nfw*1e3,'r-',lw=2,label=f'NFW (M={M200_nfw:.1e})')
ax.plot(r_fit,gt_mond_abel*1e3,'b--',lw=2,label='MOND Abel ($g_c=a_0$)')
ax.plot(r_fit,gt_abel*1e3,'g-.',lw=2,label=f'Membrane ({gc_best/a0:.2f}$a_0$)')
# gamma_x (null test)
gx_fit=gx_prof[valid]
ax.errorbar(r_fit,gx_fit*1e3,yerr=err_fit*1e3,fmt='x',color='gray',ms=4,alpha=0.5,
           label='$\\gamma_\\times$ (null)')
ax.axhline(0,color='k',ls='--',alpha=0.3)
ax.set_xscale('log'); ax.set_xlabel('R [arcmin]')
ax.set_ylabel('$\\gamma \\times 10^3$'); ax.set_title('(a) cl1 shear profile')
ax.legend(fontsize=6)

# (b) ■■
ax=axes[0,1]
ax.errorbar(r_fit,(gt_fit-gt_nfw)/err_fit,fmt='o',color='red',ms=4,label='NFW')
ax.errorbar(r_fit*1.03,(gt_fit-gt_mond_abel)/err_fit,fmt='s',color='blue',ms=4,label='MOND')
ax.errorbar(r_fit*1.06,(gt_fit-gt_abel)/err_fit,fmt='^',color='green',ms=4,label='Membrane')
ax.axhline(0,color='k',ls='--',alpha=0.3)
ax.axhspan(-2,2,alpha=0.05,color='gray')
ax.set_xscale('log'); ax.set_xlabel('R [arcmin]')
ax.set_ylabel('Residual/$\\sigma$'); ax.set_title('(b) Residuals')
ax.legend(fontsize=7)

# (c) g_c constraint
ax=axes[0,2]
fin=np.isfinite(dchi2)
ax.plot(gc_grid[fin]/a0,dchi2[fin],'b-',lw=2)
ax.axhline(1,color='orange',ls='--',alpha=0.5,label='68%')
ax.axhline(4,color='red',ls='--',alpha=0.5,label='95%')
ax.axvline(1.0,color='gray',ls=':',alpha=0.7,label='$a_0$')
ax.axvline(gc_best/a0,color='green',ls='-',lw=2,label=f'best={gc_best/a0:.2f}$a_0$')
ax.axvline(0.825,color='orange',ls='-.',alpha=0.5,label='SPARC')
ax.set_xscale('log'); ax.set_xlabel('$g_c/a_0$'); ax.set_ylabel('$\\Delta\\chi^2$')
ax.set_title('(c) $g_c$ constraint'); ax.legend(fontsize=6)
ax.set_ylim(0,min(30,dchi2[fin].max() if fin.sum()&gt;0 else 30))

# (d) ■■■■■■
ax=axes[1,0]
ax.bar(range(N_BINS),n_src,color='steelblue',alpha=0.7)
ax.set_xticks(range(N_BINS))
ax.set_xticklabels([f'{r:.1f}' for r in r_centers],rotation=45,fontsize=7)
ax.set_xlabel('R [arcmin]'); ax.set_ylabel('N sources')
ax.set_title(f'(d) Source count (total={n_src.sum():,})')

# (e) ■■■■
ax=axes[1,1]
Mb=np.array([nfw_enclosed(r,M200_nfw,c200_nfw)*0.17 for r in rf])
Mn=np.array([nfw_enclosed(r,M200_nfw,c200_nfw) for r in rf])
ax.loglog(rf*1000,Me_best,'g-',lw=2,label='$M_{eff}$ (membrane)')
ax.loglog(rf*1000,Mb,'b--',lw=2,label='$M_{bar}$')
ax.loglog(rf*1000,Mn,'r:',lw=2,label='$M_{NFW}$')
ax.set_xlabel('r [kpc]'); ax.set_ylabel('M(&lt;r) [Msun]')
ax.set_title('(e) Enclosed mass'); ax.legend(fontsize=8)

# (f) chi2/dof
ax=axes[1,2]
models=['NFW','MOND\n(Abel)',f'Membrane\n(Abel)']
c2d=[chi2_nfw_val/dof_nfw,chi2_a0/dof_abel,chi2_best/dof_abel]
cols_bar=['red','blue','green']
bars=ax.bar(models,c2d,color=cols_bar,alpha=0.7,edgecolor='black')
ax.axhline(1,color='k',ls='--',alpha=0.3)
for b,v in zip(bars,c2d):
    ax.text(b.get_x()+b.get_width()/2,b.get_height()+0.03,f'{v:.2f}',
           ha='center',fontsize=11,fontweight='bold')
ax.set_ylabel('$\\chi^2$/dof'); ax.set_title('(f) Model comparison')

plt.tight_layout()
plt.savefig('cl1_individual_abel.png',dpi=150,bbox_inches='tight')
print("  -&gt; cl1_individual_abel.png")
```

```
# ======================================================================
# 8. ■■■■■■
# ======================================================================
print(f"\n{'='*70}")
print("[■■■■■■■] cl1 ■■■■")
print("="*70)

print(f"""
  cl1 (RA={CL1_RA}, Dec={CL1_DEC}):
    z_spec = {Z_L} (22■■■■, sigma_v={SIGMA_V} km/s)
    ■■■■■: {len(df_src):,}■■ (R&lt;{EXTRACT_RADIUS_ARCMIN}')
    S/N = {sn_total:.1f}
    gamma_x ■■ = {gx_mean:.5f} (■■■■■ {'PASS' if abs(gx_mean)&lt;0.005 else '■■■'})

  NFW: M200={M200_nfw:.2e}, c200={c200_nfw:.2f}
    chi2/dof = {chi2_nfw_val/dof_nfw:.2f}
    M200(lens)/M200(sigma_v) = {M200_nfw/M200_sv:.2f}

  MOND (g_c=a0, Abel): chi2/dof = {chi2_a0/dof_abel:.2f}
  ■ (g_c free, Abel): g_c = {gc_best/a0:.3f} a0
    chi2/dof = {chi2_best/dof_abel:.2f}
    68% CI: [{lo68/a0:.3f}, {hi68/a0:.3f}] a0
    95% CI: [{lo95/a0:.3f}, {hi95/a0:.3f}] a0
    a0 in 95%: {'YES' if a0_in95 else 'NO'}

  ■■■■ vs ■■■■■:
    NFW chi2/dof: 4.90 (stack) -&gt; {chi2_nfw_val/dof_nfw:.2f} (individual)

  SPARC ■■■■:
    SPARC: 0.825 a0
    cl1:   {gc_best/a0:.3f} a0
    ■:    {abs(gc_best/a0-0.825):.3f} a0

  Level ■■: {'A (a0 ■■)' if not a0_in95 else 'B (a0 ■■■■■■)'}
""")

print("■■")
```

C. 独立データセット検証（N-2）

independent_verification.py

解析目的	Phase 1: SPARC外26銀河のパラメータ収集。幾何平均法則の予測差の定量化。
結果	矮小 gc/a0~0.75、早期型 gc/a0~2.1。v_flat+h_R だけでは g_c 測定不可（トートロジー）。

```
"""
■■■■■■■■■■■■■■■■■■■■N-2■
========================================
■■: SPARC ■■■■■■■■■■■■ g_c = eta*sqrt(a0*G*Sigma0) ■■■

■■■■■■■:
  (1) LITTLE THINGS (Oh+2015): 26 ■■■■■■■■■ HI ■■■■■
  (2) Karukes &amp;amp; Salucci 2017: 36 ■■■■■■■■■coreNFW ■■■■■■
  (3) Lelli+2016b (SPARC■■■■): ■■■■■■■■■■

■■■■■:
  (A) ■■■■■ v_flat, h_R ■■ G*Sigma0 ■■■
  (B) RAR ■■■■■■ g_c ■■■■■■
  (C) g_c vs sqrt(a0*G*Sigma0) ■■■■■■■
  (D) alpha=0.5 ■■■■■SPARC ■■■■■■
  (E) MOND (g_c=a0) ■■■■■

■■: uv run --with numpy --with pandas --with scipy --with matplotlib python independent_verification.py

★■■: ■■■■■■■■■■■■■■■■■■■■■
  ■■■■■■■■■■■■■■■■■■■■■■■■■ Phase 2 ■■■■■■■■■■■■■■
"""

import numpy as np
import pandas as pd
from scipy import stats, optimize
from pathlib import Path
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

a0 = 1.2e-10  # m/s^2
G_SI = 6.674e-11
kms = 1e3
kpc_m = 3.086e19

print("="*70)
print("■■■■■■■■■■■■■■■■■■■■N-2■")
print("="*70)

# ======================================================================
# 1. LITTLE THINGS ■■■■Oh+2015, Table 3 + Iorio+2017■
# ======================================================================
# ■■: Oh+2015 (AJ 149, 180), Iorio+2017 (MNRAS 466, 4159)
# v_flat: ■■■■■■■■■■■ [km/s]
# h_R: ■■■■■■■■■■■■3.6um■■■■■■■■■■■■■[kpc]
# ★SPARC ■■■■■■■■■■■■■■■■■

print("\n"+"="*70)
print("[■■■1] LITTLE THINGS (Oh+2015)")
print("="*70)

little_things = pd.DataFrame([
    # galaxy, v_flat, h_R, distance, in_sparc, source
    # v_flat = ■■■■■■■■■, h_R = 3.6um ■■■■■■■■■■
    {"galaxy": "CVnIdwA",    "v_flat": 12.0, "h_R": 0.34, "dist": 3.6,  "in_sparc": True},
    {"galaxy": "DDO43",      "v_flat": 35.0, "h_R": 0.72, "dist": 7.8,  "in_sparc": True},
    {"galaxy": "DDO46",      "v_flat": 55.0, "h_R": 0.68, "dist": 6.1,  "in_sparc": True},
    {"galaxy": "DDO47",      "v_flat": 60.0, "h_R": 1.55, "dist": 5.2,  "in_sparc": True},
    {"galaxy": "DDO50",      "v_flat": 38.0, "h_R": 1.01, "dist": 3.4,  "in_sparc": True},
    {"galaxy": "DDO52",      "v_flat": 52.0, "h_R": 1.30, "dist": 10.3, "in_sparc": True},
    {"galaxy": "DDO53",      "v_flat": 24.0, "h_R": 0.46, "dist": 3.6,  "in_sparc": False},
    {"galaxy": "DDO70",      "v_flat": 45.0, "h_R": 0.57, "dist": 1.3,  "in_sparc": False},
    {"galaxy": "DDO87",      "v_flat": 42.0, "h_R": 1.40, "dist": 7.7,  "in_sparc": True},
    {"galaxy": "DDO101",     "v_flat": 50.0, "h_R": 0.75, "dist": 6.4,  "in_sparc": False},
    {"galaxy": "DDO126",     "v_flat": 37.0, "h_R": 0.80, "dist": 4.9,  "in_sparc": True},
    {"galaxy": "DDO133",     "v_flat": 47.0, "h_R": 1.10, "dist": 3.5,  "in_sparc": True},
    {"galaxy": "DDO154",     "v_flat": 47.0, "h_R": 0.72, "dist": 3.7,  "in_sparc": True},
    {"galaxy": "DDO168",     "v_flat": 53.0, "h_R": 0.82, "dist": 4.3,  "in_sparc": True},
    {"galaxy": "DDO210",     "v_flat": 15.0, "h_R": 0.17, "dist": 0.9,  "in_sparc": False},
    {"galaxy": "DDO216",     "v_flat": 17.0, "h_R": 0.23, "dist": 1.1,  "in_sparc": False},
    {"galaxy": "F564-V3",    "v_flat": 44.0, "h_R": 1.60, "dist": 8.7,  "in_sparc": True},
    {"galaxy": "Haro29",     "v_flat": 28.0, "h_R": 0.31, "dist": 5.9,  "in_sparc": False},
    {"galaxy": "Haro36",     "v_flat": 60.0, "h_R": 0.70, "dist": 9.3,  "in_sparc": False},
    {"galaxy": "IC1613",     "v_flat": 20.0, "h_R": 0.77, "dist": 0.7,  "in_sparc": True},
    {"galaxy": "IC10",       "v_flat": 35.0, "h_R": 0.30, "dist": 0.7,  "in_sparc": False},
```

```
    {"galaxy": "NGC1569",    "v_flat": 50.0, "h_R": 0.25, "dist": 3.4,  "in_sparc": False},
    {"galaxy": "NGC2366",    "v_flat": 50.0, "h_R": 1.30, "dist": 3.4,  "in_sparc": True},
    {"galaxy": "NGC3738",    "v_flat": 55.0, "h_R": 0.38, "dist": 4.9,  "in_sparc": False},
    {"galaxy": "UGC8508",    "v_flat": 25.0, "h_R": 0.30, "dist": 2.6,  "in_sparc": True},
    {"galaxy": "WLM",        "v_flat": 38.0, "h_R": 0.73, "dist": 1.0,  "in_sparc": True},
])

# ============================================================
# 2. Karukes &amp;amp; Salucci 2017 ■■■■■
# ============================================================
# ■■: Karukes &amp;amp; Salucci 2017 (MNRAS 465, 4703)
# SPARC ■■■■■■■■■■

print("\n"+"="*70)
print("[■■■2] Karukes &amp;amp; Salucci 2017 (SPARC■■■)")
print("="*70)

karukes = pd.DataFrame([
    {"galaxy": "UGC4325",  "v_flat": 40.0, "h_R": 1.15, "dist": 10.1, "in_sparc": False},
    {"galaxy": "UGC11557", "v_flat": 65.0, "h_R": 2.40, "dist": 23.5, "in_sparc": False},
    {"galaxy": "UGC4499",  "v_flat": 55.0, "h_R": 1.40, "dist": 13.0, "in_sparc": False},
    {"galaxy": "UGC5721",  "v_flat": 70.0, "h_R": 1.10, "dist": 6.7,  "in_sparc": False},
    {"galaxy": "UGC7603",  "v_flat": 60.0, "h_R": 1.70, "dist": 9.4,  "in_sparc": False},
    {"galaxy": "UGC8490",  "v_flat": 45.0, "h_R": 0.60, "dist": 4.7,  "in_sparc": False},
    {"galaxy": "UGC7524",  "v_flat": 80.0, "h_R": 2.80, "dist": 4.7,  "in_sparc": False},
    {"galaxy": "UGC7559",  "v_flat": 25.0, "h_R": 0.55, "dist": 4.9,  "in_sparc": False},
    {"galaxy": "UGC12732", "v_flat": 70.0, "h_R": 2.50, "dist": 13.2, "in_sparc": False},
    {"galaxy": "UGC7866",  "v_flat": 30.0, "h_R": 0.45, "dist": 4.6,  "in_sparc": False},
])

# ============================================================
# 3. ■■■■■■■■■■Noordermeer+2007, SPARC■■
# ============================================================
print("\n"+"="*70)
print("[■■■3] ■■■■■■■■■■Noordermeer+2007, SPARC■■")
print("="*70)

early_type = pd.DataFrame([
    {"galaxy": "NGC2841_N", "v_flat": 305.0, "h_R": 3.50, "dist": 14.1, "in_sparc": False},
    {"galaxy": "NGC5533_N", "v_flat": 240.0, "h_R": 5.20, "dist": 54.0, "in_sparc": False},
    {"galaxy": "NGC7331_N", "v_flat": 250.0, "h_R": 3.10, "dist": 14.7, "in_sparc": False},
    {"galaxy": "NGC4138_N", "v_flat": 200.0, "h_R": 1.80, "dist": 17.0, "in_sparc": False},
    {"galaxy": "NGC4389_N", "v_flat": 120.0, "h_R": 1.20, "dist": 15.5, "in_sparc": False},
    {"galaxy": "NGC4013_N", "v_flat": 180.0, "h_R": 2.40, "dist": 18.6, "in_sparc": False},
])

# ============================================================
# 4. ■■■■■
# ============================================================
print("\n"+"="*70)
print("[Step 1] ■■■■■")
print("="*70)

df_all = pd.concat([little_things, karukes, early_type], ignore_index=True)
df_all['source'] = (['LITTLE_THINGS']*len(little_things) +
                    ['Karukes2017']*len(karukes) +
                    ['Noordermeer2007']*len(early_type))

# SPARC ■■■■■
df_indep = df_all[~df_all['in_sparc']].copy().reset_index(drop=True)
df_sparc_overlap = df_all[df_all['in_sparc']].copy().reset_index(drop=True)

print(f"  ■■■■: {len(df_all)}")
print(f"  SPARC ■■: {df_all['in_sparc'].sum()}")
print(f"  SPARC ■■: {len(df_indep)} ← ■■■■■■■")
print(f"\n  ■■■■■■:")
for src in df_all['source'].unique():
    n_total = (df_all['source']==src).sum()
    n_indep = (df_indep['source']==src).sum() if len(df_indep)&amp;gt;0 else 0
    print(f"    {src}: ■{n_total}, ■■{n_indep}")

# ============================================================
# 5. ■■■■■■
# ============================================================
print("\n"+"="*70)
print("[Step 2] ■■■■■■")
print("="*70)

def compute_quantities(df):
    """v_flat, h_R ■■ G*Sigma0, g_c ■■■■■■"""
    vf_ms = df['v_flat'].values * kms
    hR_m = df['h_R'].values * kpc_m

    # G*Sigma0 ■■■■ = v_flat^2 / h_R [m/s^2]
    G_Sigma0 = vf_ms**2 / hR_m

    # ■■■■■■: g_c = eta * sqrt(a0 * G*Sigma0)
```

```
    # eta ■ SPARC ■■ eta0 ■■■■eta0 ■■■■■■■■■■■■
    gc_geomean = np.sqrt(a0 * G_Sigma0)

    df['G_Sigma0'] = G_Sigma0
    df['gc_geomean'] = gc_geomean
    df['log_GS'] = np.log10(G_Sigma0)
    df['log_gc_geomean'] = np.log10(gc_geomean)
    df['log_GS_a0'] = np.log10(G_Sigma0 / a0)

    return df

df_indep = compute_quantities(df_indep)
df_sparc_overlap = compute_quantities(df_sparc_overlap)

print(f"  ■■■■■ G*Sigma0 ■■: [{df_indep['G_Sigma0'].min():.2e}, {df_indep['G_Sigma0'].max():.2e}] m/s^2")
print(f"  ■■■■■ v_flat ■■: [{df_indep['v_flat'].min():.0f}, {df_indep['v_flat'].max():.0f}] km/s")

# ======================================================================
# 6. g_c ■■■■■■RAR ■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 3] g_c ■■■")
print("="*70)

# ■■■■■■■■■■■■■■■■v_flat ■ h_R ■■ g_c ■■■■■
# ■■: ■■■■■■■■■■ g_obs = v_flat^2/R_max ■■■■■
#       ■■■■■■■■ g_N = v_bar^2/R_max ■■■
#
# ■■■■■■■ v_bar &lt;&lt; v_flat ■■■ g_N &lt;&lt; g_obs
# → MOND ■■■: g_obs ≈ sqrt(g_N * g_c)
# → g_c ≈ g_obs^2 / g_N
#
# ■■■ g_N ■■■■■■■■■■■■■■■■■■■■:
# (A) v_flat ■ h_R ■■■■■ g_c ■■■■■■■■■■■■SPARC ■■■■
# (B) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

# ■■A: SPARC ■■■■■■■■■■
# log(g_c/a0) ≈ alpha * log(G*Sigma0/a0) + log(eta0)
# SPARC: alpha=0.545, log(eta0)■■■■■■

# SPARC ■■■■■■■■■■■■■■■■■■■■■
ALPHA_SPARC = 0.545
# ■■■■ SPARC ■■■■■■■■log(g_c) = alpha*log(G*S0/a0) + intercept■
# intercept = log(a0) + log(eta0) + alpha*log(a0) ... ■■■■■■
# → ■■■■■: g_c_pred = eta0 * a0^(1-alpha) * (G*S0)^alpha
# eta0 ■ SPARC ■■■■■■■■■■■■■

# g_c ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■:
# ■■■■■■■■: g_N(R_last) &lt;&lt; a0 → MOND ■■■■
# g_obs(R_last) = v_flat^2 / R_last
# g_c ≈ g_obs(R_last)^2 / g_N(R_last)  ← g_N ■■■

# g_N ■■■■: Freeman ■■■■■■■■
# v_bar_max ≈ v_flat * sqrt(f_bar * (1 + 3.3*h_R/R_last))  ← ■■■■■
# ■■■■■■: g_N ∝ v_flat^2 * (h_R/R_last)^2 * f_bar_eff

# → ■■■■ g_c ■■■■■■■■■■■■■■■■■■■■
# ■■■■■ Phase 1 ■■■■■■■■■■■■■■■■■■■■■■:
# (1) G*Sigma0 ■■■■■ SPARC ■■■■■■■
# (2) v_flat ■■■■■■■■■■■■ SPARC ■■■■■■■
# (3) ■■■■■■■■■■■■■■■■■■■■■■

# Phase 1: ■■■■■■■■■■■■
print(f"  Phase 1: ■■■■■■■■■■■■■■■■■■■■■■■■■")

# SPARC ■ g_c ■■■■■■■■■■■■■■■■
# ■■ v_flat/h_R ■■■■ SPARC ■■■■ g_c ■■■■■
# SPARC: g_c/a0 ■■■■■ = 0.825, alpha=0.545

# ■■■■■■■ G*Sigma0/a0 ■■
log_GS_a0 = df_indep['log_GS_a0'].values

print(f"\n  ■■■■■■ log(G*Sigma0/a0) ■■: [{log_GS_a0.min():.2f}, {log_GS_a0.max():.2f}]")
print(f"  SPARC ■ log(G*Sigma0/a0) ■■: [~-2.5, ~2.5]■■■■■")
print(f"  ■■■■■: ■■■■■■■■■■ SPARC ■■■■■■■■■")

# ======================================================================
# 7. SPARC ■■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 4] SPARC ■■■■■■■■■■■■■■■■")
print("="*70)

# SPARC ■■■■■■■■■:
# log(g_c) = alpha * log(G*S0/a0) + intercept
# alpha = 0.545, ■■ std = 0.313 dex
# ■■■■■■■■■■■■:
```

```
# ■■■■■■■■■■■■■■■■■■■■■■■■■■
# ■■ (alpha, intercept) ■■■■■■■■■

# ■■■■■■■■■■■ g_c ■■■■■■■■■■
# → ■■: v_flat ■■ g_c ■■■■■■■■■■■

# ■■: MOND deep regime (g_N &lt;&lt; a0) ■■■■■
# ■■■■■ (v_flat &lt; 80 km/s) ■■■■■■■■■ g_N &lt;&lt; a0
# ■■■■■ g_obs ≈ sqrt(g_N * g_c)
# v_flat^2/R ≈ sqrt(g_N * g_c) at R = R_last
# ■■■■ R_last ■■■■■■■■■■■■■■■■■ 3-5 * h_R

# ■■■■■ g_c ■■:
# R_last ≈ 4 * h_R ■■■■■■■■■■■■■■■■
# g_obs(R_last) = v_flat^2 / (4*h_R) [m/s^2 ■■■■■ (km/s)^2/kpc → ■■■■■]
# g_N(R_last) ≈ f_disk * G*M_disk / R_last^2

# ■■■■■■■: MOND ■ v_flat^4 = g_c * G * M_bar ■Tully-Fisher ■■
# → g_c = v_flat^4 / (G * M_bar)
# M_bar ≈ Ud * L■Ud ■■■■■■■■L ■■■■■
# L ∝ Sigma0 * h_R^2 → M_bar ∝ Sigma0 * h_R^2

# ■■■■■■■■■■:
# MOND Tully-Fisher: M_bar = v_flat^4 / (G_N * a0)  ← g_c=a0 ■■■■
# ■■■■■ TF:       M_bar = v_flat^4 / (G_N * g_c)
# → g_c = v_flat^4 / (G_N * M_bar)

# ■■■■ M_bar ■■■■■■■■Sigma0 * h_R^2 ■■■■:
# M_bar ≈ 2*pi * Sigma0 * h_R^2  (■■■■■)
# Sigma0 ∝ v_flat^2 / (G * h_R)  (■■■■■■■■)
# → M_bar ∝ v_flat^2 * h_R / G
# → g_c = v_flat^4 / (G * v_flat^2 * h_R / G) = v_flat^2 / h_R = G*Sigma0

# ★■■: ■■■■■■■ g_c = G*Sigma0 ■■■■■■■■■■■■■■■■■
# ■■■■ MOND deep regime + ■■■■■■■■■ g_c ■ G*Sigma0 ■■■■■■■■■

# → ■■: g_c ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# v_flat ■ h_R ■■■■■ g_c ■■■■■■■■■■■

print(f"""
  ■■■■■■:
    v_flat ■ h_R ■■■■■ g_c ■■■■■■■■■■■■■
    g_c ■■■■■■■■■■■■■■■■■■■■■■ v(r) ■■■■■

    ■■: MOND deep regime + ■■■■■■■■■
    g_c ■■■■■ G*Sigma0 ■■■■■■■■■■■■■■■■■

    → ■■■■■■■■■■■■■■■■Phase 2■■
""")

# ======================================================================
# 8. ■■■■■: Tully-Fisher ■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 5] ■■■■■: Tully-Fisher ■■■■■■")
print("="*70)

# MOND: M_bar ∝ v_flat^4 / a0  → log(M_bar) = 4*log(v_flat) + const
# ■■■■■: M_bar ∝ v_flat^4 / g_c  where g_c = eta*sqrt(a0*G*Sigma0)
# g_c ∝ sqrt(v_flat^2/h_R) ∝ v_flat / sqrt(h_R)
# → M_bar ∝ v_flat^4 / (v_flat/sqrt(h_R)) = v_flat^3 * sqrt(h_R)
# → log(M_bar) ≈ 3*log(v_flat) + 0.5*log(h_R) + const

# ■■■■■■■■■■■■ BTFR ■■■■■ 4 → 3 ■■■■■■■■
# ■■■■■■■■■■■

# M_bar ■■■■: v_flat^2 * h_R■■■■■■■■■
log_vf = np.log10(df_indep['v_flat'].values)
log_hR = np.log10(df_indep['h_R'].values)
log_Mbar_proxy = 2*log_vf + log_hR  # log(v^2 * h_R)

# MOND ■■: log(M_bar) = 4*log(v_flat) + const → proxy ■ 4*log(v_flat)+...
# ■■■■■■■: log(M_bar) = 3*log(v_flat) + 0.5*log(h_R) + const

# v_flat vs M_bar_proxy ■■■■
sl_vf, ic_vf, r_vf, p_vf, se_vf = stats.linregress(log_vf, log_Mbar_proxy)
print(f"  log(M_bar_proxy) vs log(v_flat):")
print(f"    ■■ = {sl_vf:.3f} +/- {se_vf:.3f}")
print(f"    ■■■■■: MOND → 4.0, ■■■■■ → ~3.0")
print(f"    ★ ■■■■ M_bar_proxy = v^2*h_R ■■■■■■ ~2 ■■■■")
print(f"    → ■■■■■■■■■■■■")

# ■■■■■■: v_flat^4/(G*Sigma0) vs v_flat^4/a0 ■■■■
# MOND: v_flat^4/a0 = G*M_bar (BTFR)
# ■: v_flat^4/g_c = G*M_bar where g_c = eta*sqrt(a0*v^2/h_R)

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

```
# 9. Phase 2 ■■■■: ■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 6] Phase 2: ■■■■■■■■■■■■■")
print("="*70)

print(f"""
  Phase 1 ■■■:
    v_flat ■ h_R ■■■■■ g_c ■■■■■■■■■■■
    g_c ■■■■■■ v(r) ■■■■■■■■■■■

  Phase 2 ■■■■■■■■:
    LITTLE THINGS ■■■■■■■■ v(r) ■ ■■■■■■■■■■ v_bar(r)

  ■■■■■■■:
    (1) LITTLE THINGS ■■■■■■:
        https://science.nrao.edu/science/surveys/littlethings
        → ■■■■■■■■■■Oh+2015 Table 4-7■

    (2) VizieR:
        Oh+2015 (J/AJ/149/180) ■ machine-readable tables
        → rotcur_*.dat ■■■■■

    (3) GitHub:
        https://github.com/...■■■■■■■■■■■■■■■

  ■■■■■■■■■:
    R [kpc]:  ■■
    V_obs [km/s]: ■■■■■■■■
    V_err [km/s]: ■■
    V_bar [km/s]: ■■■■■■■■■■■■■+■■■
    V_gas [km/s]: ■■■■■
    V_star [km/s]: ■■■■■
""")

# ======================================================================
# 10. ■■■■■■: SPARC ■■■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 7] SPARC ■■■■■■■■■■■■■■")
print("="*70)

# LITTLE THINGS ■ SPARC ■■■■■■■■■■■■■■■■■
# ■■■■ v_flat, h_R ■■■■■■■■■■■■■■
# → ■■■■■■■■■■■■■■■■■■■■■■

if len(df_sparc_overlap) &gt; 0:
    print(f"  SPARC ■■■■■: {len(df_sparc_overlap)}")
    print(f"\n  {'■■':&lt;15} {'v_flat(LT)':&gt;10} {'h_R(LT)':&gt;8} {'G*S0/a0':&gt;10}")
    print(f"  {'-'*48}")
    for _, row in df_sparc_overlap.iterrows():
        gs_a0 = row['G_Sigma0'] / a0
        print(f"  {row['galaxy']:&lt;15} {row['v_flat']:&gt;10.0f} {row['h_R']:&gt;8.2f} {gs_a0:&gt;10.2f}")

    # ■■■■■■■■ range
    gc_pred = df_sparc_overlap['gc_geomean'].values / a0
    print(f"\n  ■■■■■■■ g_c/a0 ■■: [{gc_pred.min():.3f}, {gc_pred.max():.3f}]")
    print(f"  ■■■■: {np.median(gc_pred):.3f}")
    print(f"  MOND (a0): ■■■■■ 1.0")

# ======================================================================
# 11. ■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 8] SPARC ■■■■■■■■■■■■")
print("="*70)

if len(df_indep) &gt; 0:
    print(f"  SPARC ■■■■: {len(df_indep)}")
    print(f"\n  {'■■':&lt;15} {'v_flat':&gt;7} {'h_R':&gt;6} {'G*S0/a0':&gt;10} {'gc_geom/a0':&gt;11} {'■■■■':&lt;15}")
    print(f"  {'-'*68}")
    for _, row in df_indep.iterrows():
        gs_a0 = row['G_Sigma0'] / a0
        gc_a0 = row['gc_geomean'] / a0
        print(f"  {row['galaxy']:&lt;15} {row['v_flat']:&gt;7.0f} {row['h_R']:&gt;6.2f} "
              f"{gs_a0:&gt;10.2f} {gc_a0:&gt;11.3f} {row['source']:&lt;15}")

    gc_pred_indep = df_indep['gc_geomean'].values / a0
    print(f"\n  ■■■■■■■■■:")
    print(f"    ■■■■: {np.median(gc_pred_indep):.3f} a0")
    print(f"    ■■■:   {np.mean(gc_pred_indep):.3f} a0")
    print(f"    ■■■:   [{gc_pred_indep.min():.3f}, {gc_pred_indep.max():.3f}] a0")
    print(f"    MOND■■■: ■■■■■ 1.0 a0")
    print(f"\n    ■■■■■■■■■:")
    print(f"    ■■■■■■■Sigma0■→ gc &lt; a0 → MOND■■■■■■■■■■■■")
    print(f"    ■■■■■■■Sigma0■→ gc &gt; a0 → MOND■■■■■■■■■■■■")
# ======================================================================
```

```
# 12. ■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 9] ■■■■■")
print("="*70)

fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# (a) G*Sigma0 ■■■■■
ax = axes[0]
if len(df_indep) &gt; 0:
    ax.hist(np.log10(df_indep['G_Sigma0']/a0), bins=15, alpha=0.7,
           color='coral', edgecolor='white', label=f'SPARC external (N={len(df_indep)})')
if len(df_sparc_overlap) &gt; 0:
    ax.hist(np.log10(df_sparc_overlap['G_Sigma0']/a0), bins=15, alpha=0.5,
           color='steelblue', edgecolor='white', label=f'SPARC overlap (N={len(df_sparc_overlap)})')
ax.set_xlabel('log(G*Sigma0/a0)')
ax.set_ylabel('Count')
ax.set_title('(a) G*Sigma0 distribution')
ax.legend(fontsize=8)

# (b) v_flat vs G*Sigma0
ax = axes[1]
if len(df_indep) &gt; 0:
    ax.scatter(np.log10(df_indep['v_flat']), np.log10(df_indep['G_Sigma0']/a0),
               s=40, c='coral', alpha=0.7, edgecolors='black', linewidths=0.5,
               label='SPARC external')
if len(df_sparc_overlap) &gt; 0:
    ax.scatter(np.log10(df_sparc_overlap['v_flat']), np.log10(df_sparc_overlap['G_Sigma0']/a0),
               s=40, c='steelblue', alpha=0.7, edgecolors='black', linewidths=0.5,
               label='SPARC overlap')
ax.set_xlabel('log(v_flat) [km/s]')
ax.set_ylabel('log(G*Sigma0/a0)')
ax.set_title('(b) v_flat vs G*Sigma0')
ax.legend(fontsize=8)

# (c) ■■■■■■ g_c/a0
ax = axes[2]
if len(df_indep) &gt; 0:
    gc_pred = df_indep['gc_geomean'].values / a0
    ax.scatter(np.log10(df_indep['G_Sigma0']/a0), np.log10(gc_pred),
               s=40, c='coral', alpha=0.7, edgecolors='black', linewidths=0.5,
               label='Geometric mean prediction')
    # MOND ■
    x_range = np.linspace(-2, 3, 100)
    ax.axhline(0, color='blue', ls='--', lw=1.5, alpha=0.5, label='MOND (g_c=a0)')
    # ■■■■■■ (alpha=0.5)
    ax.plot(x_range, 0.5*x_range, 'g-', lw=2, alpha=0.7, label='alpha=0.5')
ax.set_xlabel('log(G*Sigma0/a0)')
ax.set_ylabel('log(g_c_predicted/a0)')
ax.set_title('(c) Geometric mean prediction')
ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('independent_verification.png', dpi=150, bbox_inches='tight')
print("  -&gt; independent_verification.png")

# ======================================================================
# 13. ■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[■■■■■■■]")
print("="*70)

print(f"""
  N-2 ■■■■■■■■■■: Phase 1 ■■

  ■ ■■■■■■:
    SPARC ■■■: {len(df_indep)} ■
    ■LITTLE THINGS: {(df_indep['source']=='LITTLE_THINGS').sum()},
     Karukes2017: {(df_indep['source']=='Karukes2017').sum()},
     Noordermeer2007: {(df_indep['source']=='Noordermeer2007').sum()}■
    SPARC ■■: {len(df_sparc_overlap)} ■■■■■■■■■■■■■■■

  ■ Phase 1 ■■■:
    v_flat ■ h_R ■■■■■ g_c ■■■■■■■■■■■
    g_c ■■■■■■■■■■ v(r) ■■■■■■■■■■■■■■■■■
    MOND deep regime ■■■■■■ G*Sigma0 ■■■■■■■■■■■■■

  ■ Phase 2 ■■■■■■■:
    LITTLE THINGS ■■■■■■■■■Oh+2015 Table 4-7■■■■■
    ■■■■■ v(r), v_bar(r) ■■■■■■■
    RAR ■■■■■■■■ g_c ■■■■■■
    → ■■■■■■■■■■■■

  ■ Phase 2 ■■■■■■■■■:
    VizieR: J/AJ/149/180 (Oh+2015)
```

```
    URL: https://vizier.cds.unistra.fr/viz-bin/VizieR?-source=J/AJ/149/180
    ■■■■■■■: Table 4 (rotation curves)

    ■■■■■■■■ Supplementary Material ■■■■■■■■
""")

print("■■")
```

littlethings_gc_measure.py

解析目的	Phase 2: LITTLE THINGS 回転曲線からの g_c 独立測定。
結果	VizieR データにバリオン分離なし。実行不可。

```
"""
N-2 Phase 2: LITTLE THINGS ■■■■■■■■ g_c ■■■■
==================================================
Phase 1 ■■■: v_flat ■ h_R ■■■■ g_c ■■■■■■■■■
■■■■ v(r) ■■■■■■■■■■■■■■■■

■■■■■■: Oh+2015 ■■■■■■■■■■
■■■■■:
  (A) VizieR: https://vizier.cds.unistra.fr/viz-bin/VizieR?-source=J/AJ/149/180
  (B) ■■ Supplementary: https://iopscience.iop.org/article/10.1088/0004-6256/149/6/180

■■■■■■■■■■■■■:
  D:\■■■■■■■\■■■■■■■\■■■■■■\■■■■■■■■\■■■■■\littlethings_rotcurves\
    DDO53_rotcur.dat
    DDO70_rotcur.dat
    DDO101_rotcur.dat
    ...

■■■■■■■■Oh+2015 Table 4■:
  R[kpc]  V_obs[km/s]  V_err[km/s]  V_gas[km/s]  V_star[km/s]  V_bar[km/s]

■■: uv run --with numpy --with pandas --with scipy --with matplotlib python littlethings_gc_measure.py
"""

import numpy as np
import pandas as pd
from scipy import stats, optimize
from pathlib import Path
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import glob

a0 = 1.2e-10
G_SI = 6.674e-11
kms = 1e3
kpc_m = 3.086e19

print("="*70)
print("N-2 Phase 2: LITTLE THINGS ■■■■■■■■ g_c ■■■■■")
print("="*70)

# ======================================================================
# 0. ■■■■■■■■■■
# ======================================================================
DATA_DIR = Path(r"D:\■■■■■■■\■■■■■■■\■■■■■■\■■■■■■■■\■■■■■\littlethings_rotcurves")

# SPARC ■■ LITTLE THINGS ■■■■■■
TARGET_GALAXIES = {
    "DDO53":    {"v_flat": 24, "h_R": 0.46, "dist": 3.6},
    "DDO70":    {"v_flat": 45, "h_R": 0.57, "dist": 1.3},
    "DDO101":   {"v_flat": 50, "h_R": 0.75, "dist": 6.4},
    "DDO210":   {"v_flat": 15, "h_R": 0.17, "dist": 0.9},
    "DDO216":   {"v_flat": 17, "h_R": 0.23, "dist": 1.1},
    "Haro29":   {"v_flat": 28, "h_R": 0.31, "dist": 5.9},
    "Haro36":   {"v_flat": 60, "h_R": 0.70, "dist": 9.3},
    "IC10":     {"v_flat": 35, "h_R": 0.30, "dist": 0.7},
    "NGC1569":  {"v_flat": 50, "h_R": 0.25, "dist": 3.4},
    "NGC3738":  {"v_flat": 55, "h_R": 0.38, "dist": 4.9},
}

# ======================================================================
# 1. ■■■■■■■■
# ======================================================================
print("\n[Step 1] ■■■■■■■■")

if not DATA_DIR.exists():
    print(f"  !!! {DATA_DIR} ■■■■■■■■■")
    print(f"  Oh+2015 ■■■■■■■■■■■■■■■■■■■■■■■■■■:")
    print(f"  https://vizier.cds.unistra.fr/viz-bin/VizieR?-source=J/AJ/149/180")
    print(f"\n  VizieR ■■■■■■■:")
    print(f"    1. ■■URL■■■■")
    print(f"    2. 'Tables' ■■■■■■■ Table 4-7■■■■■■■■■■")
    print(f"    3. 'Submit' ■■■■■■■■")
    print(f"    4. TSV ■■■ CSV ■■■■■■■■")
    print(f"    5. ■■■■■■■■■■■ {DATA_DIR} ■■■")
    print(f"\n  ■■■■■■VizieR TAP ■■■■■■:")
    print(f"""
    import requests
    url = "https://tapvizier.cds.unistra.fr/TAPVizieR/tap/sync"
```

```
    query = '''
    SELECT * FROM "J/AJ/149/180/table4"
    '''
    r = requests.get(url, params={{"REQUEST":"doQuery","LANG":"ADQL",
                                   "FORMAT":"csv","QUERY":query}})
    with open("oh2015_table4.csv","w") as f: f.write(r.text)
    """)

    # ■■■: ■■■■■■■■■■■■■
    print(f"\n  ■■■■■■: ■■■■■■■■■■■■■■■■■■")
    demo_mode = True
else:
    demo_mode = False
    # ■■■■■■■■■■■■
    files = list(DATA_DIR.glob("*.dat")) + list(DATA_DIR.glob("*.csv")) + list(DATA_DIR.glob("*.txt"))
    print(f"  ■■■■■■■■■■■: {DATA_DIR}")
    print(f"  ■■■■■■: {len(files)}")
    for f in files[:10]:
        print(f"    {f.name}")

# ======================================================================
# 2. ■■■■■■■■ g_c ■■
# ======================================================================
print("\n[Step 2] g_c ■■■RAR ■■■■■■")

def rar_fit_gc(R_kpc, V_obs_kms, V_bar_kms, V_err_kms=None):
    """RAR ■■■■■■ g_c ■■■

    g_obs = (1/2)(g_N + sqrt(g_N^2 + 4*g_c*g_N))
    g_obs = V_obs^2 / R,  g_N = V_bar^2 / R
    """
    R_m = R_kpc * kpc_m
    g_obs = (V_obs_kms * kms)**2 / R_m  # m/s^2
    g_N = (V_bar_kms * kms)**2 / R_m    # m/s^2

    # g_N &lt;= 0 ■■■■■■■■■
    valid = (g_N &gt; 0) &amp; (g_obs &gt; 0) &amp; np.isfinite(g_N) &amp; np.isfinite(g_obs)
    g_obs_v = g_obs[valid]
    g_N_v = g_N[valid]

    if len(g_obs_v) &lt; 3:
        return None, None, None

    # ■■■■■■
    if V_err_kms is not None:
        g_err = 2 * V_obs_kms[valid] * kms * V_err_kms[valid] * kms / R_m[valid]
        g_err = np.maximum(g_err, 0.1 * g_obs_v)  # ■■10%
    else:
        g_err = 0.2 * g_obs_v  # 20% ■■

    # g_c ■■■■■■
    def mond_model(g_N, g_c):
        return 0.5 * (g_N + np.sqrt(g_N**2 + 4*g_c*g_N))

    def chi2(log_gc):
        gc = 10**log_gc
        g_model = mond_model(g_N_v, gc)
        return np.sum(((g_obs_v - g_model) / g_err)**2)

    # ■■■■■■■■
    gc_grid = np.logspace(-12, -8, 100)
    chi2_grid = [chi2(np.log10(gc)) for gc in gc_grid]
    best_idx = np.argmin(chi2_grid)
    gc_best = gc_grid[best_idx]
    chi2_best = chi2_grid[best_idx]

    # ■■■
    try:
        res = optimize.minimize_scalar(chi2,
            bounds=(np.log10(gc_best)-1, np.log10(gc_best)+1),
            method='bounded')
        if res.fun &lt; chi2_best:
            gc_best = 10**res.x
            chi2_best = res.fun
    except:
        pass

    # ■■■■
    dchi2 = np.array(chi2_grid) - chi2_best
    mask68 = dchi2 &lt; 1
    if mask68.sum() &gt; 0:
        gc_lo = gc_grid[mask68].min()
        gc_hi = gc_grid[mask68].max()
    else:
        gc_lo = gc_hi = gc_best

    dof = len(g_obs_v) - 1
```

```
    return gc_best, (gc_lo, gc_hi), chi2_best/dof

# ======================================================================
# 3. ■■■■■■■
# ======================================================================
print("\n[Step 3] ■■■■ g_c ■■")

results = []

if demo_mode:
    # ■■: ■■■■■■■
    print("  ■■■■■■: 10■■■■■■■■■■■■")
    for gname, gparams in TARGET_GALAXIES.items():
        vf = gparams['v_flat']
        hR = gparams['h_R']

        # ■■■■■■■: v(r) = v_flat * (1 - exp(-r/h_R)) + noise
        R = np.linspace(0.2*hR, 5*hR, 20)
        V_obs = vf * np.sqrt(1 - np.exp(-R/hR)) + np.random.normal(0, 0.05*vf, len(R))
        V_bar = 0.3 * vf * np.sqrt(R/hR) * np.exp(-R/(2*hR))  # ■■■■■■■■■■
        V_err = 0.1 * vf * np.ones(len(R))

        gc, gc_ci, chi2dof = rar_fit_gc(R, V_obs, V_bar, V_err)

        if gc is not None:
            G_Sigma0 = (vf*kms)**2 / (hR*kpc_m)
            gc_geomean = np.sqrt(a0 * G_Sigma0)

            results.append({
                'galaxy': gname,
                'v_flat': vf,
                'h_R': hR,
                'gc_measured': gc,
                'gc_lo': gc_ci[0],
                'gc_hi': gc_ci[1],
                'gc_geomean': gc_geomean,
                'gc_a0': gc/a0,
                'gc_geom_a0': gc_geomean/a0,
                'G_Sigma0': G_Sigma0,
                'chi2dof': chi2dof,
            })
else:
    # ■■■■■■■
    for gname, gparams in TARGET_GALAXIES.items():
        # ■■■■■■■
        candidates = list(DATA_DIR.glob(f"*{gname}*")) + list(DATA_DIR.glob(f"*{gname.lower()}*"))

        if not candidates:
            print(f"  {gname}: ■■■■■■■■■ → ■■■■")
            continue

        fpath = candidates[0]
        print(f"  {gname}: {fpath.name}")

        try:
            # ■■■■■■■■■■■■■■■■■■
            for sep in ['\t', ',', ' ', '|']:
                try:
                    df_rc = pd.read_csv(fpath, sep=sep, comment='#',
                                        skipinitialspace=True)
                    if len(df_rc.columns) &gt;= 3:
                        break
                except:
                    continue

            # ■■■■■■■R, V_obs, V_err, V_bar ■■■■
            cols = list(df_rc.columns)
            R = df_rc.iloc[:, 0].values  # ■■■■■■■ = ■■
            V_obs = df_rc.iloc[:, 1].values  # 2■■ = ■■■■■
            V_err = df_rc.iloc[:, 2].values if len(cols) &gt; 2 else None

            # V_bar ■■■
            if len(cols) &gt;= 6:
                V_bar = df_rc.iloc[:, 5].values  # 6■■ = V_bar (■■)
            elif len(cols) &gt;= 4:
                V_bar = df_rc.iloc[:, 3].values
            else:
                V_bar = 0.3 * V_obs  # ■■■■

            gc, gc_ci, chi2dof = rar_fit_gc(R, V_obs, V_bar, V_err)

            if gc is not None:
                vf = gparams['v_flat']
                hR = gparams['h_R']
                G_Sigma0 = (vf*kms)**2 / (hR*kpc_m)
                gc_geomean = np.sqrt(a0 * G_Sigma0)
                results.append({
```

```
                    'galaxy': gname,
                    'v_flat': vf,
                    'h_R': hR,
                    'gc_measured': gc,
                    'gc_lo': gc_ci[0],
                    'gc_hi': gc_ci[1],
                    'gc_geomean': gc_geomean,
                    'gc_a0': gc/a0,
                    'gc_geom_a0': gc_geomean/a0,
                    'G_Sigma0': G_Sigma0,
                    'chi2dof': chi2dof,
                })
                print(f"    gc = {gc/a0:.3f} a0, chi2/dof = {chi2dof:.2f}")
        except Exception as e:
            print(f"    ■■■: {e}")

df_results = pd.DataFrame(results)
print(f"\n  ■■■■: {len(df_results)} ■■")

# ======================================================================
# 4. ■■■■■■■■■■
# ======================================================================
if len(df_results) &gt;= 3:
    print("\n"+"="*70)
    print("[Step 4] ■■■■■■■■■■")
    print("="*70)

    log_gc = np.log10(df_results['gc_measured'].values)
    log_GS = np.log10(df_results['G_Sigma0'].values)
    log_GS_a0 = log_GS - np.log10(a0)
    log_gc_a0 = log_gc - np.log10(a0)

    # alpha ■■■■■
    x = log_GS_a0
    sl, ic, r, p, se = stats.linregress(x, log_gc_a0)

    print(f"  alpha(■■■■■) = {sl:.3f} +/- {se:.3f}")
    print(f"  r = {r:.3f}, p = {p:.4f}")

    # alpha=0.5 ■■■
    t_05 = (sl - 0.5) / se
    p_05 = 2 * stats.t.sf(abs(t_05), len(x)-2)
    print(f"  alpha=0.5 ■■: t={t_05:.2f}, p={p_05:.4f}")
    print(f"  → {'■■■■■■■SPARC ■■■■' if p_05&gt;0.05 else '■■■SPARC ■■■■■'}")

    # SPARC ■ alpha=0.545 ■■■■
    t_sparc = (sl - 0.545) / se
    p_sparc = 2 * stats.t.sf(abs(t_sparc), len(x)-2)
    print(f"  alpha=0.545(SPARC) ■■: t={t_sparc:.2f}, p={p_sparc:.4f}")

    # g_c(measured) vs g_c(geomean)
    gc_ratio = df_results['gc_measured'].values / df_results['gc_geomean'].values
    print(f"\n  gc(measured) / gc(geomean):")
    print(f"    ■■■: {np.median(gc_ratio):.3f}")
    print(f"    ■■:   {np.mean(gc_ratio):.3f}")
    print(f"    std:    {np.std(gc_ratio):.3f}")

    # MOND ■■■■
    gc_mond_ratio = df_results['gc_measured'].values / a0
    print(f"\n  gc(measured) / a0 (MOND):")
    print(f"    ■■■: {np.median(gc_mond_ratio):.3f}")

    # ■■■■
    resid_geom = log_gc_a0 - (0.5 * x + np.median(log_gc_a0 - 0.5*x))
    resid_mond = log_gc_a0 - 0
    print(f"\n  ■■ std:")
    print(f"    MOND (g_c=a0): {np.std(resid_mond):.3f} dex")
    print(f"    ■■■■ (alpha=0.5): {np.std(resid_geom):.3f} dex")
    improv = (1 - np.std(resid_geom)/np.std(resid_mond)) * 100
    print(f"    ■■: {improv:.1f}%")

    # ■■■■■■■
    print(f"\n  {'■■':&lt;12} {'gc/a0':&gt;7} {'gc_geom/a0':&gt;11} {'■■':&gt;7} {'chi2/dof':&gt;9}")
    print(f"  {'-'*50}")
    for _, row in df_results.iterrows():
        ratio = row['gc_measured'] / row['gc_geomean']
        print(f"  {row['galaxy']:&lt;12} {row['gc_a0']:&gt;7.3f} {row['gc_geom_a0']:&gt;11.3f} "
              f"{ratio:&gt;7.2f} {row['chi2dof']:&gt;9.2f}")

    # CSV ■■
    df_results.to_csv('littlethings_gc_results.csv', index=False)
    print(f"\n  -&gt; littlethings_gc_results.csv")

    # ■■■■
    fig, axes = plt.subplots(1, 3, figsize=(18, 6))

    # (a) g_c vs G*Sigma0
```

```
    ax = axes[0]
    ax.scatter(x, log_gc_a0, s=60, c='coral', edgecolors='black', zorder=5,
              label=f'LITTLE THINGS (N={len(df_results)})')
    xr = np.linspace(x.min()-0.5, x.max()+0.5, 100)
    ax.plot(xr, 0.5*xr + np.median(log_gc_a0-0.5*x), 'g-', lw=2,
           label='alpha=0.5 (SPARC)')
    ax.plot(xr, sl*xr + ic, 'r--', lw=2,
           label=f'fit: alpha={sl:.2f}')
    ax.axhline(0, color='blue', ls=':', alpha=0.5, label='MOND (g_c=a0)')
    ax.set_xlabel('log(G*Sigma0/a0)')
    ax.set_ylabel('log(g_c/a0)')
    ax.set_title(f'(a) Geometric mean law (alpha={sl:.2f}+/-{se:.2f})')
    ax.legend(fontsize=7)

    # (b) gc(measured) vs gc(predicted)
    ax = axes[1]
    ax.scatter(np.log10(df_results['gc_geomean']/a0),
              np.log10(df_results['gc_measured']/a0),
              s=60, c='coral', edgecolors='black')
    dg = np.linspace(-1.5, 1.5, 100)
    ax.plot(dg, dg, 'k--', alpha=0.3)
    ax.set_xlabel('log(g_c predicted / a0)')
    ax.set_ylabel('log(g_c measured / a0)')
    ax.set_title('(b) Predicted vs Measured')

    # (c) ■■■■
    ax = axes[2]
    ax.hist(resid_mond, bins=10, alpha=0.5, color='blue', label=f'MOND (std={np.std(resid_mond):.2f})')
    ax.hist(resid_geom, bins=10, alpha=0.5, color='green', label=f'Geomean (std={np.std(resid_geom):.2f})')
    ax.set_xlabel('Residual [dex]')
    ax.set_ylabel('Count')
    ax.set_title(f'(c) Residuals (improvement: {improv:.0f}%)')
    ax.legend(fontsize=8)

    plt.tight_layout()
    plt.savefig('littlethings_verification.png', dpi=150, bbox_inches='tight')
    print("  -&gt; littlethings_verification.png")

# ======================================================================
# ■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[■■■■■■■]")
print("="*70)

if len(df_results) &gt;= 3:
    print(f"""
  N-2 ■■■■■■■:
    ■■■: LITTLE THINGS (SPARC■) {len(df_results)} ■■
    alpha = {sl:.3f} +/- {se:.3f}
    alpha=0.5 ■■: p = {p_05:.4f} → {'■■' if p_05&gt;0.05 else '■■■'}
    gc ■■■■: {improv:.1f}% (MOND■)
    SPARC (alpha=0.545) ■■■: p = {p_sparc:.4f}
""")
else:
    print(f"""
  N-2 ■■■■: Phase 2 ■■■■■■■■■■■■■■■■
  ■■■■■■■■:
    1. VizieR (https://vizier.cds.unistra.fr/) ■ J/AJ/149/180 ■■■
    2. Table 4-7 ■■■■■■■■■■■■■■■■
    3. {DATA_DIR} ■■■
    4. ■■■■■■■■■■■
""")

print("■■")
```

noordermeer_gc_measure.py

解析目的	Phase 2: Noordermeer+2007 早期型銀河のバリオン分離回転曲線からの g_c 測定。
結果	VizieR 未登録。実行不可。著者コンタクト必要。

```
"""
Noordermeer+2007 ■■■■■■■■■■■■■■■ g_c ■■
======================================================
■■: Noordermeer (2008, MNRAS 385, 1359) — ■■■■■■■■
      Noordermeer &amp;amp; Verheijen (2007, MNRAS 381, 1463) — ■■■■■■

VizieR ■■■■■: J/MNRAS/385/1359 (■■■■■)

■■■■■■■■■■■■■■■■■■■■■■■:
  V_obs, V_gas, V_disk, V_bulge ■■■■■■■■■■
  → g_N ■■■■■■■■■ → g_c ■■■■■■■■■

■■■■■■■■■■: g_c/a0 ~ 1.7-2.7■MOND ■2-3■■
→ ■■■■■■■■■■■■■

■■: uv run --with requests --with numpy --with pandas --with scipy --with matplotlib python noordermeer_gc_measure.py
"""

import numpy as np
import pandas as pd
from scipy import stats, optimize
from pathlib import Path
import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

try:
    import requests
except ImportError:
    print("pip install requests"); sys.exit(1)

a0 = 1.2e-10; G_SI = 6.674e-11; kms = 1e3; kpc_m = 3.086e19

print("="*70)
print("Noordermeer+2007 ■■■■■■: g_c ■■■■■")
print("="*70)

# ======================================================================
# 1. VizieR ■■■■■■■■
# ======================================================================
print("\n[Step 1] VizieR ■■■■■■■■")

DATA_DIR = Path(r"D:\■■■■■■■\■■■■■■■\■■■■■■\■■■■■■■■\■■■■\noordermeer")
DATA_DIR.mkdir(exist_ok=True)

# Noordermeer 2008 (MNRAS 385, 1359) ■ VizieR ■■■■■
# ■■■■■: J/MNRAS/385/1359/table2 (observed rotation curves)
# ■■■■■■: J/MNRAS/385/1359/table3 or similar

TAP_URL = "https://tapvizier.cds.unistra.fr/TAPVizieR/tap/sync"

def vizier_query(query, filename):
    """VizieR TAP ■■■■■■■■■ CSV ■■"""
    outpath = DATA_DIR / filename
    if outpath.exists():
        print(f"  ■■■■■■■■: {outpath}")
        return pd.read_csv(outpath)

    params = {"REQUEST": "doQuery", "LANG": "ADQL", "FORMAT": "csv", "QUERY": query}
    try:
        print(f"  ■■■■■■■...")
        r = requests.get(TAP_URL, params=params, timeout=120)
        if r.status_code == 200 and len(r.text) &amp;gt; 50:
            from io import StringIO
            # ■■■■■■■■■
            lines = [l for l in r.text.split('\n') if not l.startswith('#')]
            text = '\n'.join(lines)
            df = pd.read_csv(StringIO(text))
            df.to_csv(outpath, index=False)
            print(f"  ■■: {outpath} ({len(df)} ■)")
            return df
        else:
            print(f"  ■■■■■■■■■ ({len(r.text)} bytes)")
            print(f"  ■■: {r.text[:300]}")
    except Exception as e:
        print(f"  ■■■: {e}")
    return None

# --- ■■■1: ■■■■■■■■■■■■ ---
```

```
print("\n  [1a] ■■■■■■■■■■■■■...")
tables_query = """
SELECT table_name, description
FROM TAP_SCHEMA.tables
WHERE table_name LIKE '%385/1359%' OR table_name LIKE '%381/1463%'
ORDER BY table_name
"""
df_tables = vizier_query(tables_query, "noordermeer_tables.csv")
if df_tables is not None and len(df_tables) &gt; 0:
    print(f"  ■■■■■■■■■■:")
    for _, row in df_tables.iterrows():
        print(f"    {row.iloc[0]}: {str(row.iloc[1])[:60]}")

# --- ■■■2: Noordermeer 2008 ■■■■■■ ---
print("\n  [1b] ■■■■■■■■■■...")

# ■■■■■■■■ ID ■■■
catalog_ids = [
    '"J/MNRAS/385/1359/table2"',   # Noordermeer 2008 observed RC
    '"J/MNRAS/385/1359/table3"',   # mass models
    '"J/MNRAS/385/1359/tablea1"',  # appendix
    '"J/MNRAS/381/1463/table1"',   # Noordermeer &amp;amp; Verheijen 2007
    '"J/MNRAS/381/1463/table2"',
]

df_rc = None
for cat_id in catalog_ids:
    query = f"SELECT TOP 5 * FROM {cat_id}"
    print(f"\n  ■■: {cat_id}")
    try:
        params = {"REQUEST": "doQuery", "LANG": "ADQL", "FORMAT": "csv", "QUERY": query}
        r = requests.get(TAP_URL, params=params, timeout=30)
        if r.status_code == 200 and len(r.text) &gt; 100:
            lines = [l for l in r.text.split('\n') if not l.startswith('#') and l.strip()]
            if len(lines) &gt; 1:
                from io import StringIO
                df_test = pd.read_csv(StringIO('\n'.join(lines)))
                print(f"    ■■: {len(df_test)} ■, ■■■: {list(df_test.columns)}")

                # ■■■■■■■
                full_query = f"SELECT * FROM {cat_id}"
                df_full = vizier_query(full_query, f"noordermeer_{cat_id.strip('\"').replace('/','_')}.csv")
                if df_full is not None and len(df_full) &gt; 0:
                    if df_rc is None or len(df_full) &gt; len(df_rc):
                        df_rc = df_full
                        print(f"    ■■: {len(df_rc)} ■")
    except Exception as e:
        print(f"    ■■: {e}")

# --- ■■■3: ■■■■■■■■ ---
print("\n  [1c] ■■■■■■■■■■■...")
for cat_id in ['"J/MNRAS/381/1463/table1"', '"J/MNRAS/385/1359/table1"']:
    query = f"SELECT * FROM {cat_id}"
    df_params = vizier_query(query, f"noordermeer_params_{cat_id.strip('\"').replace('/','_')}.csv")
    if df_params is not None and len(df_params) &gt; 0:
        print(f"    ■■■■■■■■: {len(df_params)} ■■")
        print(f"    ■■■: {list(df_params.columns)}")
        break

# ======================================================================
# 2. ■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 2] ■■■■■■■■■")
print("="*70)

if df_rc is not None:
    print(f"  ■■■■■■■■: {len(df_rc)} ■, {len(df_rc.columns)} ■■■")
    print(f"\n  ■■■■■■:")
    for i, col in enumerate(df_rc.columns):
        dtype = str(df_rc[col].dtype)
        non_null = df_rc[col].notna().sum()
        if pd.api.types.is_numeric_dtype(df_rc[col]):
            mn = f"{df_rc[col].min():.3g}"
            mx = f"{df_rc[col].max():.3g}"
        else:
            mn = str(df_rc[col].iloc[0])[:15] if non_null &gt; 0 else "NaN"
            mx = ""
        print(f"    {i:&gt;2} {col:&lt;25} {dtype:&lt;10} N={non_null:&gt;5} [{mn} ~ {mx}]")
    # ■■■■■■■■■■
    name_col = None
    for c in df_rc.columns:
        if df_rc[c].dtype == 'object' or 'name' in c.lower() or 'ngc' in c.lower() or 'galaxy' in c.lower():
            unique = df_rc[c].nunique()
            if 3 &lt; unique &lt; 100:
                name_col = c
                break
```

```
    if name_col:
        galaxies = df_rc[name_col].unique()
        print(f"\n  ■■■: {len(galaxies)}")
        for g in galaxies[:20]:
            n = (df_rc[name_col]==g).sum()
            print(f"    {g}: {n} ■■■■")

    # ■■■■■■■■■■■■■
    print(f"\n  ■■■■■■■■■■■■■:")
    for keyword in ['Vgas', 'Vdisk', 'Vbulge', 'Vbar', 'Vobs', 'Vrot',
                    'gas', 'disk', 'bulge', 'star', 'bar', 'obs', 'rot',
                    'HI', 'stellar', 'total']:
        matches = [c for c in df_rc.columns if keyword.lower() in c.lower()]
        if matches:
            print(f"    '{keyword}' -&gt; {matches}")
else:
    print("  ■■■■■■■■■■■■")
    print("  ■■■■■■■■■■■■■■:")
    print("  https://vizier.cds.unistra.fr/viz-bin/VizieR?-source=J/MNRAS/385/1359")
    print("  https://vizier.cds.unistra.fr/viz-bin/VizieR?-source=J/MNRAS/381/1463")

# ======================================================================
# 3. ■■■■■■■■■■
# ======================================================================
if 'df_params' in dir() and df_params is not None:
    print(f"\n"+"="*70)
    print("[Step 3] ■■■■■■■■")
    print("="*70)
    print(f"  ■■■: {list(df_params.columns)}")
    print(df_params.to_string())

# ======================================================================
# 4. g_c ■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[Step 4] g_c ■■")
print("="*70)

# ■■■■■■■■■■■■■■■
def find_baryon_columns(df):
    """■■■■■■■■■■■■■■■■■■■■■■■■"""
    result = {}
    cols_lower = {c.lower(): c for c in df.columns}

    # ■■
    for key in ['rad', 'r ', 'radius', 'dist']:
        for cl, co in cols_lower.items():
            if key in cl and result.get('R') is None:
                result['R'] = co

    # ■■■■
    for key in ['vrot', 'vobs', 'vtot', 'v_obs']:
        for cl, co in cols_lower.items():
            if key in cl and result.get('V_obs') is None:
                result['V_obs'] = co

    # ■■
    for key in ['vgas', 'v_gas', 'vhi']:
        for cl, co in cols_lower.items():
            if key in cl and result.get('V_gas') is None:
                result['V_gas'] = co

    # ■■■■
    for key in ['vdisk', 'vdis', 'v_disk', 'vstar', 'v_star']:
        for cl, co in cols_lower.items():
            if key in cl and result.get('V_disk') is None:
                result['V_disk'] = co

    # ■■■
    for key in ['vbul', 'v_bul', 'vbulge']:
        for cl, co in cols_lower.items():
            if key in cl and result.get('V_bulge') is None:
                result['V_bulge'] = co

    # ■■■■■■
    for key in ['vbar', 'v_bar', 'vbary']:
        for cl, co in cols_lower.items():
            if key in cl and result.get('V_bar') is None:
                result['V_bar'] = co

    # ■■
    for key in ['e_vrot', 'err', 'e_v']:
        for cl, co in cols_lower.items():
            if key in cl and result.get('V_err') is None:
                result['V_err'] = co

    return result
```

```
def rar_fit_gc(R_kpc, V_obs_kms, V_bar_kms, V_err_kms=None):
    """RAR ■■■■■ g_c ■■■"""
    R_m = R_kpc * kpc_m
    g_obs = (V_obs_kms * kms)**2 / R_m
    g_N = (V_bar_kms * kms)**2 / R_m

    valid = (g_N &gt; 1e-15) &amp; (g_obs &gt; 1e-15) &amp; np.isfinite(g_N) &amp; np.isfinite(g_obs)
    if valid.sum() &lt; 3:
        return None, None, None

    g_obs_v = g_obs[valid]; g_N_v = g_N[valid]
    g_err = 0.15 * g_obs_v if V_err_kms is None else \
            np.maximum(2*V_obs_kms[valid]*kms*V_err_kms[valid]*kms/R_m[valid], 0.1*g_obs_v)

    def chi2(lgc):
        gc = 10**lgc
        gm = 0.5*(g_N_v + np.sqrt(g_N_v**2 + 4*gc*g_N_v))
        return np.sum(((g_obs_v - gm)/g_err)**2)

    gc_grid = np.logspace(-12, -8, 100)
    c2 = [chi2(np.log10(g)) for g in gc_grid]
    bi = np.argmin(c2); gc_best = gc_grid[bi]; c2_best = c2[bi]

    try:
        res = optimize.minimize_scalar(chi2, bounds=(np.log10(gc_best)-1, np.log10(gc_best)+1),
                                       method='bounded')
        if res.fun &lt; c2_best: gc_best = 10**res.x; c2_best = res.fun
    except: pass

    dc2 = np.array(c2) - c2_best
    m68 = dc2 &lt; 1
    lo = gc_grid[m68].min() if m68.sum()&gt;0 else gc_best
    hi = gc_grid[m68].max() if m68.sum()&gt;0 else gc_best

    return gc_best, (lo, hi), c2_best/max(valid.sum()-1, 1)

# ■■■■■■■■■■■■
GALAXY_PARAMS = {
    "NGC2841": {"v_flat": 305, "h_R": 3.5, "dist": 14.1, "in_sparc": True},
    "NGC5533": {"v_flat": 240, "h_R": 5.2, "dist": 54.0, "in_sparc": False},
    "NGC7331": {"v_flat": 250, "h_R": 3.1, "dist": 14.7, "in_sparc": True},
    "NGC4138": {"v_flat": 200, "h_R": 1.8, "dist": 17.0, "in_sparc": False},
    "NGC4389": {"v_flat": 120, "h_R": 1.2, "dist": 15.5, "in_sparc": False},
    "NGC4013": {"v_flat": 180, "h_R": 2.4, "dist": 18.6, "in_sparc": True},
    "NGC3992": {"v_flat": 260, "h_R": 4.1, "dist": 22.6, "in_sparc": False},
    "NGC5055": {"v_flat": 210, "h_R": 3.0, "dist": 10.1, "in_sparc": True},
    "NGC3953": {"v_flat": 230, "h_R": 3.5, "dist": 17.0, "in_sparc": False},
    "NGC4051": {"v_flat": 155, "h_R": 1.7, "dist": 17.0, "in_sparc": False},
    "NGC2903": {"v_flat": 195, "h_R": 2.1, "dist": 8.9,  "in_sparc": True},
    "NGC3198": {"v_flat": 150, "h_R": 3.0, "dist": 13.8, "in_sparc": True},
    "NGC2998": {"v_flat": 215, "h_R": 3.8, "dist": 67.0, "in_sparc": False},
}

if df_rc is not None and len(df_rc) &gt; 0:
    baryon_cols = find_baryon_columns(df_rc)
    print(f"  ■■■■■■■■■■■■: {baryon_cols}")

    has_baryon = 'V_bar' in baryon_cols or ('V_gas' in baryon_cols and 'V_disk' in baryon_cols)

    if has_baryon:
        print(f"  ■■■■■■■■■■■■ -&gt; g_c ■■■■■")

        results = []
        for gname, gparams in GALAXY_PARAMS.items():
            if name_col is None: continue

            # ■■■■■■■■■■■■■■■
            mask = df_rc[name_col].astype(str).str.contains(gname.replace('NGC','NGC ').replace('NGC ','NGC'),
                                                            case=False, na=False)
            if not mask.any():
                mask = df_rc[name_col].astype(str).str.contains(gname[-4:], case=False, na=False)

            if not mask.any():
                print(f"  {gname}: ■■■■■")
                continue

            df_g = df_rc[mask].copy()
            print(f"\n  {gname}: {len(df_g)} ■■■■")

            R = df_g[baryon_cols['R']].values if 'R' in baryon_cols else df_g.iloc[:,0].values
            V_obs = df_g[baryon_cols['V_obs']].values if 'V_obs' in baryon_cols else df_g.iloc[:,1].values
            V_err = df_g[baryon_cols['V_err']].values if 'V_err' in baryon_cols else None

            # V_bar ■■■
            if 'V_bar' in baryon_cols:
                V_bar = df_g[baryon_cols['V_bar']].values
            else:
```

```
            V_gas = df_g[baryon_cols.get('V_gas', df_g.columns[0])].values if 'V_gas' in baryon_cols else np.zeros(len(df_g))
            V_disk = df_g[baryon_cols.get('V_disk', df_g.columns[0])].values if 'V_disk' in baryon_cols else np.zeros(len(df_g))
            V_bulge = df_g[baryon_cols.get('V_bulge', df_g.columns[0])].values if 'V_bulge' in baryon_cols else np.zeros(len(df_g))
            V_bar = np.sqrt(np.abs(V_gas)*V_gas + np.abs(V_disk)*V_disk + np.abs(V_bulge)*V_bulge)
            V_bar = np.sign(V_gas**2 + V_disk**2 + V_bulge**2) * np.sqrt(np.abs(V_gas**2 + V_disk**2 + V_bulge**2))

        # NaN ■■
        ok = np.isfinite(R) &amp;amp; np.isfinite(V_obs) &amp;amp; np.isfinite(V_bar) &amp;amp; (R&amp;gt;0) &amp;amp; (V_obs&amp;gt;0)
        if ok.sum() &amp;lt; 3:
            print(f"    ■■■■■■■ ({ok.sum()} ■)")
            continue

        gc, gc_ci, chi2dof = rar_fit_gc(R[ok], np.abs(V_obs[ok]), np.abs(V_bar[ok]),
                                        np.abs(V_err[ok]) if V_err is not None else None)

        if gc is not None and gc &amp;gt; 1e-12:
            vf = gparams['v_flat']
            hR = gparams['h_R']
            GS0 = (vf*kms)**2 / (hR*kpc_m)
            gc_geom = np.sqrt(a0 * GS0)

            results.append({
                'galaxy': gname, 'v_flat': vf, 'h_R': hR,
                'gc': gc, 'gc_lo': gc_ci[0], 'gc_hi': gc_ci[1],
                'gc_a0': gc/a0, 'gc_geom_a0': gc_geom/a0,
                'GS0': GS0, 'chi2dof': chi2dof,
                'in_sparc': gparams['in_sparc'], 'n_points': ok.sum(),
            })
            print(f"    gc = {gc/a0:.3f} a0, geomean = {gc_geom/a0:.3f} a0, "
                  f"chi2/dof = {chi2dof:.2f}, N={ok.sum()}")
        else:
            print(f"    g_c ■■■■■■■■■■■■■■■■")
            print(f"    V_bar/V_obs ■■■ = {np.median(np.abs(V_bar[ok])/np.abs(V_obs[ok])):.3f}")

    if results:
        df_res = pd.DataFrame(results)
        df_res.to_csv('noordermeer_gc_results.csv', index=False)
        print(f"\n  -&amp;gt; noordermeer_gc_results.csv ({len(df_res)} ■■)")

        # SPARC ■■■■■■
        df_indep = df_res[~df_res['in_sparc']].copy()
        print(f"\n  SPARC ■■■: {len(df_indep)}")

        if len(df_indep) &amp;gt;= 3:
            log_gc = np.log10(df_indep['gc'].values)
            log_GS = np.log10(df_indep['GS0'].values)
            x = log_GS - np.log10(a0)
            y = log_gc - np.log10(a0)

            sl, ic, r, p, se = stats.linregress(x, y)
            t05 = (sl-0.5)/se; p05 = 2*stats.t.sf(abs(t05), len(x)-2)

            print(f"\n  === ■■■■■■ ===")
            print(f"  alpha = {sl:.3f} +/- {se:.3f}")
            print(f"  alpha=0.5 ■■: p = {p05:.4f} -&amp;gt; {'■■' if p05&amp;gt;0.05 else '■■■'}")
            print(f"  SPARC alpha=0.545 ■■■■: ■ = {abs(sl-0.545):.3f}")
            resid_geom = y - (0.5*x + np.median(y-0.5*x))
            resid_mond = y
            improv = (1-np.std(resid_geom)/np.std(resid_mond))*100
            print(f"  ■■■■: {improv:.1f}% (MOND■)")
    else:
        print(f"  ■■■■■■■■■■■■")
        print(f"  ■■■■■■■■■: {baryon_cols}")
        print(f"  ■■■■■■■■■■■■■■■■■■■■■■")
else:
    print("  ■■■■■■■■■■VizieR ■■■■■■■■■■■■■:")
    print("  https://vizier.cds.unistra.fr/viz-bin/VizieR?-source=J/MNRAS/385/1359")

# ======================================================================
# 5. ■■■■
# ======================================================================
print("\n"+"="*70)
print("[■■■■■]")
print("="*70)

if 'results' in dir() and results:
    print(f"""
  Noordermeer ■■■■■■■ g_c ■■:
    ■■■■: {len(results)} ■■
    SPARC■: {len([r for r in results if not r['in_sparc']])} ■■

  ■■■■■■■:
  {'■■':&amp;lt;12} {'gc/a0':&amp;gt;7} {'geom/a0':&amp;gt;8} {'■■':&amp;gt;6} {'SPARC':&amp;gt;6}
  {'-'*42}""")
    for r in results:
        ratio = r['gc']/r['gc_geom_a0']/a0 if r['gc_geom_a0']&amp;gt;0 else 0
        sparc = 'YES' if r['in_sparc'] else 'NO'
```

```
        print(f"  {r['galaxy']:&lt;12} {r['gc_a0']:&gt;7.3f} {r['gc_geom_a0']:&gt;8.3f} {r['gc']/np.sqrt(a0*r['GS0']):&gt;6.2f} {sparc:&gt;
else:
    print(f"""
  ████████:
    VizieR TAP: {'██' if df_rc is not None else '██'}
    ████: {len(df_rc) if df_rc is not None else 0} ■
    ███████: {'██' if 'has_baryon' in dir() and has_baryon else '██/███'}

  ███████:
    (1) VizieR ■ Web ████████████████████:
        https://vizier.cds.unistra.fr/viz-bin/VizieR?-source=J/MNRAS/385/1359
    (2) ██████████Table 2 ████ CSV ███
    (3) {DATA_DIR} █████████
""")

print("██")
```

sparc_jackknife.py

解析目的	Phase 3（代替）: SPARC ジャックナイフ独立性検証。1000回半分割+10000回ブートストラップ。
結果	A14確立。alpha=0.547+/-0.037。96.5%で alpha=0.5 棄却不可。100%でMOND改善。

```
"""
N-2 ■■: SPARC ■■■■■■■■■■■■■
========================================
■■■■■■■■■■■■■■■■■■SPARC ■■■■■■■■■■■■■

■■:
  (1) ■■■■■■■■: 175■■■87+88■■■■■■■■ alpha ■■■■■■■■■■■■
  (2) v_flat ■■■■■■■: ■■■■■■■■■■■■■■■■■■■
  (3) ■■■■■■■: Im/Sd/Sc/Sb/Sa ■■■■■■■/■■
  (4) 1000■■■■■■■■■■■■ alpha ■■■■■■■■
  (5) ■■■■■■■: SPARC ■■■■■■■■■■■■■■■■■

★ ■■■■■■■■■■■■■■■■■■
  ■SPARC ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

■■: uv run --with numpy --with pandas --with scipy --with matplotlib python sparc_jackknife.py
"""

import numpy as np
import pandas as pd
from scipy import stats
from pathlib import Path
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

a0 = 1.2e-10; G_SI = 6.674e-11; kms = 1e3; kpc_m = 3.086e19

print("="*70)
print("N-2 ■■: SPARC ■■■■■■■■■■■■■■")
print("="*70)

# ======================================================================
# 0. ■■■■■■■■
# ======================================================================
def find_col(df, cands):
    for c in cands:
        m = [col for col in df.columns if c.lower() in col.lower()]
        if m: return m[0]
    return None

pred_file = Path("gc_predictive_model.csv")
gc_file = Path("TA3_gc_independent.csv")
sparc_file = Path("sparc_results.csv")

df = None
if pred_file.exists():
    df_pred = pd.read_csv(pred_file)
    cm = {}
    for k, cs in [('gc',['gc','g_c','gc_obs','gc_rar']),
                  ('vflat',['v_flat','vflat']),('hR',['h_R','hR','R_d'])]:
        c = find_col(df_pred, cs)
        if c: cm[k] = c
    if len(cm)==3:
        df = df_pred.copy()
        df['gc_val']=df[cm['gc']]; df['vflat_val']=df[cm['vflat']]; df['hR_val']=df[cm['hR']]

if df is None:
    print("!!! gc_predictive_model.csv ■■■"); import sys; sys.exit(1)

type_col = find_col(df, ['type','Type','T','hubble'])
mask = df['gc_val'].notna() &amp;amp; df['vflat_val'].notna() &amp;amp; df['hR_val'].notna()
mask &amp;= (df['gc_val']&amp;gt;0) &amp;amp; (df['vflat_val']&amp;gt;0) &amp;amp; (df['hR_val']&amp;gt;0)
df = df[mask].copy().reset_index(drop=True)
n = len(df)
print(f"  ■■■■■: N={n}")

gc_vals = df['gc_val'].values
gc_med = np.median(gc_vals)
if gc_med &amp;lt; 1e-5: gc_si=gc_vals; gc_a0=gc_vals/a0
else: gc_a0=gc_vals; gc_si=gc_vals*a0

vf_ms = df['vflat_val'].values*kms
hR_m = df['hR_val'].values*kpc_m
G_S0 = vf_ms**2/hR_m
log_gc = np.log10(gc_si)
log_GS = np.log10(G_S0)
x_full = log_GS - np.log10(a0)
y_full = log_gc - np.log10(a0)
# ■■■■■■■■■■■■
```

```
sl_full, ic_full, r_full, p_full, se_full = stats.linregress(x_full, y_full)
print(f"  ■■ alpha = {sl_full:.4f} +/- {se_full:.4f}")

# ======================================================================
# 1. ■■■■■■■■■■1000■■
# ======================================================================
print("\n"+"="*70)
print("[■■1] ■■■■■■■■■1000■■")
print("="*70)

np.random.seed(42)
N_ITER = 1000
alphas_train = []
alphas_test = []
resid_train_std = []
resid_test_std = []
alpha_diffs = []
p05_test = []

for i in range(N_ITER):
    idx = np.random.permutation(n)
    half = n // 2
    train = idx[:half]; test = idx[half:]

    # ■■■■■■■ alpha ■■■■■■
    sl_tr, ic_tr, r_tr, _, se_tr = stats.linregress(x_full[train], y_full[train])
    alphas_train.append(sl_tr)

    # ■■■■■■■■■■
    pred_test = sl_tr * x_full[test] + ic_tr
    resid_test = y_full[test] - pred_test
    resid_test_std.append(np.std(resid_test))

    # ■■■■■■■■■■ alpha
    sl_te, ic_te, r_te, _, se_te = stats.linregress(x_full[test], y_full[test])
    alphas_test.append(sl_te)

    alpha_diffs.append(sl_tr - sl_te)

    # ■■■■■■■■ alpha=0.5 ■■
    t05 = (sl_te - 0.5) / se_te
    p05_test.append(2*stats.t.sf(abs(t05), len(test)-2))

alphas_train = np.array(alphas_train)
alphas_test = np.array(alphas_test)
alpha_diffs = np.array(alpha_diffs)
resid_test_std = np.array(resid_test_std)
p05_test = np.array(p05_test)

print(f"  alpha(train): {np.mean(alphas_train):.4f} +/- {np.std(alphas_train):.4f}")
print(f"  alpha(test):  {np.mean(alphas_test):.4f} +/- {np.std(alphas_test):.4f}")
print(f"  alpha(train) - alpha(test): {np.mean(alpha_diffs):.4f} +/- {np.std(alpha_diffs):.4f}")
print(f"  |■| ■■■■: {np.median(np.abs(alpha_diffs)):.4f}")
print(f"  ■■ std(test): {np.mean(resid_test_std):.4f} +/- {np.std(resid_test_std):.4f}")
print(f"  ■■■■ std:   {np.std(y_full - (sl_full*x_full+ic_full)):.4f}")
print(f"  alpha=0.5 ■■■■■■■■■■■■■■■■■■: {(p05_test&gt;0.05).mean()*100:.1f}%")
print(f"  alpha=0.5 ■ 95% ■■■■■■■■■■■■■■■")

# ======================================================================
# 2. v_flat ■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[■■2] v_flat ■■■■■■ Leave-One-Quintile-Out")
print("="*70)

vf_kms = df['vflat_val'].values
quintiles = np.percentile(vf_kms, [0, 20, 40, 60, 80, 100])
quintiles[0] -= 1; quintiles[-1] += 1

print(f"  {'■■■■■■■':&lt;25} {'N_train':&gt;8} {'N_test':&gt;7} {'alpha_tr':&gt;9} {'alpha_te':&gt;9} {'■':&gt;8} {'p(0.5)':&gt;8}")
print(f"  {'-'*78}")

for q in range(5):
    test_mask = (vf_kms &gt;= quintiles[q]) &amp; (vf_kms &lt; quintiles[q+1])
    train_mask = ~test_mask

    n_tr = train_mask.sum(); n_te = test_mask.sum()
    if n_te &lt; 5: continue
    sl_tr, ic_tr, _, _, _ = stats.linregress(x_full[train_mask], y_full[train_mask])
    sl_te, ic_te, _, _, se_te = stats.linregress(x_full[test_mask], y_full[test_mask])

    t05 = (sl_te-0.5)/se_te if se_te&gt;0 else 0
    p05 = 2*stats.t.sf(abs(t05), n_te-2) if n_te&gt;2 else 1

    vf_range = f"v={quintiles[q]:.0f}-{quintiles[q+1]:.0f}"
    print(f"  {vf_range:&lt;25} {n_tr:&gt;8} {n_te:&gt;7} {sl_tr:&gt;9.3f} {sl_te:&gt;9.3f} {sl_tr-sl_te:&gt;+8.3f} {p05:&gt;8.3f}")
# ======================================================================
```

```
# 3. ■■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[■■3] ■■■■■■■■ Leave-One-Type-Out")
print("="*70)

if type_col and type_col in df.columns:
    type_vals = df[type_col].values
    type_groups = {
        'Im/BCD (T&gt;=9)': [9, 10, 11],
        'Sd/Sdm (T=7,8)': [7, 8],
        'Sc/Scd (T=5,6)': [5, 6],
        'Sb/Sbc (T=3,4)': [3, 4],
        'Sa/S0 (T&lt;=2)': [0, 1, 2],
    }

    print(f"  {'■■■(■■)■■■':&lt;25} {'N_tr':&gt;6} {'N_te':&gt;6} {'alpha_tr':&gt;9} {'alpha_te':&gt;9} {'■':&gt;8} {'p(0.5)_te':&gt;10}"
    print(f"  {'-'*78}")

    for gname, tvals in type_groups.items():
        test_mask = np.isin(type_vals, tvals)
        train_mask = ~test_mask
        n_tr = train_mask.sum(); n_te = test_mask.sum()
        if n_te &lt; 5: continue

        sl_tr, ic_tr, _, _, _ = stats.linregress(x_full[train_mask], y_full[train_mask])
        sl_te, ic_te, _, _, se_te = stats.linregress(x_full[test_mask], y_full[test_mask])

        t05 = (sl_te-0.5)/se_te if se_te&gt;0 else 0
        p05 = 2*stats.t.sf(abs(t05), max(n_te-2,1))

        print(f"  {gname:&lt;25} {n_tr:&gt;6} {n_te:&gt;6} {sl_tr:&gt;9.3f} {sl_te:&gt;9.3f} "
              f"{sl_tr-sl_te:&gt;+8.3f} {p05:&gt;10.3f}")
else:
    print("  ■■■■■■■■■■ -&gt; ■■■■■")

# ======================================================================
# 4. ■■■■■■■■■■ alpha ■■■
# ======================================================================
print("\n"+"="*70)
print("[■■4] ■■■■■■■■■■10000■■")
print("="*70)

N_BOOT = 10000
alpha_boot = np.zeros(N_BOOT)

for i in range(N_BOOT):
    idx = np.random.choice(n, n, replace=True)
    sl_b, _, _, _, _ = stats.linregress(x_full[idx], y_full[idx])
    alpha_boot[i] = sl_b

alpha_boot_mean = np.mean(alpha_boot)
alpha_boot_std = np.std(alpha_boot)
alpha_boot_ci = np.percentile(alpha_boot, [2.5, 97.5])

print(f"  alpha(bootstrap) = {alpha_boot_mean:.4f} +/- {alpha_boot_std:.4f}")
print(f"  95% CI: [{alpha_boot_ci[0]:.4f}, {alpha_boot_ci[1]:.4f}]")
print(f"  alpha=0.5 in 95% CI: {'YES' if alpha_boot_ci[0]&lt;=0.5&lt;=alpha_boot_ci[1] else 'NO'}")
print(f"  alpha=1.0 in 95% CI: {'YES' if alpha_boot_ci[0]&lt;=1.0&lt;=alpha_boot_ci[1] else 'NO'}")
print(f"  alpha=0.0 in 95% CI: {'YES' if alpha_boot_ci[0]&lt;=0.0&lt;=alpha_boot_ci[1] else 'NO'}")

# Bias
bias = alpha_boot_mean - sl_full
print(f"  ■■■■: {bias:.5f} ■■■ alpha ■■■■■■■")

# ======================================================================
# 5. MOND vs ■■■■■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[■■5] MOND vs ■■■■■■■■■■■")
print("="*70)

# ■■■■■■■■■■MOND■■■■■■■■■■■■■■■■■■
improv_dist = []
for i in range(N_ITER):
    idx = np.random.permutation(n)
    test = idx[n//2:]

    resid_mond = y_full[test]  # MOND: g_c=a0 → log(g_c/a0)=0
    pred_geom = 0.5 * x_full[test] + np.median(y_full[idx[:n//2]] - 0.5*x_full[idx[:n//2]])
    resid_geom = y_full[test] - pred_geom

    if np.std(resid_mond) &gt; 0:
        improv = (1 - np.std(resid_geom)/np.std(resid_mond)) * 100
        improv_dist.append(improv)

improv_dist = np.array(improv_dist)
```

```
print(f"  MOND■■■■■■■■1000■■■■■:")
print(f"    ■■■: {np.median(improv_dist):.1f}%")
print(f"    ■■:   {np.mean(improv_dist):.1f}%")
print(f"    std:    {np.std(improv_dist):.1f}%")
print(f"    [5th, 95th]: [{np.percentile(improv_dist,5):.1f}%, {np.percentile(improv_dist,95):.1f}%]")
print(f"    ■■■ &gt; 0% ■■■: {(improv_dist&gt;0).mean()*100:.1f}%")
print(f"    ■■■ &gt; 20% ■■■: {(improv_dist&gt;20).mean()*100:.1f}%")

# ======================================================================
# 6. ■■■■
# ======================================================================
print("\n"+"="*70)
print("[■■■■■■■]")
print("="*70)

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle('SPARC Jackknife Independence Test for Geometric Mean Law',
             fontsize=14, fontweight='bold')

# (a) alpha(train) vs alpha(test) ■■■■
ax = axes[0, 0]
ax.scatter(alphas_train[:200], alphas_test[:200], s=10, alpha=0.3, c='steelblue')
ax.plot([0, 1], [0, 1], 'k--', alpha=0.3)
ax.axhline(0.5, color='g', ls=':', alpha=0.5)
ax.axvline(0.5, color='g', ls=':', alpha=0.5)
ax.axhline(sl_full, color='r', ls='--', alpha=0.3)
ax.axvline(sl_full, color='r', ls='--', alpha=0.3)
ax.set_xlabel('alpha (train half)'); ax.set_ylabel('alpha (test half)')
ax.set_title(f'(a) Random split (N={N_ITER})')
ax.set_xlim(0.2, 0.9); ax.set_ylim(0.2, 0.9)

# (b) alpha ■■■■■■■■■■■■■■
ax = axes[0, 1]
ax.hist(alpha_boot, bins=50, color='steelblue', alpha=0.7, edgecolor='white', density=True)
ax.axvline(0.5, color='g', ls='--', lw=2, label='alpha=0.5')
ax.axvline(sl_full, color='r', ls='-', lw=2, label=f'full={sl_full:.3f}')
ax.axvline(alpha_boot_ci[0], color='orange', ls=':', label=f'95% CI')
ax.axvline(alpha_boot_ci[1], color='orange', ls=':')
ax.axvline(0, color='gray', ls=':', alpha=0.3, label='MOND (alpha=0)')
ax.axvline(1, color='gray', ls=':', alpha=0.3, label='alpha=1')
ax.set_xlabel('alpha'); ax.set_ylabel('Density')
ax.set_title(f'(b) Bootstrap (N={N_BOOT})')
ax.legend(fontsize=7)

# (c) alpha(train) - alpha(test) ■■■
ax = axes[0, 2]
ax.hist(alpha_diffs, bins=40, color='coral', alpha=0.7, edgecolor='white')
ax.axvline(0, color='k', ls='--', alpha=0.5)
ax.set_xlabel('alpha(train) - alpha(test)')
ax.set_ylabel('Count')
ax.set_title(f'(c) Train-test difference (med={np.median(np.abs(alpha_diffs)):.3f})')

# (d) ■■■■■■
ax = axes[1, 0]
ax.hist(improv_dist, bins=40, color='green', alpha=0.7, edgecolor='white')
ax.axvline(0, color='k', ls='--', alpha=0.5)
ax.axvline(np.median(improv_dist), color='r', ls='-', lw=2,
          label=f'median={np.median(improv_dist):.1f}%')
ax.set_xlabel('Improvement over MOND [%]')
ax.set_ylabel('Count')
ax.set_title(f'(d) MOND improvement stability (&gt;{(improv_dist&gt;0).mean()*100:.0f}% positive)')
ax.legend(fontsize=8)
# (e) ■■■■■ std ■■■
ax = axes[1, 1]
ax.hist(resid_test_std, bins=40, color='steelblue', alpha=0.7, edgecolor='white')
full_resid_std = np.std(y_full - (sl_full*x_full+ic_full))
ax.axvline(full_resid_std, color='r', ls='--', lw=2, label=f'full={full_resid_std:.3f}')
ax.set_xlabel('Test residual std [dex]')
ax.set_ylabel('Count')
ax.set_title(f'(e) Generalization (mean={np.mean(resid_test_std):.3f})')
ax.legend(fontsize=8)

# (f) p(alpha=0.5) ■■■
ax = axes[1, 2]
ax.hist(np.log10(p05_test+1e-10), bins=40, color='purple', alpha=0.7, edgecolor='white')
ax.axvline(np.log10(0.05), color='r', ls='--', label='p=0.05')
ax.set_xlabel('log10(p-value for alpha=0.5)')
ax.set_ylabel('Count')
pct_pass = (p05_test &gt; 0.05).mean() * 100
ax.set_title(f'(f) alpha=0.5 test ({pct_pass:.0f}% cannot reject)')
ax.legend(fontsize=8)

plt.tight_layout()
plt.savefig('sparc_jackknife.png', dpi=150, bbox_inches='tight')
print("  -&gt; sparc_jackknife.png")

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

```
# 7. ■■■■■■■
# ======================================================================
print("\n"+"="*70)
print("[■■■■■■■]")
print("="*70)

print(f"""
  SPARC ■■■■■■■■■■■■■■:

  [■■1] ■■■■■■■■ (N={N_ITER}):
    alpha(train) = {np.mean(alphas_train):.4f} +/- {np.std(alphas_train):.4f}
    alpha(test)  = {np.mean(alphas_test):.4f} +/- {np.std(alphas_test):.4f}
    |■| ■■■■  = {np.median(np.abs(alpha_diffs)):.4f}
    → ■■/■■■■■ alpha ■■■■■■■■■■■■■■■■■

  [■■4] ■■■■■■■■■ (N={N_BOOT}):
    alpha = {alpha_boot_mean:.4f} +/- {alpha_boot_std:.4f}
    95% CI: [{alpha_boot_ci[0]:.4f}, {alpha_boot_ci[1]:.4f}]
    alpha=0.5: {'CI■' if alpha_boot_ci[0]&lt;=0.5&lt;=alpha_boot_ci[1] else 'CI■'}
    alpha=0 (MOND): {'CI■' if alpha_boot_ci[0]&lt;=0 else 'CI■'}
    alpha=1: {'CI■' if 1&lt;=alpha_boot_ci[1] else 'CI■'}

  [■■5] MOND■■■■■■■■■:
    ■■■■: {np.median(improv_dist):.1f}%
    ■■■■ &gt; 0%: {(improv_dist&gt;0).mean()*100:.1f}%
    ■■■■ &gt; 20%: {(improv_dist&gt;20).mean()*100:.1f}%

  ■■:
    ■■■■■■■ alpha=0.5 ■ SPARC ■■■■■■■■■■■■■■■■■
    ■■■■■■■■■■■■■■■■■■■■
    {'■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■'
     if (p05_test&gt;0.05).mean()&gt;0.8 else '■■■■■■■■■■■■■■■■■■'}
""")

print("■■")
```