膜宇宙論モデル 隔離考察チャット

観測データ解析スクリプト全文

Phase 1〜7 ／ 目的・入出力・スクリプト全文

2026年4月 | 坂口製麺所

本資料は隔離考察チャットで作成・実行した全スクリプトの目的・入出力・本文を収録したものである。循環論法が判明したスクリプト・棄却済みスクリプトには警告を明記した。

実行環境：Claude Code（VS Code・ローカル）作業ディレクトリ：D:¥ドキュメント¥エントロピー¥新膜宇宙論¥これまでの軌跡¥パイソン¥

目次

重要注意事項：循環論法と独立な結果の区別

☒ 循環論法（使用禁止）：sparc_gc.csvのgc_ratioを使った解析全般。gc_ratio = v_flat^2/(r_s*a0)の定義のため、「g_c ∝ v_flat^1.32」はトートロジー。撤回済み。

[OK] 独立・非循環（有効）：verify_f_direct.py（f^(tanh)=0.262）、verify_f_t3_rs.py（f^(T3)=0.713, p=0.32）、rc_slope_all_sparc.py（r_s_tanh ∝ v_flat^0.795）。

★ 最も信頼できる発見：r_s_tanh ∝ v_flat^0.795（r=0.562, N=175）。唯一の独立した観測的事実。

Phase 1：SPARC 175銀河 回転曲線フィッティング

v2.0の1段階tanhモデル（式7：v_c^2 = v_bar^2 + v_flat^2*T(r,rs)）をSPARC175銀河に適用し、モデルの有効性を検証する。Upsilon_d ≧ 0.30の拘束を標準とした。結果：164/175銀河（94%）で1段階が有効。v2.0の基本構造を強く支持。

スクリプト：fit_sparc_all.py

項目	内容
目的	全175銀河に1段階・2段階tanhを一括適用。sparc_results.csvを生成。
出力	sparc_results.csv (grade, chi2, ΔAIC, r_s, vflat, upsilon_d)
注意	メインのフィッティングスクリプト。全Phase の基盤となるCSVを生成。

▼ スクリプト全文：

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import os, sys, warnings, glob
from pathlib import Path
warnings.filterwarnings('ignore')

UPSILON_MIN = 0.30
UPSILON_B_FIX = 0.70
W_FIX = 1.0
DAIC_THRESHOLD = -10.0
V_ERR_MIN = 2.0

def load_sparc(filepath):
try:
data = np.loadtxt(filepath, comments='#')
except Exception:
return None
if data.ndim == 1 or len(data) < 3:
return None
r = data[:, 0]
v_obs = data[:, 1]
v_err = np.maximum(data[:, 2], V_ERR_MIN)
v_gas = data[:, 3]
v_disk= data[:, 4]
v_bul = data[:, 5] if data.shape[1] > 5 else np.zeros_like(r)
mask = (v_obs > 0) & (r > 0)
if mask.sum() < 4:
return None
return dict(r=r[mask], v_obs=v_obs[mask], v_err=v_err[mask],
v_gas=v_gas[mask], v_disk=v_disk[mask], v_bul=v_bul[mask])

def tanh_T(r, r_s):
return 0.5 * (1.0 + np.tanh(W_FIX * (r - r_s) / r_s))

def v2_baryons(r, v_gas, v_disk, v_bul, ud, ub=0.7):
v2 = np.sign(v_gas) * v_gas**2
v2 += ud * np.sign(v_disk) * v_disk**2
v2 += ub * np.sign(v_bul) * v_bul**2
return v2

def model_1s(r, vf, rs, ud, v_gas, v_disk, v_bul):
v2 = vf**2 * tanh_T(r, rs) + v2_baryons(r, v_gas, v_disk, v_bul, ud)
return np.sqrt(np.maximum(v2, 0))

def model_2s(r, vf, rs1, rs2, fe, ud, v_gas, v_disk, v_bul):
T = fe * tanh_T(r, rs1) + (1-fe) * tanh_T(r, rs2)
v2 = vf**2 * T + v2_baryons(r, v_gas, v_disk, v_bul, ud)
return np.sqrt(np.maximum(v2, 0))

def calc_aic(v_obs, v_fit, v_err, n_params):
return np.sum(((v_obs - v_fit) / v_err)**2) + 2 * n_params

def fit_one(d):
r, v_obs, v_err = d['r'], d['v_obs'], d['v_err']
v_gas, v_disk, v_bul = d['v_gas'], d['v_disk'], d['v_bul']
```

```
n = len(r)
r_max = r.max()
v_max = v_obs.max()
out = {}

def w1(r, vf, rs, ud):
return model_1s(r, vf, rs, ud, v_gas, v_disk, v_bul)
try:
p0 = [v_max*0.9, r_max*0.3, 0.5]
bd = ([5, 0.05, UPSILON_MIN], [500, r_max*2, 2.0])
po, _ = curve_fit(w1, r, v_obs, p0=p0, bounds=bd,
sigma=v_err, absolute_sigma=True, maxfev=30000)
vf1 = w1(r, *po)
out['1s'] = dict(popt=po, ok=True,
chi2=np.sum(((v_obs-vf1)/v_err)**2)/(n-3),
aic=calc_aic(v_obs,vf1,v_err,3), v_fit=vf1)
except:
out['1s'] = dict(ok=False, chi2=np.nan, aic=np.nan)

def w2(r, vf, rs1, rs2, fe, ud):
return model_2s(r, vf, rs1, rs2, fe, ud, v_gas, v_disk, v_bul)
best2 = None
for p0 in [
[v_max*0.9, r_max*0.05, r_max*0.30, 0.5, 0.5],
[v_max*0.9, r_max*0.10, r_max*0.50, 0.3, 0.5],
[v_max*0.8, r_max*0.03, r_max*0.20, 0.7, 0.4],
[v_max*0.9, r_max*0.08, r_max*0.40, 0.4, 0.3],
]:
try:
bd = ([5, 0.01, 0.1, 0.0, UPSILON_MIN],
[500, r_max, r_max*3, 1.0, 2.0])
po, _ = curve_fit(w2, r, v_obs, p0=p0, bounds=bd,
sigma=v_err, absolute_sigma=True, maxfev=30000)
if po[1] > po[2]: po[1], po[2] = po[2], po[1]
vf2 = w2(r, *po)
a2 = calc_aic(v_obs, vf2, v_err, 5)
c2 = np.sum(((v_obs-vf2)/v_err)**2)/(n-5)
if best2 is None or a2 < best2['aic']:
best2 = dict(popt=po, ok=True, chi2=c2, aic=a2, v_fit=vf2)
except:
continue
out['2s'] = best2 if best2 else dict(ok=False, chi2=np.nan, aic=np.nan)
if out['1s']['ok'] and out['2s']['ok']:
out['daic'] = out['2s']['aic'] - out['1s']['aic']
out['grade'] = '2stage' if out['daic'] < DAIC_THRESHOLD else '1stage'
else:
out['daic'] = np.nan
out['grade'] = 'failed'
return out

if __name__ == '__main__':
import csv
data_dir = sys.argv[1] if len(sys.argv) > 1 else '.'
files = sorted(glob.glob(os.path.join(data_dir, '*.dat')))
print(f"対象銀河数：{len(files)}")
results = {}
for i, fpath in enumerate(files):
gal = Path(fpath).stem
d = load_sparc(fpath)
if d is None: continue
res = fit_one(d)
results[gal] = res
print(f"[{i+1:3d}] {gal:<14} "
f"chi2_1s={res['1s'].get('chi2',np.nan):5.2f} "
f"DAIC={res.get('daic',np.nan):7.1f} -> {res.get('grade','?')}")
with open('sparc_results.csv','w',newline='',encoding='utf-8') as f:
writer = csv.writer(f)
writer.writerow(['galaxy','chi2_1s','chi2_2s','daic',
'r_s','vflat','upsilon_d','grade'])
for gal, res in sorted(results.items()):
c1 = res['1s'].get('chi2','')
c2 = res['2s'].get('chi2','')
daic = res.get('daic','')
g = res.get('grade','failed')
```

```
if res['1s'].get('ok') and res['2s'].get('ok'):
p = res['2s']['popt']
writer.writerow([gal,
f'{c1:.4f}' if c1 != '' else '',
f'{c2:.4f}' if c2 != '' else '',
f'{daic:.2f}' if daic != '' else '',
f'{res["1s"]["popt"][1]:.4f}',
f'{res["1s"]["popt"][0]:.4f}',
f'{res["1s"]["popt"][2]:.4f}', g])
print("-> sparc_results.csv 保存完了")
```

Phase 2:条件14型銀河の同定

2段階tanhが有意に有利（ΔAIC < -10）かつバルジなし（v_bul_max ≦ 5 km/s）の銀河を同定する。これらを「条件14型」（膜の二相構造が必要な銀河）の候補とした。確立：条件14型5銀河（NGC1003, NGC2403, NGC2903, NGC6015, UGC00128）。r_s1≈0.6×h_R（弾性相・バリオン連動, r=+0.72）、r_s2はバリオン独立。

スクリプト:find_cond14_galaxies.py

項目	内容
目的	全SPARC175銀河でΔAIC閾値別に条件14型を探索し、バルジなし条件で絞り込む。
出力	cond14_strict.csv / cond14_moderate.csv / cond14_loose.csv
注意	strict（ΔAIC<-10）：5銀河。moderate（ΔAIC<-5）：6銀河。loose（ΔAIC<-2）：9銀河。

▼ スクリプト全文：

```
import numpy as np
from scipy.optimize import curve_fit, brentq
from scipy import stats
import matplotlib.pyplot as plt
import csv, os, sys, warnings, glob
from pathlib import Path
warnings.filterwarnings('ignore')

UPSILON_MIN = 0.30
W_FIX = 1.0
V_ERR_MIN = 2.0
VBUL_THRESHOLD = 5.0
DAIC_LEVELS = {'strict':-10.0, 'moderate':-5.0, 'loose':-2.0}

def load_sparc(filepath):
try:
data = np.loadtxt(filepath, comments='#')
except: return None
if data.ndim==1 or len(data)<4: return None
mask = data[:,1]>0
data = data[mask]
if len(data)<4: return None
r = data[:,0]; v_obs = data[:,1]
v_err = np.maximum(data[:,2], V_ERR_MIN)
v_gas = data[:,3]; v_disk= data[:,4]
v_bul = data[:,5] if data.shape[1]>5 else np.zeros_like(r)
return dict(r=r, v_obs=v_obs, v_err=v_err,
v_gas=v_gas, v_disk=v_disk, v_bul=v_bul,
vbul_max=np.max(np.abs(v_bul)))

def tanh_T(r, r_s):
return 0.5*(1.0+np.tanh(W_FIX*(r-r_s)/r_s))

def v2_bar(r, v_gas, v_disk, v_bul, ud, ub=0.7):
return (np.sign(v_gas)*v_gas**2 + ud*np.sign(v_disk)*v_disk**2
+ ub*np.sign(v_bul)*v_bul**2)

def model_1s(r, vf, rs, ud, v_gas, v_disk, v_bul):
return np.sqrt(np.maximum(vf**2*tanh_T(r,rs)+v2_bar(r,v_gas,v_disk,v_bul,ud),0))

def model_2s(r, vf, rs1, rs2, fe, ud, v_gas, v_disk, v_bul):
T = fe*tanh_T(r,rs1)+(1-fe)*tanh_T(r,rs2)
return np.sqrt(np.maximum(vf**2*T+v2_bar(r,v_gas,v_disk,v_bul,ud),0))

def fit_one(d):
r,v_obs,v_err = d['r'],d['v_obs'],d['v_err']
vg,vd,vb = d['v_gas'],d['v_disk'],d['v_bul']
n=len(r); rm=r.max(); vm=v_obs.max()
out={}
def w1(r,vf,rs,ud): return model_1s(r,vf,rs,ud,vg,vd,vb)
try:
po,_=curve_fit(w1,r,v_obs,p0=[vm*.9,rm*.3,.5],
bounds=([5,.05,UPSILON_MIN],[500,rm*2,2.]),
sigma=v_err,absolute_sigma=True,maxfev=30000)
vf1=w1(r,*po)
out['1s']=dict(popt=po,ok=True,
```

```
chi2=np.sum(((v_obs-vf1)/v_err)**2)/(n-3),
aic=np.sum(((v_obs-vf1)/v_err)**2)+6,v_fit=vf1)
except: out['1s']=dict(ok=False,chi2=np.nan,aic=np.nan)

def w2(r,vf,rs1,rs2,fe,ud): return model_2s(r,vf,rs1,rs2,fe,ud,vg,vd,vb)
best2=None
for p0 in [[vm*.9,rm*.05,rm*.3,.5,.5],[vm*.9,rm*.1,rm*.5,.3,.5]]:
try:
po,_=curve_fit(w2,r,v_obs,p0=p0,
bounds=([5,.01,.1,0,UPSILON_MIN],[500,rm,rm*3,1,2]),
sigma=v_err,absolute_sigma=True,maxfev=30000)
if po[1]>po[2]: po[1],po[2]=po[2],po[1]
vf2=w2(r,*po)
a2=np.sum(((v_obs-vf2)/v_err)**2)+10
c2=np.sum(((v_obs-vf2)/v_err)**2)/(n-5)
if best2 is None or a2<best2['aic']:
best2=dict(popt=po,ok=True,chi2=c2,aic=a2,v_fit=vf2)
except: continue
out['2s']=best2 if best2 else dict(ok=False,chi2=np.nan,aic=np.nan)
if out['1s']['ok'] and out['2s']['ok']:
out['daic']=out['2s']['aic']-out['1s']['aic']
else: out['daic']=np.nan
return out

def extract_cond14(results, daic_threshold):
cond14={}
for gal,res in results.items():
daic=res.get('daic',np.nan)
if np.isnan(daic) or daic>=daic_threshold: continue
if res.get('vbul_max',999)>VBUL_THRESHOLD: continue
if not res['2s'].get('ok'): continue
cond14[gal]=res
return cond14

if __name__=='__main__':
data_dir=sys.argv[1] if len(sys.argv)>1 else '.'
files=sorted(glob.glob(os.path.join(data_dir,'*.dat')))
print(f"対象：{len(files)}銀河")
all_results={}
for fpath in files:
gal=Path(fpath).stem
d=load_sparc(fpath)
if d is None: continue
res=fit_one(d)
res['vbul_max']=d['vbul_max']
all_results[gal]=res

for level,threshold in DAIC_LEVELS.items():
cd=extract_cond14(all_results,threshold)
print(f"DAIC<{threshold}: {len(cd)}銀河 -> {sorted(cd.keys())}")
with open(f'cond14_{level}.csv','w',newline='',encoding='utf-8') as f:
w=csv.writer(f)
w.writerow(['galaxy','chi2_1s','chi2_2s','daic','r_s1','r_s2','f_e','upsilon_d'])
for gal,res in sorted(cd.items()):
p=res['2s']['popt']
w.writerow([gal,
f"{res['1s']['chi2']:.4f}",f"{res['2s']['chi2']:.4f}",
f"{res['daic']:.2f}",f"{p[1]:.4f}",f"{p[2]:.4f}",
f"{p[3]:.4f}",f"{p[4]:.4f}"])
print(f" -> cond14_{level}.csv 保存完了")
```

Phase 3:回転曲線形状解析

g_cスケーリング則の解析と並行して、回転曲線の形状複雑さ指標を計算し、条件14型との関係を探索した。結果（ヌル）：条件14型は形状指標（slope, vrise等）では識別不可能。注意：gc_ratioがトートロジーと判明後、g_c関連解析は全て撤回。

スクリプト:rc_slope_all_sparc.py

項目	内容
目的	全175銀河のRC slope（log-logスロープ）を計算。条件14型との分布比較。
出力	rc_slope_results.csv
注意	slope全域でMann-Whitney p=0.015（有意だが方向が予想と逆）。ヌル結果。

▼ スクリプト全文：

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import glob, os, csv, warnings
from pathlib import Path
warnings.filterwarnings('ignore')

COND14 = ['NGC1003','NGC2403','NGC2903','NGC6015','UGC00128']

def load_sparc(filepath):
try: data = np.loadtxt(filepath, comments='#')
except: return None
if data.ndim==1 or len(data)<4: return None
mask=data[:,1]>0; data=data[mask]
if len(data)<4: return None
return dict(r=data[:,0], v_obs=data[:,1],
v_bul=data[:,5] if data.shape[1]>5 else np.zeros(len(data)))

def calc_slope(r, v_obs):
mask=(r>0)&(v_obs>0)
if mask.sum()<3: return np.nan
s,_,_,_,_=stats.linregress(np.log10(r[mask]),np.log10(v_obs[mask]))
return s

def calc_slope_outer(r, v_obs):
mask=(r>np.median(r))&(v_obs>0)
if mask.sum()<3: return np.nan
s,_,_,_,_=stats.linregress(np.log10(r[mask]),np.log10(v_obs[mask]))
return s

def calc_vrise(r, v_obs):
r_med=np.median(r)
vi=np.mean(v_obs[r<=r_med]); vo=np.mean(v_obs[r>r_med])
return vo/vi if vi>0 else np.nan

if __name__=='__main__':
import sys
data_dir=sys.argv[1] if len(sys.argv)>1 else '.'
files=sorted(glob.glob(os.path.join(data_dir,'*.dat')))
grade_map={}
csv_path=os.path.join(data_dir,'sparc_results.csv')
if os.path.exists(csv_path):
with open(csv_path,newline='',encoding='utf-8') as f:
for row in csv.DictReader(f):
grade_map[row['galaxy']]=row['grade']

results=[]
for fpath in files:
gal=Path(fpath).stem
d=load_sparc(fpath)
if d is None: continue
results.append(dict(
galaxy=gal, grade=grade_map.get(gal,'unknown'),
slope=calc_slope(d['r'],d['v_obs']),
slope_outer=calc_slope_outer(d['r'],d['v_obs']),
vrise=calc_vrise(d['r'],d['v_obs']),
vbul_max=np.max(np.abs(d['v_bul'])),
```

```
no_bulge=np.max(np.abs(d['v_bul']))<=5.0,
is_cond14=gal in COND14,
))

c14 =[r for r in results if r['is_cond14']]
s1_nb=[r for r in results if r['grade']=='1stage' and r['no_bulge']]

for name,lst in [('C14',c14),('1stage_nobulge',s1_nb)]:
sl=np.array([r['slope'] for r in lst if not np.isnan(r['slope'])])
vr=np.array([r['vrise'] for r in lst if not np.isnan(r['vrise'])])
print(f"{name}: N={len(sl)} slope中央値={np.median(sl):.3f} "
f"vrise中央値={np.median(vr):.3f}")

if len(c14)>=3 and len(s1_nb)>=3:
sl_c14=np.array([r['slope'] for r in c14 if not np.isnan(r['slope'])])
sl_1s =np.array([r['slope'] for r in s1_nb if not np.isnan(r['slope'])])
_,p=stats.mannwhitneyu(sl_c14,sl_1s,alternative='two-sided')
print(f"Mann-Whitney slope: p={p:.4f}")

with open('rc_slope_results.csv','w',newline='',encoding='utf-8') as f:
w=csv.DictWriter(f,fieldnames=[
'galaxy','grade','is_cond14','no_bulge',
'slope','slope_outer','vrise','vbul_max'])
w.writeheader(); w.writerows(results)
print("-> rc_slope_results.csv 保存完了")
```

Phase 4：g_c の定義確認（循環論法の発覚）

sparc_gc.csvのgc_ratioが独立なフィットパラメータか定義式からの計算値かを判別する。結果（重大）：gc_ratio = v_flat^2/(r_s*a0) と確定（全171銀河でstd=0.00000000）。これにより「g_c ∝ v_flat^1.32」はトートロジーと判明し撤回。真の独立した発見は「r_s_tanh ∝ v_flat^0.795」のみとなった。

スクリプト：check_gc_definition.py

項目	内容
目的	gc_ratio = v_flat^2/(r_s*a0) かどうかをstd=0で判定。
出力	定義A確定の確認（コンソール出力のみ）
注意	★ 最重要スクリプト。g_c ∝ v_flat^1.32 の撤回根拠を確立。

▼ スクリプト全文：

```
import numpy as np
import csv, os, sys
from scipy import stats

a0 = 1.2e-10 # m/s^2
kpc2m = 3.086e19
kms2ms= 1e3

data_dir = sys.argv[1] if len(sys.argv)>=2 else '.'
gc_csv = os.path.join(data_dir, 'sparc_gc.csv')

with open(gc_csv, newline='', encoding='utf-8') as f:
reader = csv.DictReader(f)
headers = reader.fieldnames
rows = list(reader)

print(f"列名：{headers}")
print(f"行数：{len(rows)}")

# 定義A検定：gc = v_flat^2 / (r_s * a0) か
ratios = []
for row in rows:
try:
gc = float(row.get('gc_ratio','') or 'nan')
vf = float(row.get('vflat','') or 'nan')
rs = float(row.get('r_s','') or 'nan')
if not all(np.isfinite([gc,vf,rs])): continue
if gc<=0 or vf<=0 or rs<=0: continue
gc_SI = gc * a0
vf_SI = vf * kms2ms
rs_SI = rs * kpc2m
ratios.append(gc_SI * rs_SI / vf_SI**2)
except: pass

ratios = np.array(ratios)
print(f"¥nN = {len(ratios)}")
print(f"ratio = gc*r_s*a0/v_flat^2:")
print(f" 中央値 = {np.median(ratios):.8f}")
print(f" std = {np.std(ratios):.10f}")

if np.std(ratios) < 1e-6:
print("¥n★ 定義A確定：gc_ratio = v_flat^2 / (r_s * a0)")
print(" g_c ∝ v_flat^1.32 は r_s ∝ v_flat^0.68 の言い換え（循環論法）")
else:
print(f"¥n★ 定義B可能性：std={np.std(ratios):.4f} > 1e-6")

# log-log スロープ確認
gc_arr = np.array([float(r.get('gc_ratio','nan')) for r in rows if r.get('gc_ratio')])
vf_arr = np.array([float(r.get('vflat','nan')) for r in rows if r.get('vflat')])
rs_arr = np.array([float(r.get('r_s','nan')) for r in rows if r.get('r_s')])
```

```
valid = np.isfinite(gc_arr)&np.isfinite(vf_arr)&np.isfinite(rs_arr)
gc_v=gc_arr[valid]; vf_v=vf_arr[valid]; rs_v=rs_arr[valid]

s_vf,_,_,_,_=stats.linregress(np.log10(vf_v),np.log10(gc_v))
s_rs,_,_,_,_=stats.linregress(np.log10(rs_v),np.log10(gc_v))
print(f"¥nlog(gc) vs log(v_flat): slope={s_vf:.3f}（期待値: +2.0 if 定義A)")
print(f"log(gc) vs log(r_s) : slope={s_rs:.3f}（期待値: -1.0 if 定義A)")
print(f"共線性（r_s ∝ v_flat^0.795）のため実測はずれる（整合している）")
```

Phase 5:f = phi/2 の検証(2種類のr_sで別々に実施)

v3.0内部整合条件として導出したf = phi/2 = 0.809を観測で検証する。重要：r_s_tanh（tanhフィット）とr_s^(T3)（T-3定義）は別物であり、2種類のfは直接比較できない。f^(tanh)=0.262（独立実測・外挿多数）、f^(T3)=0.713（p=0.32, N=46）。発見：r_s_tanh ≈ 2.75×r_s^(T3)、74%の銀河は全域MOND型。

スクリプト:verify_f_direct.py

項目	内容
目的	SPARCの.datからv_bar(r_s_tanh)を直接計算。循環論法なし。
出力	f^(tanh)中央値=0.262（v_bar≈0.512*v_flat）
注意	独立・非循環的な唯一のf^(tanh)実測。ただし外挿銀河多数で堅牢性要確認。

▼ スクリプト全文：

```
import numpy as np
from scipy import stats, interpolate
import matplotlib.pyplot as plt
import csv, os, sys, warnings
from pathlib import Path
warnings.filterwarnings('ignore')

phi = (1 + 5**0.5) / 2
f_theory = phi / 2 # = 0.8090
a0 = 1.2e-10
kpc2m = 3.086e19
kms2ms = 1e3

def load_sparc_results(path):
rows = {}
with open(path, newline='', encoding='utf-8') as f:
for row in csv.DictReader(f):
try:
gal=row['galaxy']
vf=float(row.get('vflat','') or 'nan')
rs=float(row.get('r_s','') or 'nan')
ups=float(row.get('upsilon_d','') or 'nan')
if np.isfinite(vf) and np.isfinite(rs) and vf>0 and rs>0:
rows[gal]={'vflat':vf,'r_s_tanh':rs,'upsilon':ups}
except: pass
return rows

def load_baryon_profile(filepath, upsilon_d, upsilon_b=0.7):
try: data=np.loadtxt(filepath,comments='#')
except: return None
if data.ndim==1 or len(data)<4: return None
mask=data[:,1]>0; data=data[mask]
if len(data)<4: return None
r=data[:,0]; v_gas=data[:,3]; v_disk=data[:,4]
v_bul=data[:,5] if data.shape[1]>5 else np.zeros_like(r)
v2_bar=(np.sign(v_gas)*v_gas**2+upsilon_d*np.sign(v_disk)*v_disk**2
+upsilon_b*np.sign(v_bul)*v_bul**2)
v2_bar=np.maximum(v2_bar,0)
return dict(r=r, v_bar=np.sqrt(v2_bar), v_obs=data[:,1])

def interpolate_vbar_at_rs(r, v_bar, r_s):
try:
fi=interpolate.interp1d(r,v_bar,kind='linear',fill_value='extrapolate')
return float(fi(r_s))
except: return np.nan

if __name__=='__main__':
data_dir=sys.argv[1] if len(sys.argv)>=2 else '.'
results_csv=os.path.join(data_dir,'sparc_results.csv')
params_all=load_sparc_results(results_csv)
records=[]
for gal,params in params_all.items():
dat_path=os.path.join(data_dir,f'{gal}.dat')
if not os.path.exists(dat_path): continue
ups=max(params['upsilon'] if np.isfinite(params['upsilon']) else 0.5, 0.05)
d=load_baryon_profile(dat_path,upsilon_d=ups)
```

```
if d is None: continue
vbar_rs=interpolate_vbar_at_rs(d['r'],d['v_bar'],params['r_s_tanh'])
if not np.isfinite(vbar_rs) or vbar_rs<=0: continue
f_obs=(vbar_rs/params['vflat'])**2
extrapolated=(params['r_s_tanh']<d['r'].min() or
params['r_s_tanh']>d['r'].max())
records.append(dict(galaxy=gal,vflat=params['vflat'],
r_s=params['r_s_tanh'],vbar_rs=vbar_rs,
f_obs=f_obs,upsilon=ups,extrapolated=extrapolated))

f_arr=np.array([r['f_obs'] for r in records])
print(f"N={len(f_arr)}")
print(f"f_obs(tanh): 中央値={np.median(f_arr):.4f} "
f"理論値phi/2={f_theory:.4f}")
print(f"v_bar(r_s_tanh)/v_flat 中央値 = {np.median(f_arr)**0.5:.4f}")
_,p=stats.wilcoxon(f_arr-f_theory)
print(f"Wilcoxon p={p:.4f} (H0: median=phi/2)")

r_vf,p_vf=stats.spearmanr(np.array([r['vflat'] for r in records]),f_arr)
print(f"f vs v_flat: Spearman r={r_vf:+.4f}, p={p_vf:.4f}")
print(f"¥n注意：外挿銀河={sum(1 for r in records if r['extrapolated'])}個")
print(f" 0.262の堅牢性は外挿を除いた再計算で確認が必要")
```

スクリプト:verify_f_t3_rs.py

項目	内容
目的	g_N(r)=a0となるr_s^(T3)でf^(T3)を計算。g_c候補4値を比較。
出力	g_c=a0でf中央値=0.713、p=0.32（棄却できず）、N=46
注意	選択バイアスあり（中大質量銀河のみ）。Level B。

▼ スクリプト全文：

```
import numpy as np
from scipy import stats, interpolate
import matplotlib.pyplot as plt
import csv, os, sys, warnings
from pathlib import Path
warnings.filterwarnings('ignore')

phi = (1 + 5**0.5) / 2
f_theory = phi / 2
a0 = 1.2e-10
kpc2m = 3.086e19
kms2ms = 1e3

def load_sparc_results(path):
rows={}
with open(path,newline='',encoding='utf-8') as f:
for row in csv.DictReader(f):
try:
gal=row['galaxy']
vf=float(row.get('vflat','') or 'nan')
rs=float(row.get('r_s','') or 'nan')
ups=float(row.get('upsilon_d','') or 'nan')
if np.isfinite(vf) and np.isfinite(rs) and vf>0 and rs>0:
rows[gal]={'vflat':vf,'r_s_tanh':rs,'upsilon':ups}
except: pass
return rows

def load_baryon_profile(filepath, upsilon_d, upsilon_b=0.7):
try: data=np.loadtxt(filepath,comments='#')
except: return None
if data.ndim==1 or len(data)<4: return None
mask=data[:,1]>0; data=data[mask]
if len(data)<4: return None
r=data[:,0]; v_gas=data[:,3]; v_disk=data[:,4]
v_bul=data[:,5] if data.shape[1]>5 else np.zeros_like(r)
v2_bar=(np.sign(v_gas)*v_gas**2+upsilon_d*np.sign(v_disk)*v_disk**2
+upsilon_b*np.sign(v_bul)*v_bul**2)
v2_bar=np.maximum(v2_bar,0)
g_N=v2_bar*kms2ms**2/(r*kpc2m)
return dict(r=r,g_N=g_N,v_bar=np.sqrt(v2_bar))
```

```
def find_rs_t3(r, g_N, g_c_SI):
crossings=[]
for i in range(len(r)-1):
d1=g_N[i]-g_c_SI; d2=g_N[i+1]-g_c_SI
if d1*d2<=0 and d1!=d2:
crossings.append(r[i]+(r[i+1]-r[i])*(-d1)/(d2-d1))
return (crossings[0],len(crossings)) if crossings else (np.nan,0)

if __name__=='__main__':
data_dir=sys.argv[1] if len(sys.argv)>=2 else '.'
params_all=load_sparc_results(os.path.join(data_dir,'sparc_results.csv'))

gc_candidates={'a0':a0,'0.55xa0':0.55*a0,'0.28xa0':0.28*a0,'2xa0':2.0*a0}
all_results={label:[] for label in gc_candidates}

for gal,params in params_all.items():
dat_path=os.path.join(data_dir,f'{gal}.dat')
if not os.path.exists(dat_path): continue
ups=max(params['upsilon'] if np.isfinite(params['upsilon']) else 0.5, 0.05)
bary=load_baryon_profile(dat_path,upsilon_d=ups)
if bary is None: continue
for label,gc_SI in gc_candidates.items():
rs_t3,_=find_rs_t3(bary['r'],bary['g_N'],gc_SI)
if np.isnan(rs_t3): continue
try:
fi=interpolate.interp1d(bary['r'],bary['v_bar'],
kind='linear',fill_value='extrapolate')
vbar_rs=float(fi(rs_t3))
except: continue
if not np.isfinite(vbar_rs) or vbar_rs<=0: continue
f_obs=(vbar_rs/params['vflat'])**2
if not np.isfinite(f_obs) or f_obs<=0 or f_obs>100: continue
all_results[label].append(dict(
galaxy=gal,vflat=params['vflat'],
rs_tanh=params['r_s_tanh'],rs_t3=rs_t3,
vbar_rs=vbar_rs,f_obs=f_obs))

best_label=None; best_p=0.0
for label,recs in all_results.items():
if len(recs)<5: continue
f_arr=np.array([r['f_obs'] for r in recs])
_,p=stats.wilcoxon(f_arr-f_theory)
print(f"g_c={label}: N={len(f_arr)} "
f"f中央値={np.median(f_arr):.4f} p={p:.4f} "
f"{'★最良' if p>best_p else ''}")
if p>best_p: best_p=p; best_label=label

if best_label:
recs=all_results[best_label]
f_arr=np.array([r['f_obs'] for r in recs])
rs_t3=np.array([r['rs_t3'] for r in recs])
rs_th=np.array([r['rs_tanh'] for r in recs])
vf_arr=np.array([r['vflat'] for r in recs])
rs_ratio=rs_th/rs_t3
s,_,r_val,_,se=stats.linregress(np.log10(vf_arr),np.log10(rs_t3))
print(f"¥n最良候補: g_c={best_label}")
print(f"r_s^T3 ∝ v_flat^{s:.3f} (r={r_val:.3f})")
print(f"rs_tanh/rs_t3 中央値={np.median(rs_ratio):.2f}")
```

Phase 6:HSC Y3 × Miyaoka 弱重力レンズ解析

HSC-SSP Y3の弱重力レンズ形状カタログ（3580万天体）を使い、Miyaoka 2018の22銀河団周辺のタンジェンシャルシアプロファイルを計算する。根本的制約：Miyaoka22クラスターの多くがHSC Y3フットプリント外。有効クラスターは3個のみ。N=3では統計的結論不能。

スクリプト:compute_shear_profile.py

項目	内容
目的	有効3クラスター周辺のタンジェンシャルシアgamma_t(R)を計算。
出力	J1311: gamma_t=+0.222 (N_bg=25815) 、J1415: gamma_t≈0
注意	N=3のみ。統計的結論不能。方向は理論と一致 (Level B) 。

▼ スクリプト全文：

```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
import sys, os, warnings
warnings.filterwarnings('ignore')

TARGETS = [
('J1415.2-0030', 213.80917, -0.50111, 0.1403, 0.202),
('J1115.8+0129', 168.97500, 1.49556, 0.3499, 0.729),
('J1023.6+0411', 155.91167, 4.18639, 0.2850, 0.869),
('J0157.4-0550', 29.35125, -5.84000, 0.1289, 0.840),
('J0231.7-0451', 37.94708, -4.85583, 0.1843, 0.528),
('J1311.5-0120', 197.87500, -1.33528, 0.1832, 0.736),
('J1258.6-0145', 194.67125, -1.75694, 0.0845, 0.520),
('J0106.8+0103', 16.70958, 1.05472, 0.2537, 0.484),
('J1217.6+0339', 184.41917, 3.66250, 0.0766, 0.826),
]
Z_BG_MIN = 2
R_MIN_ARCMIN = 1.0; R_MAX_ARCMIN = 30.0; N_BINS = 10

def compute_tangential_shear(src_ra,src_dec,e1,e2,weight,cen_ra,cen_dec):
dra =(src_ra -cen_ra)*np.cos(np.radians(cen_dec))
ddec= src_dec-cen_dec
phi = np.arctan2(ddec,dra)
gamma_t = -(e1*np.cos(2*phi)+e2*np.sin(2*phi))
gamma_x = -(e2*np.cos(2*phi)-e1*np.sin(2*phi))
return gamma_t, gamma_x

def shear_profile(src_ra,src_dec,e1,e2,weight,cen_ra,cen_dec,
r_min=1.0,r_max=30.0,n_bins=10):
dra =(src_ra -cen_ra)*np.cos(np.radians(cen_dec))*60
ddec=(src_dec-cen_dec)*60
r =np.sqrt(dra**2+ddec**2)
gt,gx=compute_tangential_shear(src_ra,src_dec,e1,e2,weight,cen_ra,cen_dec)
bins=np.logspace(np.log10(r_min),np.log10(r_max),n_bins+1)
r_center=np.sqrt(bins[:-1]*bins[1:])
gt_mean=np.zeros(n_bins); gt_err=np.zeros(n_bins); n_src=np.zeros(n_bins,dtype=int)
for i in range(n_bins):
mask=(r>=bins[i])&(r<bins[i+1])
if mask.sum()<3:
gt_mean[i]=gt_err[i]=np.nan; continue
w=weight[mask]; w_sum=w.sum()
gt_mean[i]=np.sum(gt[mask]*w)/w_sum
gt_err[i] =0.28/np.sqrt(mask.sum())
n_src[i] =mask.sum()
return r_center,gt_mean,gt_err,n_src

def process_cluster(name,ra,dec,z,s_plastic,src_df):
bg=src_df[src_df['hsc_y3_zbin']>=Z_BG_MIN].copy()
cos_dec=np.cos(np.radians(dec))
dra=(bg['i_ra']-ra)*cos_dec*60
ddec=(bg['i_dec']-dec)*60
dist=np.sqrt(dra**2+ddec**2)
bg=bg[dist<=35.0]
if len(bg)<50:
```

```
print(f" [SKIP] {name}: 背景銀河数不足({len(bg)})"); return None
r_cen,gt,gt_err,n_src=shear_profile(
bg['i_ra'].values,bg['i_dec'].values,
bg['i_hsmshaperegauss_e1'].values,
bg['i_hsmshaperegauss_e2'].values,
bg['_derived_weight'].values,ra,dec,
R_MIN_ARCMIN,R_MAX_ARCMIN,N_BINS)
print(f" {name}: N_bg={len(bg)}, "
f"gamma_t(inner)={np.nanmean(gt[r_cen<5]):.3f}")
return dict(name=name,ra=ra,dec=dec,z=z,s_plastic=s_plastic,
r=r_cen,gt=gt,gt_err=gt_err,n_bg=len(bg))

if __name__=='__main__':
data_dir=sys.argv[1] if len(sys.argv)>1 else '.'
hsc_path=os.path.join(data_dir,'hsc_GAMA15_sources.csv')
if not os.path.exists(hsc_path):
print(f"[ERROR] {hsc_path} が見つかりません"); sys.exit(1)
print("HSCカタログ読み込み中...")
src_df=pd.read_csv(hsc_path)
print(f" -> {len(src_df)}天体")
results=[]
for name,ra,dec,z,sp in TARGETS:
print(f"¥n{name} 解析中...")
res=process_cluster(name,ra,dec,z,sp,src_df)
if res: results.append(res)
print(f"¥n解析完了：{len(results)}クラスター")
```

Phase 7:修正ポテンシャルの検証(全て棄却済み)

v3.0のU(epsilon)が単安定（epsilon=0のみ安定）であることを確認し、可能性C「二安定ポテンシャル・メタンアナロジー」を検証した。結論：修正ポテンシャルはdU/d(epsilon)=g_N/g_cを変更するためT-3（MOND補間則の第一原理導出）と共存不可能。全スクリプトの結果は参考記録。次フェーズでは使用しない。

スクリプト:double_well_final.py

項目	内容
目的	修正ポテンシャル候補B・Cの最終二安定判定。棄却理由を確立。
出力	候補B・CでAlpha≧2の二安定を確認するが、T-3との共存不可能を示す。
注意	★ 棄却の最終確認スクリプト。

▼ スクリプト全文：

```
import numpy as np
from scipy.optimize import brentq
from scipy import stats
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

phi = (1+5**0.5)/2
f_theory = phi/2

def U(eps, clip=1e-6):
eps=np.clip(eps,0,1-clip)
return -eps-eps**2/2-np.log(1-eps)

def dU(eps, clip=1e-6):
eps=np.clip(eps,0,1-clip)
return -1-eps+1/(1-eps)

def d2U(eps, clip=1e-6):
eps=np.clip(eps,0,1-clip)
return -1+1/(1-eps)**2

def U_B(eps,alpha,clip=1e-6):
eps=np.clip(eps,0,1-clip); return U(eps,clip)-alpha*eps**4
def dU_B(eps,alpha,clip=1e-6):
eps=np.clip(eps,0,1-clip); return dU(eps,clip)-4*alpha*eps**3
def d2U_B(eps,alpha,clip=1e-6):
eps=np.clip(eps,0,1-clip); return d2U(eps,clip)-12*alpha*eps**2

def U_C(eps,alpha,beta=0.5,clip=1e-6):
eps=np.clip(eps,0,1-clip); return U(eps,clip)+beta*eps**2-alpha*eps**3
def dU_C(eps,alpha,beta=0.5,clip=1e-6):
eps=np.clip(eps,0,1-clip); return dU(eps,clip)+2*beta*eps-3*alpha*eps**2
def d2U_C(eps,alpha,beta=0.5,clip=1e-6):
eps=np.clip(eps,0,1-clip); return d2U(eps,clip)+2*beta-6*alpha*eps

def find_all_equilibria(dU_func,d2U_func,alpha,extra_args=(),n=3000):
eps_grid=np.linspace(1e-6,1-1e-6,n)
stable=[]; unstable=[]
d2_at_0=d2U_func(1e-6,alpha,*extra_args)
if d2_at_0>0: stable.append(0.0)
for i in range(len(eps_grid)-1):
e1=eps_grid[i]; e2=eps_grid[i+1]
f1=dU_func(e1,alpha,*extra_args); f2=dU_func(e2,alpha,*extra_args)
if f1*f2<0:
try:
e_eq=brentq(lambda e:dU_func(e,alpha,*extra_args),e1,e2,xtol=1e-8)
d2=d2U_func(e_eq,alpha,*extra_args)
(stable if d2>0 else unstable).append(e_eq)
except: pass
return stable,unstable

if __name__=='__main__':
print(f"v3.0のU(epsilon):")
print(f" d2U(eps=0) = {d2U(1e-6):.4f} (0 → 縮退した安定点)")
print(f" U(epsilon) は単安定 (epsilon=0のみ安定) ")
```

```
print(f"¥n候補B・CのAlpha≧2での二安定確認：")
for alpha in [1.0,2.0,5.0]:
sB,uB=find_all_equilibria(dU_B,d2U_B,alpha)
sC,uC=find_all_equilibria(dU_C,d2U_C,alpha,(0.5,))
bistB=len(sB)>=2; bistC=len(sC)>=2
print(f" alpha={alpha}: B={'二安定' if bistB else '単安定'} "
f"C={'二安定' if bistC else '単安定'}")

print(f"¥n★ 棄却理由：修正ポテンシャルはdU/d(epsilon)=g_N/g_cを変更するため")
print(f" T-3（MOND補間則の第一原理導出）と共存不可能")
print(f" → 可能性C・メタンアナロジーを棄却。v3.0のU(epsilon)を維持。")
```