RLC Filter Identification from Input–Output Signals

20221242 박수빈

목차

설계 목적

1. 미지의 필터를 입출력 신호를 통해 판별

2. 1차 RC 필터와 2차 RLC 필터(LP, HP, BP)을 모두 고려하여 비교

3. 최종적으로 필터 종류, 파라미터, 신뢰도 자동으로 산출

기본 이론 - Green's Function과 전달 함수

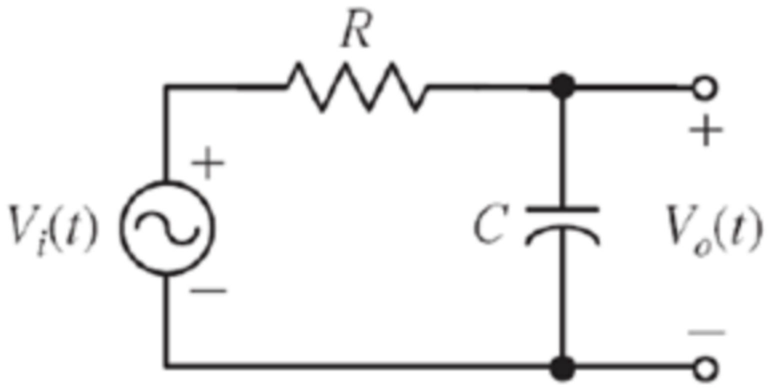
$$V_{out}(t) = \int_0^t K(t-\tau)V_{in}(\tau)d\tau$$

$$\mathcal{F}\{V_{out}(t)\} = \mathcal{F}\{K(t)\}\mathcal{F}\{V_{in}(t)\}$$

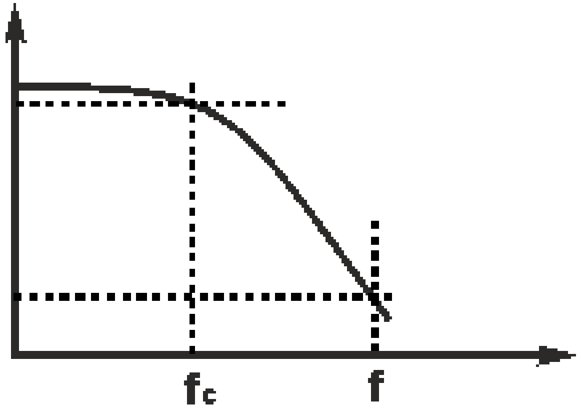
$$\mathcal{F}\{K(t)\} = H(j\omega) = \frac{V_{out}(j\omega)}{V_{in}(j\omega)}$$

기본 이론 - 1차 필터와 2차 필터

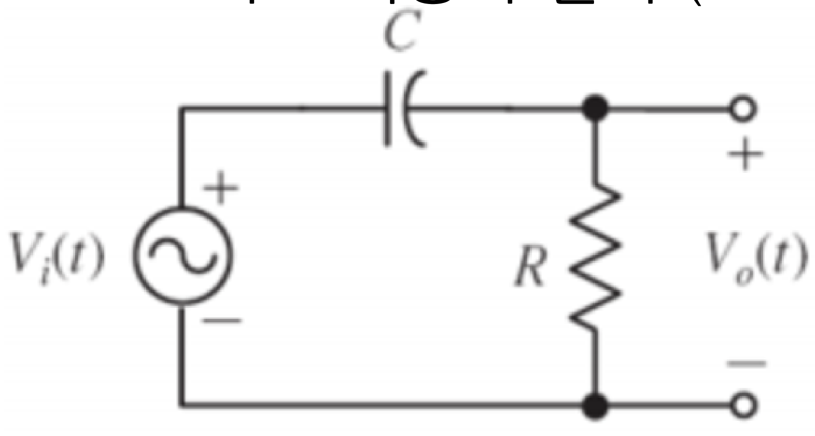
1차 저역통과 필터 (LPF)



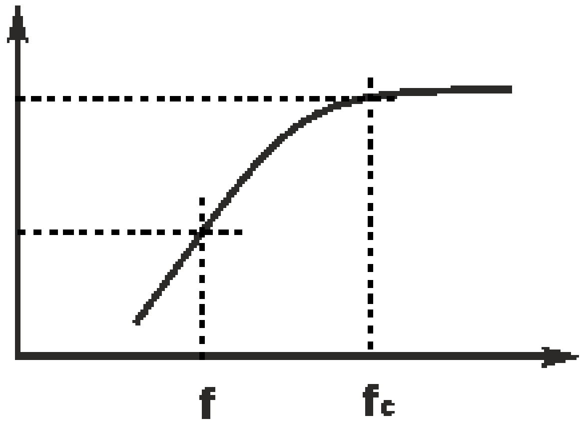
$$H(s) = \frac{1}{sRC + 1}$$



1차 고역통과 필터 (HPF)



$$H(s) = \frac{sRC}{sRC + 1}$$



$$\omega_c = \frac{1}{RC}$$

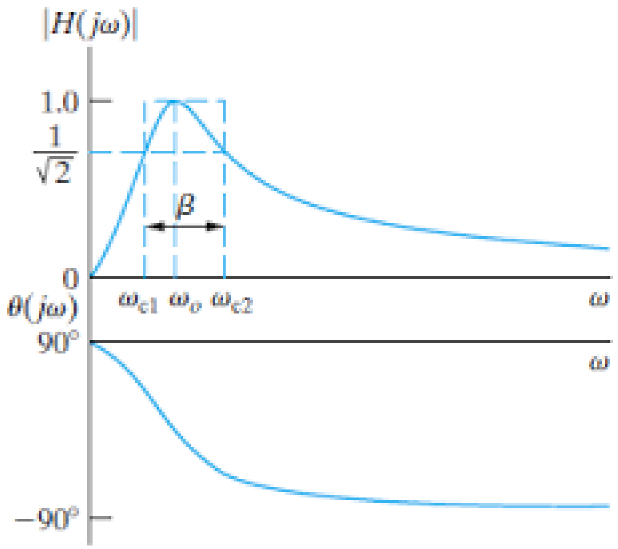
기본 이론 - 1차 필터와 2차 필터

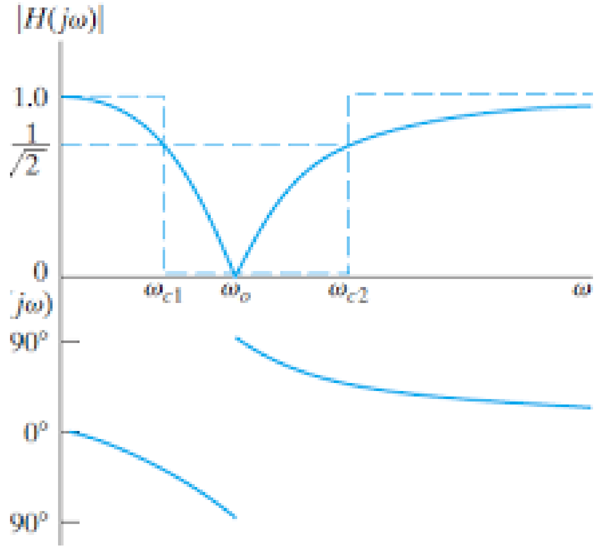
저역통과 필터 (LPF) $H(s) = \dfrac{\omega_0^2}{s^2 + \dfrac{\omega_0}{Q}s + \omega_0^2}$

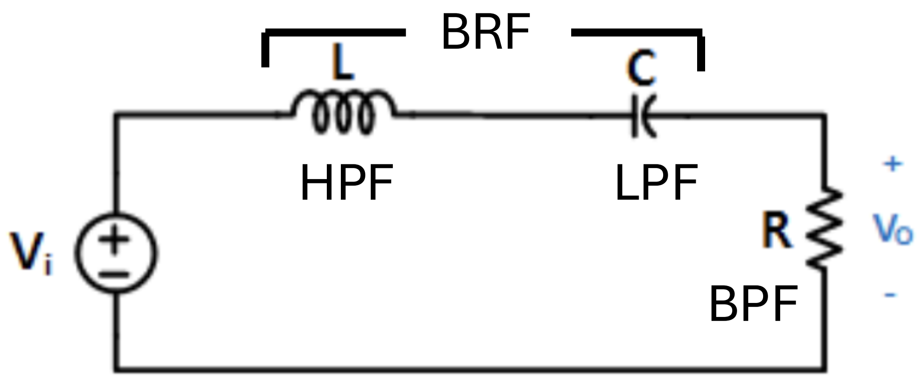
고역통과 필터 (HPF) $H(s) = \dfrac{s^2}{s^2 + \dfrac{\omega_0}{Q}s + \omega_0^2}$

대역통과 필터 (BPF) $H(s) = \dfrac{s\dfrac{\omega_0}{Q}}{s^2 + \dfrac{\omega_0}{Q}s + \omega_0^2}$

대역차단 필터 (BRF) $H(s) = \dfrac{s^2 + \omega_0^2}{s^2 + \dfrac{\omega_0}{Q}s + \omega_0^2}$







$$\omega_0 = \frac{1}{\sqrt{LC}}, \qquad Q = \frac{1}{R}\sqrt{\frac{L}{C}}$$

코드 설명

1. FFT 기반 전달함수 추정, 필터별 전달함수 정의

2. 필터 피팅 함수 정의 및 신뢰도 분석

3. 필터 결정 및 Bode Plot 그리기

4. RLC 매핑

5. 입력 CSV 로드, UI 설정

코드 설명 - FFT 기반 전달함수 추정, 필터별 전달함수

```python
def estimate_transfer_function(x, y, dt, f_max=None,
                               min_input_amp=1e-6, min_H_mag=None):
    """
    x(t), y(t)로부터 FFT 기반 H(ω) = Y(ω)/X(ω) 추정.
    """
    x = np.asarray(x, dtype=float)
    y = np.asarray(y, dtype=float)
    n = len(x)
    if len(y) != n:
        raise ValueError("x와 y의 길이가 같아야 합니다.")

    X = np.fft.rfft(x)
    Y = np.fft.rfft(y)
    freqs = np.fft.rfftfreq(n, dt)
    w = 2 * np.pi * freqs

    H = Y / X

    mask = np.abs(X) > min_input_amp  # 입력 신호가 충분히 큰 구간만 사용
    mask &= freqs > 0

    if f_max is not None:
        mask &= freqs <= f_max

    if min_H_mag is not None:   # H 크기가 너무 작은 구간은 버리기
        mask &= (np.abs(H) > min_H_mag)

    if not np.any(mask):
        raise RuntimeError("유효한 주파수 구간이 없습니다.")

    return freqs[mask], w[mask], H[mask]
```

X, Y에 대해 푸리에 변환,
전달 함수 계산

비교할 필터 전달 함수 정의

```python
# ========================================
# 1·2차 표준형 전달함수
# ========================================

def H_1st_LP(w, K, wc):
    return K / (1 + 1j*w/wc)

def H_1st_HP(w, K, wc):
    return K * (1j*w/wc) / (1 + 1j*w/wc)

def _den(w, w0, Q):
    return (1j*w)**2 + (w0/Q)*(1j*w) + w0**2

def H_2nd_LP(w, K, w0, Q):
    return K * (w0**2) / _den(w, w0, Q)

def H_2nd_HP(w, K, w0, Q):
    return K * ((1j*w)**2) / _den(w, w0, Q)

def H_2nd_BP(w, K, w0, Q):
    return K * ((w0/Q)*(1j*w)) / _den(w, w0, Q)

def H_2nd_BR(w, K, w0, Q):
    return K * ((1j*w)**2 + w0**2) / _den(w, w0, Q)
```

코드 설명 - 필터 피팅 함수 정의, 신뢰도 분석

```python
# ==========================================
# 피팅
# ==========================================

@dataclass
class FitResult:
    name: str
    params: dict
    mse: float

def fit_complex(w, H_meas, func, p0, bounds):
    """
    복소 전달함수 H_meas(w)에 대해
    H_pred(w, p)를 least_squares로 피팅.
    """
    def residual(p):
        H_pred = func(w, *p)
        diff = H_pred - H_meas
        return np.concatenate([diff.real, diff.imag])

    res = least_squares(residual, p0, bounds=bounds)
    p = res.x
    H_pred = func(w, *p)
    mse = np.mean(np.abs(H_pred - H_meas)**2)
    return p, mse
```

피팅 결과 담을 클래스 정의

예상 모델 정의

복소 오차를 실수 벡터로 반환

측정된 모델과 차이가 최소가 되는 파라미터

mse: 예측 값과 실제 값의 차이

```python
def analyze_confidence(models, ambiguous_ratio=0.1):
    """
    models: dict[str, FitResult]
    반환:
      { "best": best_result,
        "top3": [FitResult, ...],
        "rel_gap_12": 상대 차이,
        "confidence": 0~1,
        "ambiguous": True/False }
    """
    results = sorted(models.values(), key=lambda r: r.mse)
    best = results[0]
    top3 = results[:3]

    mse_best = best.mse
    if len(results) > 1:
        mse_second = results[1].mse
        rel_gap_12 = (mse_second - mse_best) / (mse_best + 1e-12)
    else:
        mse_second = None
        rel_gap_12 = float("inf")

    raw_conf = max(0.0, min(1.0, rel_gap_12 / 0.5))  # 50% 차이면 ~1
    ambiguous = (rel_gap_12 < ambiguous_ratio)

    return {
        "best": best,
        "top3": top3,
        "rel_gap_12": rel_gap_12,
        "confidence": raw_conf,
        "ambiguous": ambiguous,
    }
```

mse 작은 순으로 필터 정렬

1, 2위 모델 간 mse 차이 계산, 신뢰도 계산
신뢰도 적으면 ambiguous = 1 반환

코드 설명 - 필터 결정

```python
def identify_order12_filter(x, y, dt, f_max=None, show_bode=True):
    """
    x(t) → 시스템 → y(t) 에 대해
    1차/2차 표준형 필터(LP, HP, BP, BR) 중 하나로 피팅.
    """
    freqs, w, H_meas = estimate_transfer_function(x, y, dt, f_max=f_max,
                                                  min_input_amp=1e-6, min_H_mag=1e-3
    )
    mag = np.abs(H_meas)
    K0 = np.median(mag) if np.median(mag) > 0 else 1.0
    w_mid = np.median(w)
    wc0 = w_mid if w_mid > 0 else 1e3
    idx_pk = np.argmax(mag)
    w0_0 = w[idx_pk] if w[idx_pk] > 0 else 1e3
    Q0 = 1.0

    models = {}
```

```python
    # 1st models
    p0 = [K0, wc0]
    b1 = ([0, 1e-3], [1e3*K0, 1e9])
    # 1st LP
    p, mse = fit_complex(w, H_meas, H_1st_LP, p0, b1)
    models["1st_LP"] = FitResult(
        "1st_LP",
        {"K": float(p[0]), "wc": float(p[1])},
        float(mse)
    )
    # 1st HP
    p, mse = fit_complex(w, H_meas, H_1st_HP, p0, b1)
    models["1st_HP"] = FitResult(
        "1st_HP",
        {"K": float(p[0]), "wc": float(p[1])},
        float(mse)
    )
```

```python
# 2nd models
b2 = ([0, 1e-3, 0.1], [1e3*K0, 1e9, 100])
for name, func in {
    "2nd_LP": H_2nd_LP,
    "2nd_HP": H_2nd_HP,
    "2nd_BP": H_2nd_BP,
    "2nd_BR": H_2nd_BR,
}.items():
    p0 = [K0, w0_0, Q0]
    p, mse = fit_complex(w, H_meas, func, p0, b2)
    models[name] = FitResult(
        name,
        {"K": float(p[0]), "w0": float(p[1]), "Q": float(p[2])},
        float(mse)
    )
```

```python
best = min(models.values(), key=lambda r: r.mse)
conf_info = analyze_confidence(models, ambiguous_ratio=0.1)
```

필터 피팅 함수를 모든 모델에 적용
mse 작은 최적 모델 찾기

실행 결과 - 입력 위젯

RLC 필터 식별기

Mode: Wave (테스트용 합성)

fs [Hz]: 500000

CSV 모드에서 사용할 파일 이름

x CSV: x_input.csv

y CSV: y_output.csv

Wave 모드: x 설정

x type: sin

x A: 1

x f[Hz]: 500

x phase: 0

x t0: 0

Wave 모드: y 설정

y type: sin

y A: 0.6

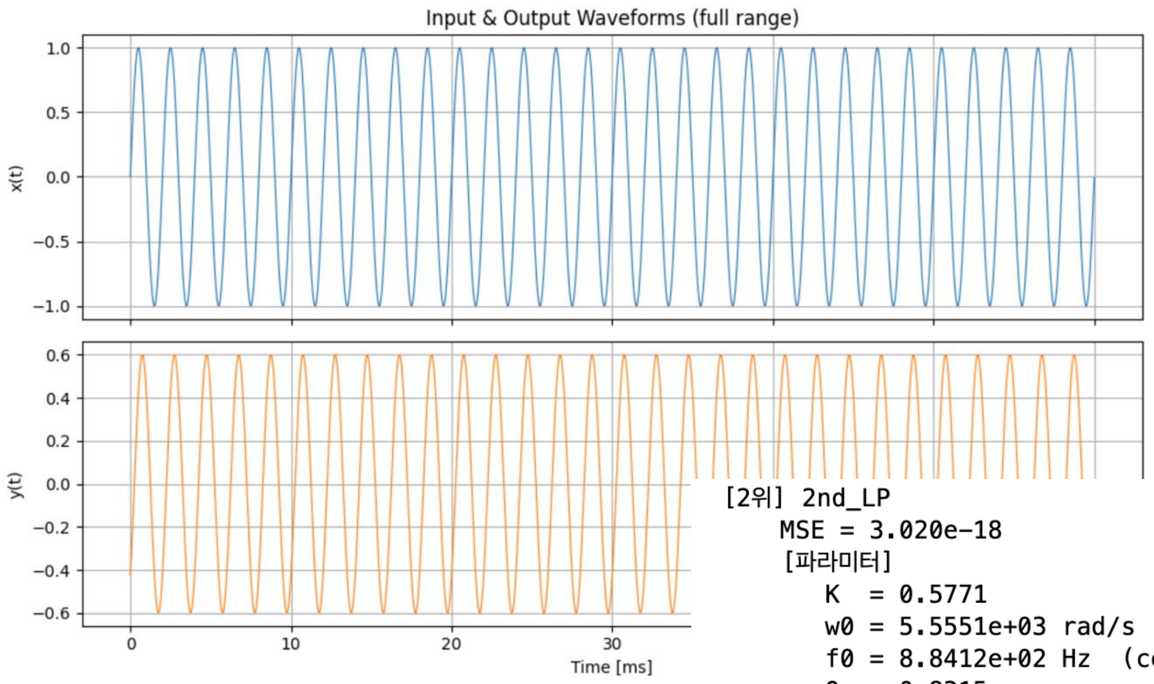
y f[Hz]: 500

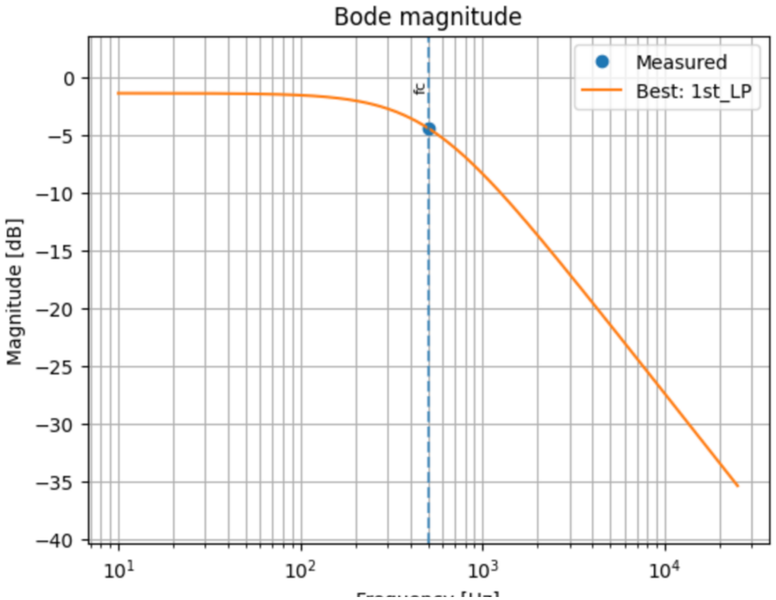
y phase: -45

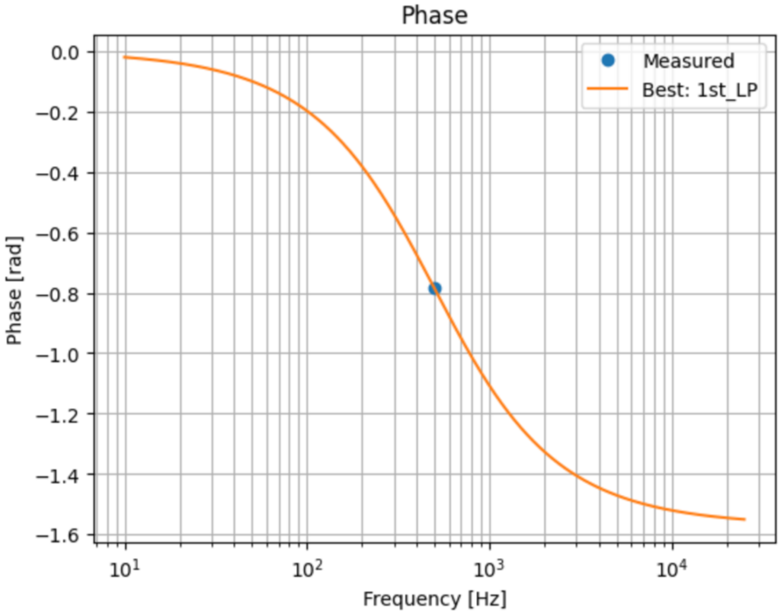
y t0: 0

Run identification

실행 결과 - 출력 결과 (사인파 입출력)



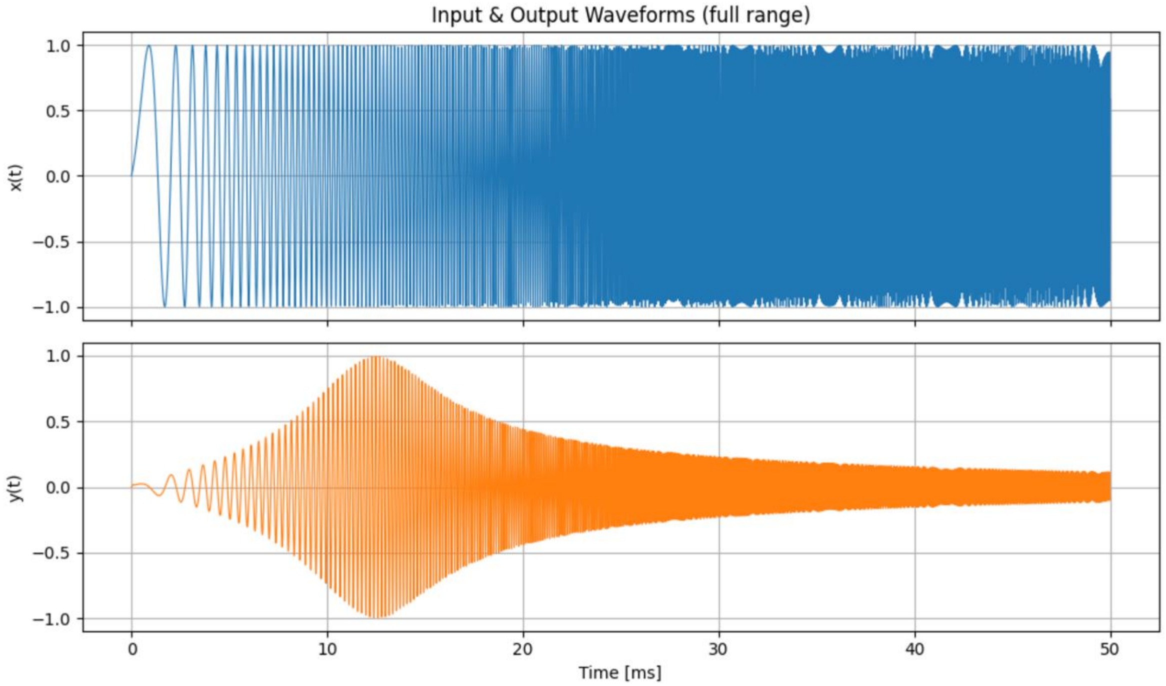


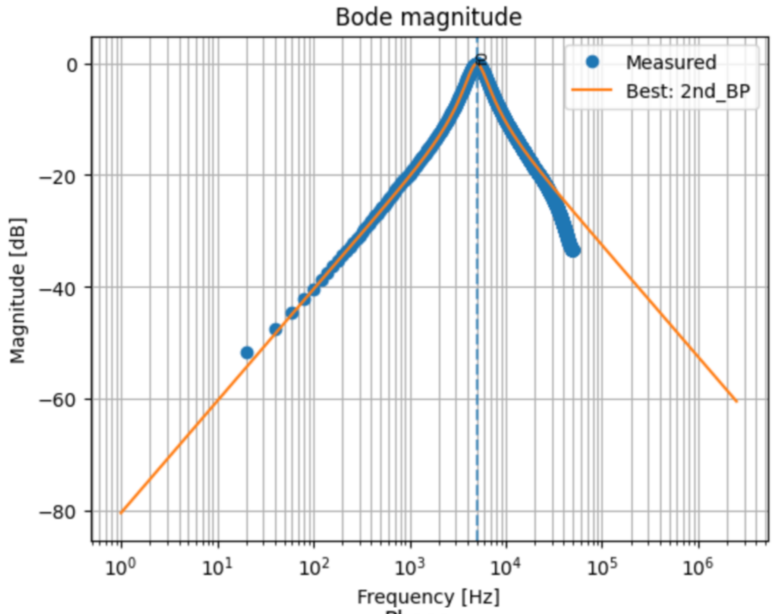


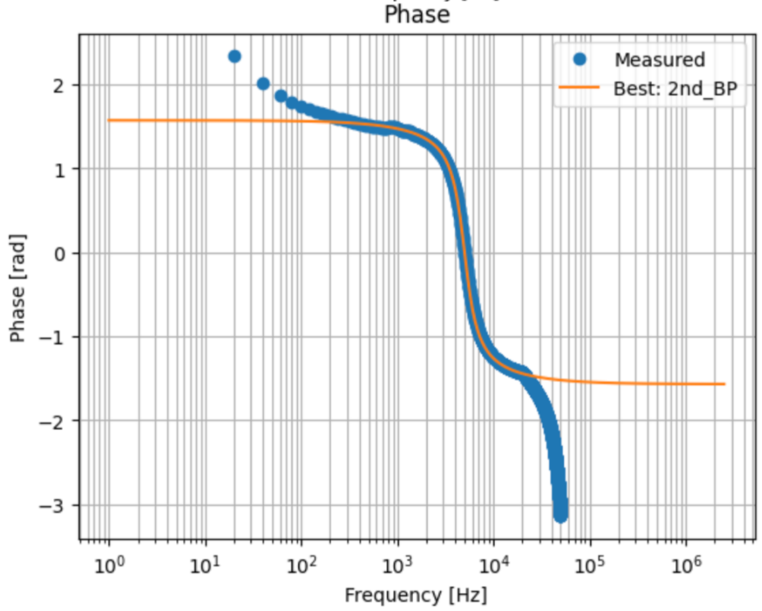

```
=== 최종 선택된 모델 (Best) ===
필터 타입: 1st_LP
  [파라미터]
    K  = 0.8485
    wc = 3.1416e+03 rad/s
    fc = 5.0000e+02 Hz  (cutoff)
  [RLC 값]
    R = 3.1831e+05
    C = 1.0000e-09
  Confidence score = 0.000
  (주의: 1위와 2위 모델의 MSE 차이가 작음.
```

```
[2위] 2nd_LP
    MSE = 3.020e-18
    [파라미터]
      K  = 0.5771
      w0 = 5.5551e+03 rad/s
      f0 = 8.8412e+02 Hz  (center)
      Q  = 0.8315
     (2nd LP fc ≈ 6.3194e+02 Hz)
    [RLC 근사]
     R ≈ 2.1651e+05
     L ≈ 3.2405e+01
     C = 1.0000e-09
[3위] 2nd_BP
    MSE = 8.337e-18
    [파라미터]
      K  = 0.8485
      w0 = 2.5808e+03 rad/s
      f0 = 4.1074e+02 Hz  (center)
      Q  = 2.5264
    [RLC 근사]
     R ≈ 1.5337e+05
     L ≈ 1.5014e+02
     C = 1.0000e-09
```

실행 결과 - 출력 결과 (Chirp 입출력 - BPF)





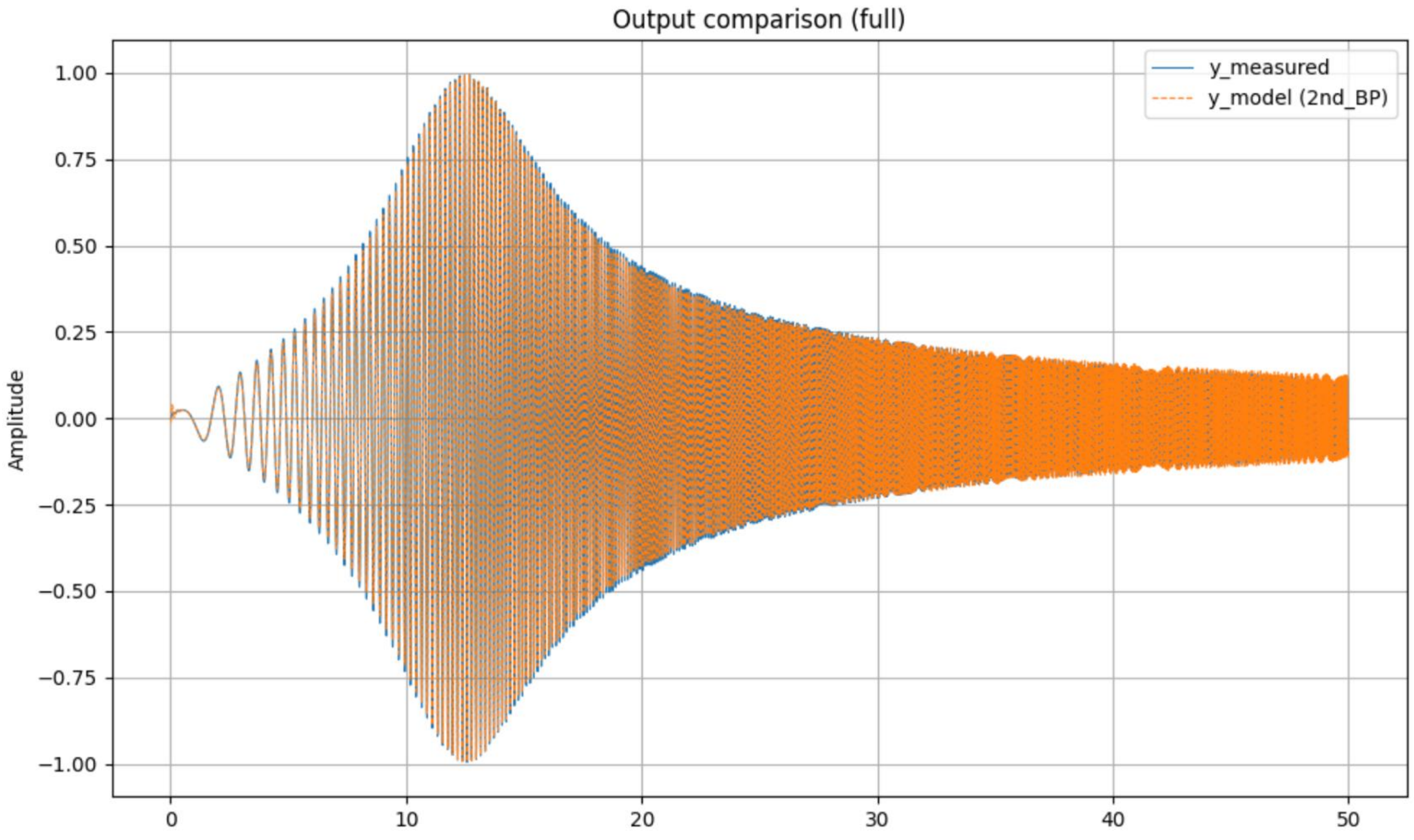



```
=== 최종 선택된 모델 (Best) ===
필터 타입: 2nd_BP
   [파라미터]
     K  = 0.9907
     w0 = 3.1386e+04 rad/s
     f0 = 4.9953e+03 Hz  (center)
     Q  = 2.0987
   [RLC 값]
     R = 1.5181e+04
     L = 1.0151e+00
     C = 1.0000e-09
  Confidence score = 1.000
```

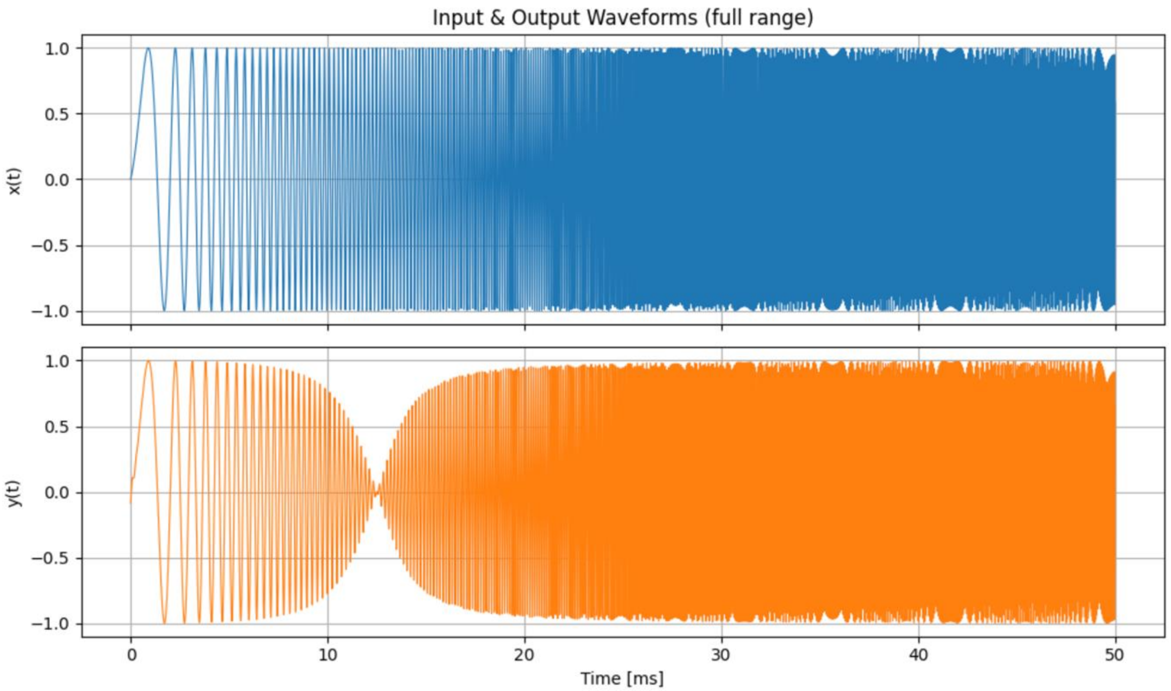
실행 결과 - 출력 결과 (Chirp 입출력 - BPF)

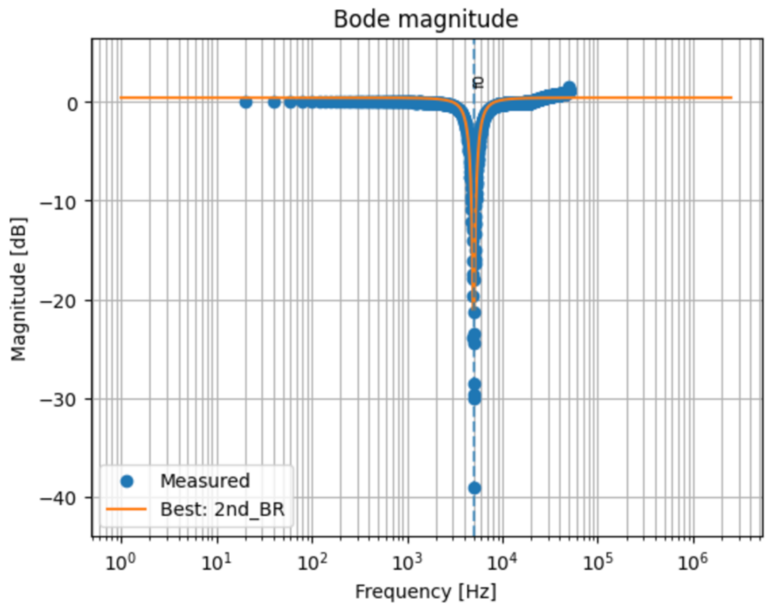
추가) 모델 검증
예측 모델에 입력 파형 넣어 얻은 출력 파형과 실제 출력 파형 비교

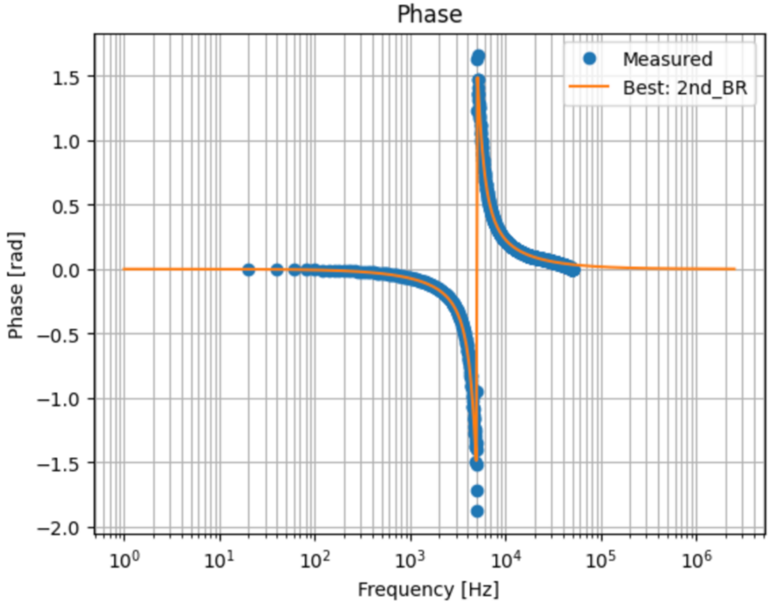
```
정렬된 신호: N=4999, dt=1.000e-05
MSE( y_model , y_meas ) = 7.501e-05
```



실행 결과 - 출력 결과 (Chirp 입출력 - BRF)

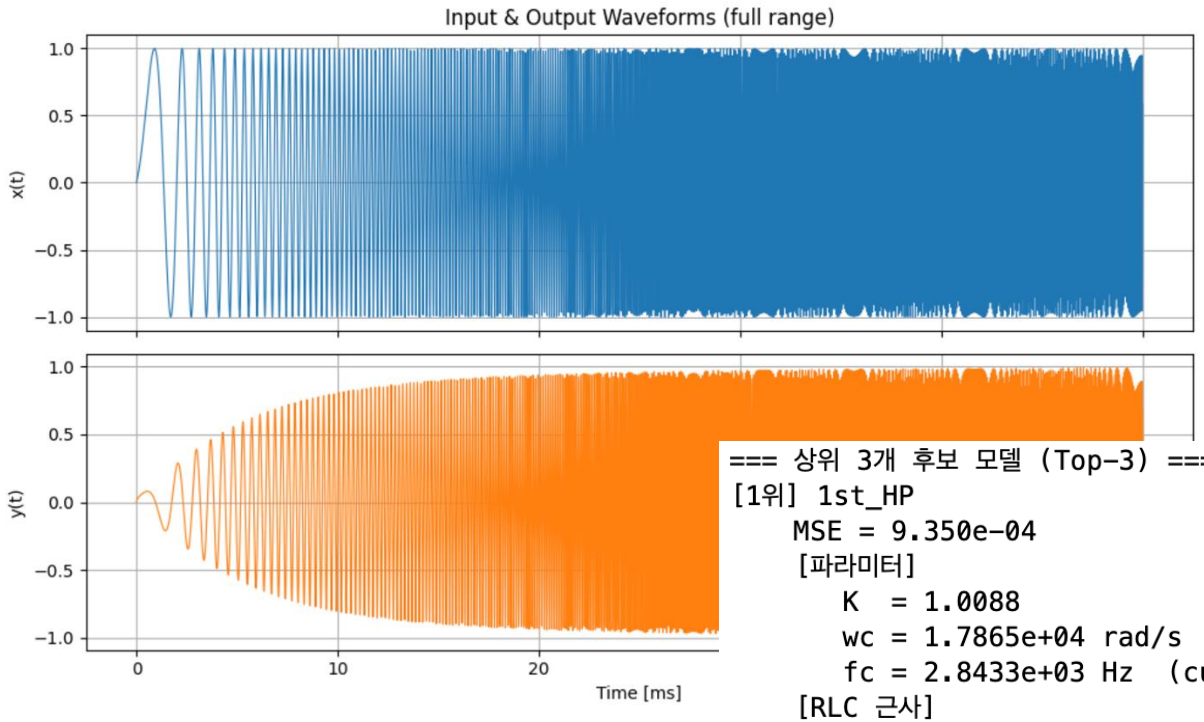






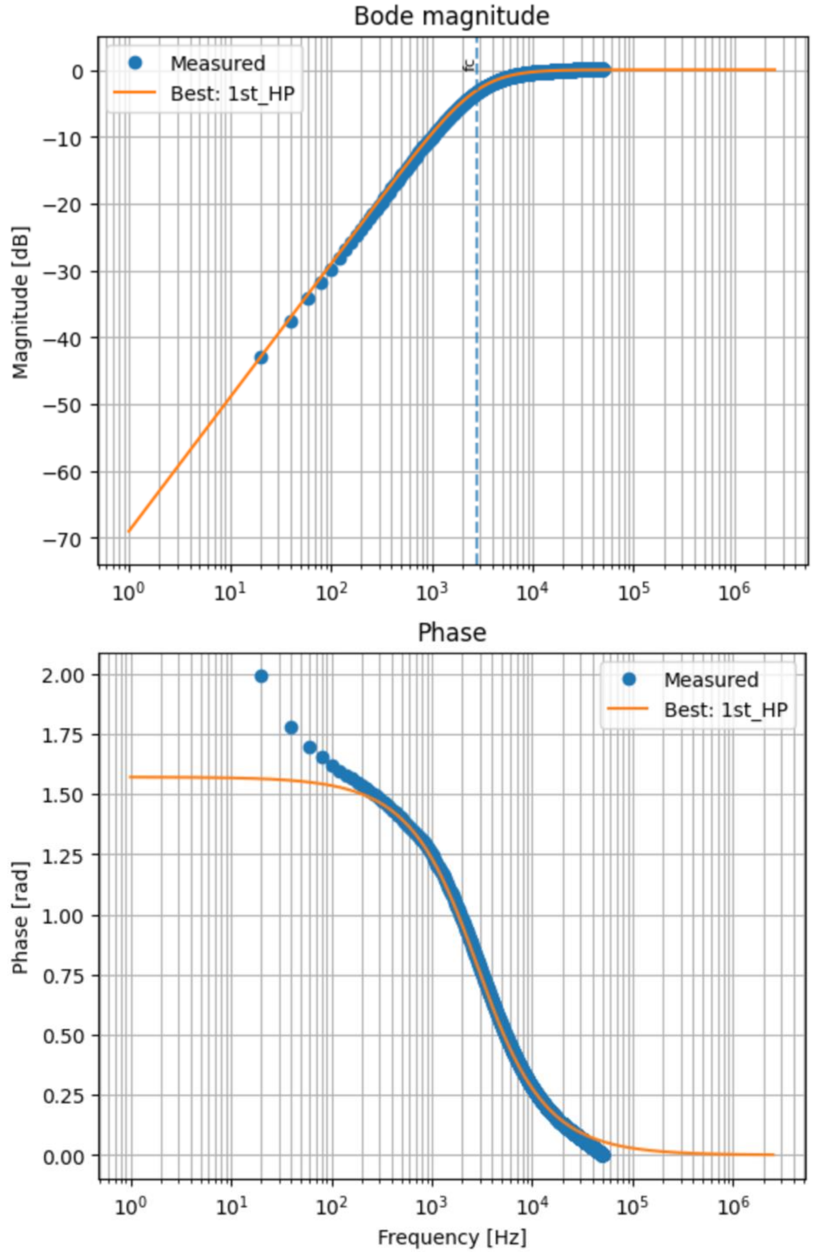

```
=== 최종 선택된 모델 (Best) ===
필터 타입: 2nd_BR
  [파라미터]
    K  = 1.0497
    w0 = 3.1416e+04 rad/s
    f0 = 5.0000e+03 Hz  (center)
    Q  = 2.8825
  [RLC 값]
    R = 1.1043e+04
    L = 1.0132e+00
    C = 1.0000e-09
  Confidence score = 1.000
```

실행 결과 - 출력 결과 (Chirp 입출력 - 1st HPF)

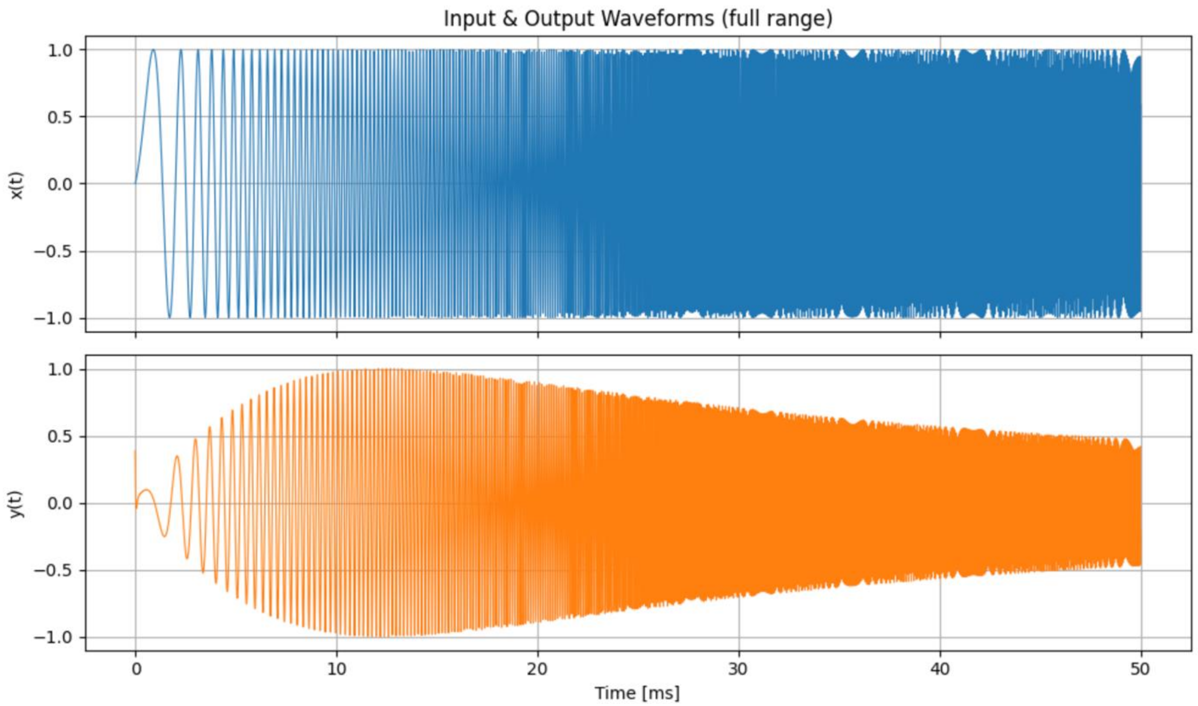


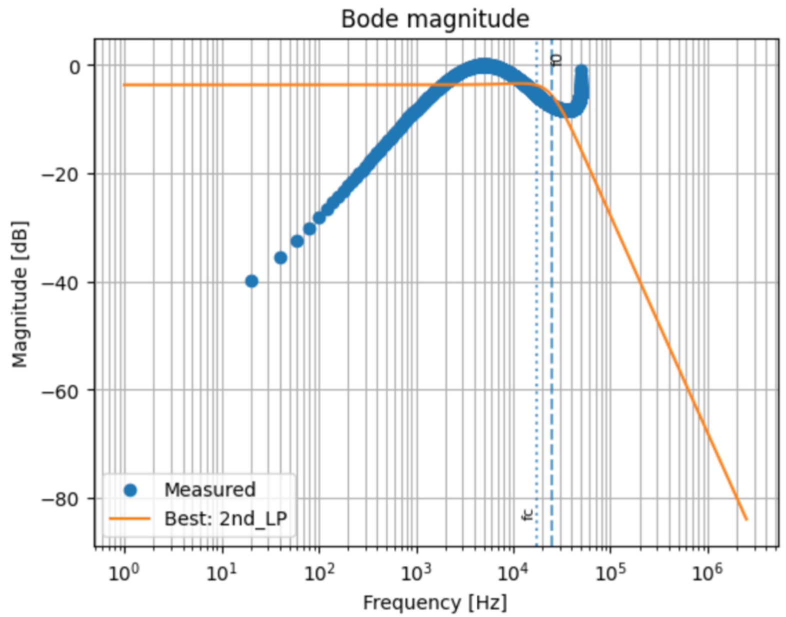

```
=== 상위 3개 후보 모델 (Top-3) ===
[1위] 1st_HP
    MSE = 9.350e-04
    [파라미터]
       K   = 1.0088
       wc = 1.7865e+04 rad/s
       fc = 2.8433e+03 Hz  (cutoff)
    [RLC 근사]
      R ≈ 5.5976e+04
      C = 1.0000e-09
[2위] 2nd_HP
    MSE = 9.477e-04
    [파라미터]
       K   = 1.0083
       w0 = 1.7857e+03 rad/s
       f0 = 2.8420e+02 Hz  (center)
       Q   = 0.1000
      (2nd HP fc ≈ 2.8702e+03 Hz)
    [RLC 근사]
      R ≈ 5.6001e+06
      L ≈ 3.1361e+02
      C = 1.0000e-09
```

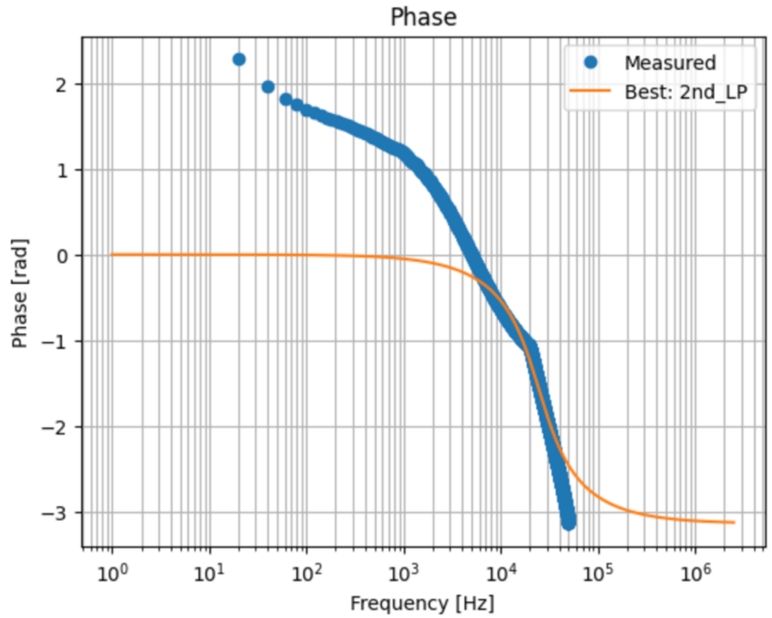
```
=== 최종 선택된 모델 (Best) ===
필터 타입: 1st_HP
  [파라미터]
    K   = 1.0088
    wc = 1.7865e+04 rad/s
    fc = 2.8433e+03 Hz  (cutoff)
  [RLC 값]
    R = 5.5976e+04
    C = 1.0000e-09
  Confidence score = 0.027
  (주의: 1위와 2위 모델의 MSE 차이가 작음.)
```



실행 결과 - 출력 결과 (Chirp 입출력 - low-Q BPF)



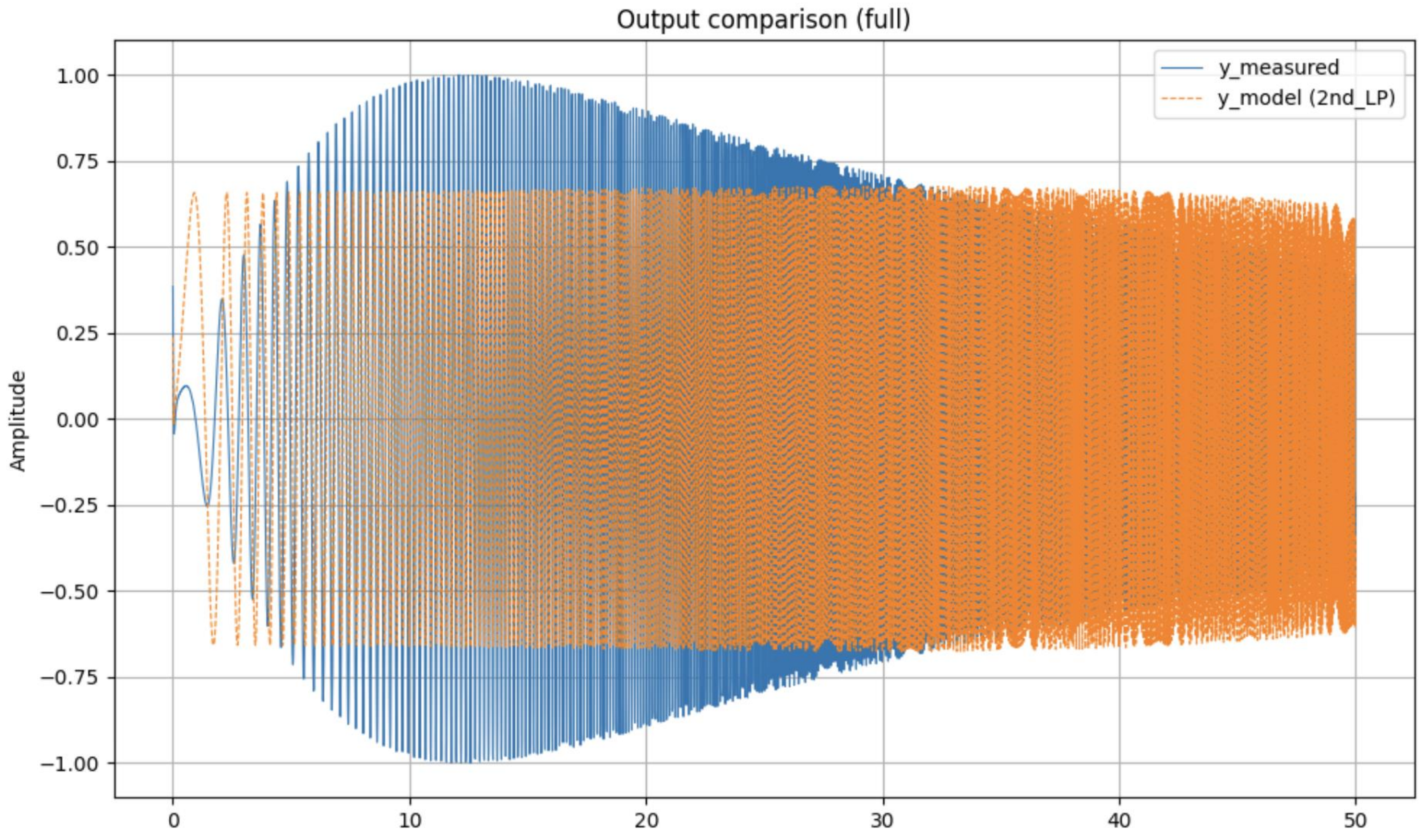





```
=== 최종 선택된 모델 (Best) ===
필터 타입: 2nd_LP
  [파라미터]
    K  = 0.6578
    w0 = 1.5469e+05 rad/s
    f0 = 2.4620e+04 Hz  (center)
    Q  = 0.8097
  [2nd LP cutoff]
    wc = 1.0883e+05 rad/s
    fc = 1.7321e+04 Hz
  [RLC 값]
    R = 7.9838e+03
    L = 4.1789e-02
    C = 1.0000e-09
  Confidence score = 0.251
```

실행 결과 - 출력 결과 (Chirp 입출력 - low-Q BPF)

```
정렬된 신호: N=4999, dt=1.000e-05
MSE( y_model , y_meas ) = 4.936e-02
```



결론

- 입출력 신호를 통해 다양한 필터 비교 후 최적의 필터 도출 가능

- 전달함수 크기가 작을 때 측정값이 튀는 불안정성

- Q가 작은 필터 식별 어려움