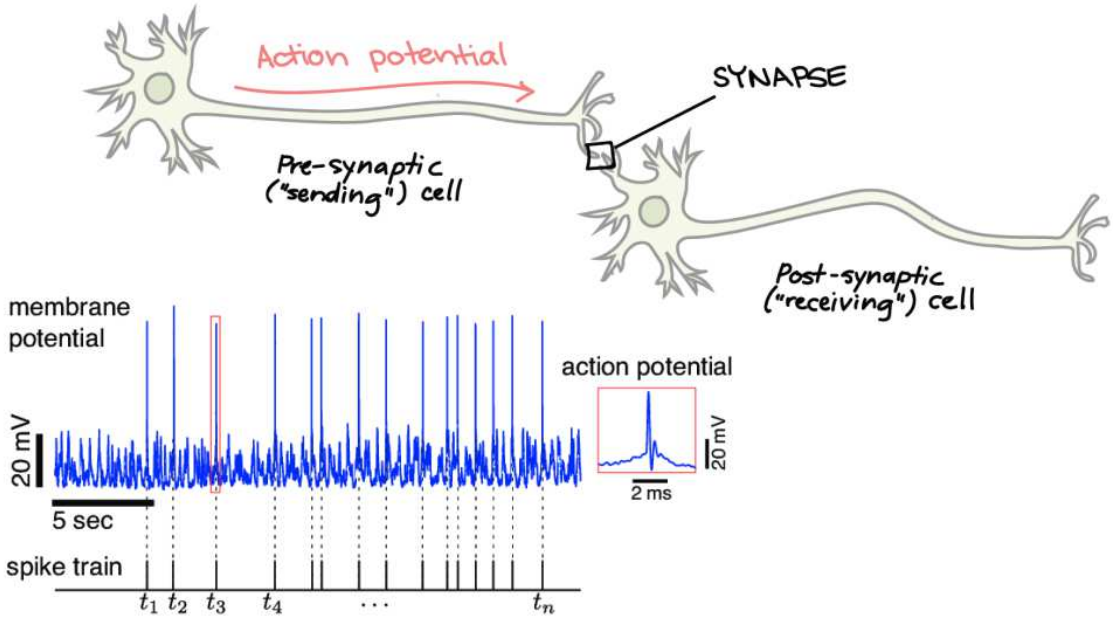
W0D1-파이썬 튜토리얼 : LIF 뉴런 1

순천향대학교 컴퓨터공학과

이 상 정

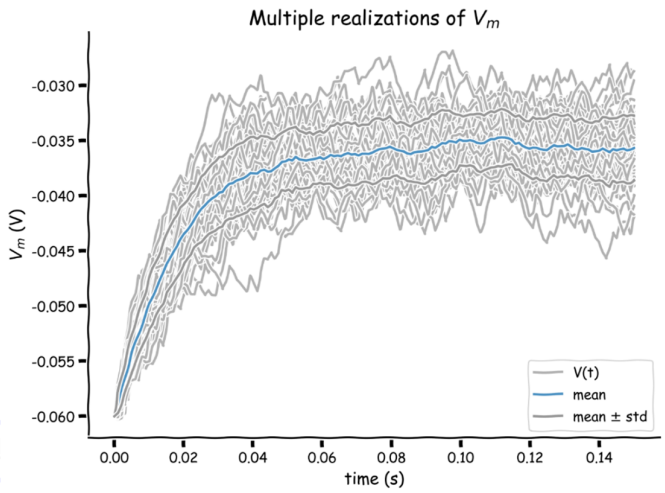
뉴런의 스파이크 동작

- 뉴런의 막전위(membrane potential)가 임계치를 넘어서면 스파이크 발생
  - 다른 뉴런으로부터 활동전위(action potential)를 받아 막전위가 상승



파이썬 튜토리얼 소개

- 파이썬(Python)과 넘파이(NumPy)를 사용하여 계산 신경과학(computational neuroscience)을 학습
- LIF(Leaky Integrate-and-Fire) 뉴런의 파이썬 구현 예
  - 스파이크 동작은 제외
  - 시간에 따른 막전위(membrane potential)의 변화를 시각화
  - 통계적 특성(평균, 표준편차)을 계산



W0D1-파이썬 튜토리얼: LIF 뉴런 1

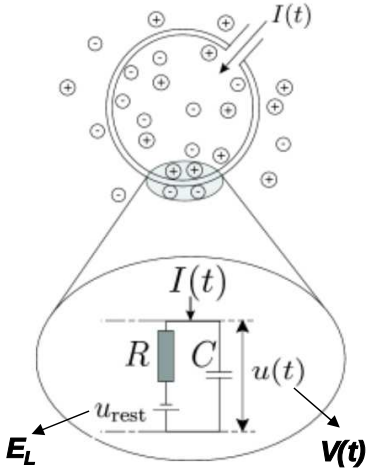
LIF 모델

- LIF 모델 방정식
  - 상미분 방정식(ordinary differential equation)
  - 시냅스 입력과 세포막의 전하 누수에 따른 막전위의 변화를 표현

$$I(t) = C\,dv/dt + (V-E_L)/R$$

$$\tau_m \frac{d}{dt} V(t) = E_L - V(t) + R\,I(t) \qquad \text{if} \quad V(t) \le V_{th}$$

$$V(t) = V_{reset} \qquad \text{otherwise}$$



- V(t), V_m : 막전위
- τ_m : 막시간상수(membrane time constant), RC
- E_L : 누수전위(leaky potential)
- R : 막저항 (membrane resistance)
- I(t) : 시냅스 입력전류
- V_{th} : 점화 임계치(firing threshold)
- V_{reset} : 리셋전위

파이썬 코드

코드: 시뮬레이션 파라미터

❑ LIF 뉴런 시뮬레이션 파라미터

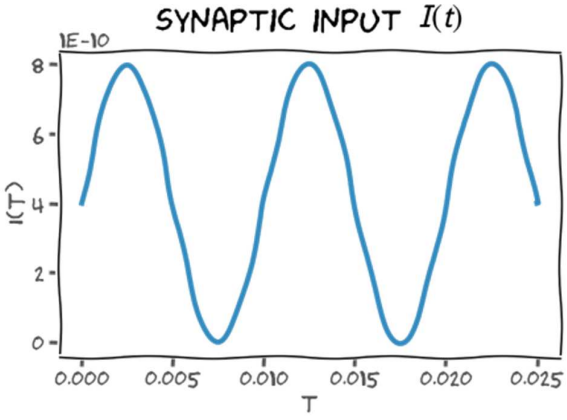
```
## Coding Exercise 1: Defining parameters
# LIF 뉴런 파라미터 정의
t_max = 150e-3          # second
dt = 1e-3               # second
tau = 20e-3             # second
el = -60e-3             # milivolt
vr = -70e-3             # milivolt
vth = -50e-3            # milivolt
r = 100e6               # ohm
i_mean = 25e-11         # ampere

print(t_max, dt, tau, el, vr, vth, r, i_mean)
```

```
0.15 0.001 0.02 -0.06 -0.07 -0.05 100000000.0 2.5e-10
```

입력전류

- 사인 함수로 시냅스 입력전류 모델링
  - t=0 에서 0.009까지 시뮬레이션
  - 간격 $\Delta t = 0.001$

$$I(t) = I_{mean}\left(1 + \sin\left(\frac{2\pi}{0.01}t\right)\right)$$



막전위 계산

- 오일러 방법(Euler's Method)으로 미분방정식 계산

$$\tau_m \frac{d}{dt} V(t) = E_L - V(t) + R I(t)$$

$$\tau_m \frac{V(t+\Delta t) - V(t)}{\Delta t} = E_L - V(t) + R I(t)$$

$$V(t+\Delta t) = V(t) + \frac{\Delta t}{\tau_m}(E_l - V(t) + RI(t))$$

코드: 막전위 계산

```
## Coding Exercise 4: 막전위 시뮬레이션
# 초기화
step_end = 10     # 스텝 횟수
v = el            # 막 전위

# 스텝 횟수 만큼 반복
for step in range(step_end):
  # 시간 t 계산
  t = step * dt

  # 시냅스 입력전류 계산
  i = i_mean * (1 + np.sin((t * 2 * np.pi) / 0.01))

  # 막전위 계산
  v = v + dt/tau * (el - v + r*i)

  # 시간, 입력전류, 막전위 출력
  print(f"{t:.3f} {i:.4e}  {v:.4e}")
```

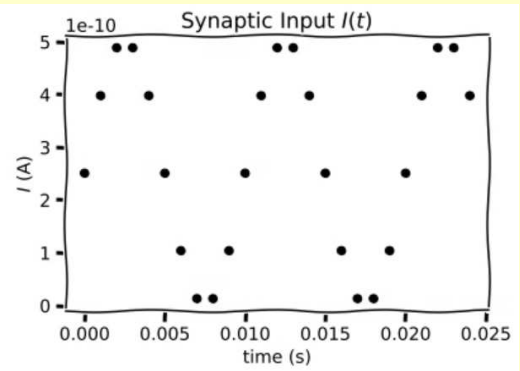
$$I(t) = I_{mean}\left(1 + \sin\left(\frac{2\pi}{0.01}t\right)\right)$$

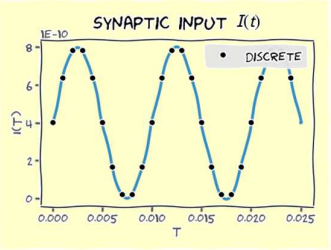
$$V(t+\Delta t) = V(t) + \frac{\Delta t}{\tau_m}(E_l - V(t) + RI(t))$$

```
0.000 2.5000e-10  -5.8750e-02
0.001 3.9695e-10  -5.6828e-02
0.002 4.8776e-10  -5.4548e-02
0.003 4.8776e-10  -5.2381e-02
0.004 3.9695e-10  -5.0778e-02
0.005 2.5000e-10  -4.9989e-02
0.006 1.0305e-10  -4.9974e-02
0.007 1.2236e-11  -5.0414e-02
0.008 1.2236e-11  -5.0832e-02
0.009 1.0305e-10  -5.0775e-02
```

코드: 입력전류 그래프

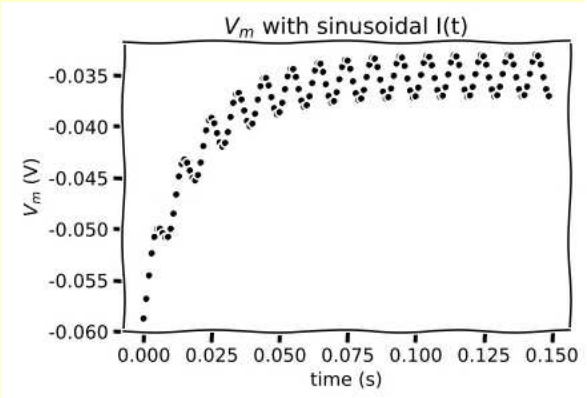
```
## Coding Exercise 5: 입력전류 그래프
# 스텝 횟수 초기화
step_end = 25

with plt.xkcd():     # xkcd (과학만화) 스타일 그래프
  # 그래프 초기화
  plt.figure()                          # 그래프 생성
  plt.title('Synaptic Input $I(t)$')    # 타이틀, 이탤릭체
  plt.xlabel('time (s)')                # x축 라벨
  plt.ylabel('$I$ (A)')                 # y축 라벨, 이탤릭체

  # 스텝 횟수만큼 반복
  for step in range(step_end):
    # 시간 t 계산
    t = step * dt

    # 시냅스 입력전류 계산
    i = i_mean * (1 + np.sin((t * 2 * np.pi) / 0.01))
  # 전류 i 그래프 그리기 (use 'ko' to get small black dots (short for color='k' and marker = 'o'))
  plt.plot(t, i, 'ko')

  # 화면에 그래프 표시
  plt.show()
```





코드: 막전위 그래프

```
## Coding Exercise 6: 막전위 그래프
# 스텝 횟수 초기화
step_end = int(t_max / dt)
# v0 초기화, 누수전위(leaky potential)
v = el
with plt.xkcd():            # xkcd (과학만화) 스타일 그래프
    # 그래프 초기화
    plt.figure()                          # 그래프 생성
    plt.title('$V_m$ with sinusoidal I(t)') # 타이틀, 이탤릭체
    plt.xlabel('time (s)')                # x축 라벨
    plt.ylabel('$V_m$ (V)');              # y축 라벨, 이탤릭체
    # 스텝 횟수만큼 반복
    for step in range(step_end):
        # 시간 t 계산
        t = step * dt
        # 시냅스 입력전류 계산
        i = i_mean * (1 + np.sin((t * 2 * np.pi) / 0.01))
        # 막전위 v 계산
        v = v + dt/tau * (el - v + r*i)
        # 막전위 v 그래프 그리기 (using 'k.' to get even smaller markers)
        plt.plot(t, v, 'k.')
    # 화면에 그래프 표시
    plt.show()
```



랜덤 시냅스 입력전류

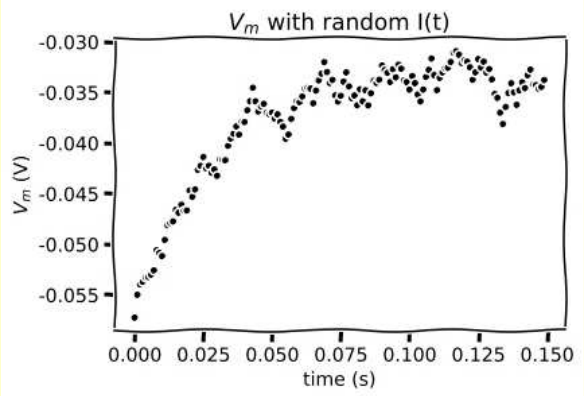
- 실제 뉴런에서 시냅스 입력전류는 랜덤(또는 확률적(stochastic))
- 입력 전류에 랜덤성(randomness) 추가

$$I(t) = I_{mean}\left(1 + 0.1\sqrt{\frac{t_{max}}{\Delta t}}\,\xi(t)\right) \qquad \text{with } \xi(t) \sim \mathcal{U}(-1, 1)$$

  - u(-1, 1)은 $x \in [-1, 1]$ 인 균등 분포(uniform distribution)

코드: 랜덤성이 추가된 막전위 그래프

```
## Coding Exercise 7: 랜덤성(randomness) 추가
# 난수 시드 설정
np.random.seed(2020)
# 스텝 횟수, v0 초기화
step_end = int(t_max / dt)
v = el
with plt.xkcd():              # xkcd (과학만화) 스타일 그래프
    plt.figure()                          # 그래프 생성
    plt.title('$V_m$ with random I(t)')   # 타이틀, 이탤릭체
    plt.xlabel('time (s)')                # x축 라벨
    plt.ylabel('$V_m$ (V)');              # y축 라벨, 이탤릭체
    # 스텝 횟수만큼 반복
    for step in range(step_end):
        t = step * dt        # 시간 t 계산
        # -1에서 1사이의 난수 생성
        random_num = 2 * np.random.random() - 1
        # 난수가 포함된 시냅스 입력전류로 막전위 계산
        i =  i_mean * (1 + 0.1 * (t_max / dt)**(0.5) * random_num)   # 입력전류
        v = v + dt/tau * (el - v + r*i)                              # 막전위
        # 막전위 v 그래프 그리기 (using 'k.' to get even smaller markers)
        plt.plot(t, v, 'k.')
    # 화면에 그래프 표시
    plt.show()
```



다중 수행 통계

- 동일 조건 하에 여러 번 수행하여 막전위 통계값 계산
  - 앙상블 통계 (Ensemble Statistics)
  - 동일 시간에 50번 반복 수행하여 리스트에 저장
  - 평균(mean), 분산(variance), 표준편차(standard deviation) 계산

$$\langle V(t) \rangle = \frac{1}{N} \sum_{n=1}^{N} V_n(t)$$

$$\mathrm{Var}(t) = \frac{1}{N-1} \sum_{n=1}^{N} \left(V_n(t) - \langle V(t) \rangle \right)^2$$

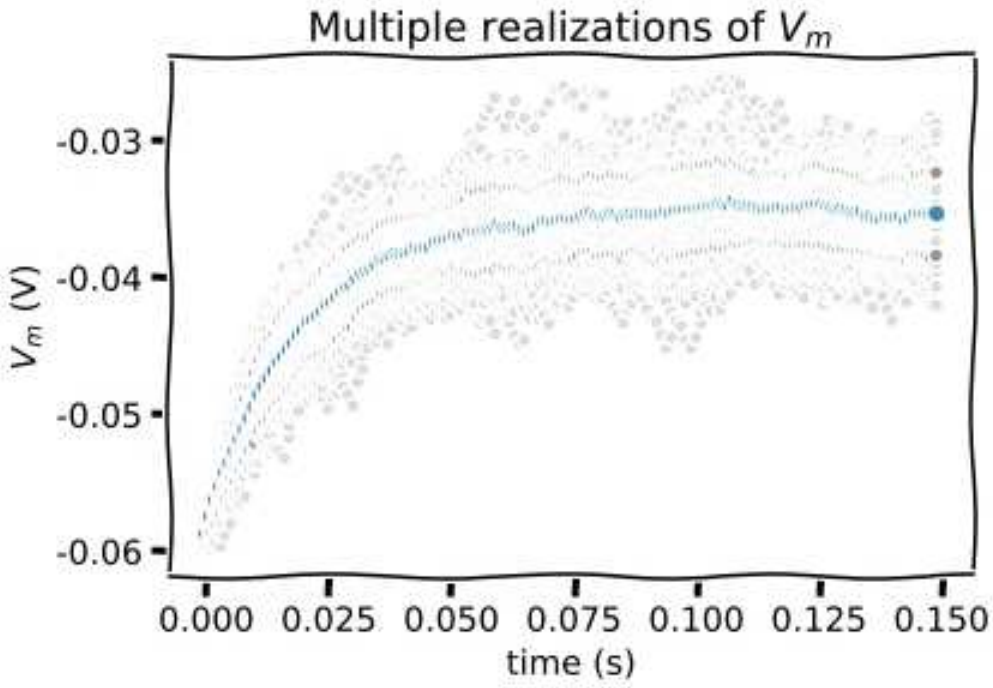
$$\sigma(t) \equiv \sqrt{\mathrm{Var}\,(t)}$$

코드: 다중 수행 통계 (평균, 표준편차)

```
## Coding Exercise 10: 다중 수행 통계 (평균, 표준편차)
# 난수 시드 설정
np.random.seed(2020)
# 스텝 횟수, 실행 횟수 초기화
step_end = int(t_max / dt)
n=50
# 리스트에 50개 막전위 초기화
v_n = [el] * n

with plt.xkcd():              # xkcd (과학만화) 스타일 그래프
    plt.figure()                                    # 그래프 생성
    plt.title('Multiple realizations of $V_m$')     # 타이틀, 이탤릭체
    plt.xlabel('time (s)')                          # x축 라벨
    plt.ylabel('$V_m$ (V)');                        # y축 라벨, 이탤릭체

    # 스텝 횟수만큼 반복
    for step in range(step_end):
        t = step * dt        # 시간 t 계산
```

```
    # 각 시간에 대해 n번 반복
    for j in range(0, n):
        # 난수가 포함된 시냅스 입력전류로 막전위 계산
        i = i_mean * (1 + 0.1 * (t_max/dt)**(0.5) * (2* np.random.random() - 1))
        # 막전위 계산하여 리스트에 추가
        v_n[j] = v_n[j] + (dt / tau) * (el - v_n[j] + r*i)
    # 평균 계산
    v_mean = sum(v_n) / n
    # 표준편차 계산
    v_var_n = [(v - v_mean)**2 for v in v_n]    # n번 반복 계산 후 리스트 생성
    v_var = sum(v_var_n) / (n - 1)              # 분산
    v_std = np.sqrt(v_var)          # 표준편차

    # 막전위 n개 그래프 그리기
    plt.plot(n*[t], v_n, 'k.', alpha=0.1)
    # 평균 그래프 그리기 (using alpha=0.8 and'C0.' for blue)
    plt.plot(t, v_mean, 'C0.', alpha=0.8, markersize=10)
    # 평균+표준편차 그래프 그리기 (with alpha=0.8 and argument 'C7')
    plt.plot(t, v_mean + v_std, 'C7.', alpha=0.8)
    # 평균-표준편차 그래프 그리기 (with alpha=0.8 and argument 'C7')
    plt.plot(t, v_mean - v_std, 'C7.', alpha=0.8)
# 화면에 그래프 표시
plt.show()
```

실행 결과: 다중 수행 통계 (평균, 표준편차)



넘파이 코드

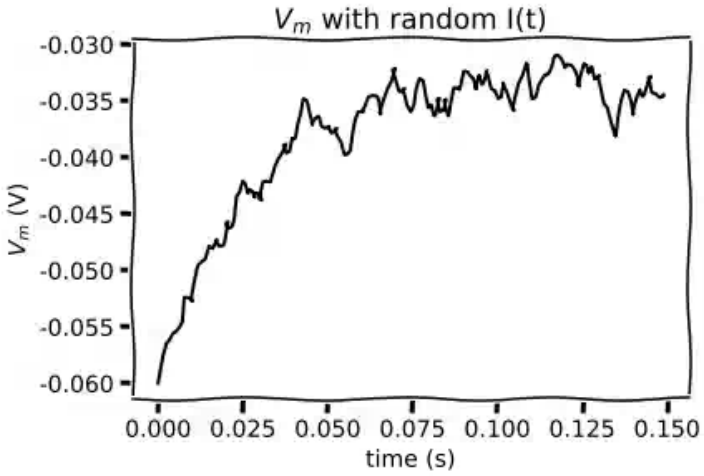
넘파이: 막전위 계산

- **넘파이 1차원 배열**을 사용하여 시간의 구간, 입력전류를 한 번에 계산
  - 스텝횟수 step_end
    step_end = int(t_max / dt) - 1
  - np.linspace() 함수를 사용하여 시간 구간을 1차원 배열에 저장
    t_range = np.linspace(0, t_max, num=step_end, endpoint=False)
  - 스텝횟수 크기의 입력전류 배열 계산
    i = i_mean * (1 + 0.1 * (t_max/dt) ** (0.5) * (2 * np.random.random(step_end) - 1))
  - 각 입력전류의 배열 원소에 대해 막전위 계산 반복
    for step, i_step in enumerate(i):
    ……
    v[step] = v[step - 1] + (dt / tau) * (el - v[step - 1] + r * i_step)

코드: 넘파이 막전위 그래프

```
## Coding Exercise 12: 넘파이로 재작성한 막전위 계산
np.random.seed(2020)          # 난수 시드 설정

# 스텝횟수 step_end, 시간 구간 배열 t_range, 막전위 배열 v 초기화
step_end = int(t_max / dt) - 1
t_range = np.linspace(0, t_max, num=step_end, endpoint=False)
                                    # 0에서 tmax까지 step_end 개의 1차원 시간 배열
v = el * np.ones(step_end)          # step_end개의 1차원배열을 el로 초기화
# 스텝횟수 크기의 입력전류 1차원 배열 계산
i = i_mean * (1 + 0.1 * (t_max/dt) ** (0.5) * (2 * np.random.random(step_end) - 1))

# 각 입력전류의 배열 원소에 대해 반복
for step, i_step in enumerate(i):
  # 첫번째 원소 스킵
  if step==0:
    continue
  # 각 스텝의 막전위 계산
  v[step] = v[step - 1] + (dt / tau) * (el - v[step - 1] + r * i_step)
```

```
# 그래프 그리기
with plt.xkcd():                    # xkcd (과학만화) 스타일 그래프
  plt.figure()                      # 그래프 생성
  plt.title('$V_m$ with random I(t)')  # 타이틀
  plt.xlabel('time (s)')            # x축 라벨
  plt.ylabel('$V_m$ (V)');          # y축 라벨

  plt.plot(t_range, v, 'k')         # 막전위 그래프 그리기
  plt.show()                        # 화면에 그래프 표시
```



넘파이: 다중 수행 통계

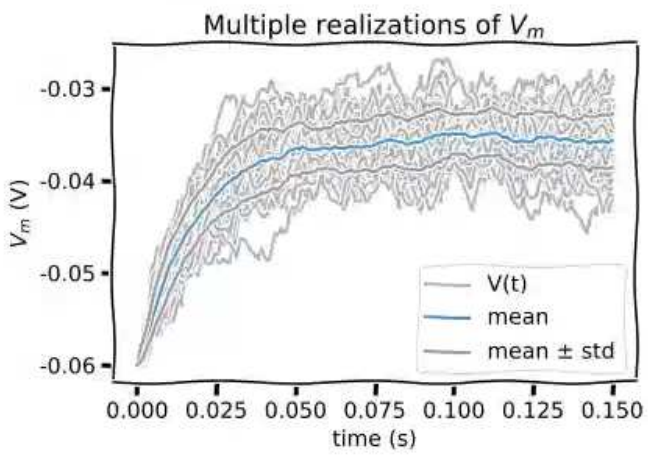
❑ 2차원 배열로 다중 수행 막전위 표현

- n x stop_end 2차원 배열
  - n: 다중 수행 횟수, stop_end: 시간 스텝 횟수
- n x step_end개의 2차원배열을 el로 초기화
  v_n = el* np.ones([n, step_end])
- n x step_end 크기의 2차원 배열 입력전류 계산
  i = i_mean * (1 + 0.1 * (t_max / dt)**(0.5) * (2 * np.random.random([n, step_end]) - 1))
- 입력전류 모든 행의 각 열 값에 대해 막전위 계산
  v_n[:, step] = v_n[:, step - 1] + (dt / tau) * (el - v_n[:, step - 1] + r * i[:, step])
- 각 행(수행)들의 원소(막전위)에 대한 평균
  v_mean = np.mean(v_n, axis=0)
- 각 행(수행)들의 원소(막전위)에 대한 표준편차
  v_std = np.std(v_n, axis=0)

코드: 넘파이 다중 수행 통계 그래프

```
## Coding Exercise 13: 넘파이 다중 수행 통계
np.random.seed(2020)          # 난수 시드 설정

# 스텝횟수 step_end, 시간 구간 배열 t_range, 막전위 배열 v 초기화
step_end = int(t_max / dt)
t_range = np.linspace(0, t_max, num=step_end)    # 0에서 tmax까지 step_end 개의 1차원 시간 배열
v_n = el* np.ones([n, step_end])                 # n x step_end개의 2차원배열을 el로 초기화

# n x step_end 크기의 2차원 배열 입력전류 계산
i = i_mean * (1 + 0.1 * (t_max / dt)**(0.5) * (2 * np.random.random([n, step_end]) - 1))

# 입력전류 모든 행의 각 열의 배열 원소에 대해 반복
for step in range(1, step_end):
    # 모든 행(수행)의 step 열 막전위 계산
    v_n[:, step] = v_n[:, step - 1] + (dt / tau) * (el - v_n[:, step - 1] + r * i[:, step])

# 각 행(수행)들의 원소에 대한 평균
v_mean = np.mean(v_n, axis=0)
# 각 행(수행)의 원소에 대한 표준편차
v_std = np.std(v_n, axis=0)
```

```
W # 그래프 그리기
  with plt.xkcd():                    # xkcd (과학만화) 스타일 그래프
    plt.figure()                      # 그래프 생성
    plt.title('Multiple realizations of $V_m$')
    plt.xlabel('time (s)')
    plt.ylabel('$V_m$ (V)')

    plt.plot(t_range, v_n.T, 'k', alpha=0.3)
                # 다중 수행, 막전위 그래프 v_n의 전치행렬(각 시간에 대한 다중 수행 값들)로 변환
  plt.plot(t_range, v_n[-1], 'k', alpha=0.3, label='V(t)')      # 마지막 수행 막전위, 범례의 라벨 표시
  plt.plot(t_range, v_mean, 'C0', alpha=0.8, label='mean')      # 평균
  plt.plot(t_range, v_mean+v_std, 'C7', alpha=0.8)              # 평균+표준편차
  plt.plot(t_range, v_mean-v_std, 'C7', alpha=0.8, label='mean $\pm$ std')    # 평균-표준편차

  plt.legend()    # 범례 표시
  plt.show()
```



참고 사이트

- Python Workshop 1 (W0D1): Tutorial: LIF Neuron Part I
  - https://compneuro.neuromatch.io/tutorials/W0D1_PythonWorkshop1/student/W0D1_Tutorial1.html

- *Neuronal Dynamics online book*
  - *Part I Foundations of Neuronal Dynamics*
    - *https://neuronaldynamics.epfl.ch/online/Pt1.html*