Modeling Recovery after Circadian Rhythm Disruption using Differential Equations

Lara Simpson, Mark Saad, Ricky Yang, Chris Demian

December 19, 2025

1 Abstract

We simulate the recovery time after three weeks of simulated circadian rhythm disruption. We compare four differential models presented in three papers: Forger's model [1], Jewett's model [2], and Hannay's single-population and two-population models [3]. Solutions were computed using the fourth-order Runge-Kutta numerical integration method. CBTmin is used as the phase marker. We found that the models produced similar results after three weekends of delayed sleep schedules of 3 hours, the Forger, Jewett, Hannay Single-Population, and Hanny Two-Population producing re-entrainment times of 59.0, 52.1, 61.1, and 59.0 hours respectively, with the Jewett model disagreeing with the other models.

2 Introduction

Modelling human circadian rhythm is a topic of great interest in biology as it is the biological mechanism that governs the periodical processes of the body, such as sleep-wake cycles. Circadian rhythm is almost exclusively modelled by ODEs, making this topic highly suitable for this course. The topic of circadian rhythms is of personal relevance to the authors, as we all share the universal university experience of sleep loss before a midterm, which leads us to wonder if we can mathematically model what happens to our circadian rhythm after repeated sleep loss. This leads to the research question: how long does it take for the circadian rhythm to recover after being disrupted? We will be comparing the models in the papers of Forger, Jewett, and Hannay.

3 Mathmatical Models

All models of circadian rhythms utilizes the Van der Pol's oscillator. The basic model that the more recent models are based on was first proposed by Richard Kronauer in 1990 [4] and consists of the following pair of coupled ODES:

$$\frac{dx}{dt} = \left(\frac{\pi}{12}\right)\left[x_c + \mu\left(x - \frac{4x^3}{3}\right) + B\right] \tag{1}$$

$$\frac{dx_c}{dt} = \left(\frac{\pi}{12}\right)\left[-\left(\frac{24}{\tau_x}\right)^2 x + Bx_c\right] \tag{2}$$

Here, x is the primary circadian oscillator indicating the position along the circadian cycle (i.e. wakefuness, body temperature, etc) . x_c is the auxiliary variable maintaining stability in the system. τ is the intrinsic period of 24.2h, and light enters the system as the perceived brightness B. (We will not be implimenting the Kronauer model) An adaptation to that model is presented by Forger et. al in 1999 [1] which attempts to model it more accurately and

simply. The cubic term is moved from the $\frac{dx}{dt}$ equation to the auxiliary equation as the amplitude of the limit cycle is more dependent on the strength of the light drive, and a small correction factor of $\left(\frac{1}{0.99669}\right)^2$ is added. A kBx term is also added to account for the effect of light on the circadian rhythm in the spirit of Aschoff's rule. Note that light intensity is converted to B through a process called Process L, consisting of the equations (5) - (9).

$$\frac{dx}{dt} = \frac{\pi}{12}(x_c + B) \tag{3}$$

$$\frac{dx_c}{dt} = \frac{\pi}{12}\left\{\mu\left(x_c - \frac{4x_c^3}{3}\right) - x\left[\left(\frac{24}{0.99669\tau_x}\right)^2 + kB\right]\right\} \tag{4}$$

$$\alpha(I) = \alpha_0\left(\frac{I^p}{I_0^p}\right) \tag{5}$$

$$\frac{dn}{dt} = 60[\alpha(I)(1-n) - \beta n] \tag{6}$$

$$\hat{B} = G(1-n)\alpha(I) \tag{7}$$

$$B = \hat{B}(1 - 0.4x)(1 - 0.4x_c) \tag{8}$$

$$\mathrm{CBT}_{\min} = x_{\min} + \phi_{ref} \tag{9}$$

Parameters: $\alpha_0 = 0.16$, $\beta = 0.013$, $G = 19.875$, $p = 0.6$, $k = 0.55$, $I_0 = 9500$ lux. where $\phi_{ref} = 0.97$ hours is a phase reference correction.

Another adaptation to Kronauer's model is presented by Jewett et al in 1999. Again, light intensity is converted to B through Process L

$$\dot{x} = \left(\frac{\pi}{12}\right)\left[x_c + \mu\left(\frac{1}{3}x + \frac{4}{3}x^3 - \frac{256}{105}x^7\right) + B\right] \tag{10}$$

$$\dot{x}_c = \left(\frac{\pi}{12}\right)\left\{qBx_c - \left[\left(\frac{24}{0.99729\tau_x}\right)^2 + kB\right]x\right\} \tag{11}$$

More recently, in 2019, Hannay et al presented two much more complex models that introduce a macroscopic approach derived from SCN neuron network models. Hannay's "single-population" model uses an explicit amplitude state R (radius of oscillation) as one of the state variables and a phase angle (or an equivalent two-state formulation where state[0] behaves like an amplitude variable). It is a self-sustained oscillator with parameters tuned to human data, and it can exhibit more realistic behaviour under strong perturbations (e.g., slower

amplitude recovery). Meanwhile, the model with two populations (often representing two subdivisions of the SCN) This model allows for internal desynchrony between two coupled oscillators, which can capture complex dynamics like transients in amplitude and phase that single-oscillator models might miss. It's an advanced model expected to handle large perturbations (like all-nighters) more realistically by, for example, showing significant amplitude suppression and slow recovery when cues are removed.

Single Population:

$$\dot{R} = -(D+\gamma)R + \frac{K}{2}\cos(\beta)R(1-R^4) + L_R(R,\psi) \tag{12}$$

$$\dot{\psi} = \omega_0 + \frac{K}{2}\sin(\beta)(1+R^4) + L_\psi(R,\psi) \tag{13}$$

$$L_R(R,\psi) = \frac{A_1}{2}B(t)(1-R^4)\cos(\psi+\beta_{L1}) + \frac{A_2}{2}B(t)R(1-R^8)\cos(2\psi+\beta_{L2}) \tag{14}$$

$$L_\psi(R,\psi) = \sigma B(t) - \frac{A_1}{2}B(t)\left(\frac{1}{R}+R^3\right)\sin(\psi+\beta_{L1}) - \frac{A_2}{2}B(t)(1+R^8)\sin(2\psi+\beta_{L2}) \tag{15}$$

Two Population:

$$\dot{R}_v = -\gamma R_v + \frac{K_{vv}}{2}R_v(1-R_v^4) + \frac{K_{dv}}{2}R_d(1-R_v^4)\cos(\psi_d-\psi_v) + L_R(R_v,\psi_v) \tag{16}$$

$$\dot{R}_d = -\gamma R_d + \frac{K_{dd}}{2}R_d(1-R_d^4) + \frac{K_{vd}}{2}R_v(1-R_d^4)\cos(\psi_d-\psi_v) \tag{17}$$

$$\dot{\psi}_v = \omega_v + \frac{K_{dv}}{2}R_d\left(\frac{1}{R_v}+R_v^3\right)\sin(\psi_d-\psi_v) + L_\psi(R_v,\psi_v) \tag{18}$$

$$\dot{\psi}_d = \omega_d - \frac{K_{vd}}{2}R_v\left(\frac{1}{R_d}+R_d^3\right)\sin(\psi_d-\psi_v) \tag{19}$$

$$L_R = \frac{A_1}{2}B(t)(1-R_v^4)\cos(\psi_v+\beta_{L1}) + \frac{A_2}{2}B(t)R_v(1-R_v^8)\cos(2\psi_v+\beta_{L2}) \tag{20}$$

$$L_\psi = \sigma B(t) - \frac{A_1}{2}B(t)\left(\frac{1}{R_v}+R_v^3\right)\sin(\psi_v+\beta_{L1}) - \frac{A_2}{2}B(t)(1+R_v^8)\sin(2\psi_v+\beta_{L2}) \tag{21}$$

Note that $K_{from,to}$ represents coupling strength from one region to another.

4 Methodology

We set up a program modelling circadian rhythm, implementing the four models mentioned above. Solutions are computed using the 4th-order Runge-Kutta method. The input is the light level as a function of time, I(t). Core body temperature minimum (CBTmin), which occurs in early morning hours, is a standard physiological marker used as the state variable, x. We set the light input to 0 when the person goes to bed, and 250 lux when the person is up.

To simulate disruptions to a normal schedule, we assume a person who wakes up at 7:00 and goes to sleep at 23:00 under normal circumstances. We first run the program under a normal light schedule for the circadian rhythm to reach an equilibrium, then we simulate disruptions to their circadian rhythm: The light schedule is normal the first 30 days for x to equilibrate. Then, for the next 3 weeks, the person stays up until 2:00 and wakes up at 10:00 (3 hour delay) for Saturday and Sunday only. The light schedule returns to normal after day 49 and runs for another 3 weeks for the circadian rhythm to re-equilibrate. Then, the simulation is repeated, except we force complete darkness after the disruption to calculate darkness recovery. (This is done to test the reliability of the models, as in darkness, the models should not re-entrain but rather free-run.)

The baseline CBTmin is computed by averaging the clock hour of CBTmin across (up to) the first 4 weeks. To compute re-entrainment time, the program checks when the model's daily CBTmin is within 15 minutes of the baseline phase (relative phase error) for three consecutive days. The streak of 3 days corresponds to common practice in jet lag studies. If reached, the function returns the time (in hours) from the start of recovery to the first day of that streak. This is reported for both darkness and LD conditions.

5 Results

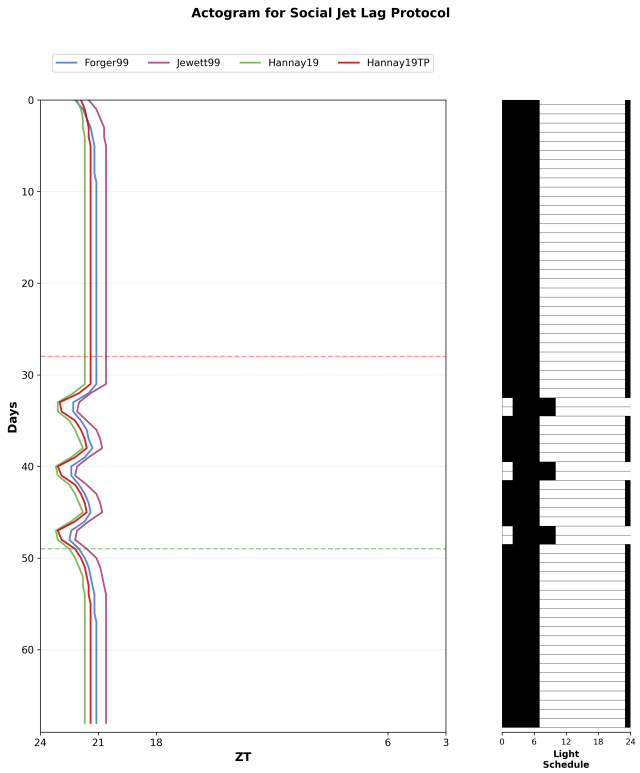

Figure 1: Actogram for Social Jet Lag Simulation across Four Models

The actogram on the left of Figure 1 displays the variation of CTBmin over time for the entire period of the simulation. As per a typical actogram graph, the days of the experiment are plotted across the y-axis, while the 24 hours of each day are plotted across the x-axis. The actogram on the right of Figure 1 displays the corresponding light schedule. As shown, a regular sleep schedule is 11 am-7 am while the delayed sleep schedule is 2 am-10 am, which is realistic. The red line signals the start of the disruption period, and the green line signals the end. As one can see, while the models differ, they all signal that the circadian rhythm recovers quickly and returns to near baseline level a few days after each weekend in the disruption period.

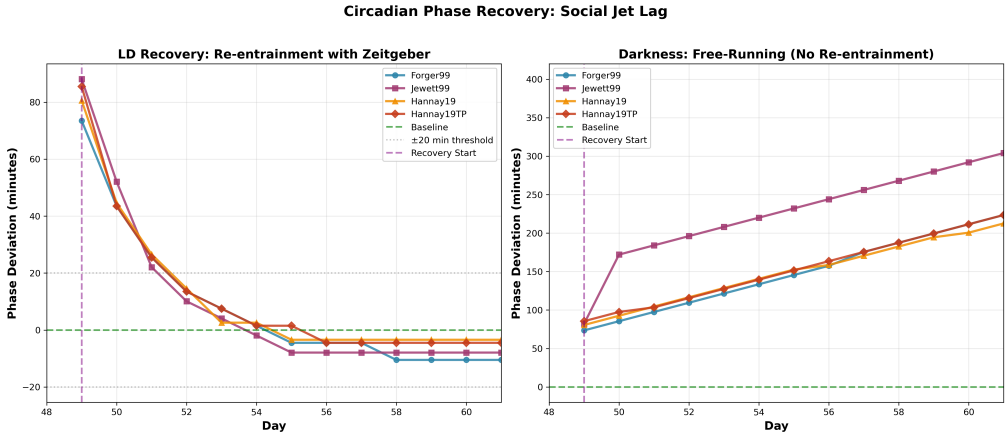

Figure 2: Circadian Phase Recovery for Social Jet Lag Across Four Models under Light-Dark (LD) and Constant Darkness (DD) Conditions

Figure 2 shows the recovery of CBTmin over time after the end of the disruption period. Phase deviation from baseline levels is plotted on the y-axis, while time in days is plotted on the x-axis. CBTmin decreases exponentially over time until it reaches around the threshold for recovery, then decreases more slowly. The graph on the right of Figure 2 shows what would happen if complete darkness is enforced: CBTmin keeps on drifting away and never returns to baseline, as expected, demonstrating the importance of light cues for re-entrainment.

Our quantitative results are summarized in the table below.

Model	**Re-entrainment Time under LD (hours)**
Forger	59.0
Jewett	52.1
Hannay (Single Population)	61.1
Hannay (Two Population)	59.0

6 Discussion

Overall, the models seem to be relatively consistent with each other, with similar graphs and a difference of less than 10 hours in re-entrainment time. However, a small outlier is Jewett's model, which shows the fastest recovery by far, being ahead of the rest of the models by almost 10, suggesting limitations to Jewett's model in predicting circaidan rhythm dynamics. (This model is the least credible of the three, as it was published the earliest, so the other two had time to build upon it. The other three models seem to agree despite being relatively different mathematically. Hence, we can predict that theoretically, a person can recover in less than 3 days following three consecutive weekends of sleeping late. Moreover,

upon close examination of the graphs, one could see that the circadian rhythm returned to baseline three days after each weekend, and there is no evidence of cumulative circadian rhythm disruption as the circadian rhythm recovers completely before the next disruptive weekend. Hence, we can conclude that established theory suggests that a person can recover in less than 3 days after a weekend of sleeping late. An important limitation is that the circadian rhythm models we examined are purely based on theory rather than empirical evidence, so a feasible suggestion for future work would be to examine how the results of the model compare to clinical trials.

7 Conclusion

The recovery time after three weeks of simulated circadian rhythm disruption is successfully simulated. We compared four differential models presented in three papers, with solutions were computed using the fourth-order Runge-Kutta numerical integration method. CBTmin is used as the phase marker. We found that the models produced similar results after three weekends of delayed sleep schedules of 3 hours, the Forger, Jewett, Hannay Single-Population, and Hanny Two-Population producing re-entrainment times of 59.0, 52.1, 61.1, and 59.0 hours respectively. The prediction of 52.1 hours from Jewett's model can be taken as an outlier and is likely less accurate than the other results.

8 References

[1] Forger, D. B., Jewett, M. E., & Kronauer, R. E. (1999). A simpler model of the human circadian pacemaker. Journal of Biological Rhythms, 14(6), 533-539.

[2] Jewett, M. E., Forger, D. B., & Kronauer, R. E. (1999). Revised limit cycle oscillator model of human circadian pacemaker. Journal of Biological Rhythms, 14(6), 493-499.

[3] Hannay, K. M., Booth, V., Forger, D. B., & Booth, V. (2019). Macroscopic models for networks of coupled biological oscillators. Scientific Reports, 9(1), 1-12.

[4] Kronauer RE (1990) A quantitative model for the effects of light on the amplitude and phase of the deep circadian pacemaker, based on human data. In Sleep '90, Proceedings of the Tenth European Congress on Sleep Research, J Horne, ed, pp 306-309, Pontenagel Press, Dusseldorf.

9 Appendix

We have experimented with many models and code implimentation, many of which did not make it into the final product. Below are some examples of our earlier code.

actogram.py

```
"""
actogram visualization
shows phase markers with light/dark schedule bars
"""

import numpy as np
import matplotlib.pyplot as plt
from models import Forger99, Jewett99, Hannay19, Hannay19TP
from metrics import get_phase_markers

# plot settings
plt.rcParams['font.size'] = 11
plt.rcParams['savefig.dpi'] = 300

# protocol parameters
DT = 0.10
BASELINE_WEEKS = 4
DISRUPTION_WEEKENDS = 3
RECOVERY_WEEKS = 3
PHASE_MARKER = "dlmo"  # can change to "cbt" if needed

# models with colors
MODELS = [
    ("Forger99", Forger99, "#4472C4"),      # blue
    ("Jewett99", Jewett99, "#A23B72"),      # purple
    ("Hannay19", Hannay19, "#70AD47"),      # green
    ("Hannay19TP", Hannay19TP, "#C00000"),  # red
]


def social_jetlag_protocol(baseline_weeks, disruption_weekends, recovery_weeks, dt):
    """create the social jet lag light schedule"""
    baseline_days = baseline_weeks * 7
    disruption_days = disruption_weekends * 7
    recovery_days = recovery_weeks * 7
    total_days = baseline_days + disruption_days + recovery_days

    t = np.arange(0.0, 24.0 * total_days, dt)
    lux = np.zeros_like(t)

    disruption_start = baseline_days
    disruption_end = baseline_days + disruption_days

    for i, ti in enumerate(t):
```

```
            day = int(ti // 24.0)
            hour = ti % 24.0
            day_of_week = day % 7
            is_weekend = (day_of_week == 5 or day_of_week == 6)  # sat/sun

            if day < disruption_start:
                # baseline: normal schedule
                lux[i] = 1000.0 if (7.0 <= hour < 23.0) else 1.0
            elif day < disruption_end:
                # disruption: late weekends
                if is_weekend:
                    lux[i] = 1000.0 if (10.0 <= hour < 24.0) or (0.0 <= hour < 2.0) else 1.0
                else:
                    lux[i] = 1000.0 if (7.0 <= hour < 23.0) else 1.0
            else:
                # recovery: back to normal
                lux[i] = 1000.0 if (7.0 <= hour < 23.0) else 1.0

        return t, lux, baseline_days, disruption_end


    def allnighter_protocol(baseline_days, recovery_days, dt):
        """create one all-nighter light schedule"""
        allnighter_day = baseline_days
        total_days = baseline_days + 1 + recovery_days

        t = np.arange(0.0, 24.0 * total_days, dt)
        lux = np.zeros_like(t)

        for i, ti in enumerate(t):
            day = int(ti // 24.0)
            hour = ti % 24.0

            if day < allnighter_day:
                # baseline
                lux[i] = 1000.0 if (7.0 <= hour < 23.0) else 1.0
            elif day == allnighter_day:
                # all-nighter = 24h light
                lux[i] = 1000.0
            else:
                # recovery
                lux[i] = 1000.0 if (7.0 <= hour < 23.0) else 1.0

        return t, lux, baseline_days, allnighter_day + 1
```

```
def create_actogram(protocol_type="social_jetlag"):
    """make the actogram figure"""

    print("\n" + "="*70)
    print(f"GENERATING ACTOGRAM - {protocol_type.upper()}")
    print("="*70)

    # set up protocol
    if protocol_type == "social_jetlag":
        t, lux, baseline_days, recovery_start = social_jetlag_protocol(
            BASELINE_WEEKS, DISRUPTION_WEEKENDS, RECOVERY_WEEKS, DT
        )
        title = 'Actogram for Social Jet Lag Protocol'
        filename = 'actogram_social_jetlag.png'
    else:  # allnighter
        t, lux, baseline_days, recovery_start = allnighter_protocol(
            baseline_days=30, recovery_days=14, dt=DT
        )
        title = 'Actogram for One All-Nighter Protocol'
        filename = 'actogram_allnighter.png'

    total_days = int(t[-1] / 24.0)

    # set up figure with two panels
    fig = plt.figure(figsize=(10, 12))

    # left panel: phase markers
    ax_main = plt.subplot2grid((1, 20), (0, 0), colspan=14)

    # right panel: light schedule
    ax_light = plt.subplot2grid((1, 20), (0, 15), colspan=5, sharey=ax_main)

    # run all models
    print("\nAnalyzing models...")
    all_markers = {}
    for model_name, model_class, color in MODELS:
        print(f"  - {model_name}")

        model = model_class()
        traj = model.integrate(t, input=lux)
        markers = get_phase_markers(model, traj, marker=PHASE_MARKER)

        all_markers[model_name] = (markers, color)

    # plot phase markers
    for model_name, (markers, color) in all_markers.items():
```

```
    marker_days = [int(m // 24) for m in markers]
    marker_times = [m % 24 for m in markers]

    ax_main.plot(marker_times, marker_days, '-', color=color,
                 label=model_name, linewidth=2.0, alpha=0.85)

# add phase boundary lines
ax_main.axhline(baseline_days, color='red', linestyle='--',
                linewidth=1.5, alpha=0.4)
ax_main.axhline(recovery_start, color='green', linestyle='--',
                linewidth=1.5, alpha=0.4)

# format main plot
ax_main.set_xlim([18, 6])  # show 18:00 to 06:00 (wrapping around midnight)
ax_main.set_ylim([total_days, 0])  # day 0 at top
ax_main.set_xlabel('ZT', fontsize=13, fontweight='bold')
ax_main.set_ylabel('Days', fontsize=13, fontweight='bold')

# x-axis ticks
ax_main.set_xticks([18, 21, 24, 3, 6])
ax_main.set_xticklabels(['18', '21', '24', '3', '6'])

# legend at top
ax_main.legend(loc='upper center', bbox_to_anchor=(0.5, 1.08),
               ncol=4, fontsize=11, frameon=True)

ax_main.grid(True, alpha=0.2, axis='y')
ax_main.set_axisbelow(True)

# draw light/dark schedule bars
for day in range(total_days):
    # figure out light schedule for this day
    if protocol_type == "social_jetlag":
        day_of_week = day % 7
        is_weekend = (day_of_week == 5 or day_of_week == 6)

        if day < baseline_days:
            # normal schedule
            light_periods = [(7, 23)]
        elif day < recovery_start:
            # disruption phase
            if is_weekend:
                # late weekend
                light_periods = [(10, 24), (0, 2)]
            else:
                # normal weekday
```

```
                light_periods = [(7, 23)]
            else:
                # recovery
                light_periods = [(7, 23)]
        else:  # allnighter
            if day < baseline_days:
                light_periods = [(7, 23)]
            elif day == baseline_days:
                # all-nighter day
                light_periods = [(0, 24)]
            else:
                # recovery
                light_periods = [(7, 23)]

        # draw black background (darkness)
        ax_light.barh(day, 24, left=0, height=1.0,
                      color='black', edgecolor='black', linewidth=0.3)

        # draw white bars for light
        for start, end in light_periods:
            ax_light.barh(day, end - start, left=start, height=1.0,
                          color='white', edgecolor='black', linewidth=0.3)

# format light schedule panel
ax_light.set_xlim([0, 24])
ax_light.set_ylim([total_days, 0])
ax_light.set_xticks([0, 6, 12, 18, 24])
ax_light.set_xticklabels(['0', '6', '12', '18', '24'], fontsize=9)
ax_light.yaxis.set_visible(False)
ax_light.spines['left'].set_visible(False)
ax_light.spines['top'].set_visible(False)
ax_light.spines['right'].set_visible(False)
ax_light.set_xlabel('Light\nSchedule', fontsize=10, fontweight='bold')

# save figure
fig.suptitle(title, fontsize=14, fontweight='bold', y=0.99)

plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.savefig(filename, bbox_inches='tight', dpi=300)
print(f"\nSaved: {filename}")

# print baseline stats
print("\n" + "="*70)
print(f"BASELINE {PHASE_MARKER.upper()} TIMES (average)")
print("="*70)
```

```
    for model_name, (markers, _) in all_markers.items():
        baseline_markers = [m % 24 for m in markers if m < baseline_days * 24]
        if baseline_markers:
            avg_time = np.mean(baseline_markers)
            hours = int(avg_time)
            minutes = int((avg_time - hours) * 60)
            print(f"{model_name:12} - {hours:02d}:{minutes:02d}")

    print("="*70)
    print()

    plt.close()


if __name__ == "__main__":
    print("\n" + "="*70)
    print("ACTOGRAM GENERATION - CIRCADIAN RHYTHM PROJECT")
    print("="*70)
    print(f"\nPhase Marker: {PHASE_MARKER.upper()}")
    print(f"Protocol: {BASELINE_WEEKS}w baseline → {DISRUPTION_WEEKENDS}w disruption → {
    print("="*70)

    # run actogram generation
    create_actogram(protocol_type="social_jetlag")

    print("\n" + "="*70)
    print("ACTOGRAM COMPLETE!")
    print("="*70)
    print("\nOutput:")
    print("  actogram_social_jetlag.png - Social jet lag (late weekends)")
    print("="*70)
```

amplitudeplot.py

```
import importlib
import numpy as np
import math
import matplotlib.pyplot as plt
from metrics import get_phase_markers, compute_baseline_marker, reentrainment_hours, amp
from protocols import allnighter_protocol, branch_ld_after, branch_dark_after, branch_sh
from models import Jewett99, Forger99, Hannay19, Hannay19TP, Breslow13,Skeldon23
'''
ld, LD -> a regular 'light-dark' cycle (16 hour day, 8 hour nights)
sn, SN-> a 'short night' cycle that implements a poor sleep schedule (5 hour nights, 19
d, dark -> simulates a constant darkness condition (reflects how the a external zeitgebe
```

```
,,,
# Which phase marker to use for the analysis ("cbt" or "dlmo")
PHASE_MARKER = "dlmo" #"cbt"    "dlmo"
MODEL_CLASSES = [
    ("Jewett99",   Jewett99),
    ("Forger99",   Forger99),
    ("Hannay19",   Hannay19),
    ("Hannay19TP", Hannay19TP),
    ("Breslow13", Breslow13),
    ('Skeldon23', Skeldon23)
]

# ==== protocol and light schedule ====
DT = 0.1
BASELINE_DAYS = 25
RECOVERY_DAYS = 15
HOURS = 24* (BASELINE_DAYS + 1 + RECOVERY_DAYS)
# Re-entrainment definition
TOL_MIN = 15     # tolerance in minutes = 0.25 hours
STREAK = 3       # consecutive days within tolerance
DAY_START = 6.0
DAY_END = 22.0
PCT = 0.9999
# Baseline light levels (lux)
DAY_LUX = 3000.0
NIGHT_LUX = 0.0
DARK_LUX = 1.0
# All-nighter
ALLNIGHTER_LUX = 1000.0
ALL_NIGHTER_START = 22.0
ALL_NIGHTER_END = 6.0   # next morning
# short-night light delay (in hours) - we are assuming a lux level equal to the allnight
PAST_BEDTIME = 3
t, lux_all, baseline_days, recovery_start_day = allnighter_protocol(
    baseline_days=BASELINE_DAYS,
    recovery_days=RECOVERY_DAYS,
    dt=DT,
    day_start=DAY_START,
    day_end=DAY_END,
    day_lux=DAY_LUX,
    night_lux=NIGHT_LUX,
    allnighter_lux=ALLNIGHTER_LUX,
)

def daily_mean_marker(marker_times):
    """
```

```
    Given absolute marker times (hours), return (days, daily_mean_clock_hour).
    Uses circular mean within each day.
    """
    marker_times = np.asarray(marker_times)
    if marker_times.size == 0:
        return np.array([]), np.array([])

    day_dict = {}
    for t in marker_times:
        d = int(t // 24.0)
        day_dict.setdefault(d, []).append(t)

    days = sorted(day_dict.keys())
    means = []
    for d in days:
        hs = [ti % 24.0 for ti in day_dict[d]]
        means.append(circular_mean_hours(hs))

    return np.array(days), np.array(means)

def reentrainment_hours(cbt_times, baseline_marker_h, start_day=31, tol_min=15, streak=3
    """
    Return the time to re-entrainment in HOURS (float), measured from
    the start of the recovery phase (start_day*24 h).
    """
    tol_h = tol_min / 60.0
    # consider only markers during the recovery phase
    days = sorted(set(int(t // 24) for t in cbt_times if t >= start_day * 24))

    for d in days:
        # markers on this day
        todays = [t for t in cbt_times if d * 24 <= t < (d + 1) * 24]
        if not todays:
            continue

        # mean clock hour for that day
        h = np.mean([ti % 24.0 for ti in todays])
        err = (((h - baseline_marker_h + 12) % 24) - 12)   # wrapped error in hours
        if abs(err) <= tol_h:
            # check streak on following days
            good = True
            for k in range(1, streak):
                d2 = d + k
                toms = [t for t in cbt_times if d2 * 24 <= t < (d2 + 1) * 24]
                if not toms:
                    good = False
```

```
                    break
                h2 = np.mean([ti % 24.0 for ti in toms])
                err2 = (((h2 - baseline_marker_h + 12) % 24) - 12)
                if abs(err2) > tol_h:
                    good = False
                    break

            if good:
                # absolute re-entrainment time in hours
                t_abs = d * 24.0 + h
                # convert to hours since start of recovery
                return t_abs - start_day * 24.0

    return None

'''
def plot_phase_amp(t, model, traj_dark, traj_ld, title):
    # phase markers
    marker_dark = get_phase_markers(model, traj_dark, marker=PHASE_MARKER)
    marker_ld   = get_phase_markers(model, traj_ld, marker=PHASE_MARKER)

    # phase error lines
    # compute baseline
    baseline_h = compute_baseline_marker(marker_ld)  # both had baseline; use ld to get
    def err_series(cbt_times):
        days = sorted(set([int(tt//24.00) for tt in cbt_times]))
        errs = []
        xs = []
        for d in days:
            hs = [tt%24.00 for tt in cbt_times if d*24.00 <= tt < (d+1)*24.00]
            if not hs: continue
            xs.append(d)
            # circular wrap error
            e = (((np.mean(hs) - baseline_h + 12) % 24.00) - 12)
            errs.append(e)
        return np.array(xs), np.array(errs)
    xd, ed = err_series(marker_dark)
    xl, el = err_series(marker_ld)

    # amplitude
    Rd = amp_series(model, traj_dark)
    Rl = amp_series(model, traj_ld)

    fig, axs = plt.subplots(2, 1, figsize=(8,6), sharex=True)
    axs[0].plot(xd, ed, label='Darkness')
    axs[0].plot(xl, el, label='LD')
```

```
    axs[0].axhline(0, lw=0.8, alpha=0.6)
    axs[0].set_ylabel('Phase error (h)')
    axs[0].set_title(title)
    axs[0].legend()

    # downsample amplitude daily mean
    days = np.arange(0, int(t[-1]//24.00)+1)
    def daily_mean(R):
        out=[]
        for d in days:
            mask = (t>=d*24.00)&(t<(d+1)*24.00)
            if np.any(mask): out.append(np.mean(R[mask]))
            else: out.append(np.nan)
        return np.array(out)
    axs[1].plot(days, daily_mean(Rd), label='Darkness')
    axs[1].plot(days, daily_mean(Rl), label='LD')
    axs[1].set_xlabel('Day')
    axs[1].set_ylabel('Daily mean amplitude')
    axs[1].legend()
    plt.tight_layout()
    plt.show()
'''
def plot_phase_amp(
    t,
    model,
    traj_dark,
    traj_ld,
    traj_sn,
    baseline_h,
    recovery_start_day,
    title,
    phase_marker_name="CBTmin",
):
    """
    Two-panel plot:
      top: daily phase error vs baseline
      bottom: daily mean amplitude vs day (dark vs LD), with 90% baseline line.
    """

    # --- Phase error series ---
    markers_dark = get_phase_markers(model, traj_dark, marker=PHASE_MARKER)
    markers_ld   = get_phase_markers(model, traj_ld,   marker=PHASE_MARKER)
    markers_sn   = get_phase_markers(model, traj_sn,   marker=PHASE_MARKER)

    days_d, mean_d = daily_mean_marker(markers_dark)
    days_l, mean_l = daily_mean_marker(markers_ld)
```

```
days_s, mean_s = daily_mean_marker(markers_sn)

err_d = np.array(
    [circular_distance(h, baseline_h) if h is not None else np.nan for h in mean_d]
)
err_l = np.array(
    [circular_distance(h, baseline_h) if h is not None else np.nan for h in mean_l]
)
err_s = np.array(
    [circular_distance(h, baseline_h) if h is not None else np.nan for h in mean_s])
# --- Amplitude series (daily means) ---
Rd = amp_series(model, traj_dark)
Rl = amp_series(model, traj_ld)
Rsn = amp_series(model, traj_sn)


def daily_mean_amp(R, t):
    if len(R) == 0:
        return np.array([]), np.array([])
    total_days = int(t[-1] // 24) + 1
    days = np.arange(total_days)
    out = []
    for d in days:
        mask = (t >= d * 24.0) & (t < (d + 1) * 24.0)
        if np.any(mask):
            out.append(np.mean(R[mask]))
        else:
            out.append(np.nan)
    return days, np.array(out)


days_amp_d, daily_Rd = daily_mean_amp(Rd, t)
days_amp_l, daily_Rl = daily_mean_amp(Rl, t)
days_amp_s, daily_Rsn = daily_mean_amp(Rsn, t)

# Baseline amplitude (from LD branch only)
start_baseline = (recovery_start_day - 10) * 24.0
end_baseline = recovery_start_day * 24.0
mask = (t >= start_baseline) & (t < end_baseline)

if np.any(mask):
    baseline_R = np.nanmean(Rl[mask])
else:
    baseline_R = np.nan  # or handle as you prefer

target_90 = PCT * baseline_R if not np.isnan(baseline_R) else None
```

```
    '''CHANGED TO 99% FOR NOW'''
    ##plot
    fig, axs = plt.subplots(2, 1, figsize=(8, 6), sharex=True)

    # Top: phase error
    if days_d.size > 0:
        axs[0].plot(days_d, err_d, "o-", label="Darkness", markersize=3)
    if days_l.size > 0:
        axs[0].plot(days_l, err_l, "x-", label="LD", markersize=3)
    if days_s.size > 0:
        axs[0].plot(days_s, err_s, "s-", label="Short-night", markersize=3)

    axs[0].axhline(0.0, color="k", linestyle="--", linewidth=0.8)
    axs[0].axvline(recovery_start_day, color="grey", linestyle=":", linewidth=0.8)
    axs[0].set_ylabel(f"Phase error ({phase_marker_name}) [h]")
    axs[0].legend()
    axs[0].grid(True, alpha=0.3)

    # Bottom: amplitude
    if days_amp_d.size > 0:
        axs[1].plot(days_amp_d, daily_Rd, "o-", label="Darkness Recovery", markersize=3)
    if days_amp_l.size > 0:
        axs[1].plot(days_amp_l, daily_Rl, "x-", label="LD Recovery", markersize=3)
    if days_amp_s.size > 0:
        axs[1].plot(days_amp_s, daily_Rsn, "s-", label="Short-night", markersize=3)
    if target_90 is not None:
        axs[1].axhline(
            target_90,
            color="green",
            linestyle=":",
            linewidth=0.8,
            alpha=0.6,
            label="AVG baseline amplitude",
        )
    axs[1].axvline(recovery_start_day, color="grey", linestyle=":", linewidth=0.8)
    axs[1].set_xlabel("Day")
    axs[1].set_ylabel("Daily mean amplitude")
    axs[1].legend()
    axs[1].grid(True, alpha=0.3)

    fig.suptitle(f"{title} { {phase_marker_name}-based analysis")
    plt.tight_layout()
    plt.show()

def main():
    # Build common protocol (baseline + all-nighter + recovery)
```

```
t, lux_all, baseline_days, recovery_start_day = allnighter_protocol(
    baseline_days=BASELINE_DAYS,
    recovery_days=RECOVERY_DAYS,
    dt=DT,
    day_start=DAY_START,
    day_end=DAY_END,
    day_lux=DAY_LUX,
    night_lux=NIGHT_LUX,
    allnighter_lux=ALLNIGHTER_LUX,
    allnighter_start=ALL_NIGHTER_START,
    allnighter_end=ALL_NIGHTER_END)

lux_ld   = branch_ld_after(t, lux_all, recovery_start_day)
lux_dark = branch_dark_after(t, lux_all, recovery_start_day, dark_lux=DARK_LUX)
lux_sn   = branch_shortnight_after(t,lux_all,recovery_start_day,
    day_start=DAY_START,
    day_lux=DAY_LUX,
    night_lux=NIGHT_LUX,
    allnighter_lux=ALLNIGHTER_LUX,
    past_bedtime = PAST_BEDTIME)

print(f"Using phase marker: {PHASE_MARKER.upper()}")
print(f"Baseline days: {BASELINE_DAYS}, recovery days: {RECOVERY_DAYS}")
print(f"Recovery starts on day {recovery_start_day}")

for name, cls in MODEL_CLASSES:
    print("\n==============================")
    print(f"Model: {name}")

    try:
        model = cls()
    except Exception as e:
        print(f"[ERROR] Could not construct model {name}: {e}")
        continue

    # Integrate under each branch
    try:
        traj_dark = model.integrate(t, input=lux_dark)
        traj_ld   = model.integrate(t, input=lux_ld)
        traj_sn   = model.integrate(t, input=lux_sn)
    except Exception as e:
        print(f"[ERROR] Integration failed for {name}: {e}")
        continue

    # Phase markers and baseline
    markers_ld = get_phase_markers(model, traj_ld, marker=PHASE_MARKER)
```

```
baseline_h = compute_baseline_marker(markers_ld, baseline_days=BASELINE_DAYS)

if baseline_h is None:
    print(f"[ERROR] No reliable baseline {PHASE_MARKER.upper()} for {name}, skip
    continue

print(f"Baseline {PHASE_MARKER.upper()}: {baseline_h:.2f} h")

# Re-entrainment times (hours from start of recovery)
markers_dark = get_phase_markers(model, traj_dark, marker=PHASE_MARKER)
markers_sn   = get_phase_markers(model, traj_sn,   marker=PHASE_MARKER)

# For DLMO we anchor re-entrainment to the last full baseline day
reentrain_start_day = baseline_days if PHASE_MARKER.lower() == "dlmo" else recov

rdark = reentrainment_hours(
    markers_dark,
    baseline_h,
    start_day=reentrain_start_day,
    tol_min=TOL_MIN,
    streak=STREAK,
)
rld = reentrainment_hours(
    markers_ld,
    baseline_h,
    start_day=reentrain_start_day,
    tol_min=TOL_MIN,
    streak=STREAK,
)
rsn = reentrainment_hours(
    markers_sn,
    baseline_h,
    start_day=reentrain_start_day,
    tol_min=TOL_MIN,
    streak=STREAK,
)
# Amplitude recovery (hours from start of recovery)
Rd = amp_series(model, traj_dark)
Rl = amp_series(model, traj_ld)
Rsn = amp_series(model, traj_sn)

start_baseline = (recovery_start_day - 10) * 24.0
end_baseline   = recovery_start_day * 24.0
baseline_mask  = (t >= start_baseline) & (t < end_baseline)
baseline_R_ld  = np.nanmean(Rl[baseline_mask]) if np.any(baseline_mask) else np.
```

```
    dA_d  = hours_to_90pct(Rd,  t, pct=PCT, start_day=recovery_start_day, sustain_da
    dA_l  = hours_to_90pct(Rl,  t, pct=PCT, start_day=recovery_start_day, sustain_da
    dA_sn = hours_to_90pct(Rsn, t, pct=PCT, start_day=recovery_start_day, sustain_da


    if dA_d is not None and dA_l is not None and dA_l != 0:
        recovery_ratio = dA_d / dA_l
    # Recovery ratio (short night/LD)
    recovery_ratio_dark = None
    recovery_ratio_sn   = None

    if dA_d is not None and dA_l is not None and dA_l != 0:
        recovery_ratio_dark = dA_d / dA_l

    if dA_sn is not None and dA_l is not None and dA_l != 0:
        recovery_ratio_sn = dA_sn / dA_l

    # Print metrics
    if rdark is not None:
        print(f"Re-entrainment (Dark): {rdark:.2f} h ({rdark/24.0:.2f} days)")
  # else:
#       print("Re-entrainment (Dark): N/A (In constant darkness, re-entrainment to

    if rld is not None:
        print(f"Re-entrainment (LD): {rld:.2f} h ({rld/24.0:.2f} days)")
    else:
        print("Re-entrainment (LD): not reached")

    if rsn is not None:
        print(f"Re-entrainment (SN): {rsn:.2f} h ({rsn/24.0:.2f} days)")
    else:
        print("Re-entrainment (SN): not reached")

    if dA_d is not None:
        print(f"90% amplitude (Dark): {dA_d:.2f} h ({dA_d/24.0:.2f} days)")
    else:
        print("90% amplitude (Dark): not reached")

    if dA_l is not None:
        print(f"90% amplitude (LD): {dA_l:.2f} h ({dA_l/24.0:.2f} days)")
    else:
        print("90% amplitude (LD): not reached")

    if dA_sn is not None:
        print(f"90% amplitude (SN): {dA_sn:.2f} h ({dA_sn/24.0:.2f} days)")
    else:
```

```
            print("90% amplitude (SN): not reached")

        if recovery_ratio_dark is not None:
            print(f"Recovery ratio Dark/LD: {recovery_ratio_dark:.2f}")
        else:
            print("Recovery ratio Dark/LD: N/A")

        if recovery_ratio_sn is not None:
            print(f"Recovery ratio SN/LD: {recovery_ratio_sn:.2f}")
        else:
            print("Recovery ratio SN/LD: N/A")

        phase_label = "CBTmin" if PHASE_MARKER.lower() == "cbt" else "DLMO"
        plot_phase_amp(
            t,
            model,
            traj_dark,
            traj_ld,
            traj_sn,
            baseline_h,
            recovery_start_day,
            title=name,
            phase_marker_name=phase_label,
        )

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