Computational Neuroscience 2025

Lecture 4

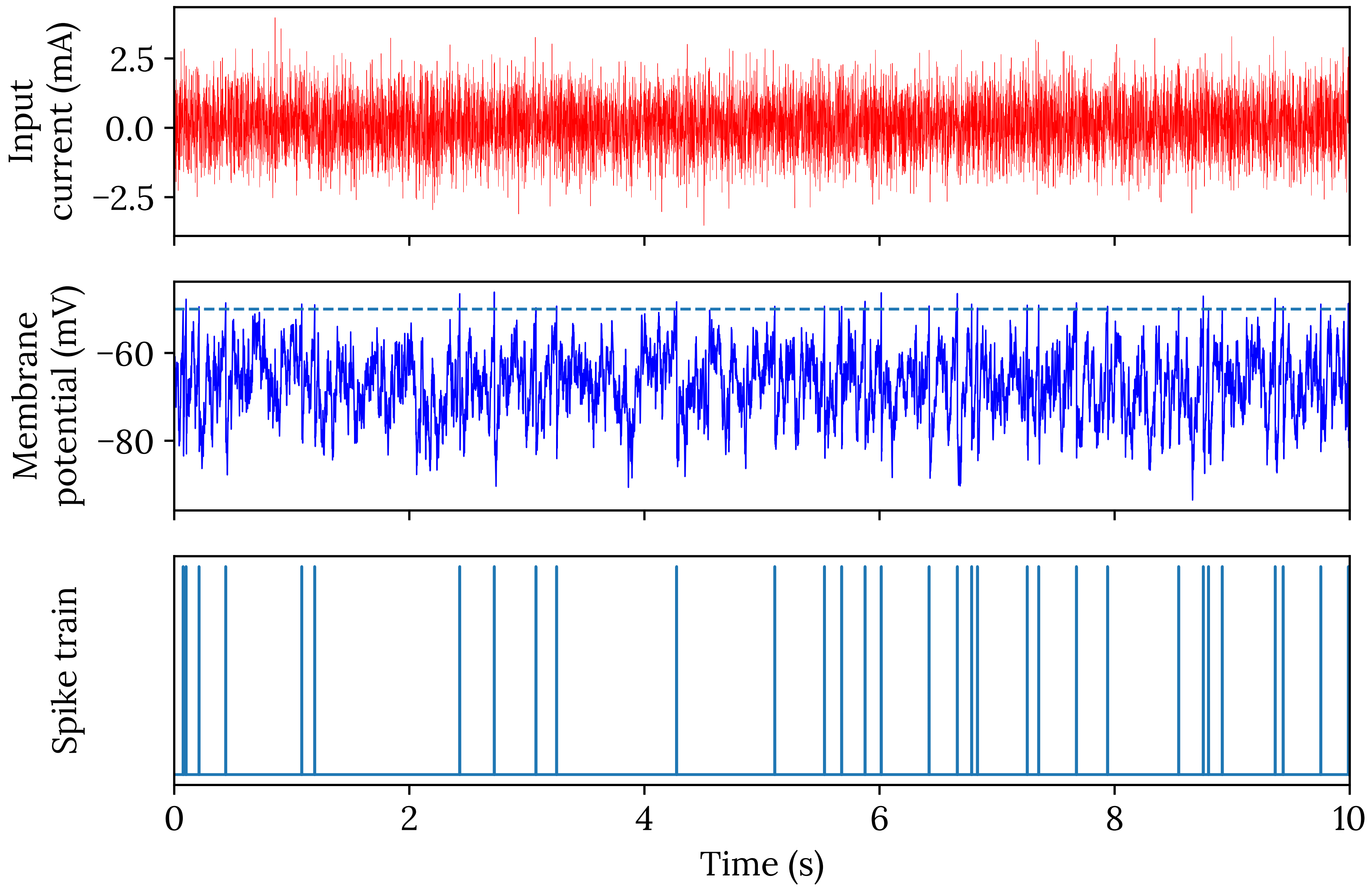
2025-09-24

What makes a neuron fire: Spike-trigger average; Noise and correlation; Linear–nonlinear Poisson cascade model

- Online slides: lec04-spike_trigger.html
- Code: code04.ipynb, lec04-data.npz
- Homework: hw04.pdf, hw04-data.npz, c1p8.mat

Username: `cns`, Password: `nycu2025`

Stimulus, membrane potential, and spikes



Simple model that generates the signal

Leaky Integrate-and-fire neuron

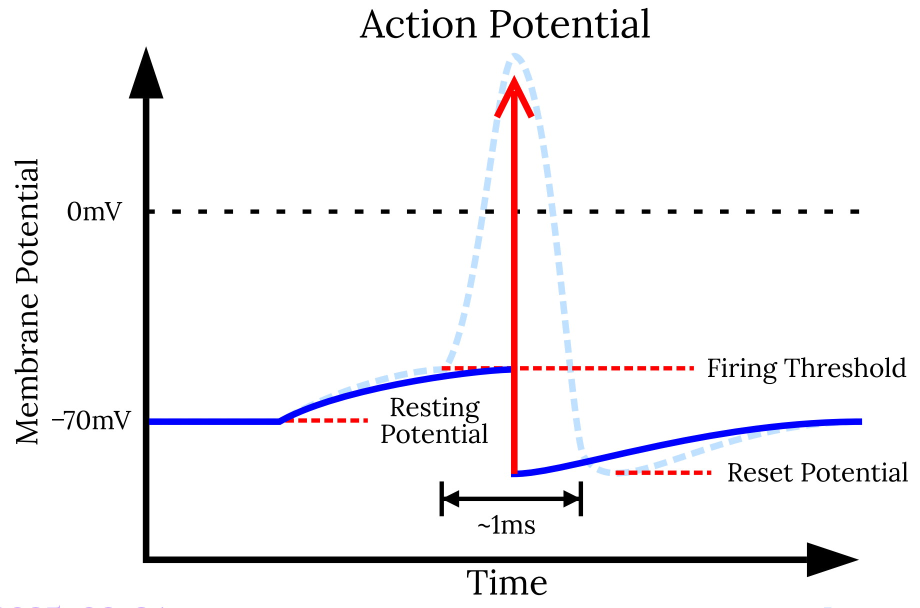
Membrane potential dynamics:

$$\tau \frac{dv}{dt} = v_0 - v + Ir$$

driven by $I = \eta + I_{\text{bg}}$, while, the noise

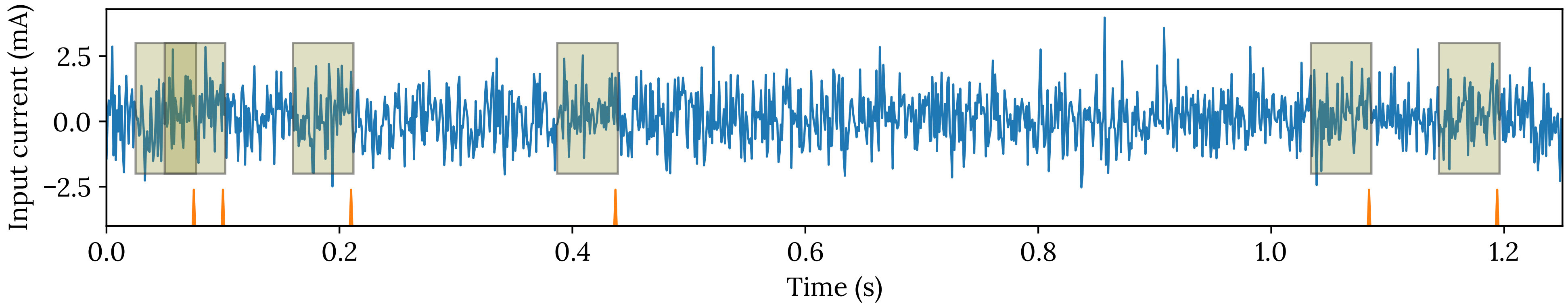
$$\langle \eta(t)\eta(t')\rangle = \sigma_\eta^2 \delta(t - t')$$

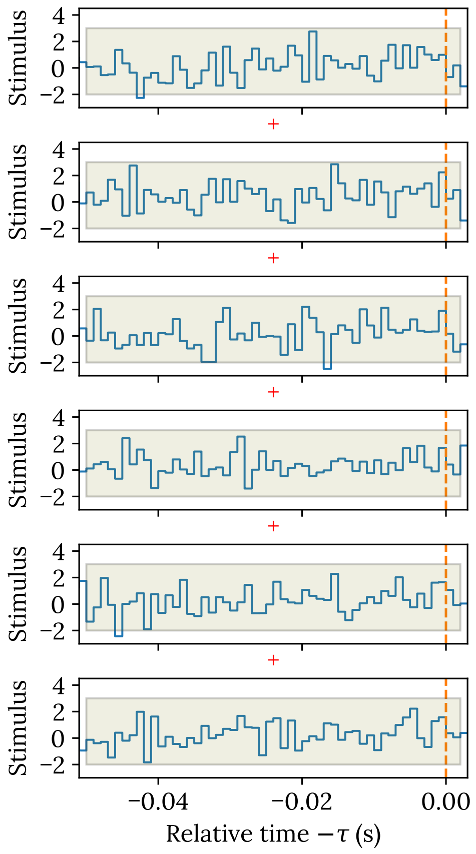
is an uncorrelated, white noise.



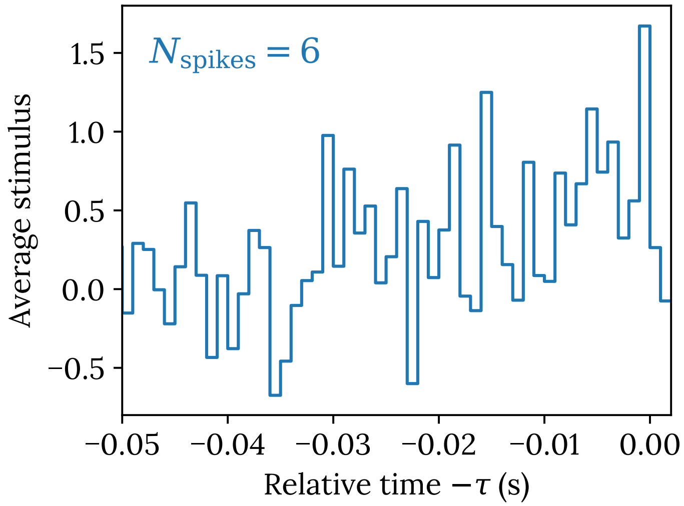
```
T = 10  # simulate for 10s
v = vi # initial state
n = int(T/dt) # number of time frames
# traces
vs = []  # membrane potential
ii = []  # input stimulus current
ts = []  # time
st = []  # spike train
rng = np.random.default_rng(1234)
t = 0
for i in range(n):
   ic = sgm*rng.normal()/np.sqrt(dt)+bg
   v += dt*(v0-v+ic*r)/tau
   t += dt
   vs.append(v)
   ii.append(ic)
   ts.append(t)
   if v>vth:
      v = vr
      st.append(1)
   else: st.append(0)
```

Spike-trigger average of stimulus





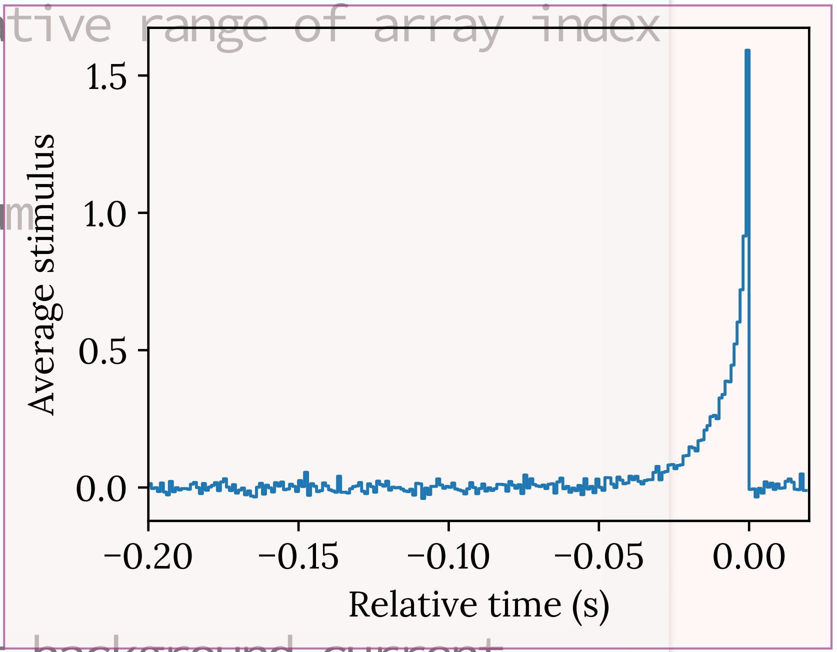
$$/\, N_{\text{spikes}} =$$



Spike-triggered average

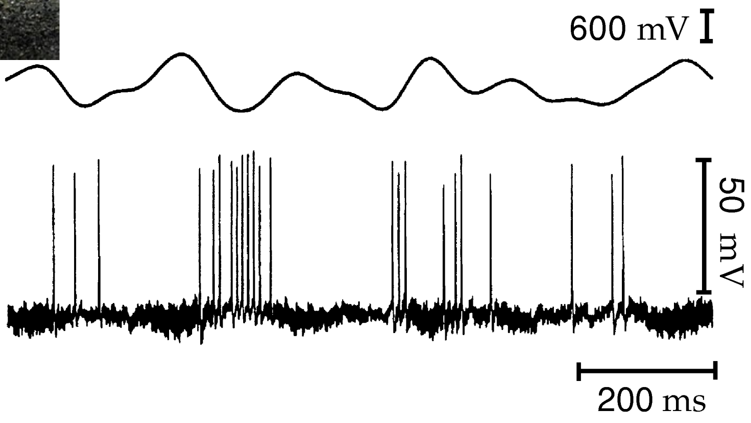
Implementing spike-triggered average

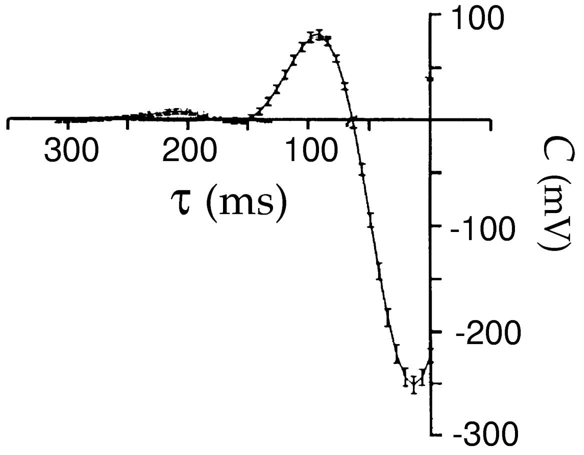
```python
win_range = np.array([-0.2,0.02]) # Relative range of window
[i0,i1] = (win_range/dt).astype(int) # Relative range of array index

# Loop implementation
wsum = np.zeros(i1-i0) # Spike-triggered sum
cnt = 0
for i in range(-i0,n-i1):
    if st[i]==0: continue
    wsum += ii[i+i0:i+i1]
    cnt += 1
print(f'Average over {cnt} spikes')
sta1 = wsum/cnt-ii.mean() # Remove constant background current
```



```python
plt.step(np.arange(*win_range,dt),sta1,lw=1)
plt.xlim(win_range)
plt.xlabel('Relative time (s)')
plt.ylabel('Average stimulus')
save_plot('spike_trigger_average')
plt.show()
```

Real example of STA

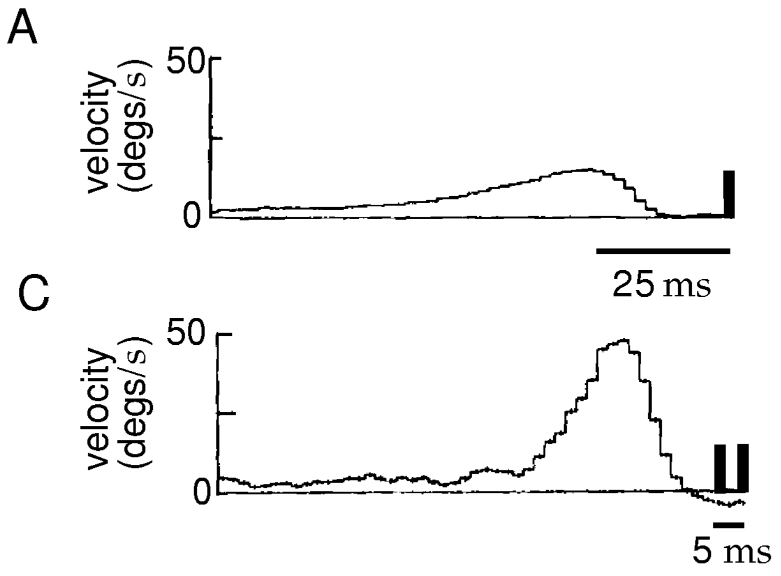




The spike-triggered average stimulus for a neuron of the electrosensory lateral-line lobe of the weakly electric fish Eigenmannia. The upper left trace is the potential used to generate the electric field to which this neuron is sensitive. The evoked spike train is plotted below the stimulus potential. The plot on the right is the spike-triggered average stimulus. (Adapted from Gabbiani et al., 1996.)

Figure 1.9, Dayan & Abbott 2001

Triggering on combinations of spikes



Single- and multiple-spike-triggered average stimuli for a blowfly H1 neuron responding to a moving visual image. (A) The average stimulus velocity triggered on a single spike. (B) The average stimulus velocity before two spikes with a separation of 10 ± 1 ms. (C) The average stimulus before two spikes with a separation of 5 ± 1 ms. (Data from de Ruyter van Steveninck and Bialek, 1988;Figure adapted from Rieke et al., 1997.)

Figure 1.10, Dayan & Abbott 2001

Mathematical details of STA

$$C(\tau) = \left\langle \frac{1}{n} \sum_{i=1}^{n} s(t_i - \tau) \right\rangle \approx \frac{1}{\langle n \rangle} \left\langle \sum_{i=1}^{n} s(t_i - \tau) \right\rangle \quad (1)$$

$$C(\tau) = \frac{1}{\bar{n}} \int_0^T dt \, \langle \rho(t) \rangle \, s(t - \tau) \quad (2)$$

Correlation between time-dependent firing rate and the input stimulus:

$$Q_{\mathrm{rs}}(\tau) = \frac{1}{T} \int_0^T dt \, \mathrm{r}(t) s(t + \tau)$$

$$C(\tau) = \frac{1}{r} Q_{\mathrm{rs}}(-\tau)$$

Reverse correlation function

Auto correlation and white noise

- Cross correlation

$$Q_{ab}(\tau) = \frac{1}{T}\int_0^T dt a(t) b(t-\tau) \tag{3}$$

- Auto correlation

$$Q_{ss}(\tau) = \frac{1}{T}\int_0^T dt s(t) s(t-\tau) \tag{4}$$

- White condition: $Q_{ss}(\tau) = \sigma_s^2 \delta(\tau)$
- Power spectrum

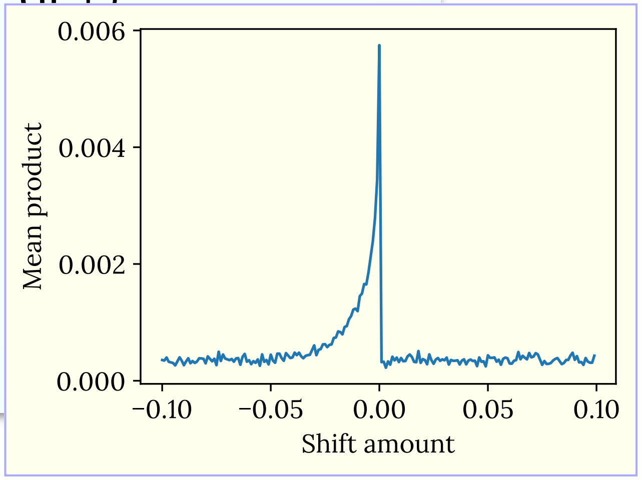
$$\tilde{Q}_{ss}(\omega) = \frac{1}{T}\int_{-T/2}^{T/2} d\tau Q_{ss}(\tau) \exp(i\omega\tau) \tag{5}$$

$$= \frac{\sigma_s^2}{T} \tag{6}$$

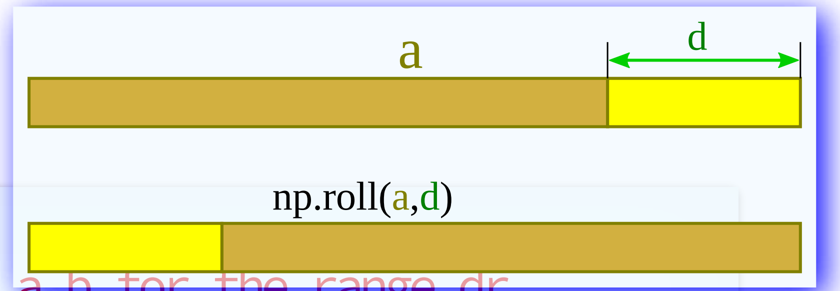
Cross and auto correlation implementation

```python
def shifted_product(a,b,d):
    '''Calculate the mean of a(t)b(t+d) over t

    Parameters
    ==========
    a,b :   Two 1d numpy arrays of the same size
      d :   Relative shift of b from a'''
    if d>0: return (a[:-d]*b[d:]).mean()
    if d<0: return (a[-d:]*b[:d]).mean()
    return (a*b).mean()
dr = np.arange(-100,100) # Range of displaced index
qrs1 = np.array([shifted_product(st,ii,d) for d in dr])
# Alternative code using ternary operator
qrs2 = np.array([(
    st[:-d]*ii[d:] if d>0 else
    st[-d:]*ii[:d] if d<0 else
    st*ii
).mean() for d in dr])
```

```python
plt.plot(dr*dt,qrs)
plt.xlabel('Shift amount')
plt.ylabel('Mean product')
plt.show()
```

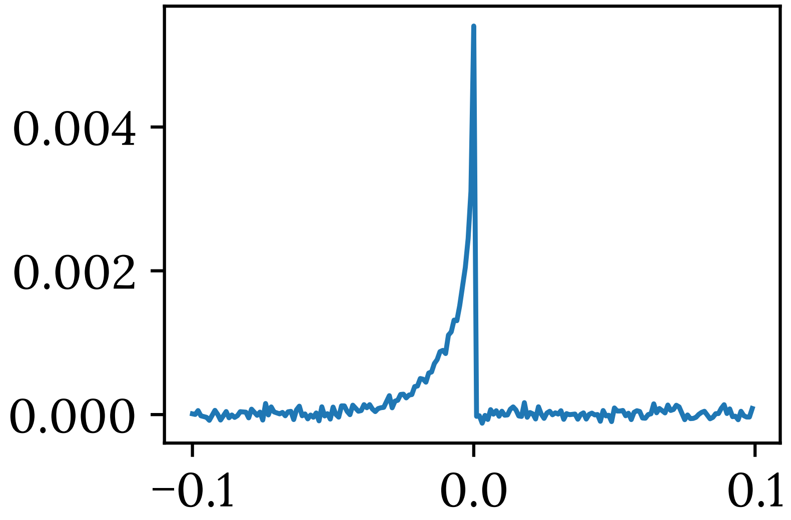


Using numpy.roll for shift

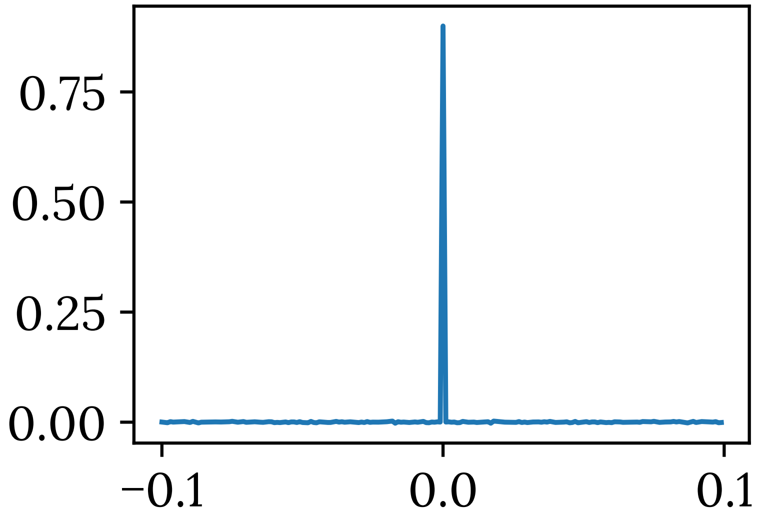


```python
def corr(a,b,dr):
   '''Calculate the cross correlation between a,b for the range dr

   Parameters
   ==========
   a,b :   Two 1d numpy arrays of the same size
    dr :   Range of relative shifts of b from a'''
   return np.array([(np.roll(a,d)*b).mean()-a.mean()*b.mean() for d in dr])
```

a=spike_train, b=input_currents



a=input_currents, b=input_currents



Discrete-time white noise

White condition:

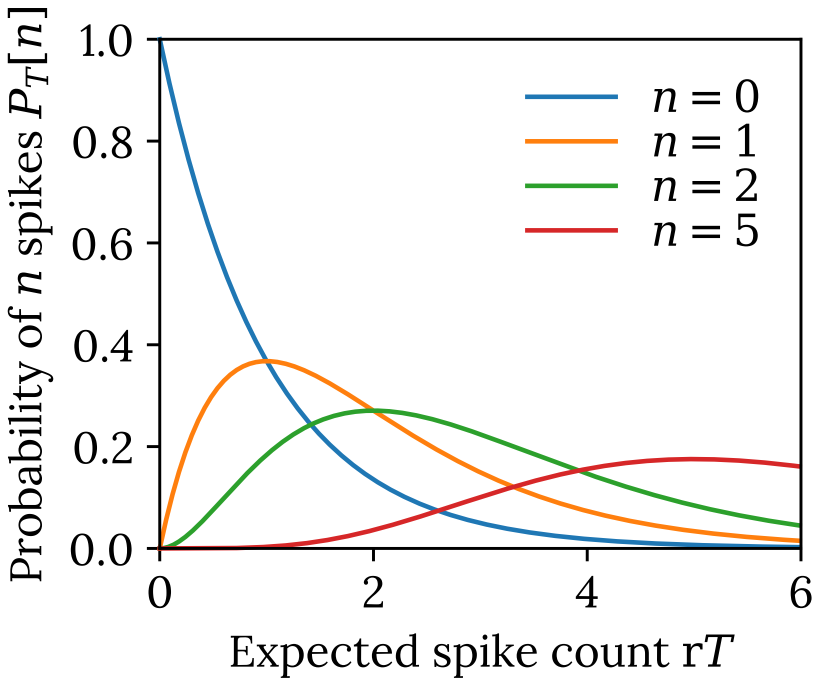
$$Q_{ss}(d) \equiv \frac{1}{M}\sum_{m=1}^{M} s_m s_{m+d} = \begin{cases} \sigma^2/\Delta t, & \text{if } \ d = 0 \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

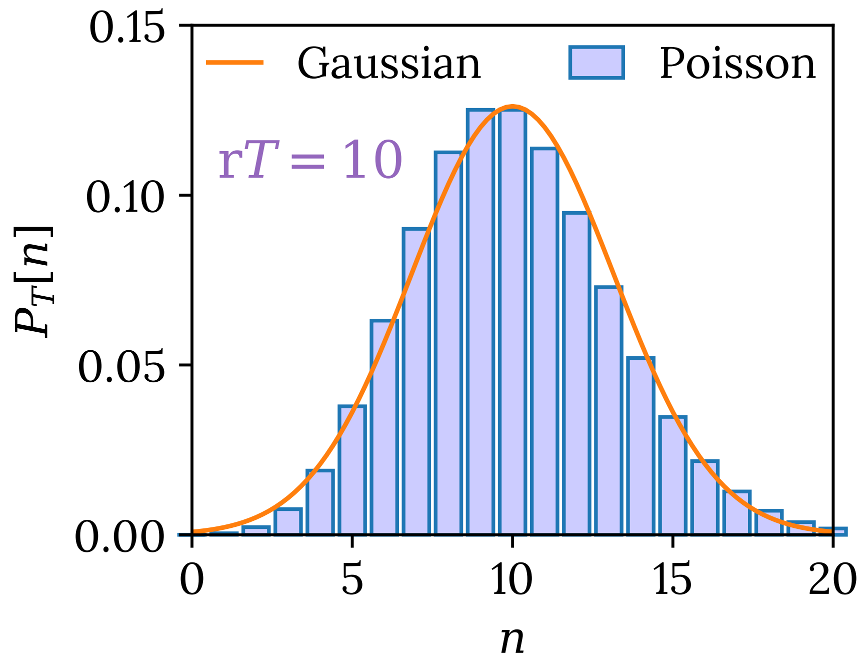
The noise amplitude should be adjust with the size of the time step as $1/\sqrt{\Delta t}$. This can be understand by considering how a discrete random walk can approach a continuous limit:

$$\frac{dv}{dt} \sim \frac{v_{i+1} - v_i}{\Delta t} = \eta_i, \quad \text{where } |\eta_i| = \sigma. \tag{8}$$

During T, the average drift of v is estimated by $\sigma \Delta t \sqrt{T/\Delta t}$. The scaling of σ as stated above is needed to keep this drift constant while sending $\Delta t \to 0$.

Distributions of Poisson process





- Depends on its mean rT; Approaches Gaussian for large rT.
- Variance $\sigma_n^2 = \langle n^2 \rangle - \langle n \rangle^2$ is the same as mean.
- Fano factor $\sigma_n^2 / \langle n \rangle$ is 1 for homogeneous Poisson process.
- Interspike interval follows exponential $P(\tau) = \mathrm{r} \exp(-\mathrm{r}\tau)$ distribution with mean $\langle \tau \rangle = \mathrm{r}^{-1}$ and variance $\sigma_\tau^2 = \mathrm{r}^{-2}$.

Neural code: Rate or temporal?

For single neuron

- Independent-spike code characterized by $\mathrm{r}(t)$ completely, easy for analysis
- Correlation code exists but commonly less than 10%

For neural population

- Independent-neuron code: each neuron responds indepedently to input stimulus
- Synchronous firing of neurons and Rhythmic oscillations of population are mechanisms for conveying information $\Leftarrow$ Not necessary against independent-neuron assumption...

Temporal codes

- Precise spike timing or high-frequency firing-rate fluctuation are carrying information
- Still in debate with challenge in identifying how relationships between firing patterns of different neurons contribute to neural coding...

Modeling firing of a neuron

Consider how each moment in the history of the stimulus input contributes to the current firing rate of the neuron.

For a linear system, the corresponding weight should be proportional to spike-triggered average of the stimulus.

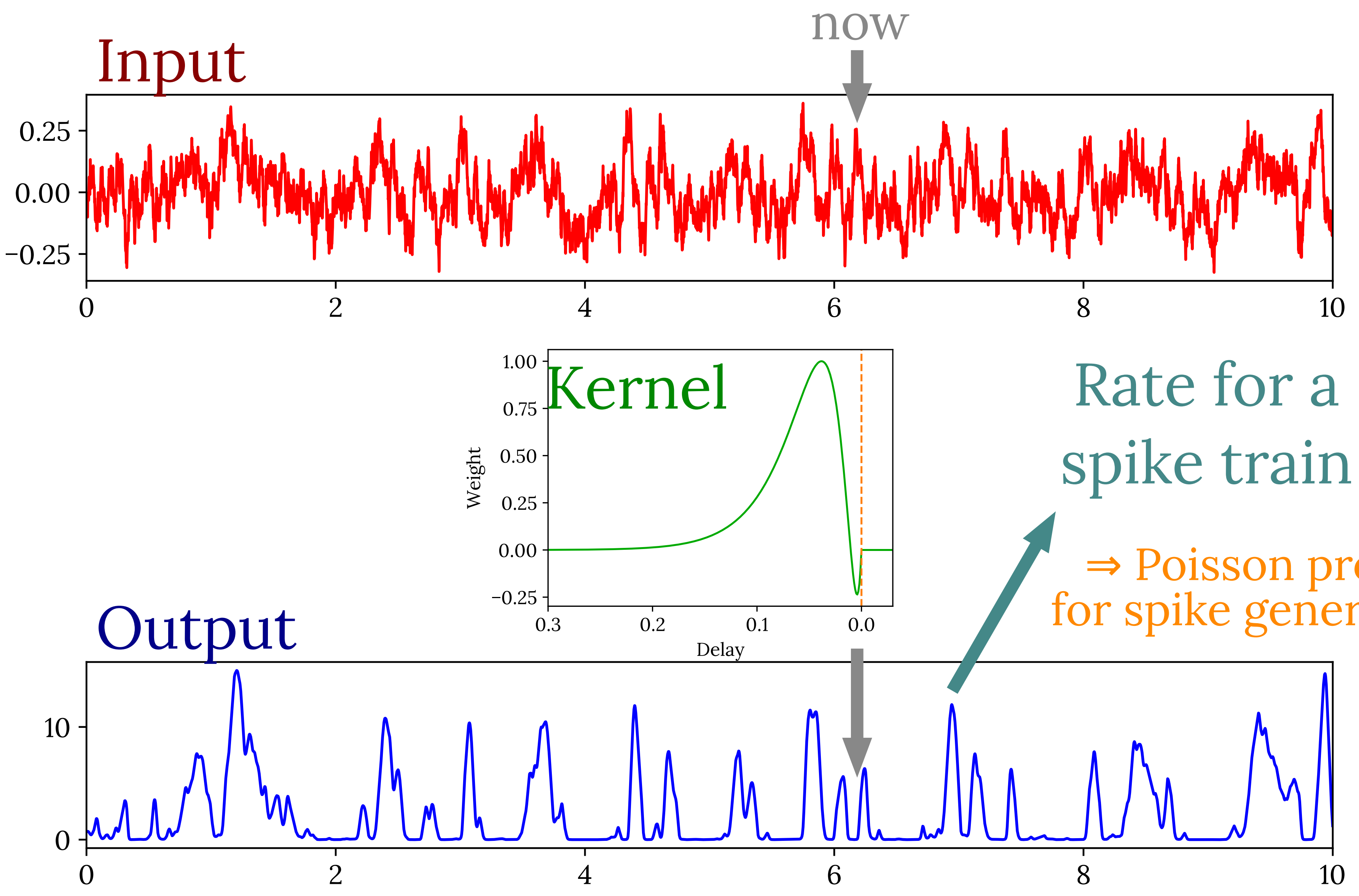
We can thus convolve the STA with the input to estimate how strong the neuron is driven to fire at each moment.

To turn this driving force (which is a signed number) to firing-rate (which is non-negative), we use a nonlinear function, such as, `relu`.

Once we have the firing rate, spiking events can be generated using a inhomogeneous Poisson process

Linear–nonlinear model

– AKA: *Linear–nonlinear–Poisson cascade model* (LNP)



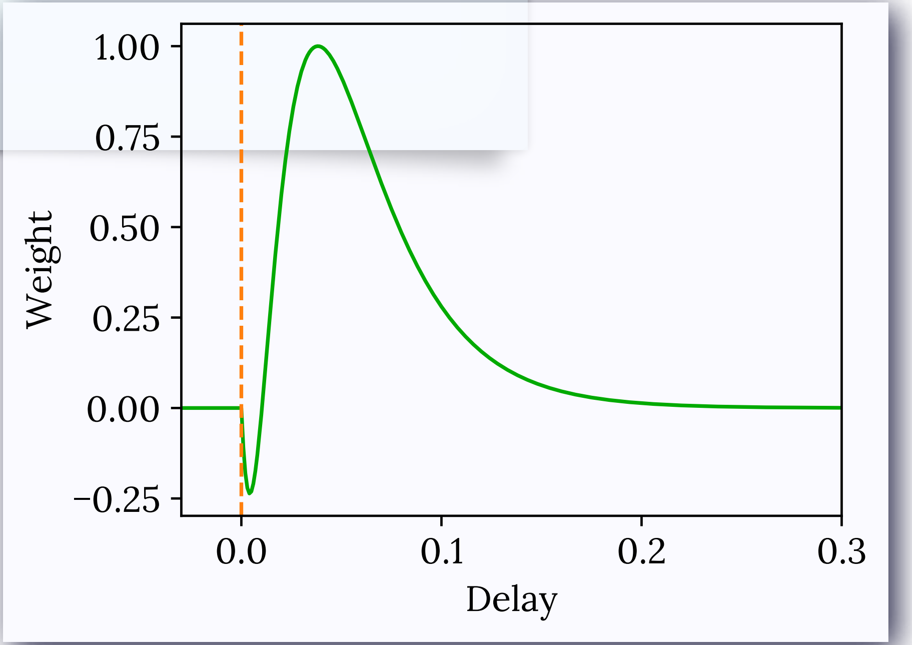
Rate for a spike train

⇒ Poisson process for spike generation...

Implementing LNP: the made-up kernel

```
kktr = np.array([-0.03,0.3]) # Kernel time range
kts = np.arange(*ktr,dt) # Kernel time frames
# Make up a nontrivil kernel
knl = np.exp(-kts/.032)-1.8*np.exp(-kts/.016)+0.8*np.exp(-kts/.008)
knl *= (kts>0)  # Make negative delay part zero (causal)
knl /= max(knl) # Normalize max to 1

plt.plot(kts,knl,color='#0a0')
plt.xlim(*ktr)
plt.vlines([0],[min(knl)],[max(knl)],ls='--',color='C1')
plt.ylabel('Weight')
plt.xlabel('Delay')
save_plot('lnp_kernel')
plt.show()
```

When convolving with the input, beginning part gets multiplied first. Since it is before $t = 0$, we will drop the frames until $t = 0$ hits the multiplication zone.

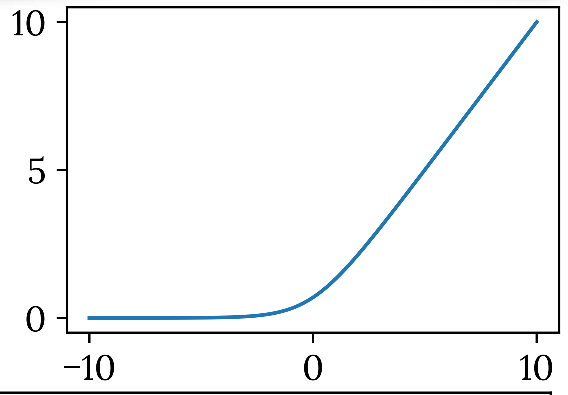


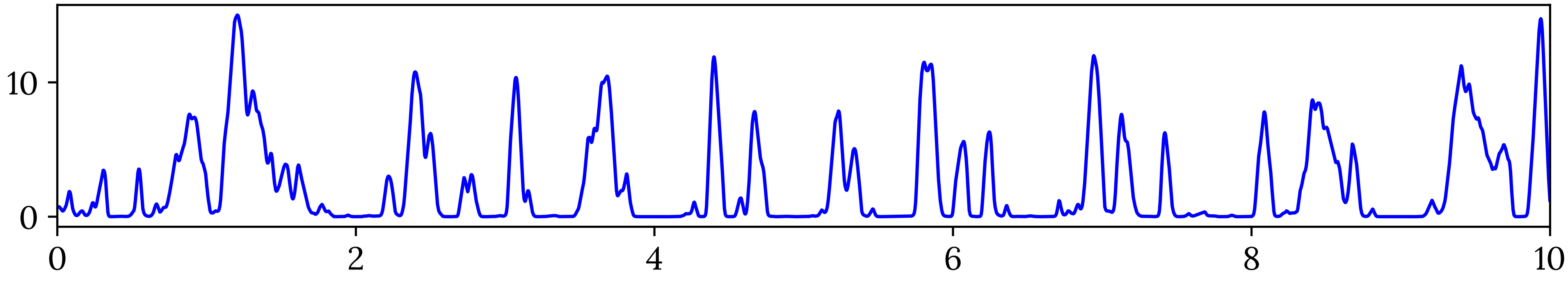
Implementing LNP: convolution & rectification

```
# Linear filter and nonlinear rectification...
fd = (kts<0).sum() # Number of frames to drop in front
rect = lambda x:np.log(1+np.exp(x)) # Some rectifying function
mu = rect(np.convolve(ii,knl)[fd:fd-len(knl)+1])
```

```
# Show rectification function
x = np.linspace(-10,10,101)
plt.figure(figsize=(3,2))
plt.plot(x,rect(x))
plt.show()
# Show resulting firing rate
plt.figure(figsize=(10,1.5))
plt.plot(ts,mu,color='#00f')
plt.xlim(0,n*dt)
plt.show()
```

After convolving kernel with the stimulus, we get an array of some influence strength of the input on the neuron. A non-linear rectifying function or some bias can be added to transform the influence into a non-negative firing rate.

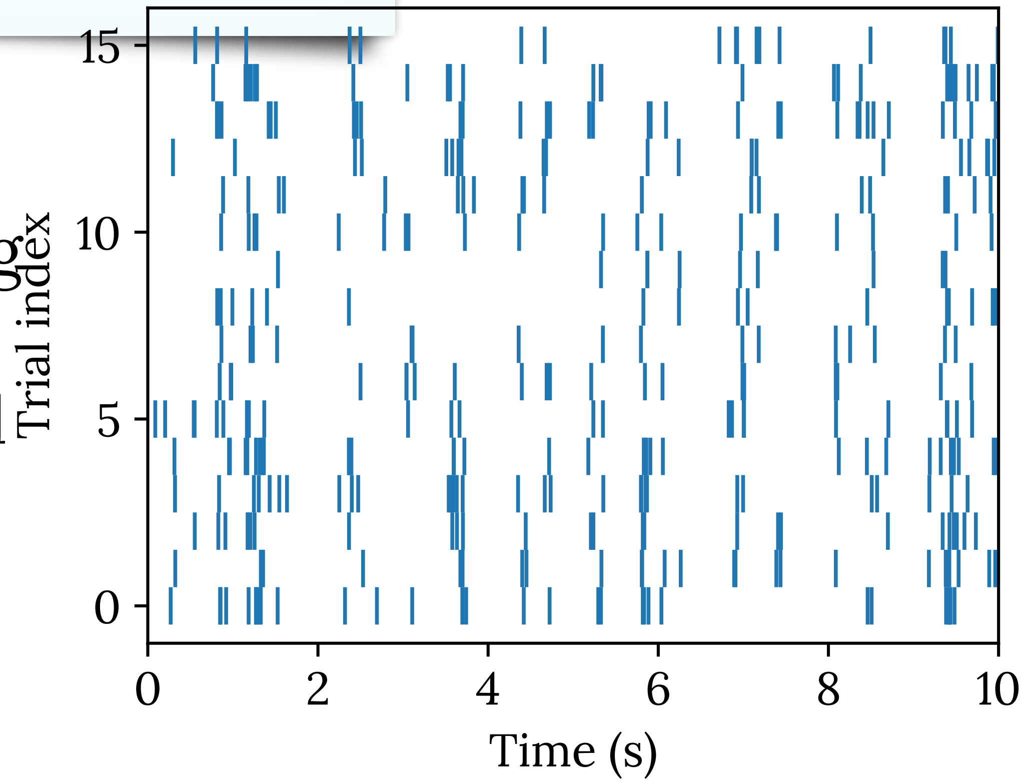




Implementing LNP: *spike generation*

```
ntrial = 16
for sd in range(ntrial):
    rng = np.random.default_rng(sd)
    st = rng.uniform(size=mu.shape)<mu*dt
    plt.vlines(np.where(st)[0]*dt,sd,sd+1,lw=1)
plt.xlim(tr)
plt.ylim(0,ntrial)
plt.show()
```

The above is one of the quickest ways of implementing the Poisson process. But, the spike-time precision is limited by `dt` and requires good precision of random number when `mu*dt` is small.


Trial index
Time (s)