Exercise 1（Rate vs Time）

- 使用 np.load("hw03-data.npz") 在 google colab 讀入 rates（單位 Hz）與 delta_t（單位 s），時間軸 t = np.arange(len(rates)) x delta_t，總時長 duration = len(rates) * delta_t。
- 以 plt.plot(t, rates) 繪製 **rate–time 曲線**，標註 x 軸（Time, s）、y 軸（Rate, Hz）。

Exercise 2（100-trial Raster；非齊次 Poisson 的 thinning）

- 設上界速率 M = max(rates)，總時長 T = duration。
- 每個 trial：先抽候選得點數 N ~ Poisson(M·T)，候選時間 cand ~ Uniform(0,T)；對每個候選地時間 u，以機率 r(u)/M 接受（保留），得到該 trial 的 spike 時間。重複產生 **100 條**。
- 以 plt.eventplot(spikes_all) 繪製 **raster**（每列一個 trial；x 軸為時間）。

Exercise 3（Frame-based + Gaussian smoothing）

- 從 Ex.2 選 **trial 100**（index 99），該 trial 以 time-of-spike 表示。
- 轉為 **frame-based**：取 frame_dt = 0.0001 s（注意不同於 delta_t），建立 frame_times = np.arange(0, duration, frame_dt) 與同長度 0/1 陣列 frames，對每個 spike 位置填 1。
- 建立 **Gaussian 核**（σ = 0.1 s），離散核以 kernel /= kernel.sum() * frame_dt 正規化，使核對時間的「面積 = 1」（輸出還是 Hz）。
- 以 np.convolve(frames, kernel, mode="same") 得到預估地 **firing rate vs time**，並與原始 rates 圖片比較；原始曲線用以**虛線**表示。

Results

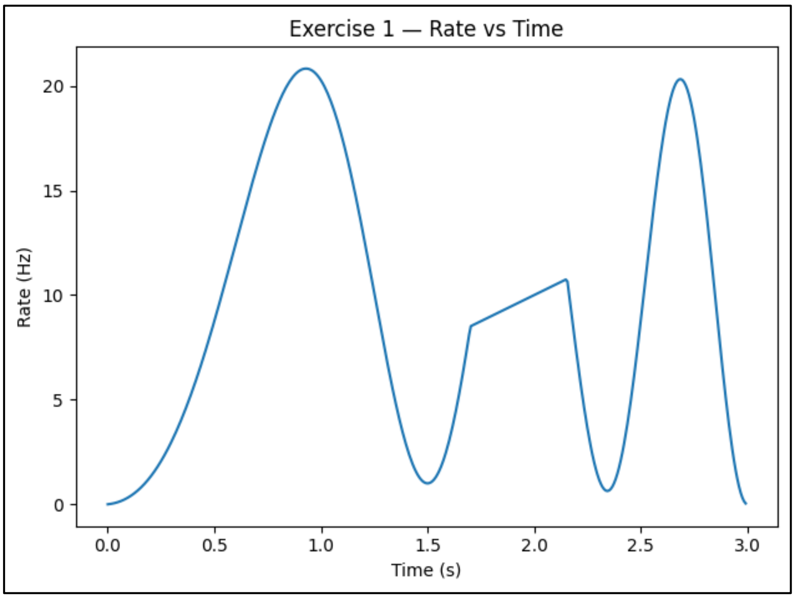

圖 1（Exercise 1）

rates(t) 呈現數個明顯的高低起伏：前段隨時間上升至一個高峰，中段短暫回落並伴隨一段近似線性的上升，後段再次形成高峰後迅速下降。簡單說，這條曲線就是**刺激強度隨時間在變**的樣子，後面我們就**拿它當生成 spike trains 的目標發放率**。

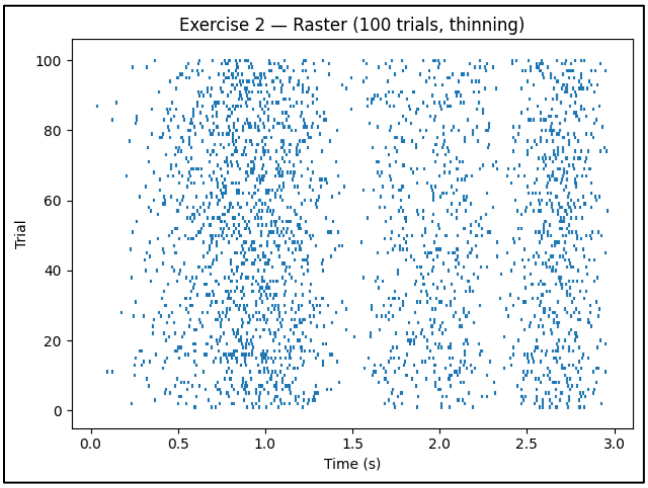

圖 2（Exercise 2 — Raster（100 trials, thinning））

我們用 thinning（random points in a box） 這招，照著 rates(t) 去生 100 條 spike trains。看 raster 圖就很明顯：**rates(t) 高的時候**，點點**擠在一起**，整片像**直直的帶狀**；**rates(t) 低的時候**，點就**稀稀落落**。這種「點的密度」跟 rates(t) 的起伏**一一對得上**，等於在告訴我們：**模擬出來的非齊次**

Poisson spike trains 跟目標速率是對的。

BTW 我們也計算了經驗平均發放率 total_spikes / (duration × trials)，其數值與 rates.mean() 接近，作為 sanity check。

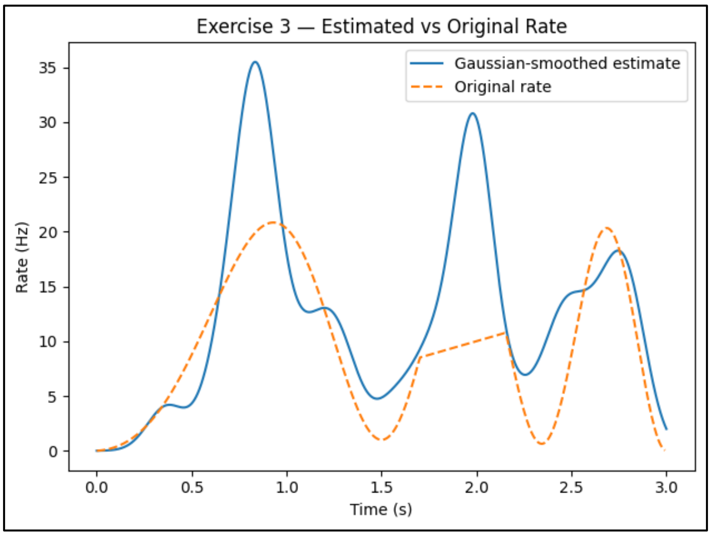

圖 3（Exercise 3 — Gaussian-smoothed rate vs Original rate）

先把 Ex.2 生出的 **第 100 條 spike train** 抓出來，接著用 frame_dt = 0.0001 s 把它轉成每格不是 0 就是 1 的 **frame-based** 序列。然後套一個 **Gaussian 核**（σ = 0.1 s，面積有做歸一化），用 np.convolve(..., "same") 去平滑，得到這條 trial 的 **firing rate**。

圖上**實線**是平滑後的估計，**虛線**是原本的 rates(t)：你會看到估計曲線在高峰處比較圓、在谷底不會整個貼到 0；這就是平滑帶來的 **偏差 / 變異** 取捨。σ **開大**，線會更順但峰值會被「壓扁」；σ **開小**，線更貼近原始起伏但抖動也會變多。整體趨勢和原始速率是對得上的。