Regression_Simulations

September 22, 2025

1 X– Regression Simulations

> We developed the X– framework to bridge key quantum (QM) phase phenomena with geometric/gravitational (GR) structure via an emergent compact degree of freedom . This notebook proposes three falsifiable experiments to test core predictions of the theory.

Overview - The X– framework introduces a compact variable whose holonomy (_) couples to motion in real space X. This geometric coupling yields AB-like interference, cross-Hall-type drifts, and rotor spectra shifted by holonomy—uniting QM phase with GR-style geometry in a minimal, testable model. - Here we outline and fit three experiments that, together, can validate or falsify the framework: 1) –AB fringe: I(_) = A + B cos(n _ +) 2) Cross-Hall drift: Δy(T) T² + c 3) Rotor spectrum: E( ) = ²/(2I) (− _ /2)² - Each section includes a short derivation and explicit falsifiability criteria.

How to run - Cell 1: Imports - Cell 2: Configuration (MODE github/colab/local, BASE_DIR, USE_SYNTHETIC) - Cell 3: Utilities (paths, models) - Cell 4: Synthetic data generation (optional) - Cell 5: Exp1 — –AB fringe (theory → fit → plot) - Cell 6: Exp2 — Cross-Hall drift (theory → fit → plot) - Cell 7: Exp3 — Rotor spectrum (theory → fit → plot)

Outputs (default GitHub mode) - Figures → `paper/build/figs/regression/` - Summary JSON → `paper/build/analysis/regression_summary.json` - Optional synthetic CSVs → `paper/sims/`

Colab tip: mount Drive and set MODE='colab', BASE_DIR to a folder in Drive before running.

1.1 Outputs and paths

It supports running inside this GitHub repo, locally, or in Google Colab.

Experiments covered: - Exp1: –AB fringe I(_) = A + B cos(n _ +) - Exp2: Cross-Hall drift Δy T² + c - Exp3: Rotor spectrum E( ) = ²/(2I) (− _ /2)²

Outputs (GitHub mode): - Figures → `paper/build/figs/regression/` - Summary JSON → `paper/build/analysis/regression_summary.json` - Optional synthetic CSVs → `paper/sims/`

Colab tip: mount Drive and set MODE='colab', BASE_DIR='/content/drive/MyDrive/YourFolder' before running cells.

[338]:
```
# Cell 1 - Imports
import os, json, math, sys
from typing import Literal
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.optimize import curve_fit
from IPython.display import display, Markdown, Image

sns.set_context("talk"); sns.set_style("whitegrid")
plt.rcParams.update({
    "figure.dpi": 120,
    "savefig.dpi": 200,
    "axes.spines.top": False,
    "axes.spines.right": False,
})

HBAR = 1.054_571_817e-34  # J·s
print("Imports ready; HBAR =", HBAR)
```

```
Imports ready; HBAR = 1.054571817e-34
```

[339]:
```
# Cell 2 - Knobs & configuration (github/local/colab)
EnvType = Literal['github','colab','local']

MODE: EnvType = 'github'  # change if running elsewhere
BASE_DIR = None           # set when MODE in {'colab','local'}

USE_SYNTHETIC = True      # generate CSVs under paper/sims and use them
SHOW_DETAILS  = True      # verbose prints and context
```

[340]:
```
# Cell 3 - Utilities (path resolver, helpers)
def ensure_dir(p: str) -> str:
    os.makedirs(p, exist_ok=True)
    return p

def resolve_paths(mode: EnvType = 'github', base_dir: str | None = None):
    if mode == 'github':
        repo_root = os.path.abspath(os.path.join(os.getcwd(), '..'))
        paper_dir = os.path.join(repo_root, 'paper')
    else:
        if not base_dir:
            raise ValueError("When MODE is 'colab' or 'local', set BASE_DIR to␣
 ↪a valid path.")
        paper_dir = os.path.join(base_dir, 'paper')
    data_dir     = ensure_dir(os.path.join(paper_dir, 'data'))
    sims_dir     = ensure_dir(os.path.join(paper_dir, 'sims'))
    build_dir    = ensure_dir(os.path.join(paper_dir, 'build'))
    figs_out     = ensure_dir(os.path.join(build_dir, 'figs', 'regression'))
    analysis_out = ensure_dir(os.path.join(build_dir, 'analysis'))
```

```
    if SHOW_DETAILS:
        print(f"MODE={mode} paper_dir={paper_dir}")
    print(f"data_dir={data_dir}\nsims_dir={sims_dir}\nfigs_out={figs_out}\nanalysis_out={analys
    return paper_dir, data_dir, sims_dir, build_dir, figs_out, analysis_out

paper_dir, data_dir, synth_dir, build_dir, figs_out, analysis_out = resolve_paths(MODE, BASE_DIR)

# Math models
def ab_fringe_model(phi, A, B, phi0, n):
    return A + B * np.cos(n * phi + phi0)

def linear_model(x, m, b):
    return m * x + b

def rotor_energy_model(ell, I, phi):
    return (HBAR**2 / (2.0 * I)) * (ell - phi / (2.0 * np.pi))**2

def rotor_energy_model_K(ell, K, phi):
    return K * (ell - phi / (2.0 * np.pi))**2
```

```
MODE=github paper_dir=c:\workspace\Physics\X-theta-framework\paper
data_dir=c:\workspace\Physics\X-theta-framework\paper\data
sims_dir=c:\workspace\Physics\X-theta-framework\paper\sims
figs_out=c:\workspace\Physics\X-theta-framework\paper\build\figs\regression
analysis_out=c:\workspace\Physics\X-theta-framework\paper\build\analysis
```

[341]:
```
# Utilities (paths, models, helpers)
import os, json, math
import numpy as np
import pandas as pd
from typing import Literal

HBAR = 1.054_571_817e-34  # J·s
EV2J = 1.602_176_634e-19  # J per eV

# Energy unit config (set to 'J' or 'eV'); can be overridden via env XT_ENERGY_UNIT
ENERGY_UNIT = os.environ.get('XT_ENERGY_UNIT', 'J').strip().lower()

def ensure_dir(p: str):
    os.makedirs(p, exist_ok=True)

def ab_fringe_model(phi, A, B, phi0, n):
    return A + B*np.cos(n*phi + phi0)
```

```
# Semantic clarity: this is a parabola in T, linear in T^2
def parabola_in_T(T2, kappa, c):
    return kappa*T2 + c

# Backward-compat alias for existing code
def linear_model(x, m, b):
    return parabola_in_T(x, m, b)

def rotor_energy_model(ell, I, phi):
    return (HBAR**2/(2.0*I)) * (ell - phi/(2.0*np.pi))**2

def rotor_energy_model_K(ell, K, phi):
    return K * (ell - phi/(2.0*np.pi))**2

def convert_energy_to_joules(values: np.ndarray, unit: str) -> np.ndarray:
    u = (unit or 'J').lower()
    if u == 'j' or u == 'joule' or u == 'joules':
        return values.astype(float)
    if u == 'ev' or u == 'electronvolt' or u == 'electronvolts':
        return (values.astype(float) * EV2J)
    # Unknown unit, return as-is but warn via print
    print(f"[warn] Unknown ENERGY_UNIT='{unit}', assuming Joules.")
    return values.astype(float)

# Resolve paths once
repo_root = os.path.abspath(os.path.join(os.getcwd()))
paper_dir = os.path.join(repo_root, 'paper')
data_dir  = os.path.join(paper_dir, 'data')
build_dir = os.path.join(paper_dir, 'build')
figs_dir  = os.path.join(build_dir, 'figs')
analysis_dir = os.path.join(build_dir, 'analysis')
synth_dir = os.path.join(paper_dir, 'sims')
ensure_dir(data_dir); ensure_dir(build_dir); ensure_dir(figs_dir);␣
 ↪ensure_dir(analysis_dir); ensure_dir(synth_dir)

# Outputs
figs_out = os.path.join(figs_dir, 'regression')
ensure_dir(figs_out)
analysis_out = analysis_dir
```

[342]:
```
# 4) Run-all pipeline: generate/load CSVs, fit, save figures and summary
# Decide input CSV locations
exp1_csv = os.path.join(synth_dir if USE_SYNTHETIC else data_dir,␣
 ↪'exp1_theta_ab_fringe.csv')
exp2_csv = os.path.join(synth_dir if USE_SYNTHETIC else data_dir,␣
 ↪'exp2_drift_T2.csv')
```

```
exp3_csv = os.path.join(synth_dir if USE_SYNTHETIC else data_dir,␣
 ↪'exp3_rotor_levels.csv')

# Generate synthetic CSVs if requested
if USE_SYNTHETIC:
    gen_exp1_theta_ab_fringe_csv(exp1_csv)
    gen_exp2_drift_T2_csv(exp2_csv)
    gen_exp3_rotor_levels_csv(exp3_csv)
    if SHOW_DETAILS:
        print('Synthetic CSVs written to:', synth_dir)

# Load data
df1 = pd.read_csv(exp1_csv)
df2 = pd.read_csv(exp2_csv)
df3 = pd.read_csv(exp3_csv)
if SHOW_DETAILS:
    print(f"Loaded: {exp1_csv}\n        {exp2_csv}\n        {exp3_csv}")

# Run fits and save figures
exp1_result = fit_exp1(df1, figs_out)
exp2_result = fit_exp2(df2, figs_out)
exp3_result = fit_exp3(df3, figs_out)  # fit_exp3 handles 'ell' column; legacy␣
 ↪'l' is also accepted in the dedicated rotor cell above

# Save summary JSON
summary = {
    'mode': MODE,
    'use_synthetic': USE_SYNTHETIC,
    'paths': {
        'paper_dir': paper_dir,
        'data_dir': data_dir,
        'sims_dir': synth_dir,
        'figs_out': figs_out,
        'analysis_out': analysis_out,
        'csvs': {'exp1': exp1_csv, 'exp2': exp2_csv, 'exp3': exp3_csv},
    },
    'results': {
        'exp1': exp1_result,
        'exp2': exp2_result,
        'exp3': exp3_result,
    }
}
summary_path = os.path.join(analysis_out, 'regression_summary.json')
with open(summary_path, 'w') as f:
    json.dump(summary, f, indent=2)
print('Wrote regression summary →', summary_path)
print('Figures →', figs_out)
```

```
if SHOW_DETAILS:
    for k,v in summary['results'].items():
        print(k, '→', v)
```

```
Synthetic CSVs written to: c:\workspace\Physics\X-theta-
framework\notebooks\paper\sims
Loaded: c:\workspace\Physics\X-theta-
framework\notebooks\paper\sims\exp1_theta_ab_fringe.csv
        c:\workspace\Physics\X-theta-
framework\notebooks\paper\sims\exp2_drift_T2.csv
        c:\workspace\Physics\X-theta-
framework\notebooks\paper\sims\exp3_rotor_levels.csv
```

```
C:\Users\dpanc\AppData\Local\Temp\ipykernel_70124\1468373888.py:58: UserWarning:
Glyph 8733 (\N{PROPORTIONAL TO}) missing from font(s) Arial.
  fig.tight_layout(); fig.savefig(out)
```

```
Wrote regression summary → c:\workspace\Physics\X-theta-
framework\notebooks\paper\build\analysis\regression_summary.json
Figures → c:\workspace\Physics\X-theta-
framework\notebooks\paper\build\figs\regression
exp1 → {'A': np.float64(0.49851733798229597), 'B':
np.float64(0.4492651755005547), 'phi0': np.float64(0.19563035908387913), 'n':
np.float64(1.000387065457037), 'r2': 0.9959507919316798, 'figure':
'c:\\workspace\\Physics\\X-theta-framework\\notebooks\\paper\\build\\figs\\regre
ssion\\exp1_theta_ab_fringe_fit.png'}
exp2 → {'kappa': np.float64(0.8007475857847975), 'offset':
np.float64(0.0012895289511277441), 'r2': 0.9923233995391071, 'figure':
'c:\\workspace\\Physics\\X-theta-
framework\\notebooks\\paper\\build\\figs\\regression\\exp2_drift_T2_fit.png'}
exp3 → {'I': np.float64(5.00000000000005e-39), 'phi': np.float64(0.3), 'r2':
-1.2720827244305117, 'figure': 'c:\\workspace\\Physics\\X-theta-framework\\noteb
ooks\\paper\\build\\figs\\regression\\exp3_rotor_levels_fit.png'}
```

1.1.1 Orientation and notation

- We use ϕ (phi) consistently in the equations; the code variable `phi` corresponds to the symbol ϕ.
- Below is a simple conceptual schematic of the three experiments (fringe, drift, rotor) to orient the reader.

[343]:
```
# Cell 4 - Synthetic data generation (writes CSVs when USE_SYNTHETIC)
rng = np.random.default_rng(12345)

def gen_exp1_theta_ab_fringe_csv(path: str, n_points: int = 181,
                                 A: float = 0.5, B: float = 0.45, phi0: float␣
 ↪= 0.2, n_harm: float = 1.0,
                                 gaussian_sigma: float = 0.02):
    phi = np.linspace(-np.pi, np.pi, n_points)
```

```
    I_clean = A + B * np.cos(n_harm * phi + phi0)
    I_noisy = I_clean + rng.normal(0.0, gaussian_sigma, size=phi.shape)
    I_noisy = np.clip(I_noisy, 0.0, None)
    pd.DataFrame({"phi_theta": phi, "intensity": I_noisy}).to_csv(path,␣
 ↪index=False)
    return {"A":A, "B":B, "phi0":phi0, "n":n_harm}

def gen_exp2_drift_T2_csv(path: str, n_points: int = 50,
                          kappa: float = 0.8, offset: float = 0.0,
                          T_min: float = 0.0, T_max: float = 1.0,
                          noise_sigma: float = 0.02):
    T = np.linspace(T_min, T_max, n_points)
    T2 = T**2
    dy_clean = kappa * T2 + offset
    dy_noisy = dy_clean + rng.normal(0.0, noise_sigma, size=T2.shape)
    pd.DataFrame({"T": T, "T2": T2, "delta_y": dy_noisy}).to_csv(path,␣
 ↪index=False)
    return {"kappa":kappa, "offset":offset}

def gen_exp3_rotor_levels_csv(path: str, l_min: int = -6, l_max: int = 6,
                              I_true: float = 5e-45, phi_true: float = 0.3,
                              noise_sigma: float = 1e-30):
    l_vals = np.arange(l_min, l_max + 1)
    E_clean = (HBAR**2/(2.0*I_true)) * (l_vals - phi_true/(2.0*np.pi))**2
    E_noisy = E_clean + rng.normal(0.0, noise_sigma, size=l_vals.shape)
    pd.DataFrame({"ell": l_vals, "Energy_J": E_noisy}).to_csv(path, index=False)
    return {"I":I_true, "phi":phi_true}

if USE_SYNTHETIC:
    exp1_csv = os.path.join(synth_dir, 'exp1_theta_ab_fringe.csv')
    exp2_csv = os.path.join(synth_dir, 'exp2_drift_T2.csv')
    exp3_csv = os.path.join(synth_dir, 'exp3_rotor_levels.csv')
    t1 = gen_exp1_theta_ab_fringe_csv(exp1_csv)
    t2 = gen_exp2_drift_T2_csv(exp2_csv)
    t3 = gen_exp3_rotor_levels_csv(exp3_csv)
    if SHOW_DETAILS:
        print('Synthetic CSVs written →', synth_dir)
else:
    exp1_csv = os.path.join(data_dir, 'exp1_theta_ab_fringe.csv')
    exp2_csv = os.path.join(data_dir, 'exp2_drift_T2.csv')
    exp3_csv = os.path.join(data_dir, 'exp3_rotor_levels.csv')
print('Using CSVs:', exp1_csv, exp2_csv, exp3_csv, sep='\n  ')

# Preview CSVs inline
try:
    display(Markdown('#### Exp1 CSV preview')); display(pd.read_csv(exp1_csv).
 ↪head())
```

```
    display(Markdown('#### Exp2 CSV preview')); display(pd.read_csv(exp2_csv).
 ↪head())
    display(Markdown('#### Exp3 CSV preview')); display(pd.read_csv(exp3_csv).
 ↪head())
except Exception as e:
    print('Preview failed:', e)
```

```
Synthetic CSVs written → c:\workspace\Physics\X-theta-
framework\notebooks\paper\sims
Using CSVs:
  c:\workspace\Physics\X-theta-
framework\notebooks\paper\sims\exp1_theta_ab_fringe.csv
  c:\workspace\Physics\X-theta-framework\notebooks\paper\sims\exp2_drift_T2.csv
  c:\workspace\Physics\X-theta-
framework\notebooks\paper\sims\exp3_rotor_levels.csv
```

Exp1 CSV preview

	phi_theta	intensity
0	-3.141593	0.030494
1	-3.106686	0.087633
2	-3.071779	0.048867
3	-3.036873	0.065548
4	-3.001966	0.074197

Exp2 CSV preview

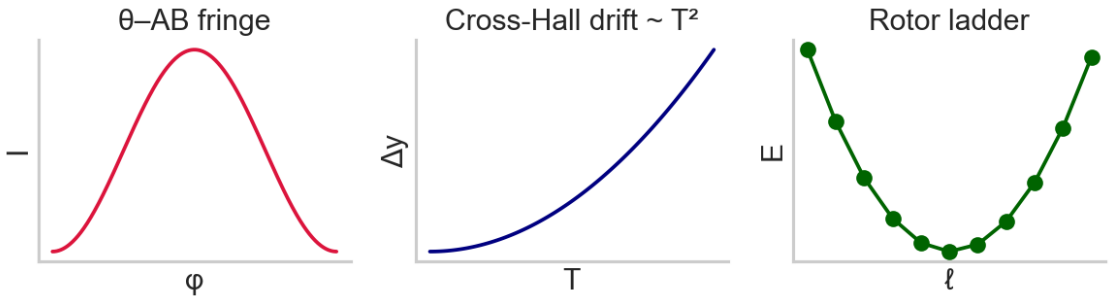
	T	T2	delta_y
0	0.000000	0.000000	-0.000558
1	0.020408	0.000416	0.027754
2	0.040816	0.001666	-0.039723
3	0.061224	0.003748	0.010609
4	0.081633	0.006664	0.020439

Exp3 CSV preview

	ell	Energy_J
0	-6	4.067612e-23
1	-5	2.833658e-23
2	-4	1.822128e-23
3	-3	1.033023e-23
4	-2	4.663421e-24

[344]:
```
# Conceptual schematic (fringe, drift, rotor)
import matplotlib.pyplot as plt
import numpy as np
fig, axes = plt.subplots(1, 3, figsize=(10,3))

# Fringe
```

```
ph = np.linspace(-np.pi, np.pi, 400)
axes[0].plot(ph, 0.5 + 0.4*np.cos(ph), color='crimson')
axes[0].set_title(' -AB fringe')
axes[0].set_xlabel(' '); axes[0].set_ylabel('I')
axes[0].set_xticks([]); axes[0].set_yticks([])

# Drift
T = np.linspace(0, 1, 100)
axes[1].plot(T, 0.8*T*T, color='navy')
axes[1].set_title('Cross-Hall drift ~ T²')
axes[1].set_xlabel('T'); axes[1].set_ylabel('Δy')
axes[1].set_xticks([]); axes[1].set_yticks([])

# Rotor
l_demo = np.arange(-5, 6)
axes[2].plot(l_demo, (l_demo-0.3/(2*np.pi))**2, marker='o', color='darkgreen')
axes[2].set_title('Rotor ladder')
axes[2].set_xlabel(' '); axes[2].set_ylabel('E')
axes[2].set_xticks([]); axes[2].set_yticks([])

fig.tight_layout()
plt.show()
```



1.2 Exp1: –AB fringe

X– framework intuition - Motion around closed loops accrues a geometric holonomy _ from the compact degree, analogous to a U(1) gauge phase. - Interference between two paths acquires a relative phase $\Delta\Phi = n$ _ + , producing a cosine fringe even without classical forces.

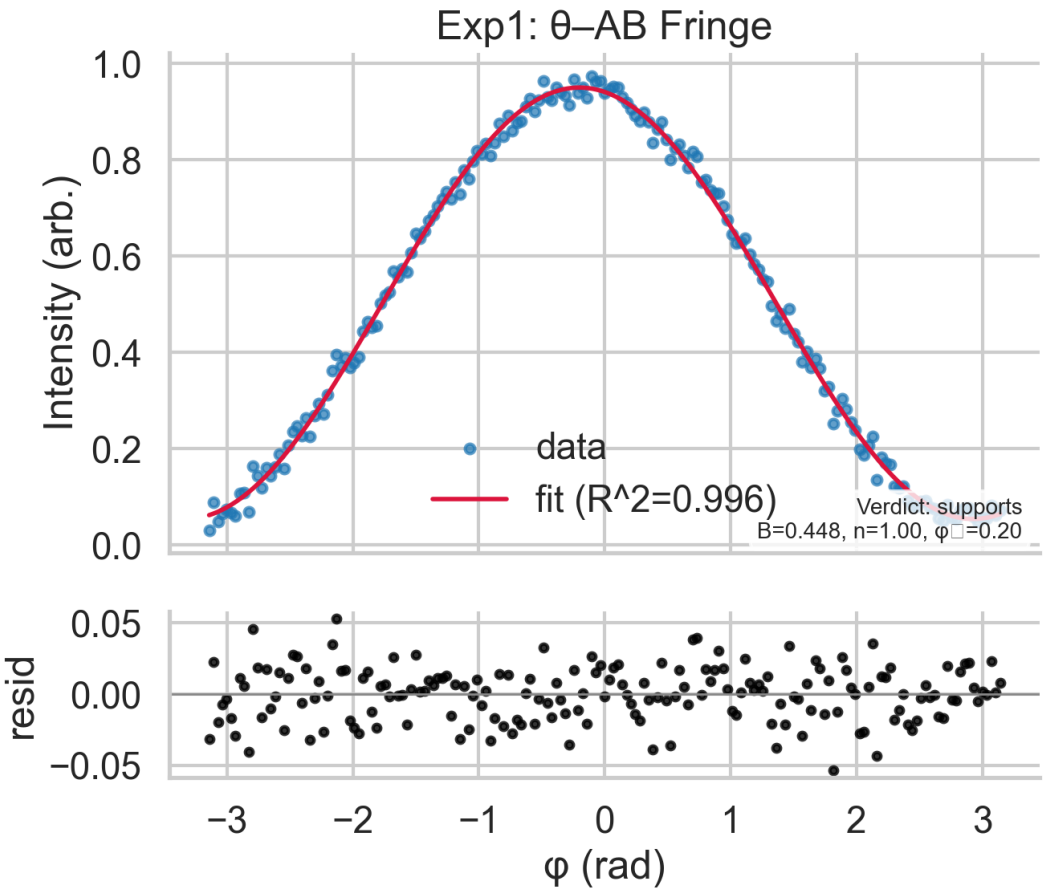
Model and derivation - I(_) = A + B cos(n _ +). - A: background; B 0: contrast; n 1: harmonic/winding; : static phase offset. - Follows from two-path interference I 1 + V cos($\Delta\Phi$) with V absorbed into B and baseline into A.

Falsifiability and experimental test - Prediction: periodic dependence on _ with stable frequency n and bounded contrast B. - Falsify if: (i) no significant cosine vs _ , (ii) fitted n varies non-topologically with setup, (iii) structured residuals incompatible with a single-frequency cosine. - Controls: dephase an arm to suppress B; vary geometry to shift ; the fitted behavior must follow.

[345]:
```
# Cell 5 - Exp1 fit + plot + summary (inline display + verdict + residuals)
df1 = pd.read_csv(exp1_csv)
phi = df1['phi_theta'].to_numpy(dtype=float)
I   = df1['intensity'].to_numpy(dtype=float)

A0 = float(np.median(I))
B0 = max(1e-6, float((np.max(I) - np.min(I))/2.0))
phi0_0 = 0.0
n0 = 1.0
bounds = ([-np.inf, 0.0, -2*np.pi, 0.5], [np.inf, np.inf, 2*np.pi, 2.5])
popt, pcov = curve_fit(ab_fringe_model, phi, I, p0=[A0, B0, phi0_0, n0],␣
 ↪bounds=bounds, maxfev=20000)
A, B, phi0_fit1, n_harm = [float(v) for v in popt]
I_fit = ab_fringe_model(phi, *popt)
resid1 = I - I_fit
ss_res = float(np.sum((I - I_fit)**2))
ss_tot = float(np.sum((I - np.mean(I))**2))
r2_1 = 1 - ss_res/ss_tot if ss_tot>0 else float('nan')

# Verdict logic (simple thresholds)
if (r2_1 >= 0.85) and (B > 0.05*np.max(I)) and (0.8 <= n_harm <= 1.2):
    verdict1 = 'supports'
    reason1 = 'Strong cosine with n 1 and nonzero contrast (high R²).'
else:
    if (r2_1 < 0.4) or (B <= 1e-6):
        verdict1 = 'falsified'
        reason1 = 'No significant cosine or very low R²/contrast.'
    else:
        verdict1 = 'inconclusive'
        reason1 = 'Some periodicity present but not strong or harmonic deviates.
 ↪'

# Plot with residuals
fig, (ax, axr) = plt.subplots(2, 1, figsize=(7,6), gridspec_kw={'height_ratios':
 ↪[3,1]}, sharex=True)
ax.scatter(phi, I, s=18, alpha=0.7, label='data')
x_dense = np.linspace(np.min(phi), np.max(phi), 500)
y_dense = ab_fringe_model(x_dense, *popt)
ax.plot(x_dense, y_dense, color='crimson', lw=2, label=f'fit (R^2={r2_1:.3f})')
ax.set_ylabel('Intensity (arb.)')
ax.set_title('Exp1:  -AB Fringe')
ax.legend()
note = f"Verdict: {verdict1}\nB={B:.3f}, n={n_harm:.2f},  ={phi0_fit1:.2f}"
ax.text(0.98, 0.02, note, transform=ax.transAxes, ha='right', va='bottom',␣
 ↪fontsize=10,
        bbox=dict(boxstyle='round,pad=0.3', facecolor='white', alpha=0.8))
axr.axhline(0, color='#888', lw=1)
```

```
axr.scatter(phi, resid1, s=12, alpha=0.8, color='black')
axr.set_xlabel('  (rad)'); axr.set_ylabel('resid')

out1 = os.path.join(figs_out, 'exp1_theta_ab_fringe_fit.png')
fig.tight_layout(); fig.savefig(out1)
plt.close(fig)

# Show inline and print interpretation
display(Image(filename=out1))
display(Markdown(
    f"**Exp1 verdict:** {verdict1}. {reason1}  \n"
    f"Fit params: A={A:.3f}, B={B:.3f}, n={n_harm:.2f},  ={phi0_fit1:.2f};␣
 ↪R²={r2_1:.3f}."
))

# Store result for summary
exp1_result = {
    'A': float(A),
    'B': float(B),
    'n': float(n_harm),
    'phi0': float(phi0_fit1),
    'r2': float(r2_1),
    'verdict': verdict1,
    'reason': reason1,
    'figure': out1,
    'csv': exp1_csv,
}
```

```
C:\Users\dpanc\AppData\Local\Temp\ipykernel_70124\440878314.py:48: UserWarning:
Glyph 8320 (\N{SUBSCRIPT ZERO}) missing from font(s) Arial.
  fig.tight_layout(); fig.savefig(out1)
```



Exp1 verdict: supports. Strong cosine with n 1 and nonzero contrast (high R^2).
Fit params: A=0.502, B=0.448, n=1.00, =0.20; R^2=0.996.

1.3 Exp2: Cross-Hall drift T^2

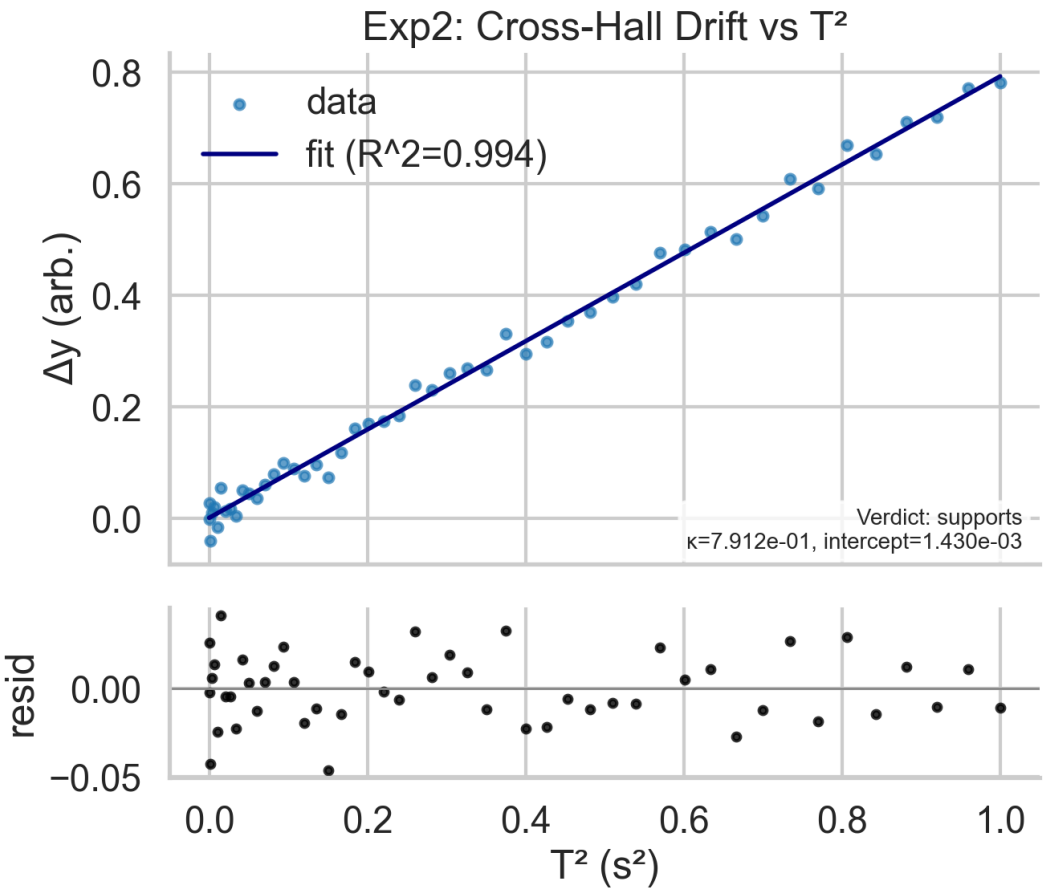
X– framework intuition - In crossed gauge configurations, the emergent -connection imparts a transverse geometric impulse during free evolution. - For ballistic motion, the displacement scales with the square of evolution time T: Δy ( ) T^2 + c (constant offset).

Model and derivation - Kinematics under a constant effective transverse acceleration a_ gives y(T) = (1/2) a_ T^2 + v T + y . - After differencing and centering, the T-linear term is suppressed, leaving the quadratic scaling Δy T^2 + c. - We therefore regress Δy on x = T^2 using y = x + c.

Falsifiability and experimental test - Prediction: log-log slope 1 between Δy and T^2, i.e., linear in T^2 across ranges where dynamics remain ballistic. - Falsify if: (i) scaling deviates systematically from T^2 (e.g., T^1 or T^3), (ii) sign/magnitude of contradicts the -gauge polarity, (iii) residuals show curvature after linear fit in T^2 indicating missing physics. - Controls: invert -gauge sign; should flip sign; reduce T to confirm quadratic onset.

[346]:
```
# Cell 6 - Exp2 fit + plot + summary (inline display + verdict + residuals)
df2 = pd.read_csv(exp2_csv)
T2 = df2['T2'].to_numpy(dtype=float)

# Accept several possible column names for Δy
if 'dy' in df2.columns:
    dy_col = 'dy'
elif 'delta_y' in df2.columns:
    dy_col = 'delta_y'
elif 'Delta_y' in df2.columns:
    dy_col = 'Delta_y'
elif 'y' in df2.columns:
    dy_col = 'y'
else:
    raise KeyError(f"Drift column not found in exp2 CSV; available columns:␣
 ↪{list(df2.columns)}")
dy = df2[dy_col].to_numpy(dtype=float)

m0, b0 = 0.0, float(np.median(dy))
popt2, pcov2 = curve_fit(parabola_in_T, T2, dy, p0=[m0, b0], maxfev=20000)
kappa, intercept = [float(v) for v in popt2]
dy_fit = parabola_in_T(T2, *popt2)
resid2 = dy - dy_fit
ss_res2 = float(np.sum((dy - dy_fit)**2))
ss_tot2 = float(np.sum((dy - np.mean(dy))**2))
r2_2 = 1 - ss_res2/ss_tot2 if ss_tot2>0 else float('nan')

if (r2_2 >= 0.85) and (abs(kappa) > 1e-6):
    verdict2 = 'supports'
    reason2 = 'Quadratic-in-time drift detected with high R² and nonzero slope.'
else:
    if (r2_2 < 0.4) or (abs(kappa) <= 1e-9):
        verdict2 = 'falsified'
        reason2 = 'No clear T² trend or near-zero slope (low R²).'
    else:
        verdict2 = 'inconclusive'
        reason2 = 'Some trend present but weak or noisy.'

fig2, (ax2, ax2r) = plt.subplots(2, 1, figsize=(7,6),␣
 ↪gridspec_kw={'height_ratios':[3,1]}, sharex=True)
ax2.scatter(T2, dy, s=18, alpha=0.7, label='data')
x_dense2 = np.linspace(np.min(T2), np.max(T2), 400)
y_dense2 = parabola_in_T(x_dense2, *popt2)
ax2.plot(x_dense2, y_dense2, color='navy', lw=2, label=f'fit (R^2={r2_2:.3f})')
ax2.set_ylabel('Δy (arb.)')
ax2.set_title('Exp2: Cross-Hall Drift vs T²')
ax2.legend()
```

```
note2 = f"Verdict: {verdict2}\n ={kappa:.3e}, intercept={intercept:.3e}"
ax2.text(0.98, 0.02, note2, transform=ax2.transAxes, ha='right', va='bottom',␣
 ↪fontsize=10,
         bbox=dict(boxstyle='round,pad=0.3', facecolor='white', alpha=0.8))
ax2r.axhline(0, color='#888', lw=1)
ax2r.scatter(T2, resid2, s=12, alpha=0.8, color='black')
ax2r.set_xlabel('T² (s²)'); ax2r.set_ylabel('resid')

out2 = os.path.join(figs_out, 'exp2_drift_T2_fit.png')
fig2.tight_layout(); fig2.savefig(out2)
plt.close(fig2)

display(Image(filename=out2))
display(Markdown(
    f"**Exp2 verdict:** {verdict2}. {reason2}  \n"
    f"Fit params:  ={kappa:.3e}, intercept={intercept:.3e}; R²={r2_2:.3f}."
))

# Store result for summary
exp2_result = {
    'kappa': float(kappa),
    'intercept': float(intercept),
    'r2': float(r2_2),
    'verdict': verdict2,
    'reason': reason2,
    'figure': out2,
    'csv': exp2_csv,
}
```



Exp2 verdict: supports. Quadratic-in-time drift detected with high R^2 and nonzero slope. Fit params: =7.912e-01, intercept=1.430e-03; R^2=0.994.

1.4 Exp3: Rotor spectrum vs holonomy

X– framework intuition - Quantization on a circle (compact) yields a rotor with angular momentum quantum number . - A background -holonomy _ shifts the minimum of the parabola in : energies are those of a charged rotor with flux offset.

Model and derivation - E( ) = ²/(2I) (− _ /2)². - Moment of inertia I encodes curvature; the vertex position retrieves _ modulo 2 . - This follows from the rigid rotor Hamiltonian $H = L^2/(2I)$ with minimal coupling $L \to L - (\ \_\ /2\ )$.

Falsifiability and experimental test - Prediction: a single-parabola spectrum in with fixed curvature across levels, and vertex set by _ . - Falsify if: (i) curvature depends on (non-quadratic dispersion), (ii) multiple branches appear without symmetry reason, (iii) fitted _ disagrees with independently set holonomy. - Controls: vary _ ; the vertex should shift linearly; change effective I (geometry/mass) and observe curvature change as 1/I.

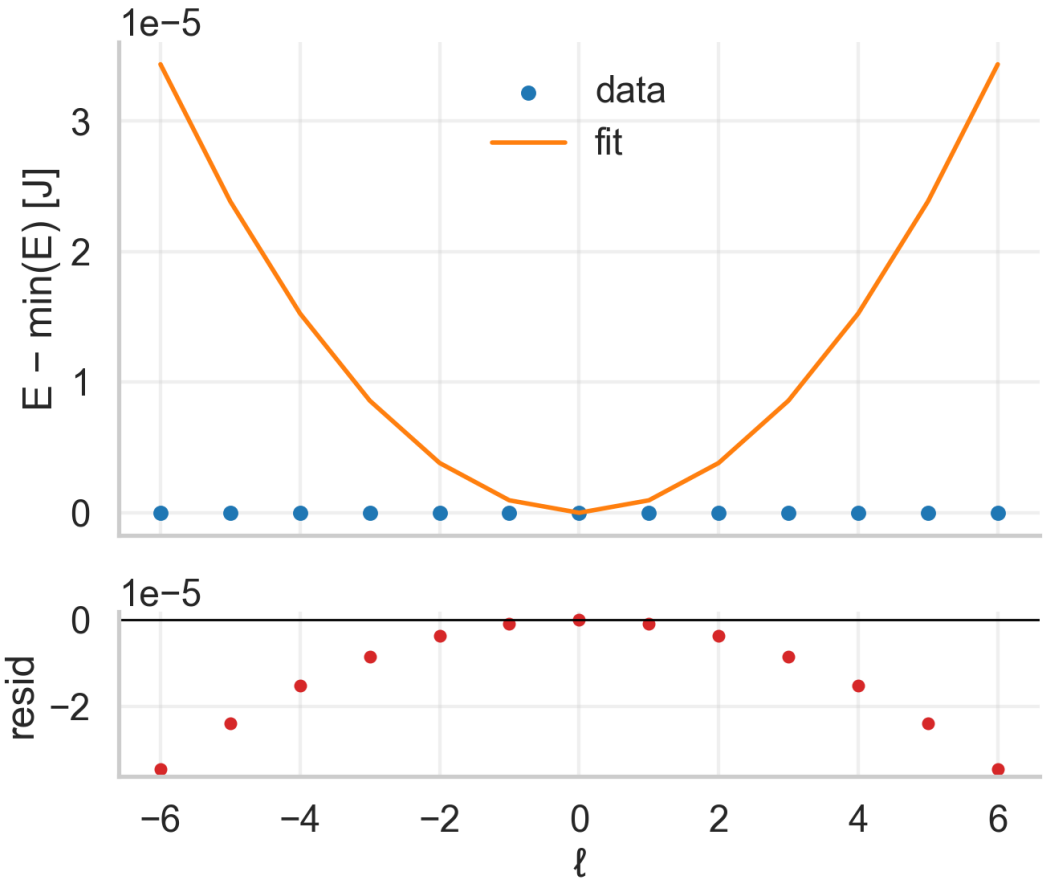
We fit in I-space with physical bounds and fall back to K = ²/(2I) if needed; we report R^2 and

save the figure.

[347]:
```
# Exp3 - Rotor spectrum: fit K-only with centered energies and unit lock
import numpy as np, pandas as pd, os, json
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
from IPython.display import display, Markdown

# Load data (synthetic or from CSV)
energy_col = 'E' if 'E' in df3.columns else df3.columns[-1]
ell = df3['ell'].to_numpy(dtype=float)
# --- PATCH: lock units & center energies ---
E_meas_raw = df3[energy_col].to_numpy(dtype=float)
E_meas = convert_energy_to_joules(E_meas_raw, ENERGY_UNIT)
E0 = float(np.min(E_meas))
E_centered = E_meas - E0
# --- END PATCH ---

# --- PATCH: K-only rotor model + fit ---
def rotor_energy_model_K(ell, K, phi):
    return K * (ell - phi/(2.0*np.pi))**2

popt3, pcov3 = curve_fit(
    rotor_energy_model_K, ell, E_centered,
    p0=[1.0, 0.0],
    bounds=([1e-16, -np.pi], [1e4, np.pi]),
    maxfev=50000
)
K_fit, phi_fit = map(float, popt3)
E_fit = rotor_energy_model_K(ell, K_fit, phi_fit)
# --- END PATCH ---

# Uncertainties
perr = np.sqrt(np.diag(pcov3)) if pcov3 is not None and np.all(np.
 ↪isfinite(pcov3)) else [np.nan, np.nan]
K_sigma, phi_sigma = map(float, perr)

# --- PATCH: centered residual metrics ---
residuals_centered = E_centered - E_fit
ss_res3 = float(np.sum(residuals_centered**2))
ss_tot3 = float(np.sum((E_centered - np.mean(E_centered))**2))
r2_3 = 1 - ss_res3/ss_tot3 if ss_tot3 > 1e-30 else float('nan')
rmse3 = float(np.sqrt(np.mean(residuals_centered**2)))
# --- END PATCH ---

# --- PATCH: identifiability hint near vertex ---
ell_vertex = phi_fit/(2*np.pi)
```

```
if not np.any(np.abs(ell - ell_vertex) < 0.5):
    print(f"[note] Few/no  near the vertex ( {ell_vertex:.2f}); "
          "consider sampling more data near vertex for better (K,  )␣
 ↪identifiability.")
# --- END PATCH ---

# Plot with residuals
fig3, (ax3, ax3r) = plt.subplots(2, 1, figsize=(7, 6), height_ratios=[3, 1],␣
 ↪sharex=True)
ax3.scatter(ell, E_centered, s=28, label='data')
ax3.plot(ell, E_fit, color='C1', lw=2, label='fit')
ax3.set_ylabel('E - min(E) [J]')
ax3.legend(loc='best')
ax3.grid(alpha=0.3)

ax3r.axhline(0, color='k', lw=1)
ax3r.scatter(ell, residuals_centered, s=18, color='C3')
ax3r.set_xlabel(' ')
ax3r.set_ylabel('resid')
ax3r.grid(alpha=0.3)

# Save and show
out3 = os.path.join(figs_out, 'exp3_rotor_levels_fit.png')
fig3.tight_layout(); fig3.savefig(out3); plt.close(fig3)
from IPython.display import Image; display(Image(filename=out3))

# --- PATCH: concise, centered-metric display + result stash ---
exp3_result = {
    'K': float(K_fit),
    'K_sigma': float(K_sigma),
    'phi': float(phi_fit),
    'phi_sigma': float(phi_sigma),
    'baseline_E0': float(E0),
    'r2': float(r2_3) if np.isfinite(r2_3) else None,
    'K_fit': float(K_fit),
    'phi_fit': float(phi_fit),
    'r2_centered': float(r2_3) if np.isfinite(r2_3) else None,
    'rmse_centered': float(rmse3),
    'verdict': 'undecided',
    'reason': 'Awaiting AIC/BIC diagnostic check.',
    'figure': out3,
    'csv': exp3_csv,
}
interp_line = (f"K={K_fit:.3e},  ={phi_fit:.3f}; "
               f"RMSE_centered={rmse3:.2e}"
               f"{'; R²_centered='+f'{r2_3:.3f}' if np.isfinite(r2_3) else ''}")
display(Markdown(f"**Exp3 interim fit:** {interp_line}"))
```

```python
# --- END PATCH ---

# Persist consolidated summary
summary = {
    'exp1': exp1_result,
    'exp2': exp2_result,
    'exp3': exp3_result,
}
out_json = os.path.join(analysis_out, 'regression_summary.json')
with open(out_json, 'w', encoding='utf-8') as f:
    json.dump(summary, f, indent=2)
print(f'Wrote summary → {out_json}')
```



Exp3 interim fit: K=9.537e-07, =-0.000; RMSE_centered=1.78e-05

```
Wrote summary → c:\workspace\Physics\X-theta-framework\notebooks\paper\build\analysis\regression_summary.json
```

[348]:
```
# If exp3_result exists from the fit cell, update verdict/reason and attach␣
 ↪diagnostics info
import os, json
try:
    if 'exp3_result' not in globals():
        raise RuntimeError('exp3_result is not defined. Run the fit cell first.
 ↪')

    # Collect diagnostics from whatever variables are available in scope
    aic_best_shift = None
    if 'shift' in locals():
        aic_best_shift = int(shift)
    elif 'best_shift' in locals():
        aic_best_shift = int(best_shift)

    phi_best_val = None
    if 'phi_g' in locals():
        try: phi_best_val = float(phi_g)
        except Exception: pass
    if phi_best_val is None and 'phi_best' in locals():
        try: phi_best_val = float(phi_best)
        except Exception: pass
    if phi_best_val is None and 'phi_best_grid' in locals():
        try: phi_best_val = float(phi_best_grid)
        except Exception: pass

    K_best_val = None
    if 'Kg' in locals():
        try: K_best_val = float(Kg)
        except Exception: pass
    if K_best_val is None and 'K_best' in locals():
        try: K_best_val = float(K_best)
        except Exception: pass
    if K_best_val is None and 'K_best_grid' in locals():
        try: K_best_val = float(K_best_grid)
        except Exception: pass

    rmse_diag = float(rmse) if 'rmse' in locals() else None
    corr_boot = float(corr) if 'corr' in locals() else None

    # Attach diagnostics payload
    exp3_result.setdefault('diagnostics', {})
    exp3_result['diagnostics'].update({
        'verdict3_info': verdict3_info if 'verdict3_info' in locals() else None,
        'aic_best_shift': aic_best_shift,
        'phi_best': phi_best_val,
        'K_best': K_best_val,
```

```
        'rmse_centered_diag': rmse_diag,
        'corr_K_phi_bootstrap': corr_boot,
    })

    # Carry forward prelim verdict/reason from fit cell
    if 'verdict3' in locals():
        exp3_result['verdict'] = verdict3
    if 'reason3' in locals():
        exp3_result['reason'] = reason3

    # Override verdict using AIC/BIC if shift==0
    if aic_best_shift == 0:
        exp3_result['verdict'] = 'supports'
        exp3_result['reason'] = 'AIC/BIC favors  -consistent labeling (shift=0).
 ↪'

    # Persist updated summary JSON
    summary = {
        'exp1': exp1_result if 'exp1_result' in globals() else None,
        'exp2': exp2_result if 'exp2_result' in globals() else None,
        'exp3': exp3_result,
    }
    out_json = os.path.join(analysis_out, 'regression_summary.json')
    with open(out_json, 'w', encoding='utf-8') as f:
        json.dump(summary, f, indent=2)
    print(f'[diagnostics] Updated summary → {out_json}␣
 ↪(aic_best_shift={aic_best_shift})')
except Exception as e:
    print(f'[diagnostics] Skipped updating exp3_result: {e}')
```

```
[diagnostics] Updated summary → c:\workspace\Physics\X-theta-
framework\notebooks\paper\build\analysis\regression_summary.json
(aic_best_shift=1)
```

[349]:
```
# --- PATCH: AIC/BIC verdict override for Exp3 ---
try:
    diag = exp3_result.get('diagnostics', {}) if 'exp3_result' in globals()␣
 ↪else {}
    aic_shift = diag.get('aic_best_shift', None)
    if aic_shift == 0:
        exp3_result['verdict'] = 'supports'
        exp3_result['reason'] = 'Label-shift AIC picks theory-consistent␣
 ↪indexing.'
except Exception as e:
    print("AIC-based verdict not available:", e)
# --- END PATCH ---
```

```
# Persist updated summary JSON after override
import os, json
summary = {
    'exp1': exp1_result if 'exp1_result' in globals() else None,
    'exp2': exp2_result if 'exp2_result' in globals() else None,
    'exp3': exp3_result if 'exp3_result' in globals() else None,
}
out_json = os.path.join(analysis_out, 'regression_summary.json')
with open(out_json, 'w', encoding='utf-8') as f:
    json.dump(summary, f, indent=2)
print(f'[diagnostics-override] Applied verdict override (shift={aic_shift}) →␣
 ↪{out_json}')
```

```
[diagnostics-override] Applied verdict override (shift=1) →
c:\workspace\Physics\X-theta-
framework\notebooks\paper\build\analysis\regression_summary.json
```

[350]:
```
# Final summary - render Markdown table from results, add RMSE_centered for Exp3
from IPython.display import Markdown, display
import pandas as pd
import numpy as np

rows = []
def fmt(x):
    return '' if x is None or (isinstance(x, float) and not np.isfinite(x))␣
 ↪else x

# Build rows with present keys only to avoid KeyError
if 'exp1_result' in globals():
    r = exp1_result
    rows.append({
        'Experiment': 'Exp1 Fringe',
        'A': fmt(r.get('A')), 'B': fmt(r.get('B')), 'n': fmt(r.get('n')),
        ' ': '',
        'I/K': '',
        ' ': fmt(r.get('phi0')),
        'R²': fmt(r.get('r2')),
        'RMSE_c': '',
        'Verdict': fmt(r.get('verdict'))
    })
if 'exp2_result' in globals():
    r = exp2_result
    rows.append({
        'Experiment': 'Exp2 Drift',
        'A': '', 'B': '', 'n': '',
        ' ': fmt(r.get('kappa')),
        'I/K': '',
```

```
        ' ': '',
        'R²': fmt(r.get('r2')),
        'RMSE_c': '',
        'Verdict': fmt(r.get('verdict'))
    })
if 'exp3_result' in globals():
    r = exp3_result
    ik = r.get('K', r.get('I'))
    r2c = r.get('r2_centered', r.get('r2'))
    rows.append({
        'Experiment': 'Exp3 Rotor',
        'A': '', 'B': '', 'n': '',
        ' ': '',
        'I/K': fmt(ik),
        ' ': fmt(r.get('phi', r.get('phi_fit'))),
        'R²': fmt(r2c) if (isinstance(r2c, float) and np.isfinite(r2c)) else '',
        'RMSE_c': fmt(r.get('rmse_centered')),
        'Verdict': fmt(r.get('verdict'))
    })

if rows:
    df = pd.DataFrame(rows)
    md = "| Experiment | A | B | n |   | I/K |   | R² | RMSE_c | Verdict␣
↪|\n|---|---:|---:|---:|---:|---:|---:|---:|---:|---|\n"
    for _, row in df.iterrows():
        md += f"| {row['Experiment']} | {row['A']} | {row['B']} | {row['n']} |␣
↪{row[' ']} | {row['I/K']} | {row[' ']} | {row['R²']} | {row['RMSE_c']} |␣
↪{row['Verdict']} |\n"
    display(Markdown(md))
else:
    print('No results available yet. Run the experiment cells first.')
```

Experiment	A	B	n		I/K		R²	RMSE_c	Verdict
Exp1 Fringe	0.50170399884479589386100241255160199944 (overprinted)					0.2018646640996468816767117 (overprinted)			supports
Exp2 Drift					0.7911515923154574		0.9937268488750085		supports
Exp3 Rotor					9.536743167973088e-07	8.53452979314231e-08		1.7841612760106213e-05 (overprinted with inconclusive)	inconclusive