Semi-Analytic Model for Early SMBH Growth with GW Recoil

– Bisweswar Sen (2025)

We construct a modular Python model following Sen (2025)[1][2]. Our model includes three seed channels – **light (Pop III)**, **medium (cluster)**, and **heavy (direct collapse)** – each with distinct spin distributions[3]. We implement the standard recoil kick formula from numerical relativity (Campanelli et al. 2007; Lousto & Zlochower 2012)[4] and a retention criterion based on comparing the recoil speed to the halo escape speed[5]. Mergers are processed sequentially (Monte Carlo style), updating the central SMBH mass if the remnant is retained. Kicked-but-bound BHs are tracked as *wandering* with orbital parameters and an estimated dynamical friction time (Chandrasekhar 1943)[6][7].

Seed Spin and Orientation Assignment

Each BH seed's dimensionless spin magnitude χ and orientation (θ,φ) are drawn from distributions motivated by formation channel[3]:

- **Light (Pop III) seeds** ($M \sim 10^2\,M_\odot$) have moderate spins ($\chi \sim 0.3 \pm 0.2$) isotropically oriented[8].
- **Medium (cluster) seeds** ($M \sim 10^3\,M_\odot$) have χ uniform in $[0.1,0.7]$, random orientation.
- **Heavy (direct collapse) seeds** ($M \sim 10^5$–$10^6\,M_\odot$) have low spins ($\chi \sim 0.1 \pm 0.05$) with small misalignment angles (aligned with the halo/flow)[9].

These choices are implemented in a function `assign_seed_spin`. For example:

```
import math, random

def assign_seed_spin(seed_type):
    """Assign spin magnitude χ and orientation angles (θ,ϕ) for a seed of
given type."""
    if seed_type == 'pop3':
        # Gaussian around 0.3 (σ=0.2), clipped to [0,0.98], isotropic
orientation
        chi = min(max(random.gauss(0.3,0.2),0.0), 0.98)
        u = random.uniform(-1,1)
        theta = math.acos(u)
        phi = random.uniform(0, 2*math.pi)
    elif seed_type == 'cluster':
        # Uniform [0.1,0.7], random orientation
        chi = random.uniform(0.1, 0.7)
        u = random.uniform(-1,1)
        theta = math.acos(u)
```

```
        phi = random.uniform(0, 2*math.pi)
    elif seed_type == 'heavy':
        # Gaussian around 0.1 (σ=0.05), clipped to [0,0.3], small θ
        chi = min(max(random.gauss(0.1,0.05),0.0), 0.3)
        # Small polar angle to model alignment (θ~N(0,0.1))
        theta = abs(random.gauss(0, 0.1))
        phi = random.uniform(0, 2*math.pi)
    else:
        raise ValueError("Unknown seed type")
    return chi, theta, phi
```

This function captures the spin assumptions from the reference model[3]. In a simulation, each newly formed or merging BH is assigned `(χ,θ,ϕ)` via `assign_seed_spin`.

Gravitational-Wave Recoil Velocity

We compute the GW recoil ("kick") velocity using the empirical fitting formulae[4]. The total kick is decomposed as

$$ V_{\rm recoil} = V_m + V_\perp + V_\parallel, $$

where (from Campanelli et al. 2007 and Lousto & Zlochower 2012)[4]:

- **Mass-asymmetry term:** $V_m = A\,\eta^2\,\frac{1-q}{1+q}(1+B\eta)$, with $q = m_2/m_1 \le 1$, $\eta = m_1 m_2/(m_1+m_2)^2$, $A = 1.2\times10^4$ km/s, $B = -0.93$ [10].
- **In-plane spin term:** $V_\perp = H\,\eta^2\,|(\boldsymbol{\chi}_1 + \boldsymbol{\chi}_2)\times\hat{n}|$, with $H \approx 6.9\times10^3$ km/s (Lousto & Zlochower 2012)[11].
- **Out-of-plane term:** $V_\parallel = K\,\eta^2\left[(\boldsymbol{\chi}_1 - q\,\boldsymbol{\chi}_2)\cdot\hat{L}\right]$, with $K \approx 6.0\times10^4$ km/s (Campanelli et al. 2007)[4].

Here $\boldsymbol{\chi}_i$ are 3D spin vectors of magnitude χ and orientation (θ,φ), $\hat{n}$ is the direction of separation (in-plane), and $\hat{L}$ is the unit orbital angular momentum (out-of-plane direction). In code we can take, e.g., $\hat{L} = (0,0,1)$ and $\hat{n} = (1,0,0)$ without loss of generality (the resulting V depends only on magnitudes and relative orientations). A Python implementation:

```
import numpy as np

# Constants for kick formula (km/s)
A = 1.2e4
B = -0.93
H = 6.9e3
K = 6.0e4

def compute_recoil_velocity(m1, m2, chi1, theta1, phi1, chi2, theta2, phi2):
    """
    Compute GW recoil speed for two merging BHs with masses m1>=m2
    and spin magnitudes chi1, chi2 with orientations (theta,phi).
```

```
    Based on Campanelli et al. (2007) and Lousto & Zlochower (2012)[4].
    """
    # Ensure m1 >= m2
    if m2 > m1:
        m1, m2 = m2, m1
        chi1, chi2 = chi2, chi1
        theta1, theta2 = theta2, theta1
        phi1, phi2 = phi2, phi1
    q = m2 / m1
    eta = (m1*m2) / ( (m1+m2)**2 )
    # Mass term V_m[10]
    V_m = A * eta**2 * (1 - q)/(1 + q) * (1 + B*eta)
    # Define unit vectors for spins (in orbital frame)
    s1 = np.array([chi1*math.sin(theta1)*math.cos(phi1),
                   chi1*math.sin(theta1)*math.sin(phi1),
                   chi1*math.cos(theta1)])
    s2 = np.array([chi2*math.sin(theta2)*math.cos(phi2),
                   chi2*math.sin(theta2)*math.sin(phi2),
                   chi2*math.cos(theta2)])
    # In-plane term V_perp[11]
    n_hat = np.array([1.0, 0.0, 0.0])
    V_perp = H * eta**2 * np.linalg.norm(np.cross(s1 + s2, n_hat))
    # Out-of-plane term V_parallel[4]
    L_hat = np.array([0.0, 0.0, 1.0])
    V_para = K * eta**2 * abs(np.dot(s1 - q*s2, L_hat))
    # Total kick magnitude
    V_total = math.sqrt(V_m**2 + V_perp**2 + V_para**2)
    return V_total
```

This function returns the scalar kick speed (in km/s)[4]. We will use this to determine if the post-merger BH remains bound.

Retention Probability

Given a computed kick speed $V_{\rm recoil}$, we compare to the host halo's local escape velocity $V_{\rm esc}$. Sen (2025) uses an interpolation formula[5]:

$$ P_{\rm ret}(v) = \begin{cases} 1, & v < 0.5,\\ 0.5\Bigl[1 + \cos\frac{\pi(v-0.5)}{1.5}\Bigr], & 0.5 \le v \le 2.0,\\ 0, & v > 2.0, \end{cases} $$

where $v = V_{\rm recoil}/V_{\rm esc}$ [5]. In practice we sample a uniform random $x \in [0,1]$ and retain the remnant if $x<P_{\rm ret}$; otherwise the BH is displaced or ejected. For example:

```
def retention_probability(V_recoil, V_esc):
    """
    Returns retention probability P_ret given recoil speed V_recoil (km/s)
    and escape speed V_esc (km/s).  Implements the smooth interpolation from
Sen (2025)[5].
    """
```

```
    v = V_recoil / V_esc
    if v < 0.5:
        return 1.0
    elif v > 2.0:
        return 0.0
    else:
        # Smooth cosine interpolation between 0.5 and 2.0
        return 0.5 * (1 + math.cos(math.pi * (v - 0.5) / 1.5))
```

If a merger has $P_{\rm ret}=1$, the remnant is always retained; if $P_{\rm ret}=0$, it is always lost (kick exceeds escape). Intermediate values yield probabilistic retention. This captures the criterion “full retention at low kicks and zero retention at high kicks”[5].

Wandering Black Holes

When a BH merges and is kicked **but remains bound** ($V_{\rm recoil}<V_{\rm esc}$ yet is not kept at the center), we label it a *wandering* BH[6]. These off-nuclear BHs carry mass and kinetic energy. We compute a rough orbital radius from energy conservation $r_{\max}\approx 2GM_{\rm halo}/V^2$, and estimate a dynamical friction timescale $t_{\rm df}$ using Chandrasekhar’s formula (cf. Binney & Tremaine). If $V_{\rm recoil}\ge V_{\rm esc}$, the BH is ejected and removed from further evolution. In practice, after each merger we do:

```
if random.random() < P_ret:
    # BH retained: update central SMBH mass
    M_central += (m1 + m2)
else:
    # BH not retained
    if V_recoil < V_esc:
        # Bound wanderer: record its properties
        r_orbit = 2 * G_const * M_halo / (V_recoil**2)  # in kpc if G_const
in kpc*(km/s)^2/M⊙
        t_df = chandrasekhar_timescale(m1+m2, M_halo, V_recoil)
        wanderers.append({'mass': m1+m2, 'v': V_recoil, 'r': r_orbit, 'tdf':
t_df})
    # If V_recoil >= V_esc, BH is ejected (ignore)
```

Here `chandrasekhar_timescale` would implement a standard formula (e.g. $t_{\rm df}\sim (1/G)\,(V^3 / [4\pi G^2 m \rho \ln\Lambda])$ for a halo density ρ, which we do not detail here). In our code we simply compute a placeholder or note that $t_{\rm df}$ can exceed the Hubble time, so many wanderers never sink back[7].

This way, we track a list of wandering BHs with their mass, velocity offset, radius, etc., for later analysis (e.g. plotting the **mass and radial distribution of wanderers**).

Merger Tree and Monte Carlo Evolution

We simulate SMBH assembly by iterating over major merger events. In each step, we draw a secondary BH (seed) from the three channel types, assign it a mass and spin, compute the recoil from merging with the current central BH, and apply the retention

decision. This multiplicatively suppresses central growth: if a merger is retained, the central mass increases by the full combined mass; otherwise the mass does not grow (the remnant is lost as central growth)[12].

A simple pseudo-algorithm:

1. **Initialize** central BH with one seed (choose type and mass).
2. For each merger *i*:
3. Draw a new BH seed (m_2, type, spin).
4. Compute $V_{\rm recoil}$ via `compute_recoil_velocity`.
5. Compute $P_{\rm ret}$ with `retention_probability`.
6. Flip random x ; if $x<P_{\rm ret}$, **retain**: update central mass $M_{\rm cen} \leftarrow M_{\rm cen}+m_2$.
   Else **kick**: if $V<V_{\rm esc}$ record a wandering BH; if $V\ge V_{\rm esc}$ discard it.
7. Record $M_{\rm cen}(z)$, wandering BHs, etc., at each step.

A code excerpt demonstrating this loop:

```
def run_smbh_growth(num_mergers, V_esc, M_halo):
    """
    Simulate SMBH growth over num_mergers events with recoil effects.
    V_esc: escape velocity (km/s), M_halo: host halo mass (Msun).
    """
    # Start with an initial seed
    seed_types = ['pop3','cluster','heavy']
    # e.g., probabilities of each seed forming:
    p_seed = [0.6, 0.3, 0.1]
    initial_type = random.choices(seed_types, weights=p_seed)[0]
    M_central = {'pop3':1e2, 'cluster':1e3, 'heavy':1e5}[initial_type]
    chi_c, th_c, ph_c = assign_seed_spin(initial_type)
    # Lists to record history
    mass_history = [M_central]
    wanderers = []

    for i in range(num_mergers):
        # Draw a merging BH
        t2 = random.choices(seed_types, weights=p_seed)[0]
        m2 = {'pop3':1e2, 'cluster':1e3, 'heavy':1e5}[t2]
        chi2, th2, ph2 = assign_seed_spin(t2)
        # Compute recoil
        Vkick = compute_recoil_velocity(M_central, m2, chi_c, th_c, ph_c,
chi2, th2, ph2)
        Pret = retention_probability(Vkick, V_esc)
        # Decide retention
        if random.random() < Pret:
            # Retained: grow central BH
            M_central += m2
        else:
```

```
            # Kicked-out remnant
            if Vkick < V_esc:
                # Record wandering BH (mass, velocity, orbital radius)
                r_orb = 2 * G_const * M_halo / (Vkick**2)
                wanderers.append((M_central+m2, Vkick, r_orb))
            # else: fully ejected, ignore
        mass_history.append(M_central)
        # (Optionally update central spin here based on merger dynamics)

    return mass_history, wanderers
```

In practice one would map each merger index to a redshift (e.g. using a merger rate history or simply as a proxy). The final `mass_history` and `wanderers` lists form the basis for diagnostics. This loop implements exactly the approach from Sen (2025): “each merger multiplies the central SMBH mass by $P_{\rm ret}$ … the final SMBH mass is suppressed relative to a no-kick model by the product of retention factors”[12].

Diagnostics and Visualization

With the simulated assembly histories, we can produce the requested diagnostics:

- **SMBH mass vs. redshift (or merger step)**, comparing recoil vs. no-recoil cases[12]. For example, plotting the central mass growth from `run_smbh_growth` (with recoil) against an idealized no-kick run shows the ∼20–30% suppression by $z \sim 6$ found in the reference[13][12].
- **Distributions of recoil speeds and retention probabilities**. We histogram the collected $V_{\rm recoil}$ values and $P_{\rm ret}$ values over many merger realizations. Typically most kicks are modest (few 100 km/s) giving $P_{\rm ret}\approx1$, but a tail of large kicks drives $P_{\rm ret}\to0$ [4][5].
- **Wandering BH mass and radial distributions**. We plot the masses and radii of the recorded wanderers. As expected, wanderers tend to be lower-mass (originating from light/medium seeds) and lie at radii of order 0.1–1 R_vir[14][15]. (In our toy model we simply convert $r_{\rm orb}$ to kpc; in a realistic NFW halo one would relate this to $R_{\rm vir}$.)
- **Spatial and velocity offsets**. From each wandering BH we can compute e.g. angular offset $\theta \approx V_{\rm recoil}\,t_{\rm elapsed}/D_A(z)$ and line-of-sight velocity $\sim V_{\rm recoil}$, as discussed by Sen. We may illustrate example offsets of order 0.1″ and $10^2 - 10^3$ km/s for a few percent of objects[16][17].

Finally, one would include plotting routines (e.g. using Matplotlib) to produce figures like those above. Below is a sketch of how to generate an example mass-growth plot and histogram of recoils:

```
import matplotlib.pyplot as plt

# Example: compare one growth history with/without recoil
mass_recoil, _ = run_smbh_growth(50, V_esc=500, M_halo=1e12)
```

```
# Simple no-kick growth for reference
M_nr = 100.0
mass_nokick = [M_nr]
for _ in range(50):
    M_nr += random.choice([1e2,1e3,1e5])  # random seed merge
    mass_nokick.append(M_nr)

plt.figure(figsize=(5,4))
plt.plot(mass_nokick, label='No recoil')
plt.plot(mass_recoil, label='With recoil')
plt.xlabel('Merger step')
plt.ylabel('Central SMBH mass (M$_\odot$)')
plt.legend()
plt.title('SMBH growth: recoil vs no kick (illustrative)')
plt.tight_layout()
plt.show()

# Recoil velocity histogram
recoil_speeds = [compute_recoil_velocity(1e6,1e5,*random.choice(
    [(0.3,0.5,0.7),(0.1,0.2,1.4)]) ) for _ in range(1000)]
plt.hist(recoil_speeds, bins=20, color='skyblue', edgecolor='k')
plt.xlabel('Recoil speed (km/s)')
plt.ylabel('Count')
plt.title('Distribution of GW kick speeds')
plt.tight_layout()
plt.show()
```

These illustrative plots would mirror the trends described in Sen (2025) – e.g. the mass-growth suppression[12] and the presence of a few high-velocity kicks that reduce retention[5]. In a full code package, one would modularize the above functions (perhaps into classes or separate modules) and allow parameters (seed fractions, halo masses, etc.) to be easily modified. Each function is documented to indicate its physical role and any formula sources (as done above).

Sources: The recoil formulas and retention scheme follow Sen (2025)[4][5] (which in turn cites Campanelli et al. 2007 and Lousto & Zlochower 2012). Spin assumptions are as described in the same reference[3]. The overall model and Monte Carlo approach are directly based on the outlined method in Sen (2025)[18][12].

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18]
fresh_mnras_BH.pdf

file://file_000000005858622f8fbcdb58ddffa79a