Python Script for Grim's Heart:

```
Import numpy as np
From scipy.integrate import solve_ivp
Import matplotlib.pyplot as plt  # Optional for plotting


Def closure_gap(G):
   “””Compute Delta = det(G) – tr(G)^2 / 4”””
   Det_G = np.linalg.det(G)
   Tr_G = np.trace(G)
   Return det_G – (tr_G ** 2) / 4


Def commutator(G, J):
   “””Compute [G, J] = G J – J G”””
   Return G @ J – J @ G


Def dynamical_law(t, flat_G, J):
   “””Flattened version for solve_ivp: dot{G} = Delta [G,J] – 2 Delta^2 G”””
   G = flat_G.reshape(2, 2)
   Delta = closure_gap(G)
   dG = Delta * commutator(G, J) – 2 * (Delta ** 2) * G
   return dG.flatten()


J = np.array([[0, -1], [1, 0]])  # Rotation matrix


Def simulate_bulge(initial_G):
   “””Simulate one trajectory until Delta > -1e-10, return final eigenvalue ratio”””
```

```
    Def event(t, y):  # Termination event
        G = y.reshape(2, 2)
        Return closure_gap(G) + 1e-10  # Halt when Delta >= -1e-10
    Event.terminal = True
    Event.direction = 1  # Only from below

    Sol = solve_ivp(dynamical_law, [0, 10], initial_G.flatten(), args=(J,),
                Method='DOP853', rtol=1e-8, atol=1e-10, events=event)

    If sol.status == 1:  # Event triggered
        Final_G = sol.y[:,-1].reshape(2, 2)
        Eigvals = np.linalg.eigvals(final_G)
        Abs_eig = np.abs(eigvals)
        Ratio = np.max(abs_eig) / np.min(abs_eig) if np.min(abs_eig) > 0 else np.inf
        Return ratio
    Return np.nan  # Failed sim

# Run over 1000 random initial conditions (Delta_0 < 0)
N_runs = 1000
Ratios = []
For _ in range(n_runs):
    G0 = np.random.uniform(-1, 1, (2, 2))
    If closure_gap(G0) < 0:  # Only valid starts
        Ratio = simulate_bulge(G0)
        If not np.isnan(ratio):
            Ratios.append(ratio)
```

```
Mean_ratio = np.mean(ratios)

Std_ratio = np.std(ratios)

Print(f"Mean bulge ratio over {len(ratios)} runs: {mean_ratio:.4f} ± {std_ratio:.0e}")


# Optional: Plot a single trajectory's Delta evolution

G_example = np.array([[0.5, -0.2], [0.1, 0.3]])  # Example with Delta < 0

Sol_example = solve_ivp(dynamical_law, [0, 5], G_example.flatten(), args=(J,),
dense_output=True)

T = np.linspace(0, 5, 100)

Deltas = [closure_gap(sol_example.sol(ti).reshape(2,2)) for ti in t]

Plt.plot(t, deltas)

Plt.xlabel('Time')

Plt.ylabel('Delta')

Plt.title('Approach to Instability Threshold')

Plt.show()
```