Lecture 3

2025-09-17

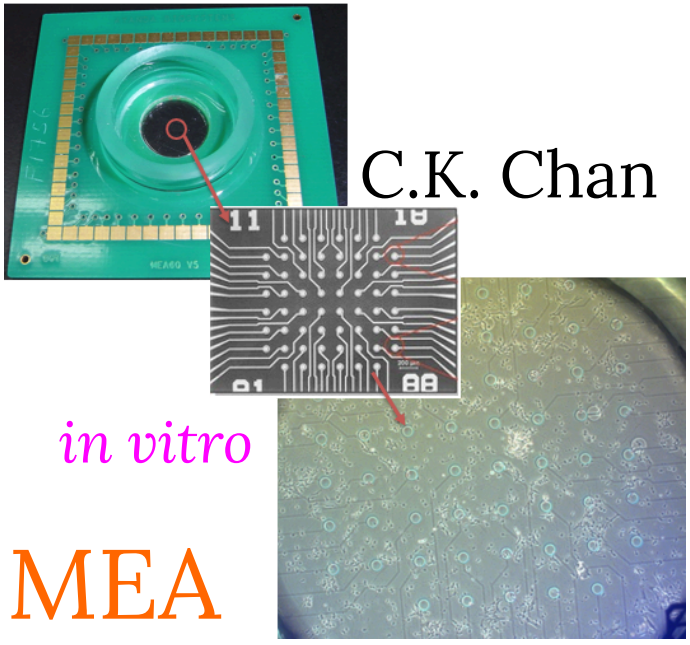
Spike trains with time-varying rates; Bias–variance tradeoff; Poisson process; Random number generator

- Online slides: lec03-poisson_process.html
- Code: code03.ipynb, lec03-data.npz
- Homework: hw03.pdf, hw03-data.npz

Username: cns, Password: nycu2025

Laboratory of Computing Biological Networks

Neural signals from a variety of sources

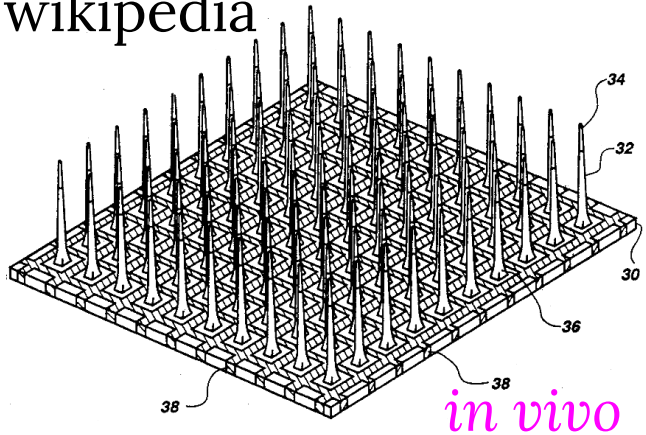


C.K. Chan

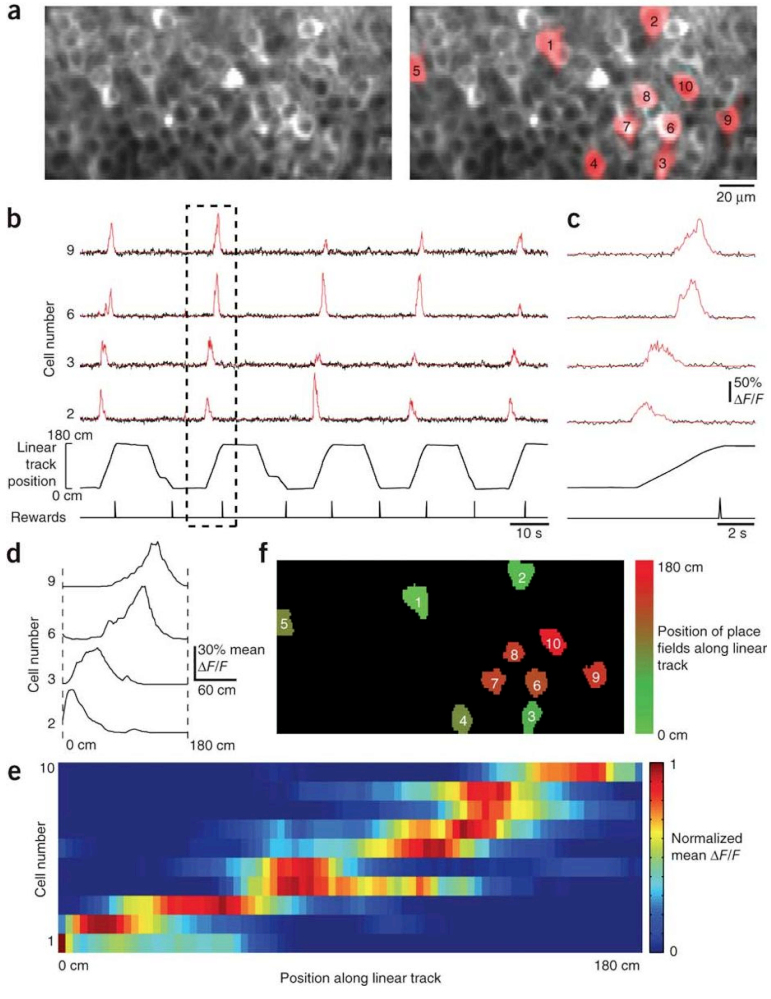
in vitro

MEA

wikipedia



in vivo



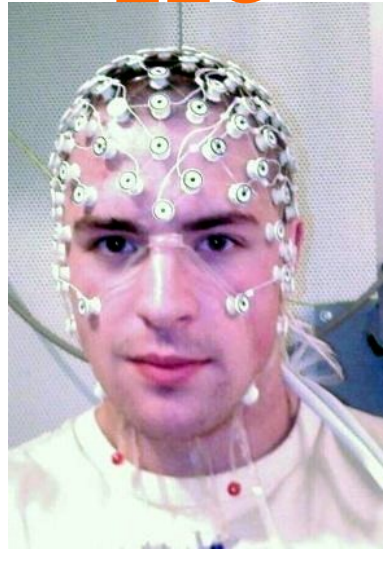
Optical imaging

Meshulem 2017

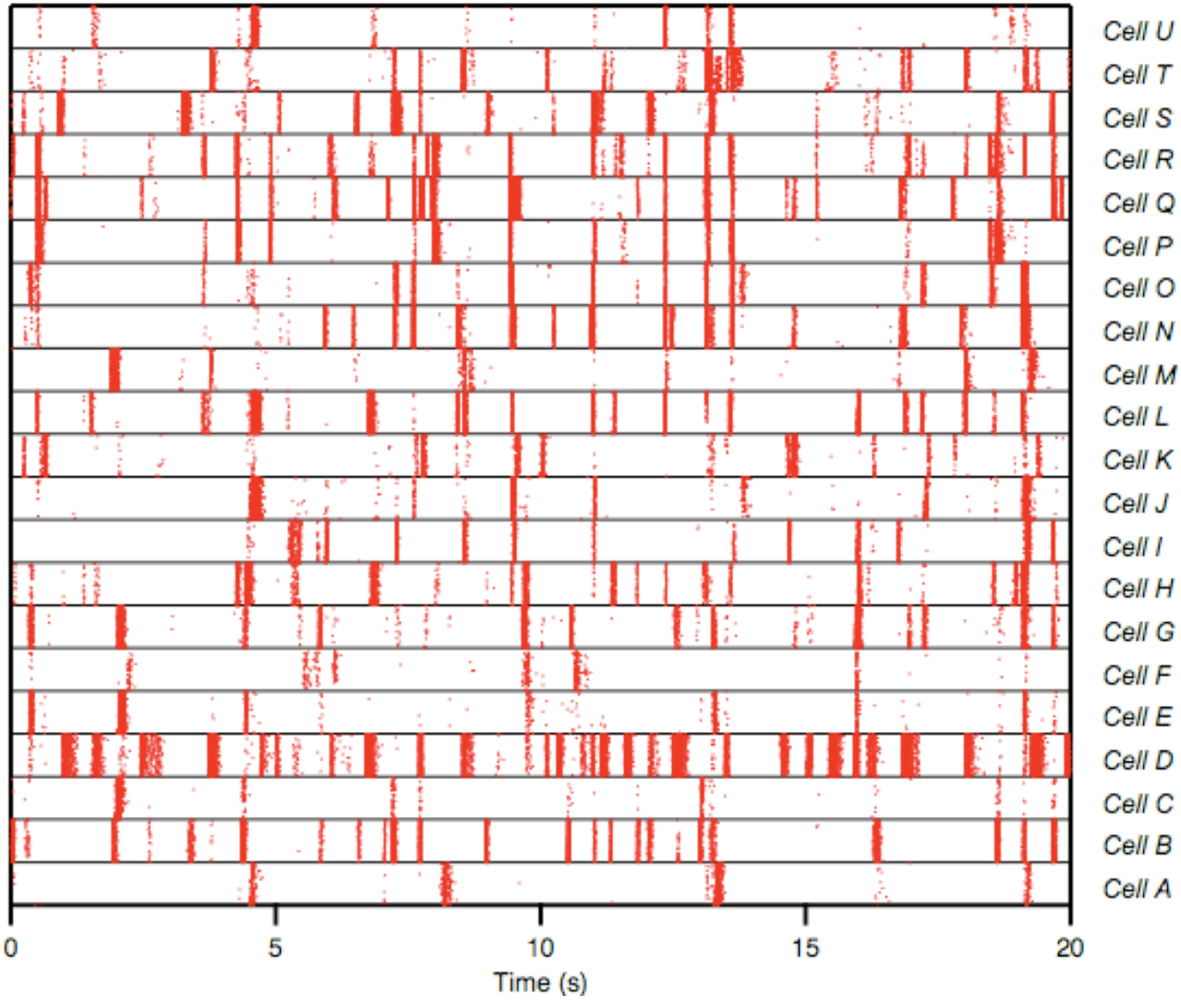
MEG

wikipedia

EEG



Spikes segmented from neural recordings



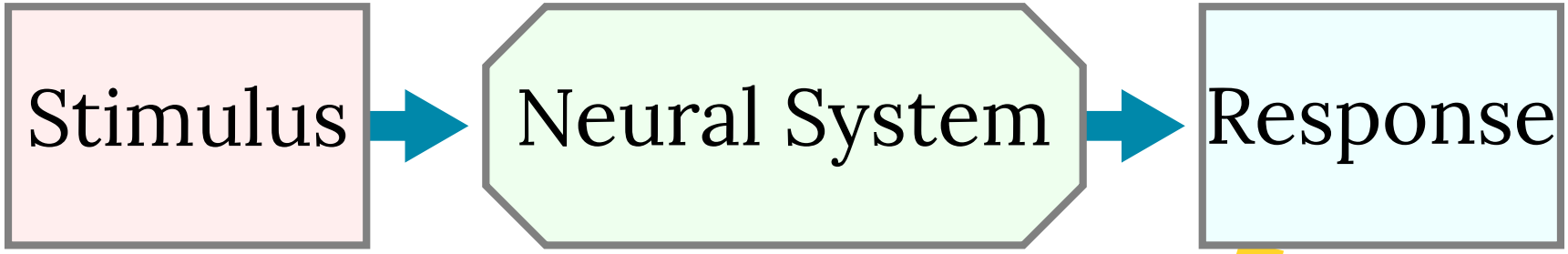
Recording of retina ganglion cells (Michael II Berry)

Raster plot

Each strap contains spike trains from multiple trials of the same stimulus.

Each dot represents a spike time.

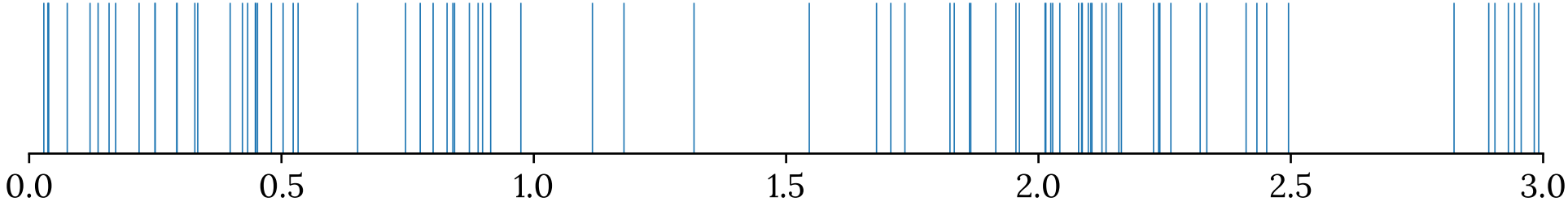
Characterizing neural response to stimuli



Q: *What aspect of the stimulus is baeing processed (encoded) by the neural system?*

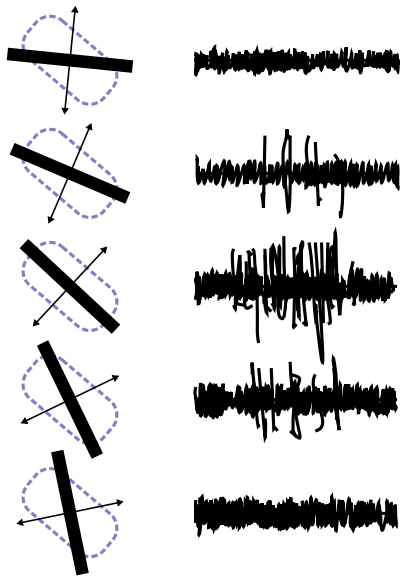
Q: *How can we read (decode) the input stimulus from the spike trains?*

Spike trains



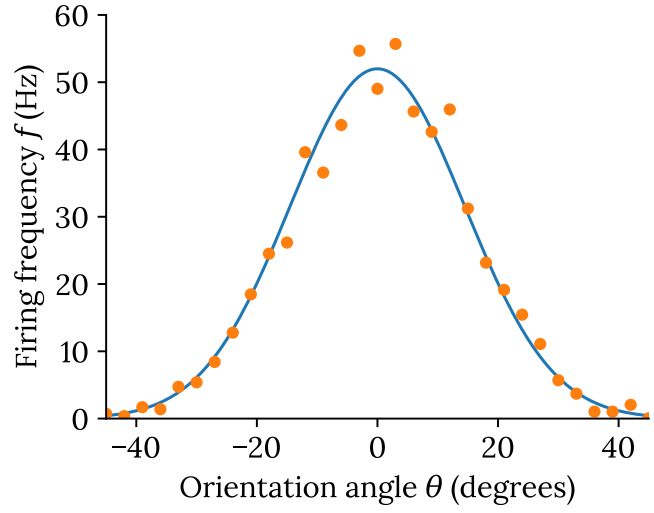
Characterize the neural responses

Tuning curve



Statistics

⇒



— *Fine for static input...*

For time-varying input, instantaneous response can only be estimated approximately. (This is what we did in the last lecture with window functions, kernels, convolution, etc.)

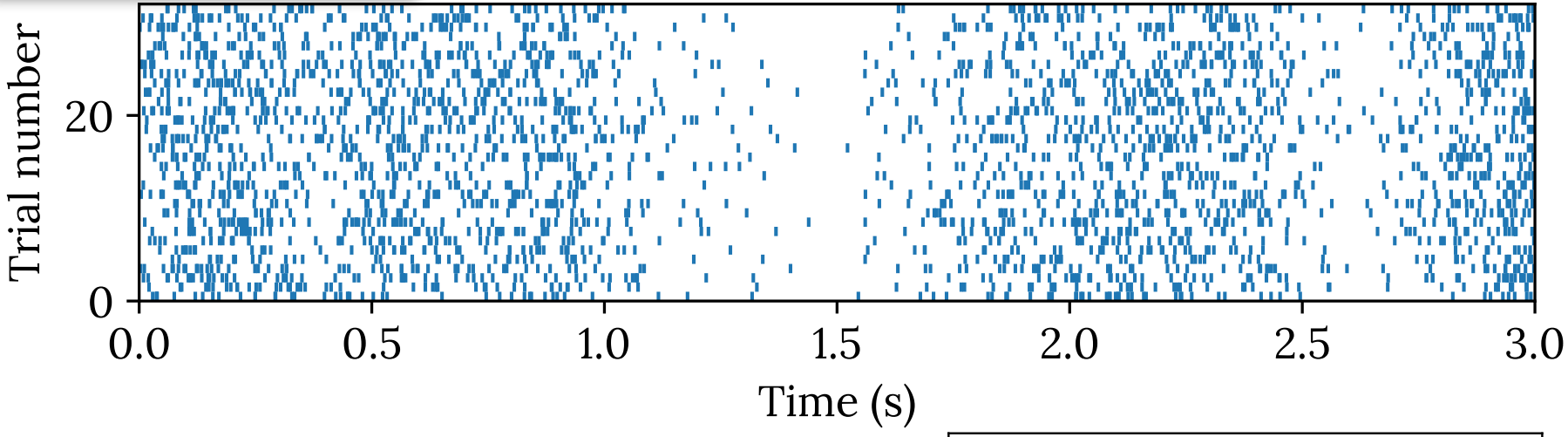
In the above analysis,

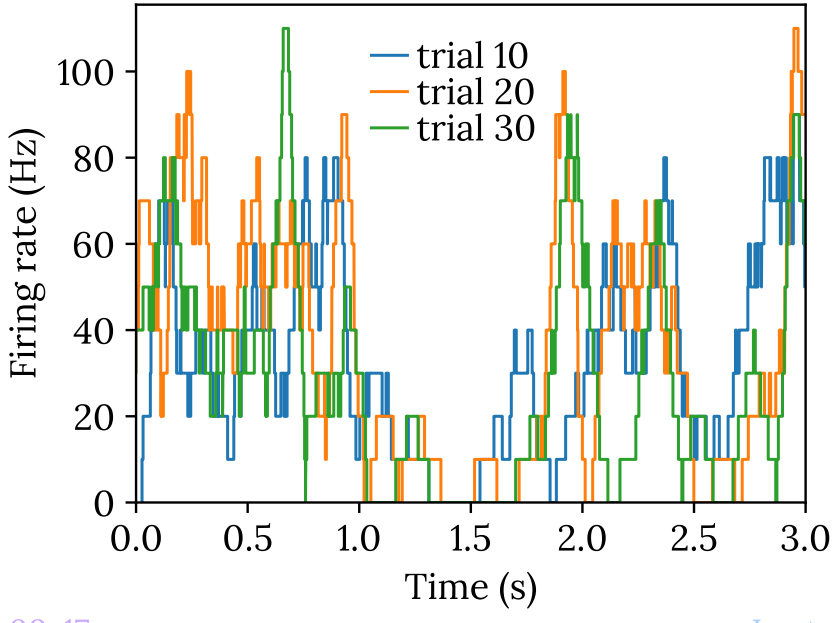
- only a single degree of freedom is considered...
- exact (absolute or relative) timing of spikes is ignored...
- a trade-off exists between statistics and precision...

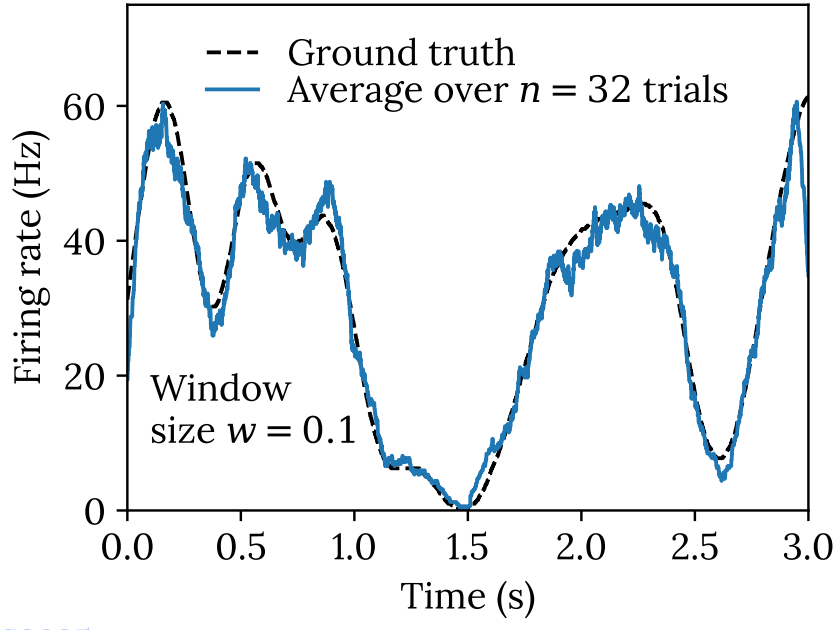
Improve time precision with trial average

Average over many repeated measurements for the same time-varying input...

Raster plot of spikes

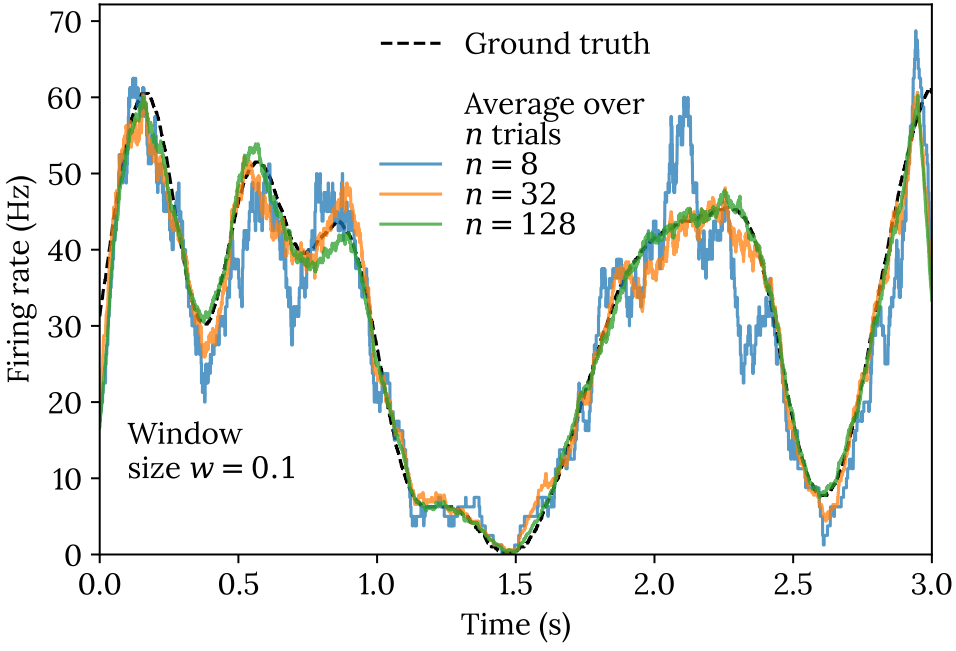




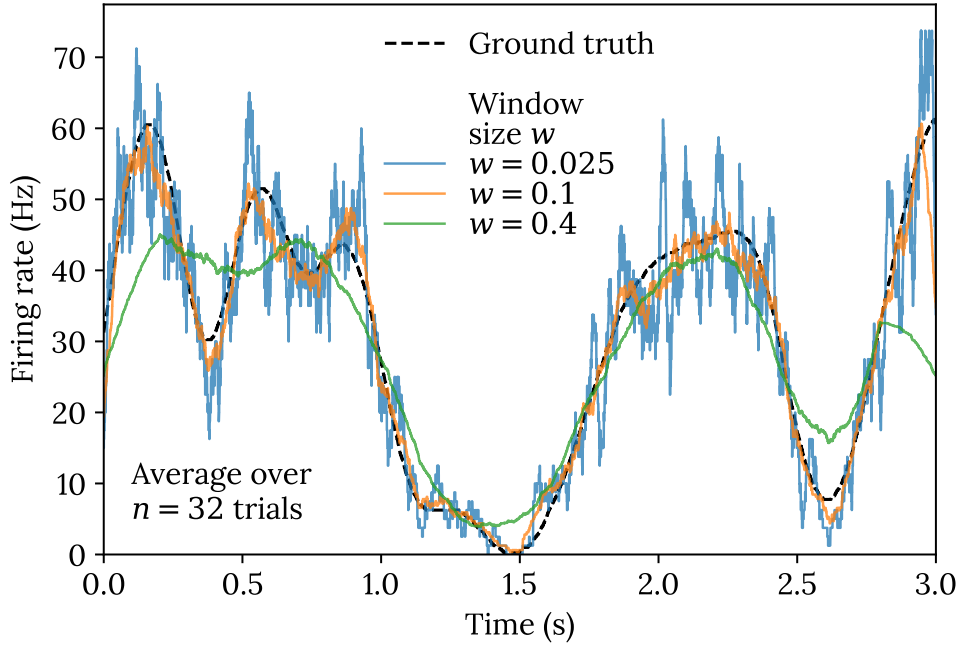


Tradeoff between bias and variance

Varying number of trials



Varying size of sliding window



To reach a good statistics and precision for the approximated firing-rate function, two limits should be taken: infinite number of trials and infinitesimal window size.

Event-generating processes

- **Point process**
  A stochastic process that generates a sequence of events.

- **Renewal process**
  A point process whose intervals between successive events are independent.

- **Poisson process**
  A point process whose events themselves are statistically independent.

$$\text{Poisson process} \subset \text{Renewal process} \subset \text{Point process}$$

Event: *a location in space time, can be described by a coordinate.*

Poisson process

"Events themselves are statistically independent"

⇑ What does this mean?

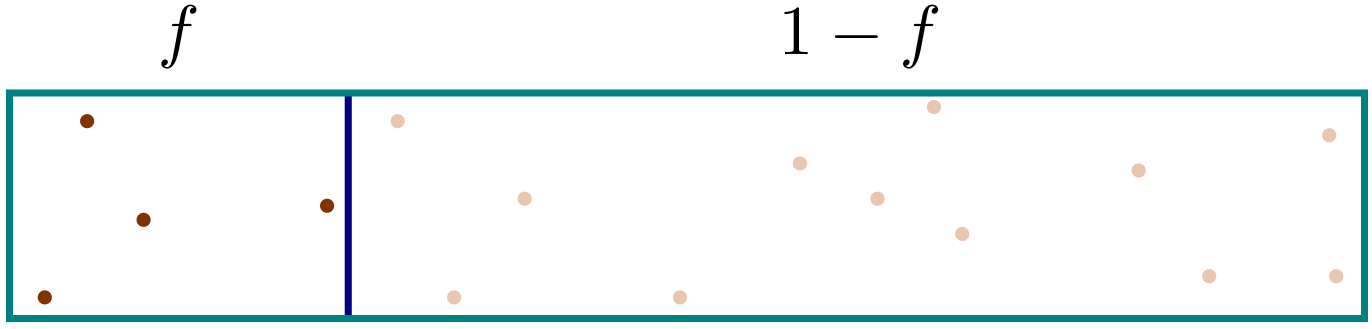
During an interval, event times can be generated by independent and identically distributed (iid) random variables. While, the number of these random variables follows the Poisson distribution

$$P(k) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{1}$$

where $\lambda = \langle k \rangle = \sigma^2(k)$ is both the mean and the variance of the distribution.

In a simple (but profound) term, it is a rate (density) process.

From fixed averages to rate



Consider a total of N points randomly distributed in a box of size 1. Among the N points, there are k of them happen to locate in the given fraction f area of the box. We can expect the mean *outcome* of k to be $\bar{k} = Nf \equiv \lambda$.

The probability to get a given k is given by

$$C_k^N f^k (1-f)^{N-k}. \tag{2}$$

Taking the limits $N \to \infty$, $f \to 0$ while keeping the product $\lambda = Nf$ fixed leads to the Poisson distribution (1).

Details of the limit

$$
\begin{aligned}
& C_k^N f^k (1-f)^{N-k} \\
=& \frac{N!}{k!(N-k)!} \frac{\lambda^k}{N^k} \left(1 - \frac{\lambda}{N}\right)^{N-k} \\
=& \frac{\lambda^k}{k!} \left(1 - \frac{\lambda}{N}\right)^{N} \frac{N(N-1)\cdots(N-k+1)}{N^k} \left(1 - \frac{\lambda}{N}\right)^{-k} \\
\approx& \frac{\lambda^k}{k!} e^{-\lambda}
\end{aligned}
$$

See Wikipedia for more properties of Poisson distribution.

Poisson process for bus arrival

Consider arrival rate of bus being 1 bus per 10 min.

For each minute, there is 1 out of 10 chance for a bus to arrive. Thus, upon arriving at the stop, you are expected to wait for 10 min for the next bus. Looking back in time, the last bus is also expected to have left, on average, 10 min ago. Therefore, you are encountering a 20-min gap on average between bus arrivals.

Questions:

1. What is the average gap between bus arrivals in the above process?
2. Why are you expected to hit a bus gap that is twice as long as the average?
3. How long are you expected to wait for bus if the schedule is regular instead of Poisson?

Poisson spike train generation

Two approximations

1. For given rate μ, generate μT spike times for a length T time interval.

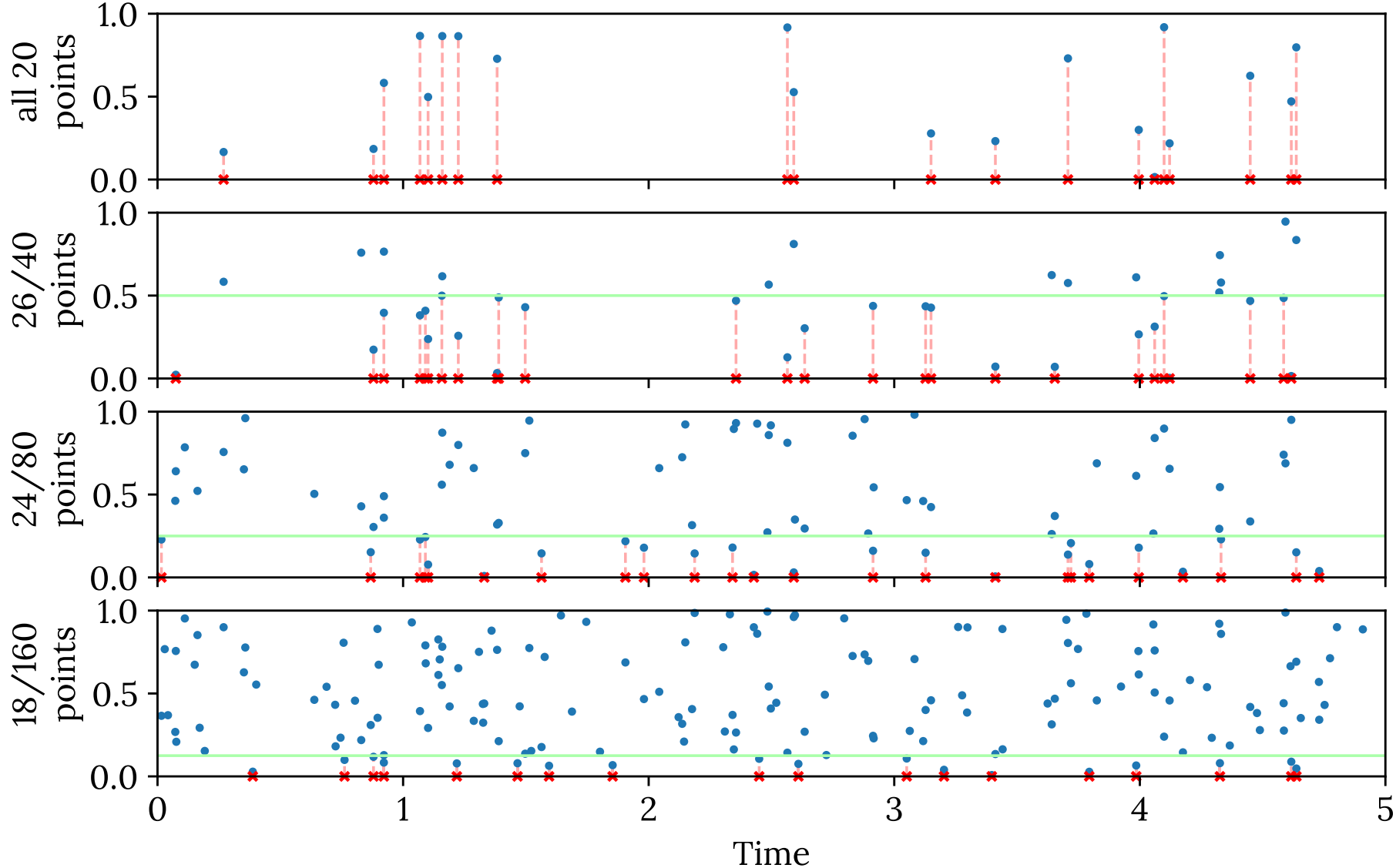
   High spike-time precision; Need $\mu T \to \infty$ for exact Poisson statistics; Need to generate spikes outside interested interval; Nontrivial for time-varying μ

2. For each size Δt time step, decide whether a spike occurs with probability $\mu \Delta t$.

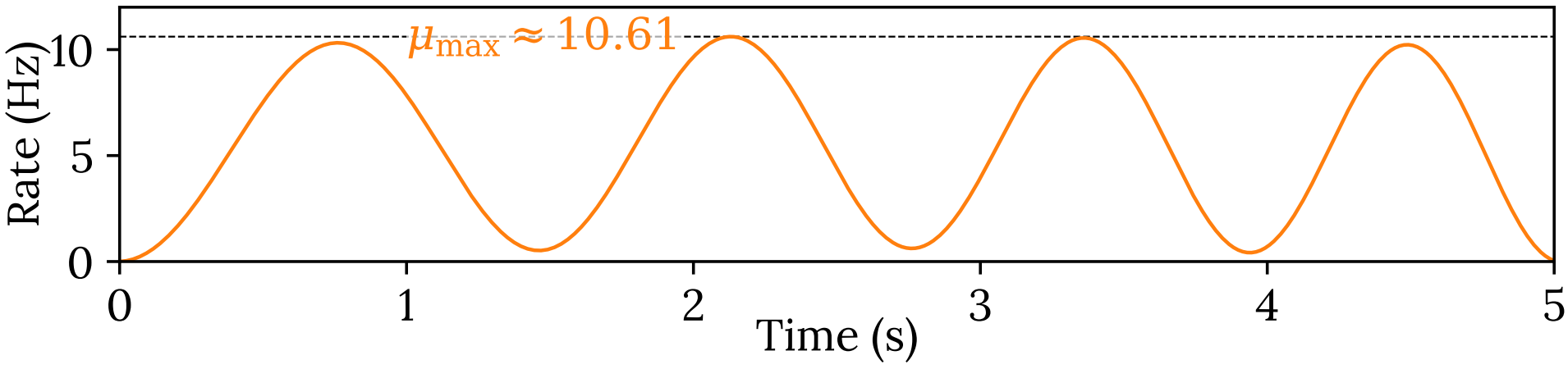
   Precision is limited to Δt; Only single spike is possible in each time step; Need $\Delta t \to 0$ for exact Poisson statistics; May encounter numerical problem; Easy for time-varying μ

Approaching Poisson with an accepting rate

Simulate Poisson process with rate $\mu = 4$ for interval $T = 5$:

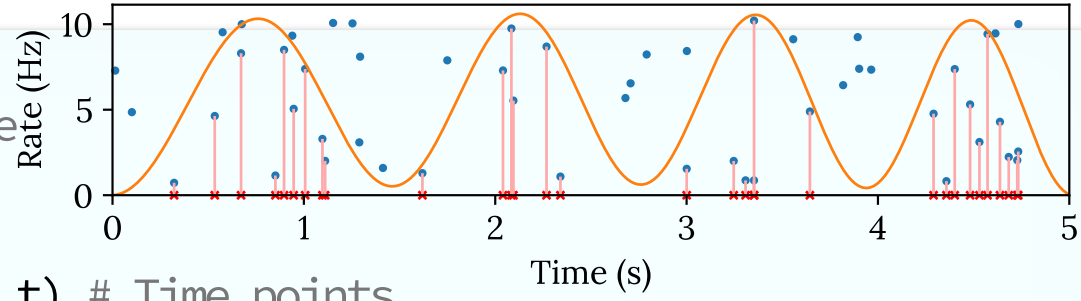


Time-varying-rate Poisson spike train



- Turn the threshold line into a curve
- Minimize rejection by choosing $\mu_{\max}$ as box height
- Generate $N = \lfloor \mu_{\max} T \rfloor$ random points (density $\rho \approx 1$)
- Keep the points under the curve

Details of the implementation




```
# Make up a time-varying spike rate
time_range = 5  # Total range of time
delta_t = 1/100 # Time step size

times = np.arange(0,time_range,delta_t) # Time points
rates = 10*np.sin(times*(2+times/10))**2+times*(5-times)/10 # Rates

# Create box and find the number of points needed in the box
rate_max = np.max(rates)
num_points = round(rate_max*time_range) # Total number of points

# Generate random points in the box
rng = np.random.default_rng(12)
xs = rng.uniform(0,time_range,size=num_points)
ys = rng.uniform(0,rate_max,size=num_points)

# Pick points under the rate curve
kept_indices = ys<np.take(rates,np.floor(xs/delta_t).astype(int))
xs_kept,ys_kept = xs[kept_indices],ys[kept_indices]

final_spikes = np.sort(xs_kept) # Resulting sorted spike train
```

(Pseudo) random number generator

To account for unknown factors of flucation in experimental systems, stochastic (random) variables are widely included in theoreical models.

Because of the lack of true randomness in classical computers and for the reproducibility of numerical results, pseudo random numbers (pRNG or simply RNG) are used to simulate the stochastic effects.

Many pRNG are based on recurrent maps:

Example sequence of random numbers

1, 5, 3, 4, 8, 6, 7, 2, ...

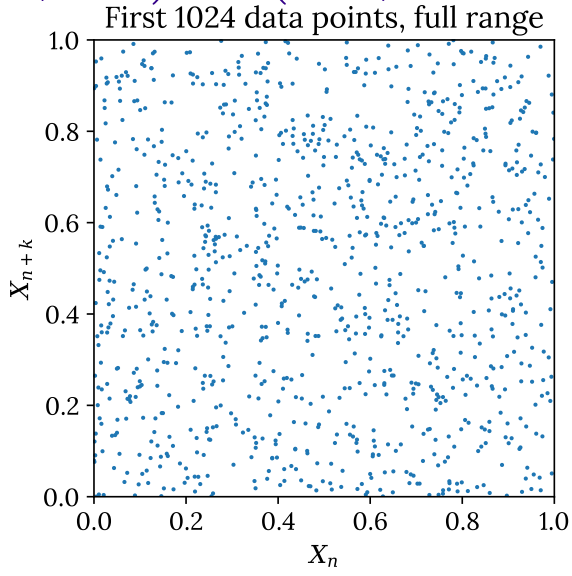
Linear congruential generator

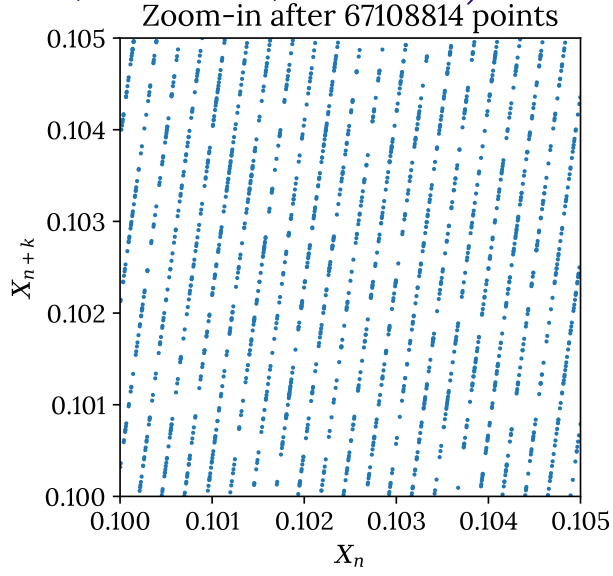
$$X_{n+1} = (aX_n + c) \mod m$$

Parameters for the above sequence: $m = 9$, $a = 4$, $c = 1$

ANSI C `rand()`: $(m, a, c, X_0) = (2^{31}, 1103515245, 12345, 12345)$

Perform k-step test
in s dimension
$(k, s) = (25, 2)$





Some notes on using RNGs

- Seed your RNG for reproducible results
- Use different seeds to simulate different trials
- Don't implement! Use well-established ones, e.g.,

  MT19937

  Available in C++ standard library; Used by numpy for legacy random number generation ⇒ Used to be the way.

  PCG64

  New default bit generator of numpy ⇒ Will use this now.

- Store the state of the generator to resume the random sequence

```
A: [6 2 7 3 2 7 6 6]
B: [9 3 8 3 5 5 2 1]
C: [0 7 9 8 6 4 6 0]
D: [9 3 8 3 5 5 2 1]
```

```python
rng = np.random.default_rng(12345)
print('A:',rng.integers(10,size=8))
sr = rng.bit_generator.state # Save the state
print('B:',rng.integers(10,size=8))
rng = np.random.default_rng(54321)
print('C:',rng.integers(10,size=8))
rng.bit_generator.state = sr # Load the state
print('D:',rng.integers(10,size=8))
```