Lecture 8. Interactions in time series

Alexander Zhigalov / Dept. of CS, University of Helsinki and Dept. of NBE, Aalto University

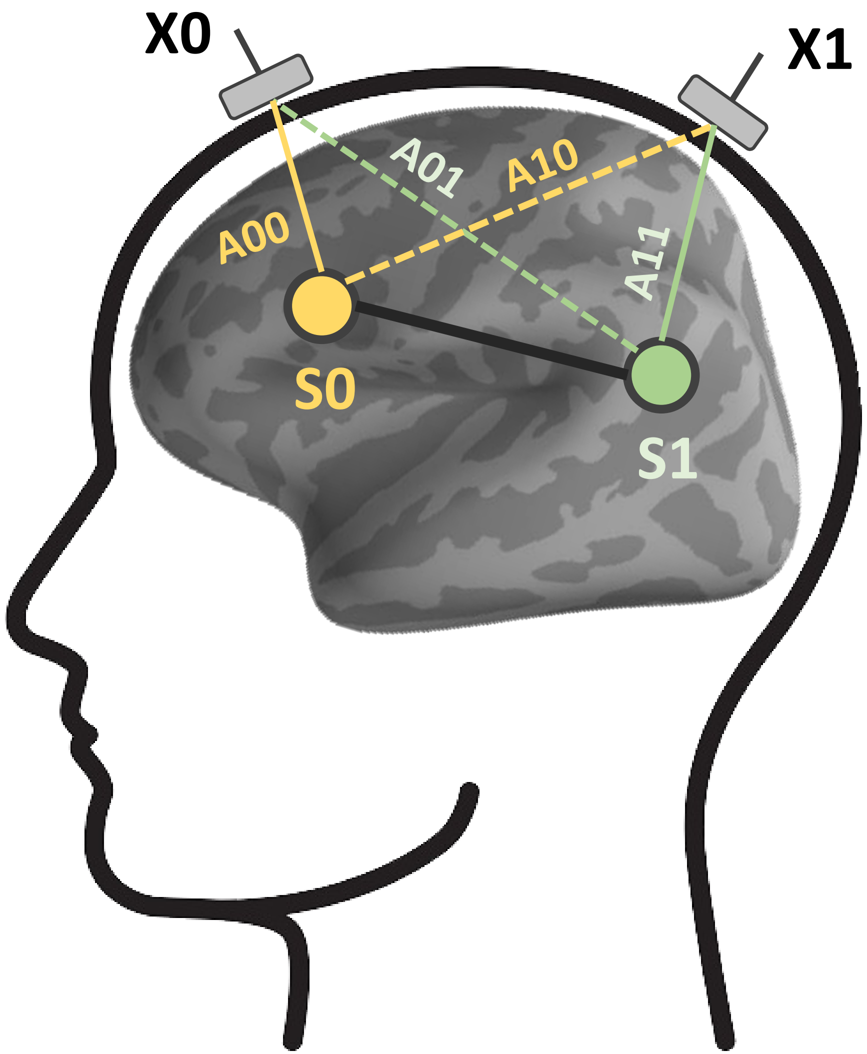
Outline / overview

Section 1. Source mixing

Linear mixing of sources

What is source mixing?

```
# measurements
X0 = A00 * S0 + A01 * S1
X1 = A10 * S0 + A11 * S1

# matrix notation
X = np.dot(A, S)
```

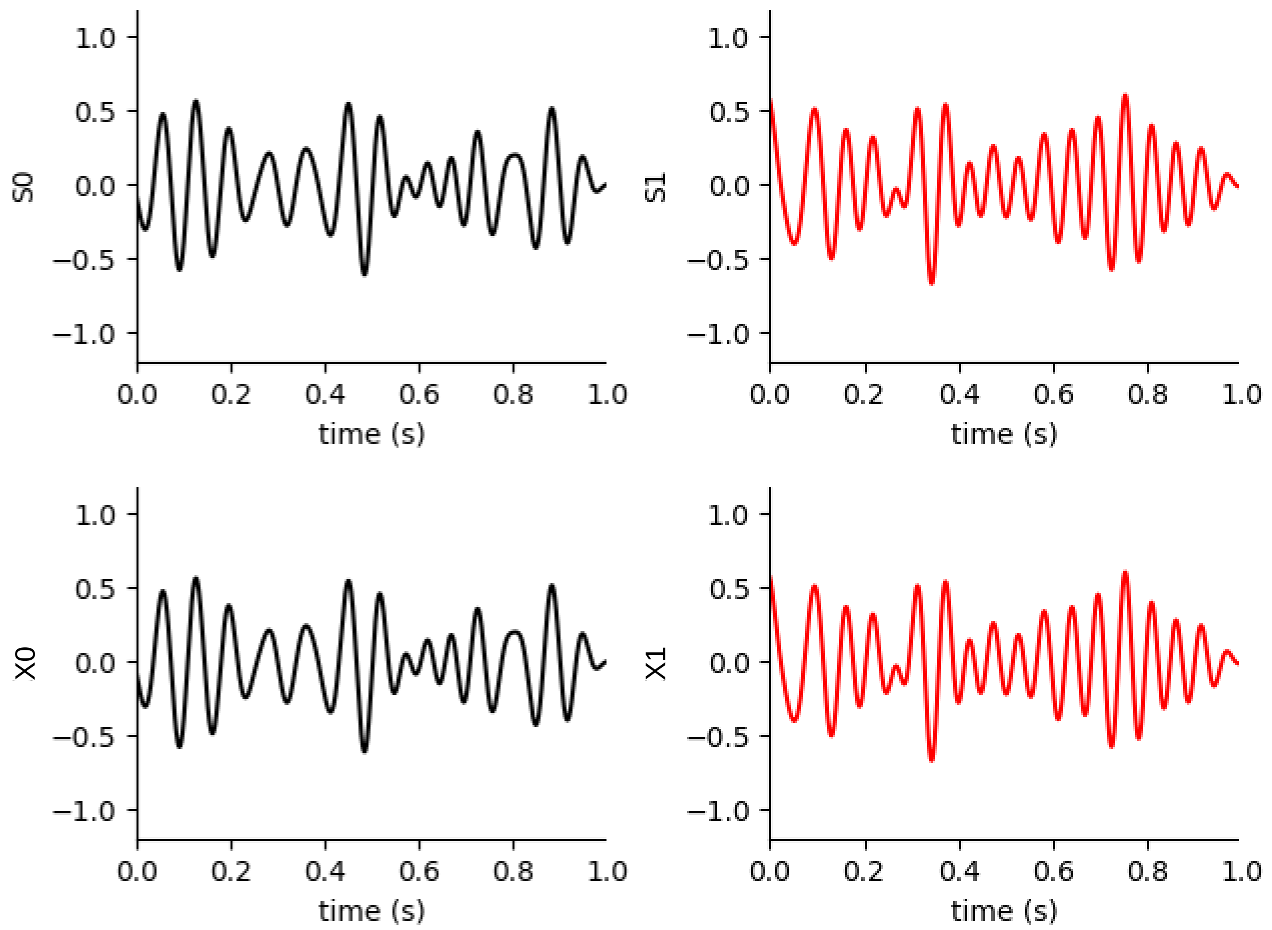


See, “L08_source_mixing.py”

Mixing "strength" (1/2)

Mixing matrix (A) determines the contribution of different sources.

```
# signal
S0 = np.random.randn(1, N)
S1 = np.random.randn(1, N)
S = np.concatenate((S0, S1))

# mixing matrix
A = np.array([[1.0, 0.0], \
              [0.0, 1.0]])

# mixing
X = np.dot(A, S)
X0 = X[0, :]
X1 = X[1, :]

# what is dot?
X[0,:] = A[0,0]*X[0,:] + A[0,1]*X[1,:]
X[1,:] = A[1,0]*X[0,:] + A[1,1]*X[1,:]
```

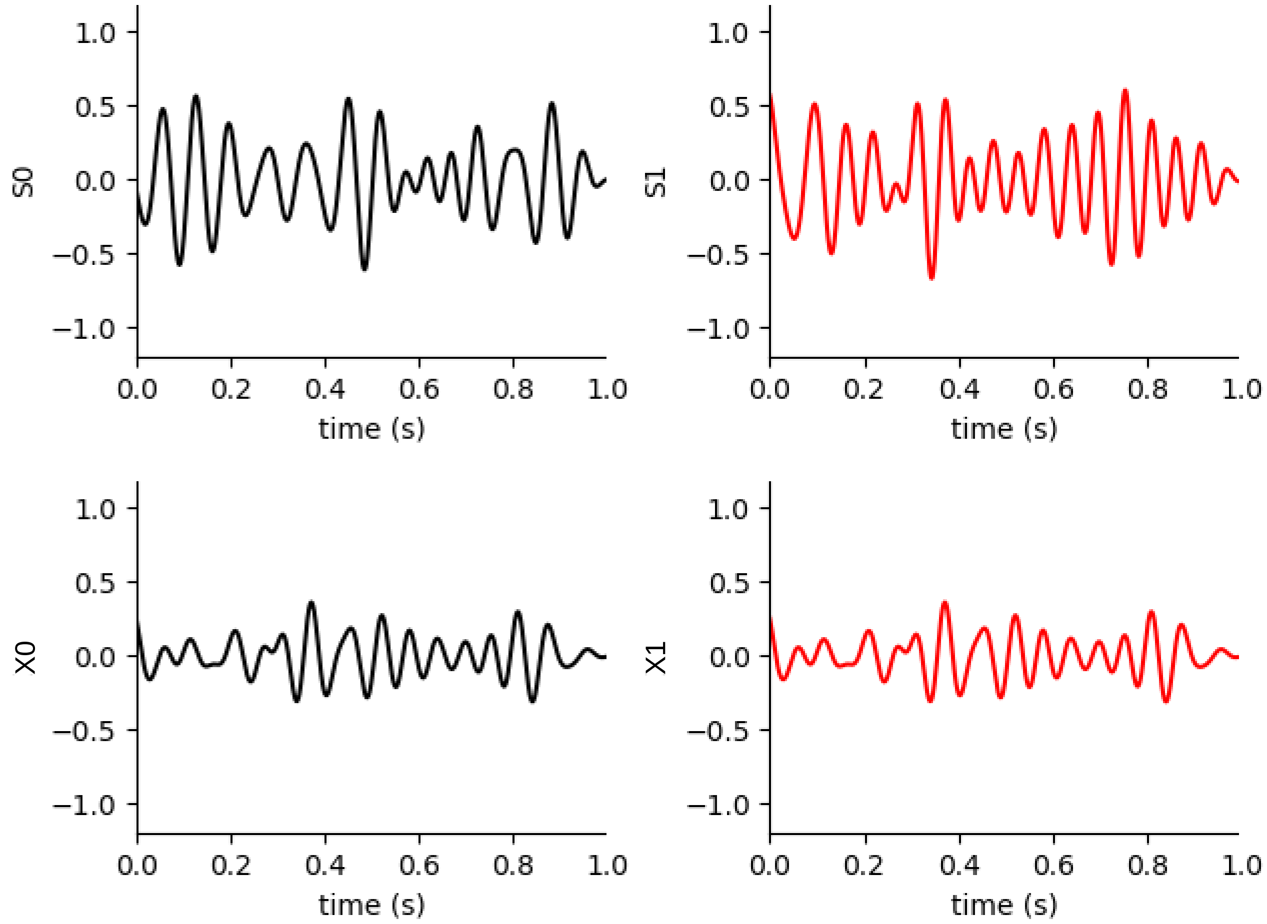


See, "L08_source_mixing.py"

Mixing "strength" (2/2)

Mixing matrix (A) determines the contribution of different sources.

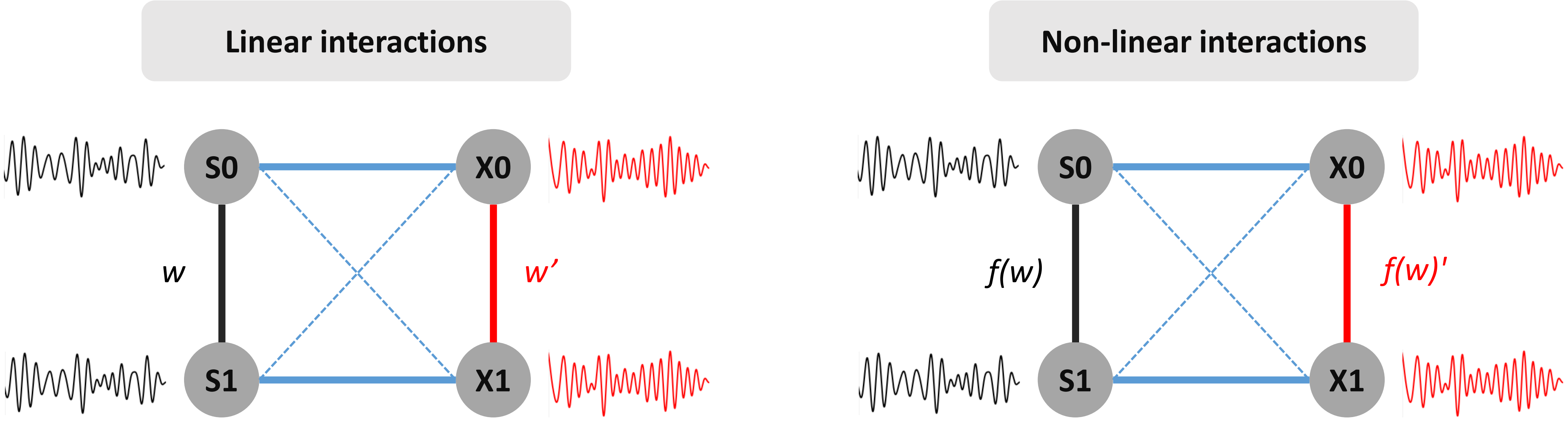
```
# signal
S0 = np.random.randn(1, N)
S1 = np.random.randn(1, N)
S = np.concatenate((S0, S1))

# mixing matrix
A = np.array([[0.5, 0.5], \
              [0.5, 0.5]])

# mixing
X = np.dot(A, S)
X0 = X[0, :]
X1 = X[1, :]
```



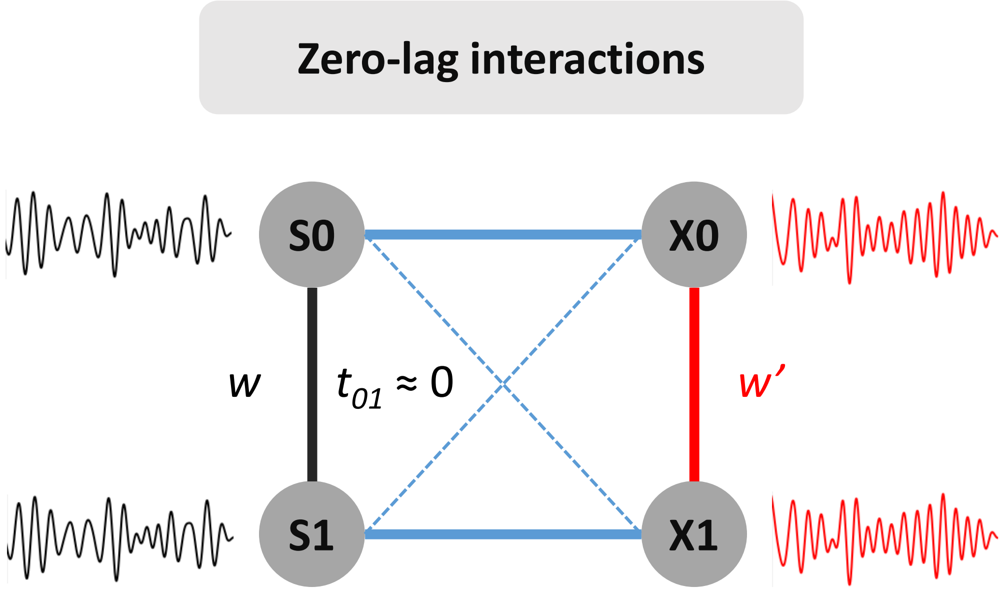
See, "L08_source_mixing.py"

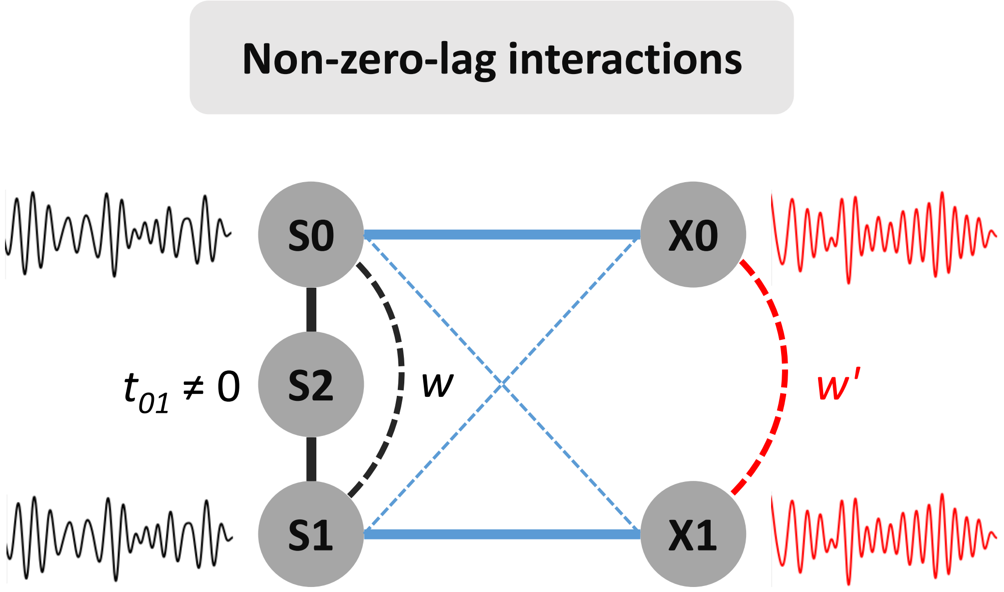
Section 2. Interactions between sources

Linear and Non-linear interactions



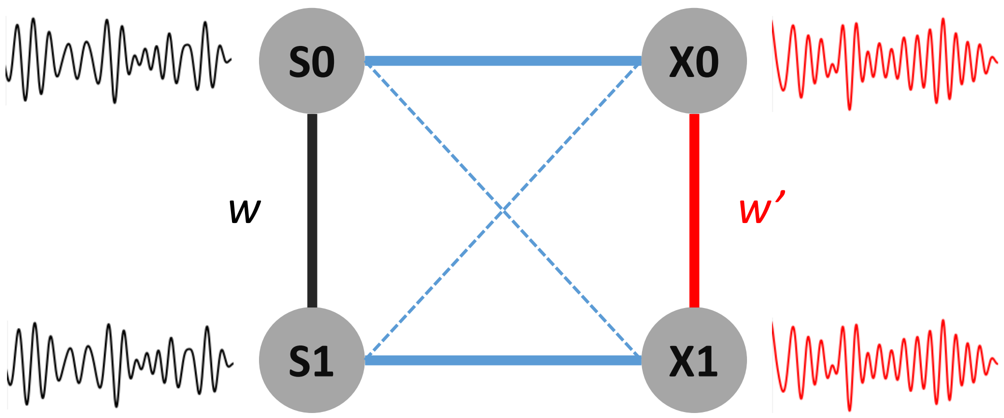
Zero-lag and Non-zero-lag interactions



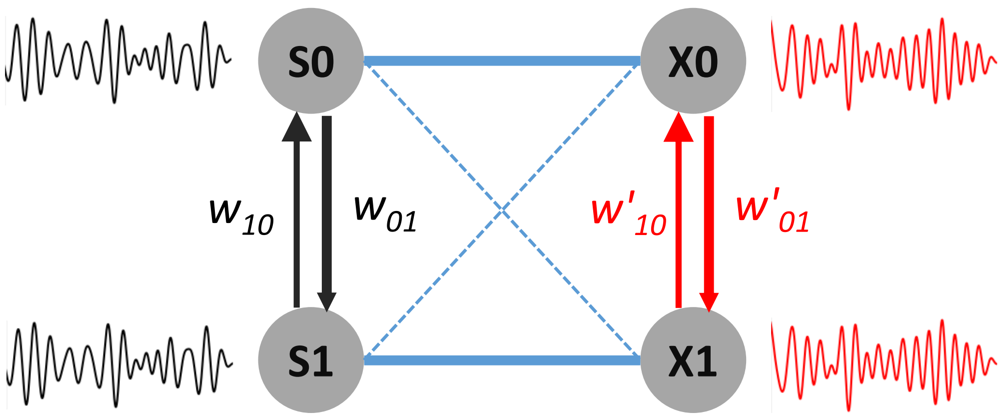


Directed/Causal and Non-directed interactions

Non-directed interactions



Directed interactions

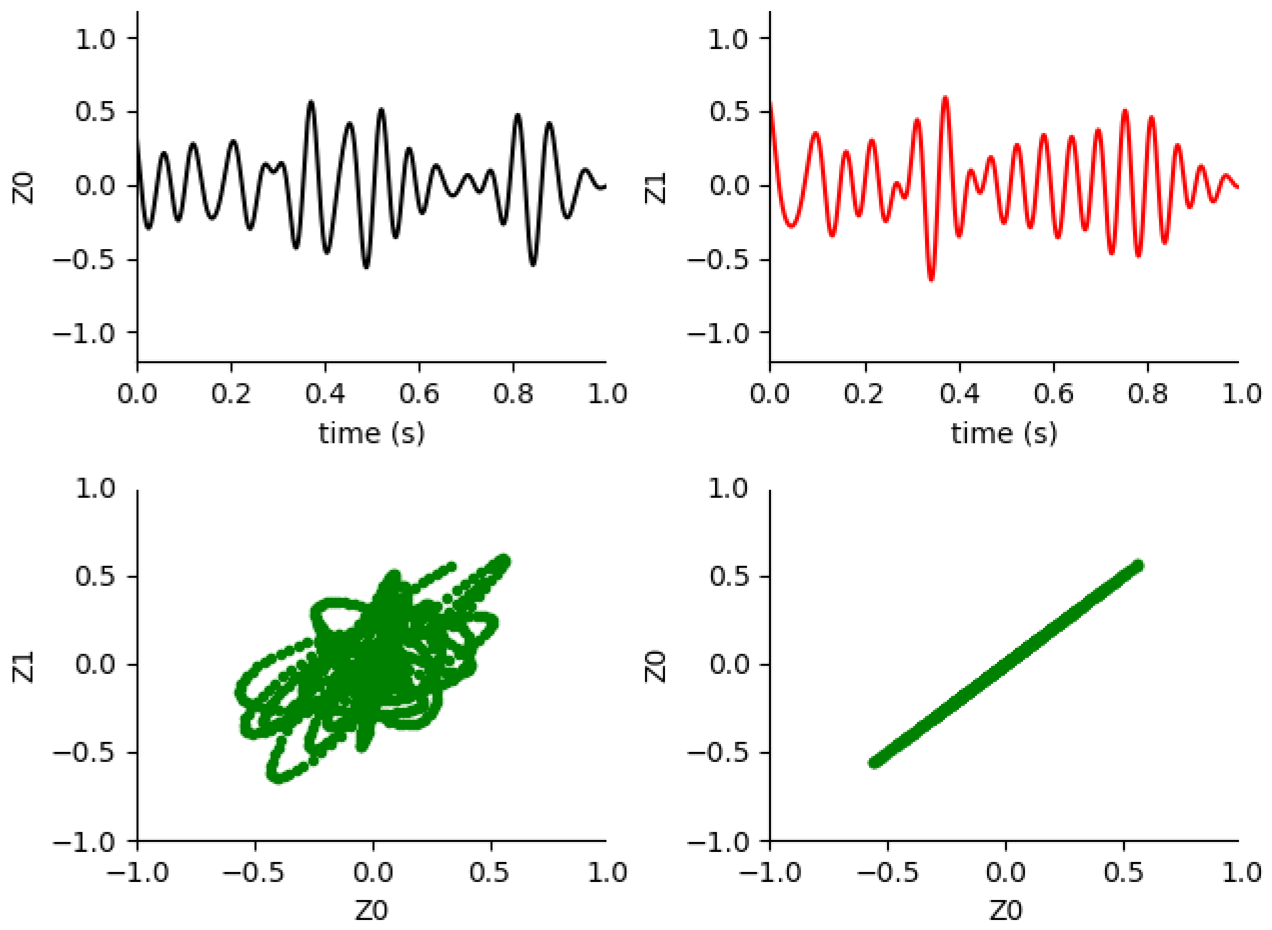


Section 3. Covariance and Correlation

Covariance

What is covariance? It is a number that shows similarity between signals.

```
# mixing matrix
A = np.array([[1.0, 0.7], \
              [0.3, 1.0]])

# mixing
Z = np.dot(A, S)
Z0 = Z[0, :]
Z1 = Z[1, :]

# covariance
r = np.sum(((Z0 - np.mean(Z0)) *
            (Z1 - np.mean(Z1)))) / N
```

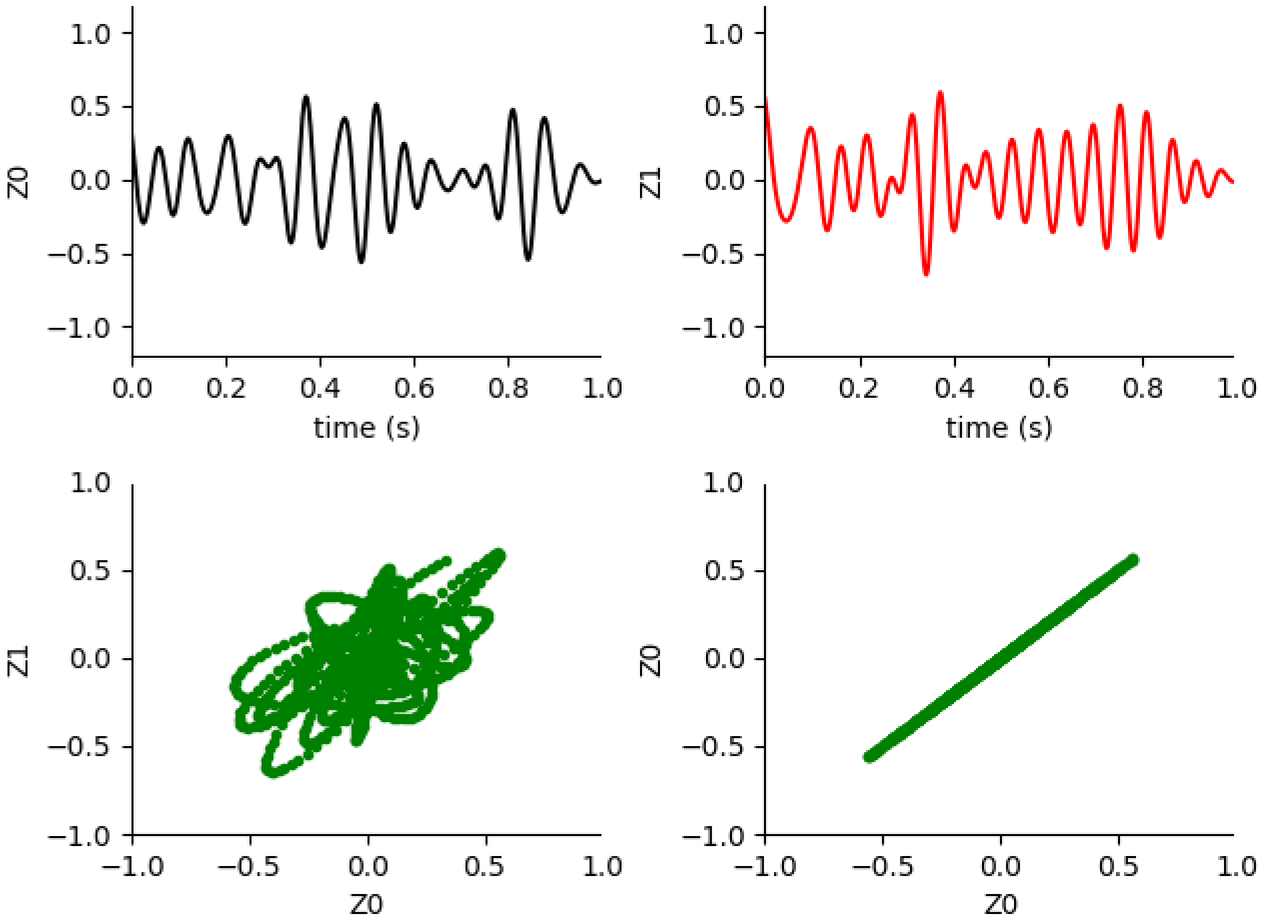


See, “L08_covariance_and_correlation.py”

Correlation

What is correlation? Correlation is the normalized covariance.

```
# mixing matrix
A = np.array([[1.0, 0.7], \
              [0.3, 1.0]])

# mixing
Z = np.dot(A, S)
Z0 = Z[0, :]
Z1 = Z[1, :]

# correlation, range [-1, 1]
r = np.sum(((Z0 - np.mean(Z0)) *
            (Z1 - np.mean(Z1))) /
           (np.std(Z0) * np.std(Z1))) / N
```



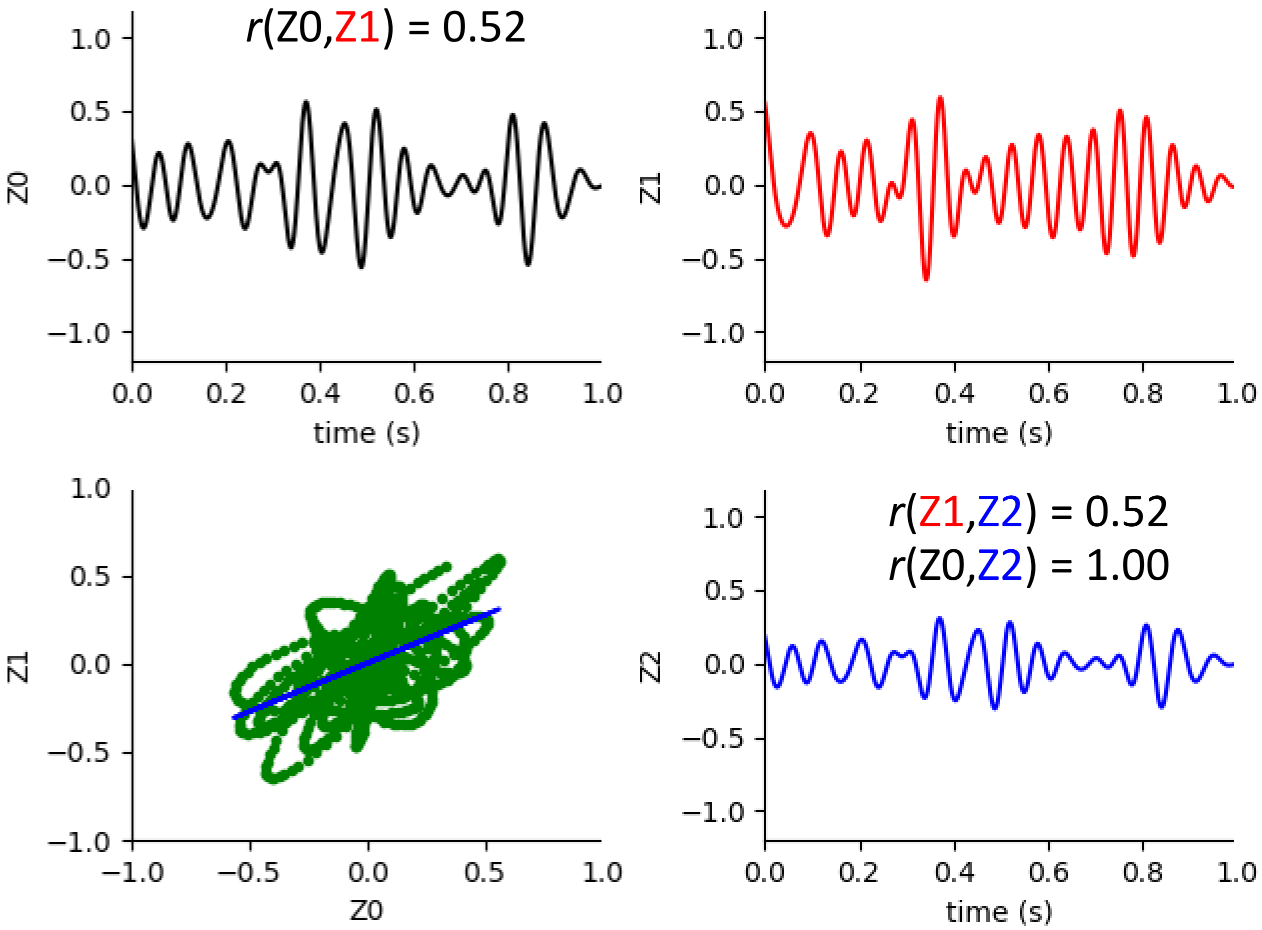
See, “L08_covariance_and_correlation.py”

Section 4. Regression and Partial correlation

Regression

What is regression? Regression tries to represent y as a weighted version of x.

```
# mixing matrix
A = np.array([[1.0, 0.7], \
              [0.3, 1.0]])

# mixing
Z = np.dot(A, S)
Z0 = Z[0, :]
Z1 = Z[1, :]

# regression
p = np.polyfit(Z0, Z1, 1)
Z2 = p[0] * Z0 + p[1]
```

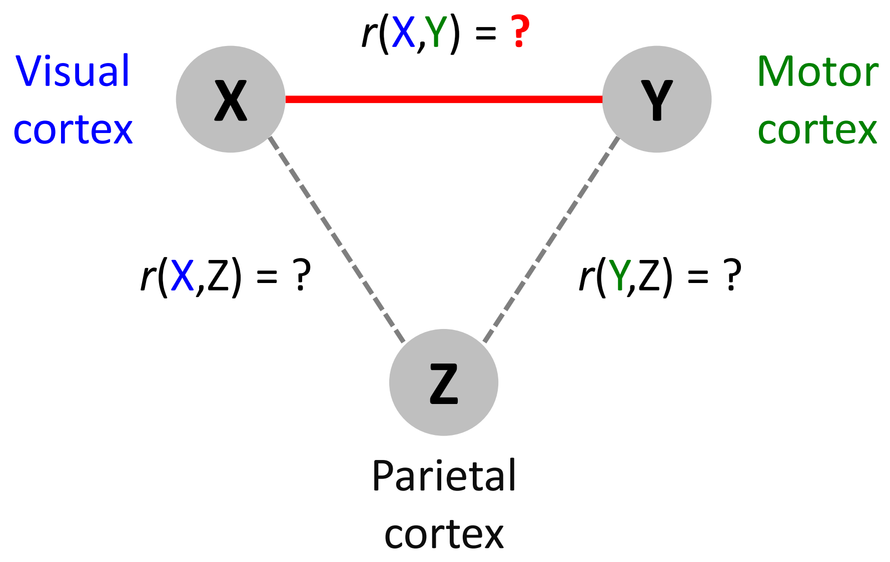


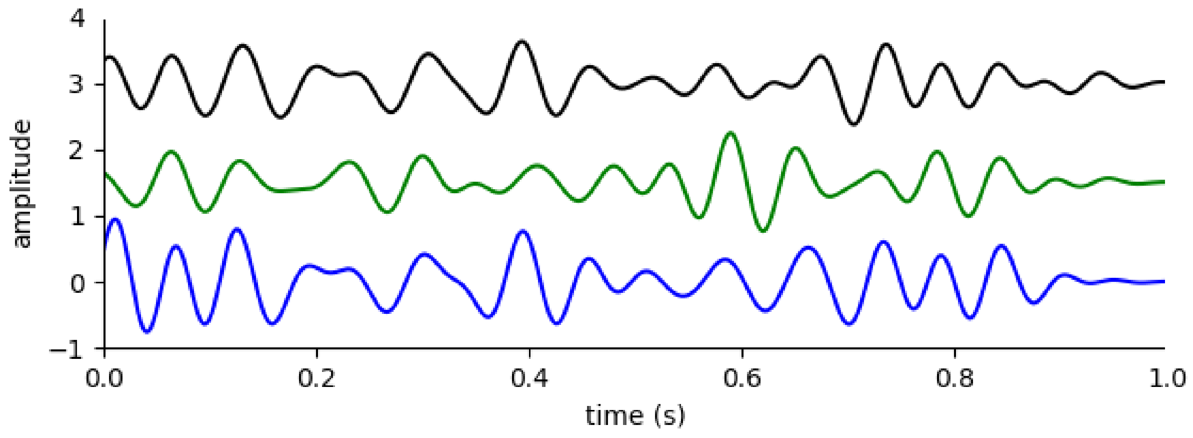
See, “L08_regression.py”

Partial correlation (1/3)

What is partial correlation? It is correlation between X and Y with regressed out Z.

```
# mixing matrix
A = np.array([[1.0, 0.7, 0.3], \
              [0.3, 1.0, 0.1], \
              [0.5, 0.5, 1.0]])
# mixing
U = np.dot(A, S)
X = U[0, :] # visual cortex
Y = U[1, :] # motor cortex
Z = U[2, :] # parietal cortex
```



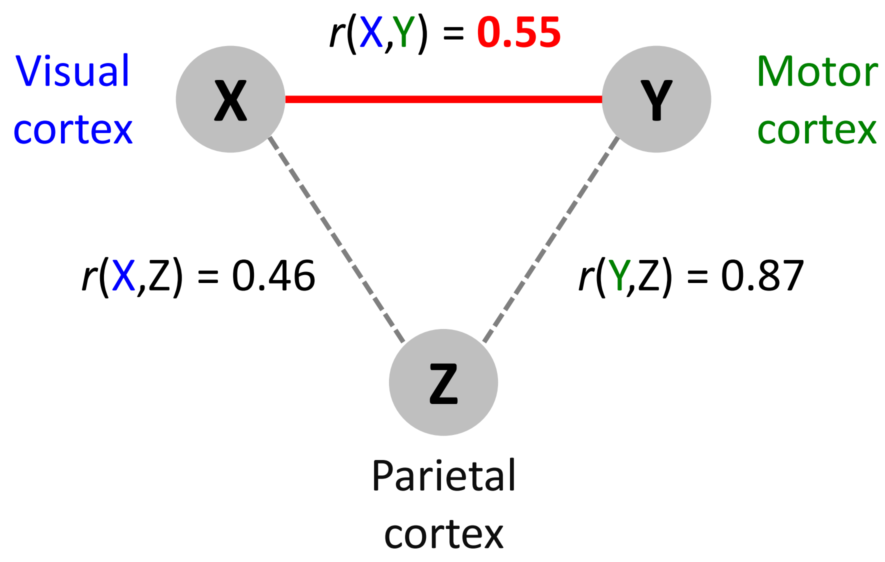


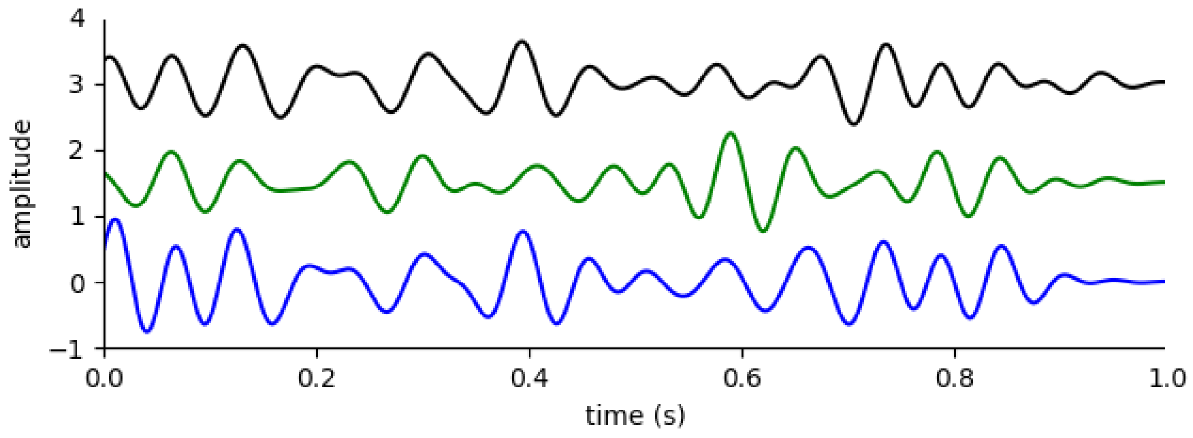
See, “L08_partial_correlation.py”

Partial correlation (2/3)

We can compute correlation coefficients between all pairs.

```
# mixing matrix
A = np.array([[1.0, 0.7, 0.3], \
              [0.3, 1.0, 0.1], \
              [0.5, 0.5, 1.0]])
# mixing
U = np.dot(A, S)
X = U[0, :] # visual cortex
Y = U[1, :] # motor cortex
Z = U[2, :] # parietal cortex

# correlation
rXY = np.corrcoef(X, Y)[0, 1]
rXZ = np.corrcoef(X, Z)[0, 1]
rYZ = np.corrcoef(Y, Z)[0, 1]
```



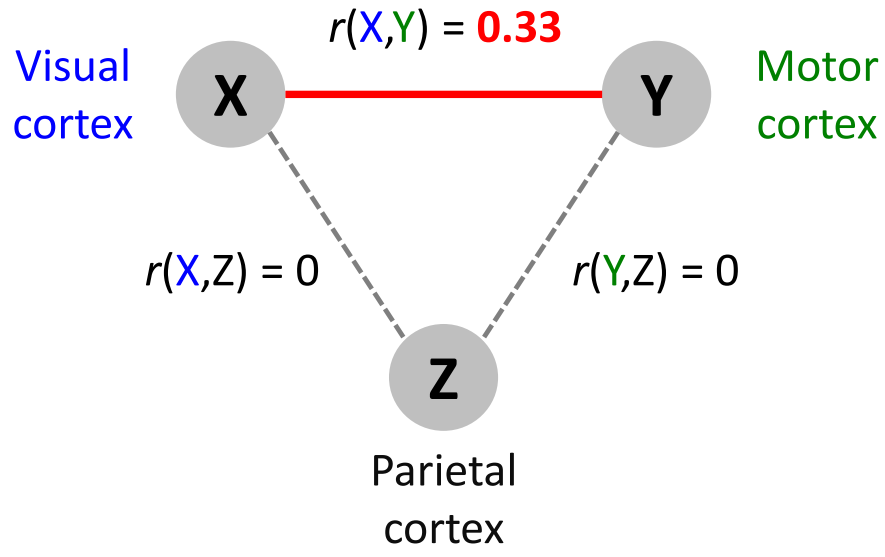


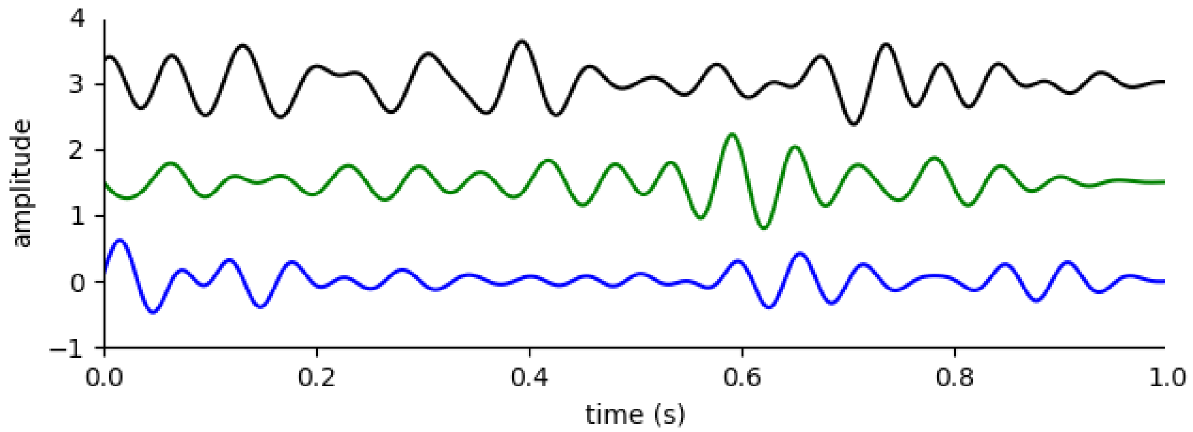
See, “L08_partial_correlation.py”

Partial correlation (3/3)

Can we assess correlation between X and Y without influence of Z?

```
# fitting and regressing out
p = np.polyfit(Z, X, 1)
fX = X – (p[0] * Z + p[1])

p = np.polyfit(Z, Y, 1)
fY = Y – (p[0] * Z + p[1])

# correlation
rXY = np.corrcoef(fX, fY)[0, 1]
rXZ = np.corrcoef(fX, Z)[0, 1]
rYZ = np.corrcoef(fY, Z)[0, 1]
```

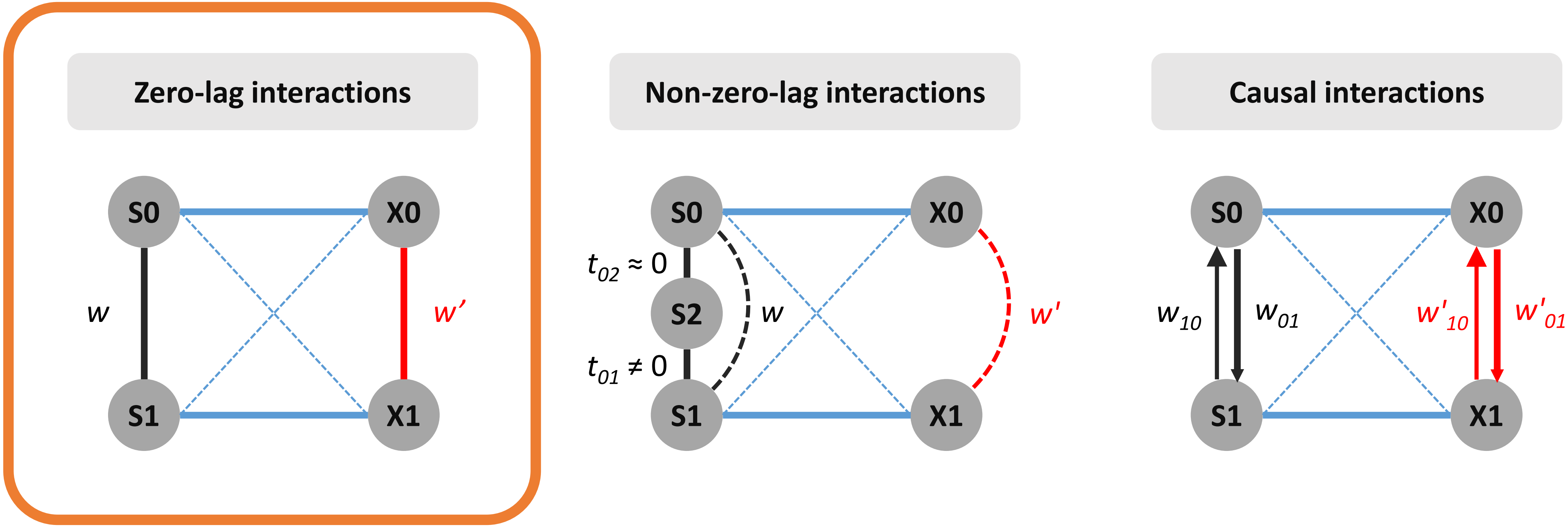




See, “L08_partial_correlation.py”

Section 5. Zero-lag interactions

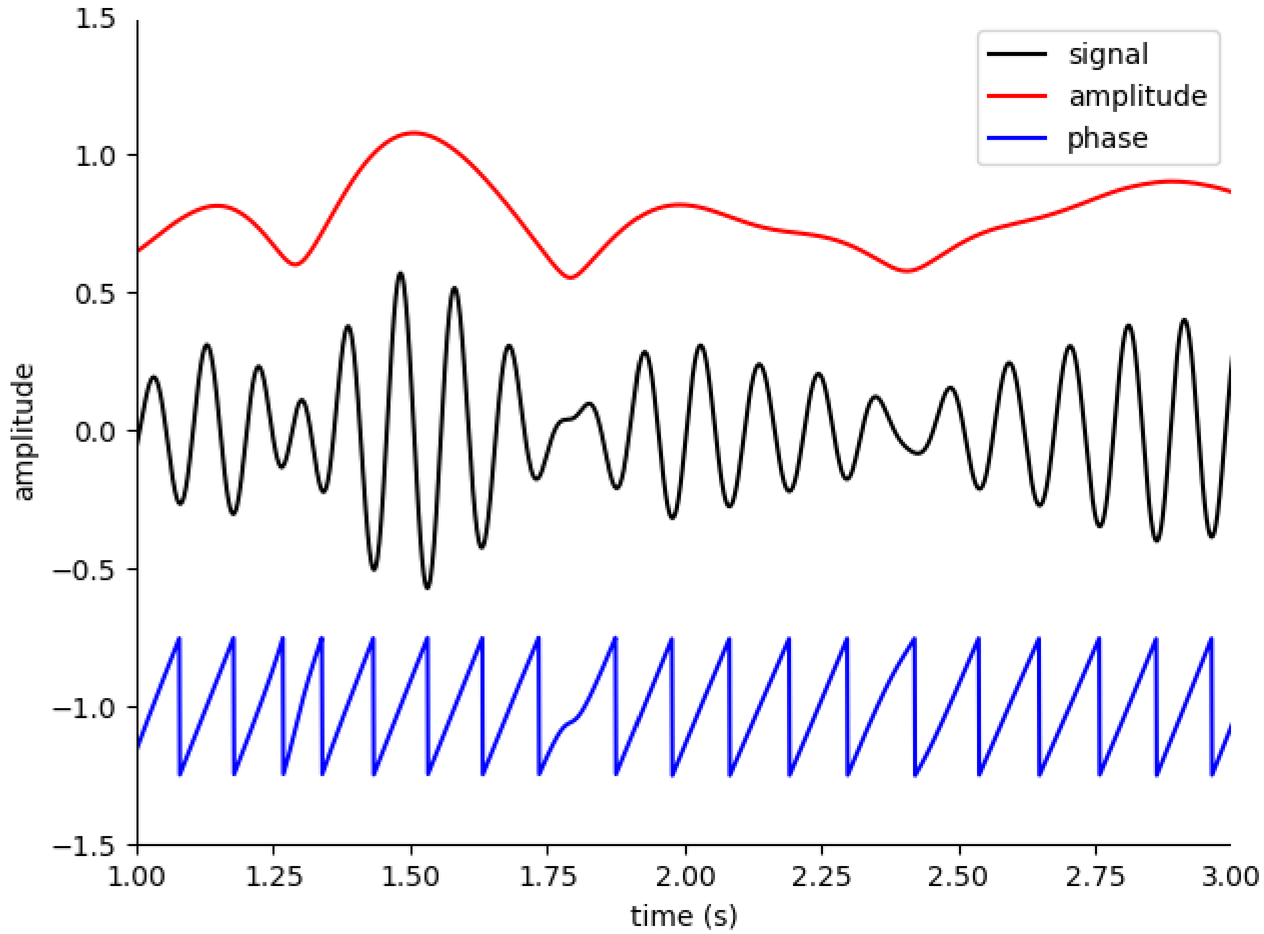
Interactions



Signal parameters

Signal can be represented by its amplitude and phase.

```
# signal
S = np.random.randn(1, N)

# filtering
[b, a] = signal.butter(4,
        [8.0 / (fs/2), 12.0 / (fs/2)],
        'bandpass')
X = signal.filtfilt(b, a, S)

# amplitude
AX = np.abs(signal.hilbert(X))

# phase
PX = np.angle(signal.hilbert(X))
```

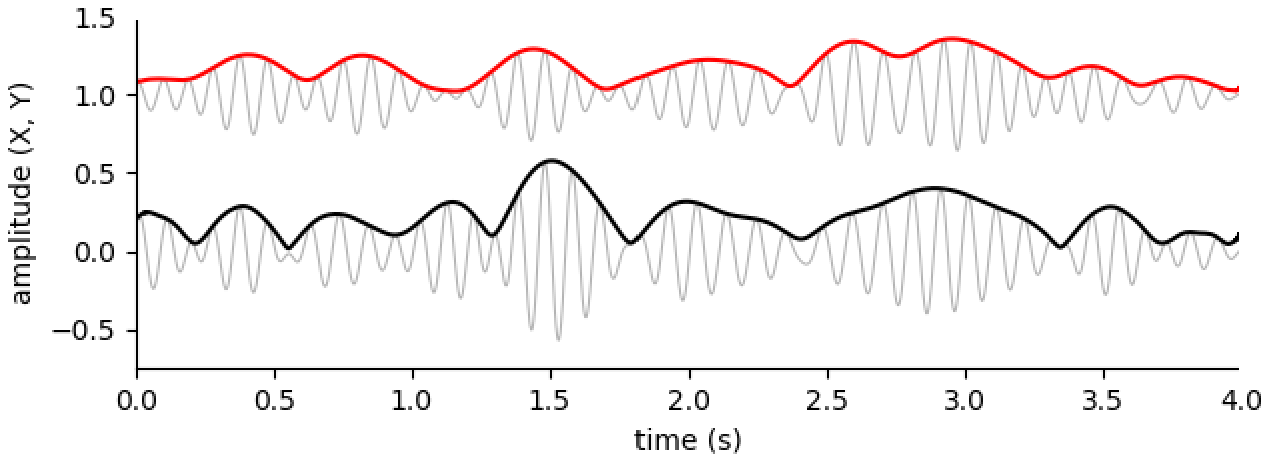


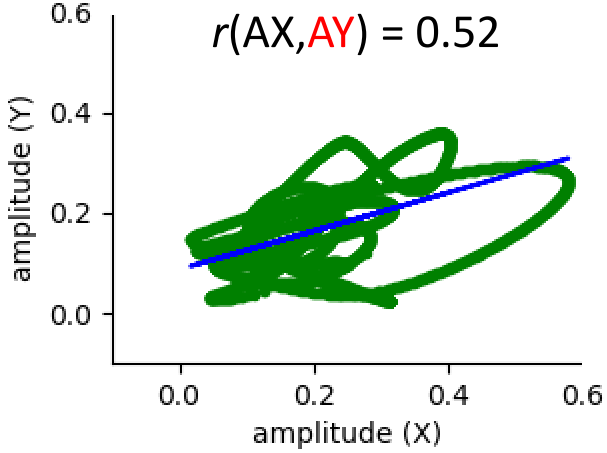
See, “L08_zero_lag_interactions_signal.py”

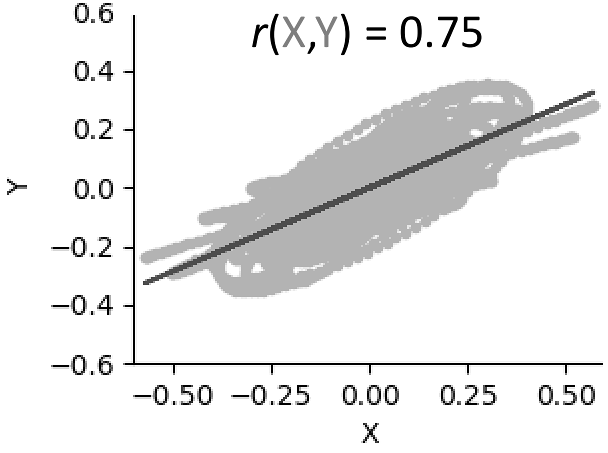
I. Amplitude-amplitude correlations

Correlation between amplitudes of two signals.

```
# amplitude
AX = np.abs(signal.hilbert(X))
AY = np.abs(signal.hilbert(Y))

# linear fit (amplitudes)
p = np.polyfit(AX, AY, 1)
AU = p[0] * AX + p[1]

# linear fit (signal)
p = np.polyfit(X, Y, 1)
U = p[0] * X + p[1]

# correlation
rAA = np.corrcoef(AX, AY)[0, 1]
rXY = np.corrcoef(X, Y)[0, 1]
```





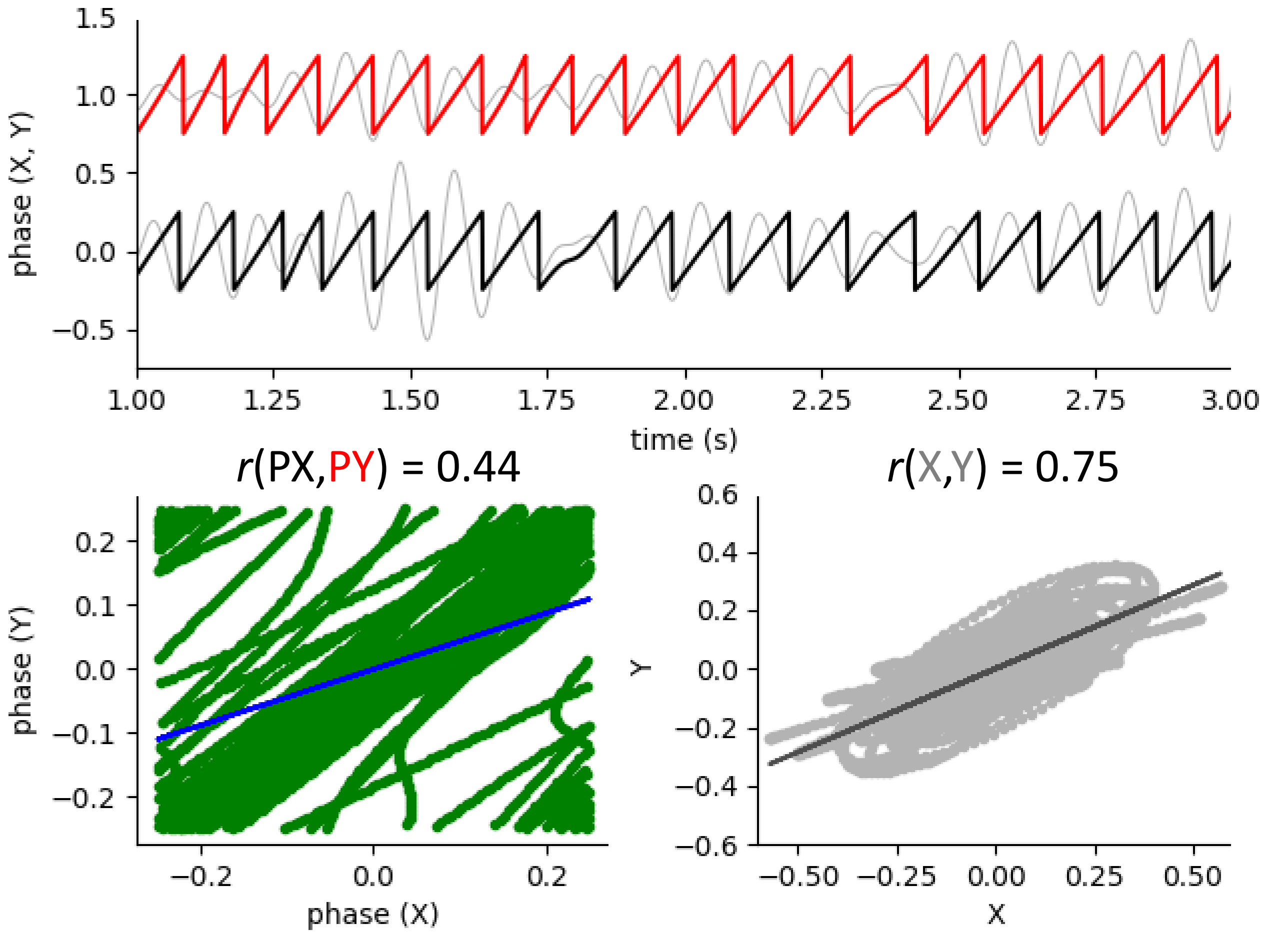


See, "L08_zero_lag_interactions_amplitude.py"

II. Phase-phase coupling (1/2)

Coupling between phases of two signals.

```
# phase
PX = np.angle(signal.hilbert(X))
PY = np.angle(signal.hilbert(Y))

# linear fit (phases)
p = np.polyfit(PX, PY, 1)
PU = p[0] * PX + p[1]

# linear fit (signal)
p = np.polyfit(X, Y, 1)
U = p[0] * X + p[1]

# correlation
rPP = np.corrcoef(PX, PY)[0, 1]
rXY = np.corrcoef(X, Y)[0, 1]
```

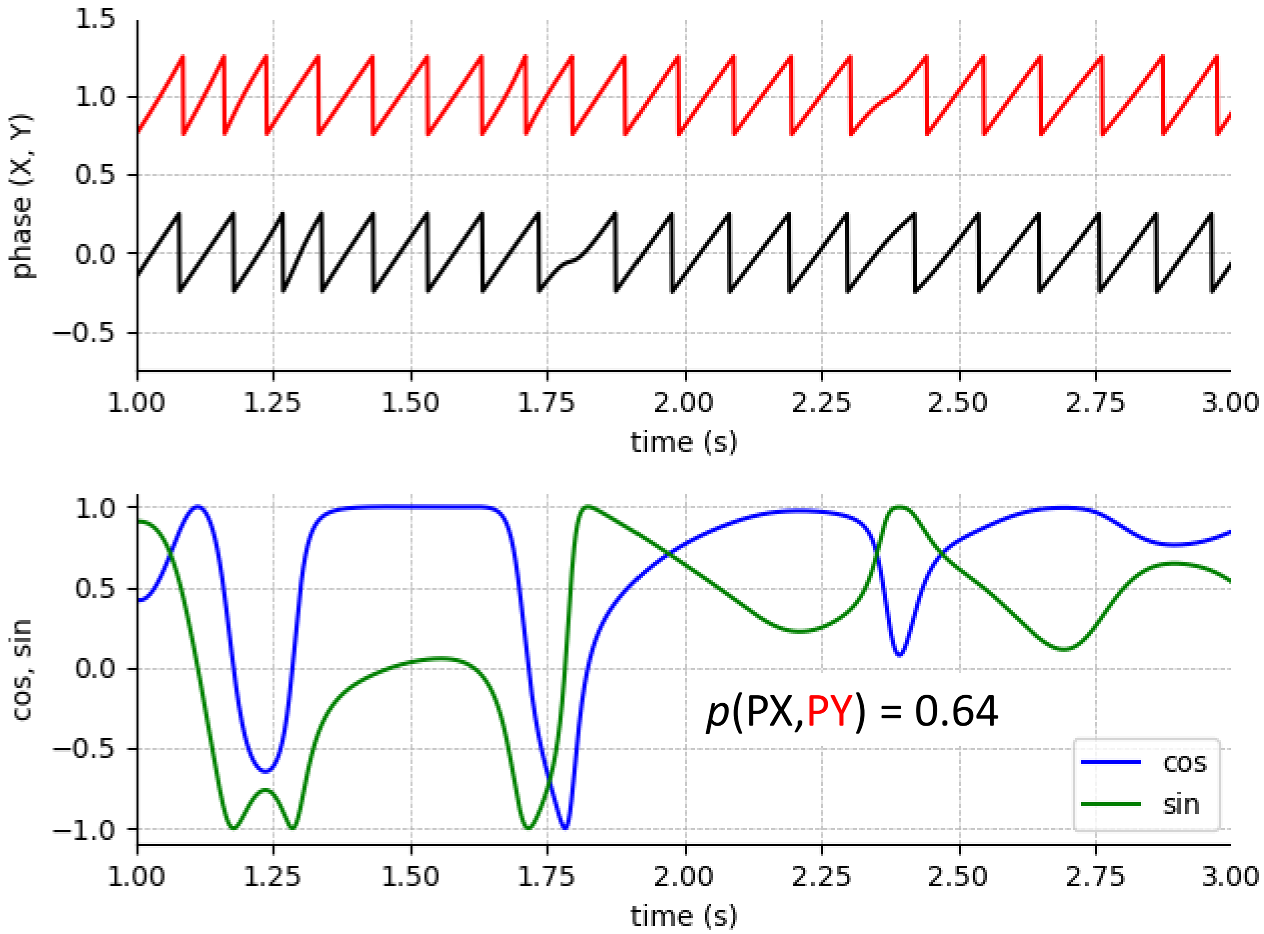


See, “L08_zero_lag_interactions_phase.py”

II. Phase-phase coupling (2/2)

Coupling between phases of two signals can be assessed via phase-locking value.

```
# phase
PX = np.angle(signal.hilbert(X))
PY = np.angle(signal.hilbert(Y))

# phase-locking value
p = np.abs(np.sum(np.exp(1j * (PX - PY))) / N)

# phase-locking value via sine
p = np.abs(np.sum(np.cos(PX - PY) +
              1j * np.sin(PX - PY)) / N)
```

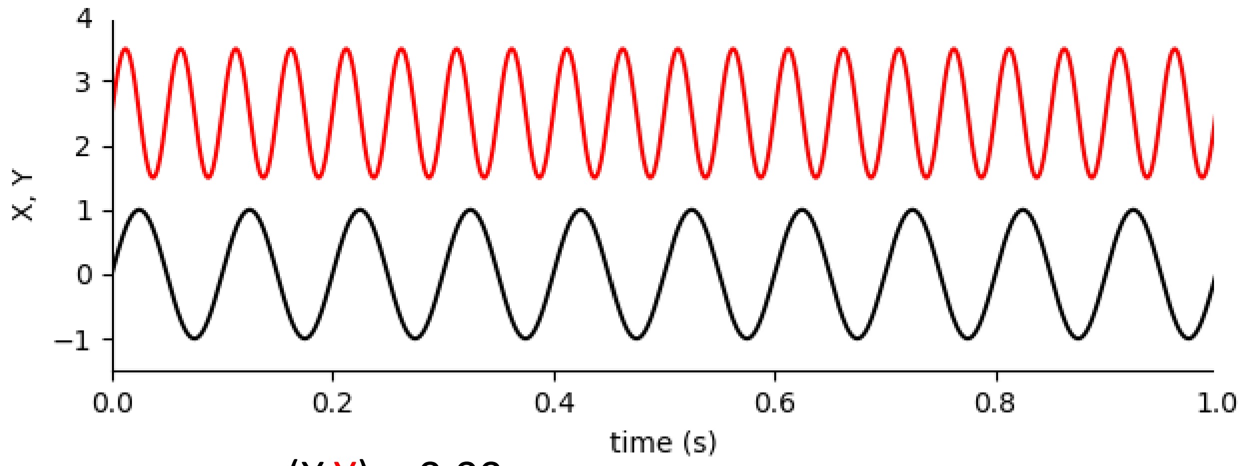


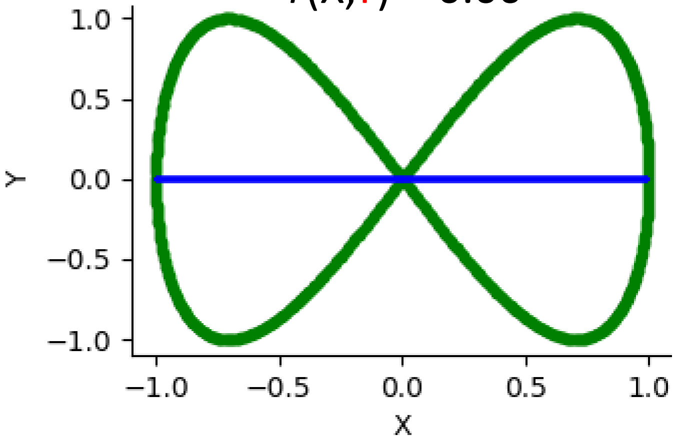
See, “L08_zero_lag_interactions_phase.py”

III. Cross-frequency phase-phase coupling (1/3)

How could we quantify interactions between two signals at different frequencies?

```
# signal
ratio = 2
f0 = 10
X = np.sin(2 * np.pi * f0 * t)
Y = np.sin(2 * np.pi * (f0 * ratio) * t)

# linear relationship
p = np.polyfit(X, Y, 1)
U = p[0] * X + p[1]

# correlation
r = np.corrcoef(X, Y)[0, 1]
```



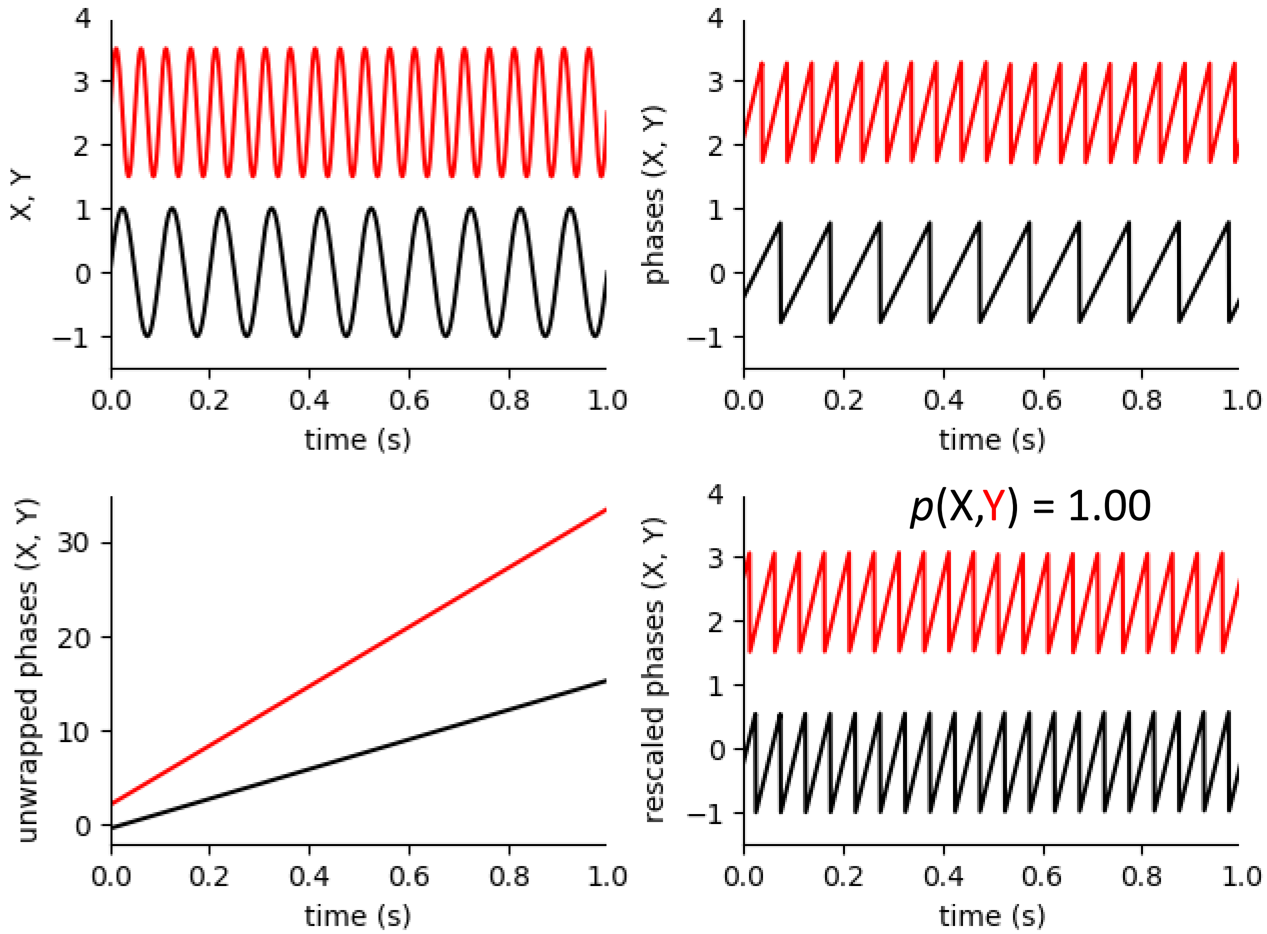


See, “L08_zero_lag_interactions_cf.py”

III. Cross-frequency phase-phase coupling (2/3)

Rescaling the phase time series could solve this problem.

```
# get phase
LF_ph = np.angle(signal.hilbert(X))
HF_ph = np.angle(signal.hilbert(Y))

# unwrap phases
LF_unwrap_ph = np.unwrap(LF_ph)
HF_unwrap_ph = np.unwrap(HF_ph)

# rescale phase
LF_res_ph = (LF_unwrap_ph % (2 * np.pi /
             ratio)) * ratio
HF_res_ph = (HF_unwrap_ph % (2 * np.pi))

# compute PLV
p = np.abs(np.sum(np.exp(1j *
(LF_res_phase - HF_res_phase))) / N)
```

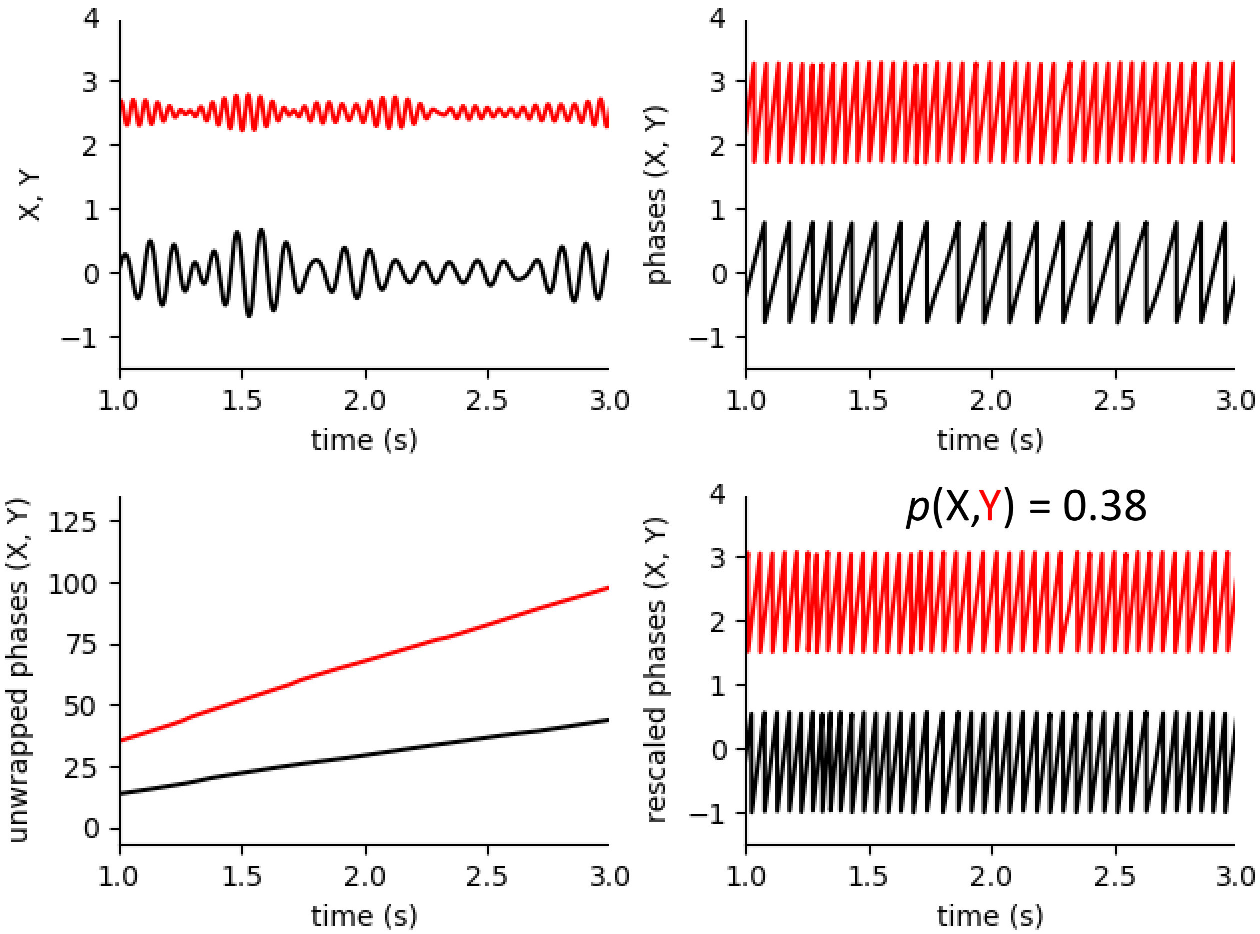


See, “L08_zero_lag_interactions_cf.py”

III. Cross-frequency phase-phase coupling (3/3)

How do it work for realistic data?

```
# get phase
LF_ph = np.angle(signal.hilbert(X))
HF_ph = np.angle(signal.hilbert(Y))

# unwrap phases
LF_unwrap_ph = np.unwrap(LF_ph)
HF_unwrap_ph = np.unwrap(HF_ph)

# rescale phase
LF_res_ph = (LF_unwrap_ph % (2 * np.pi /
            ratio)) * ratio
HF_res_ph = (HF_unwrap_ph % (2 * np.pi))

# compute phase-locking value
p = np.abs(np.sum(np.exp(1j *
(LF_res_phase - HF_res_phase))) / N)
```

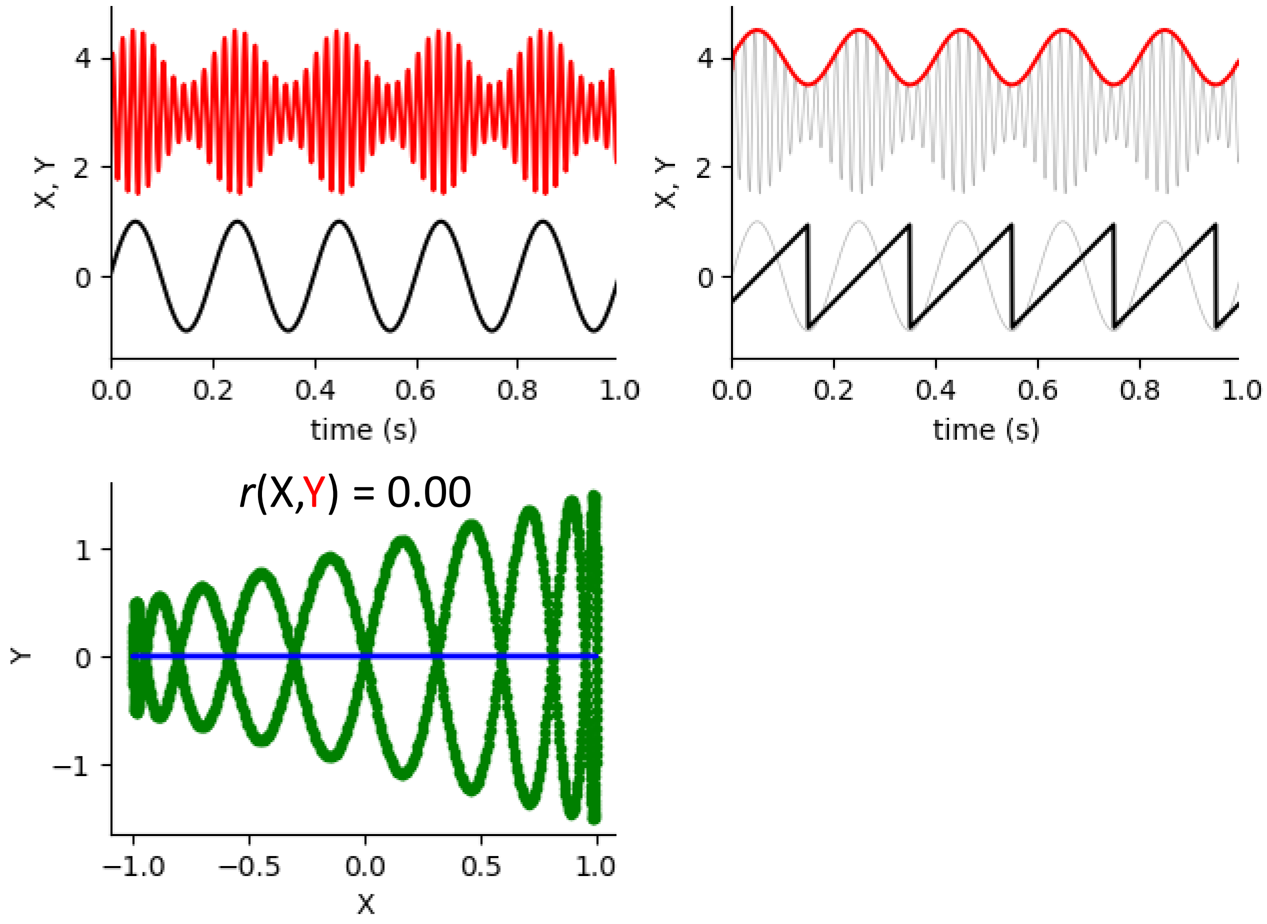


See, “L08_zero_lag_interactions_cf.py”

IV. Phase-amplitude modulation (1/2)

There are cases when phase of slow signal modulates amplitude of fast signal.

```
# signal
X = np.sin(2 * np.pi * 5 * t)
Y = np.sin(2 * np.pi * 50 * t) *
(1 + 0.5 * np.sin(2 * np.pi * 5 * t)) # AM

# linear relationship
p = np.polyfit(X, Y, 1)
U = p[0] * X + p[1]

# correlation
r = np.corrcoef(X, Y)[0, 1]
```

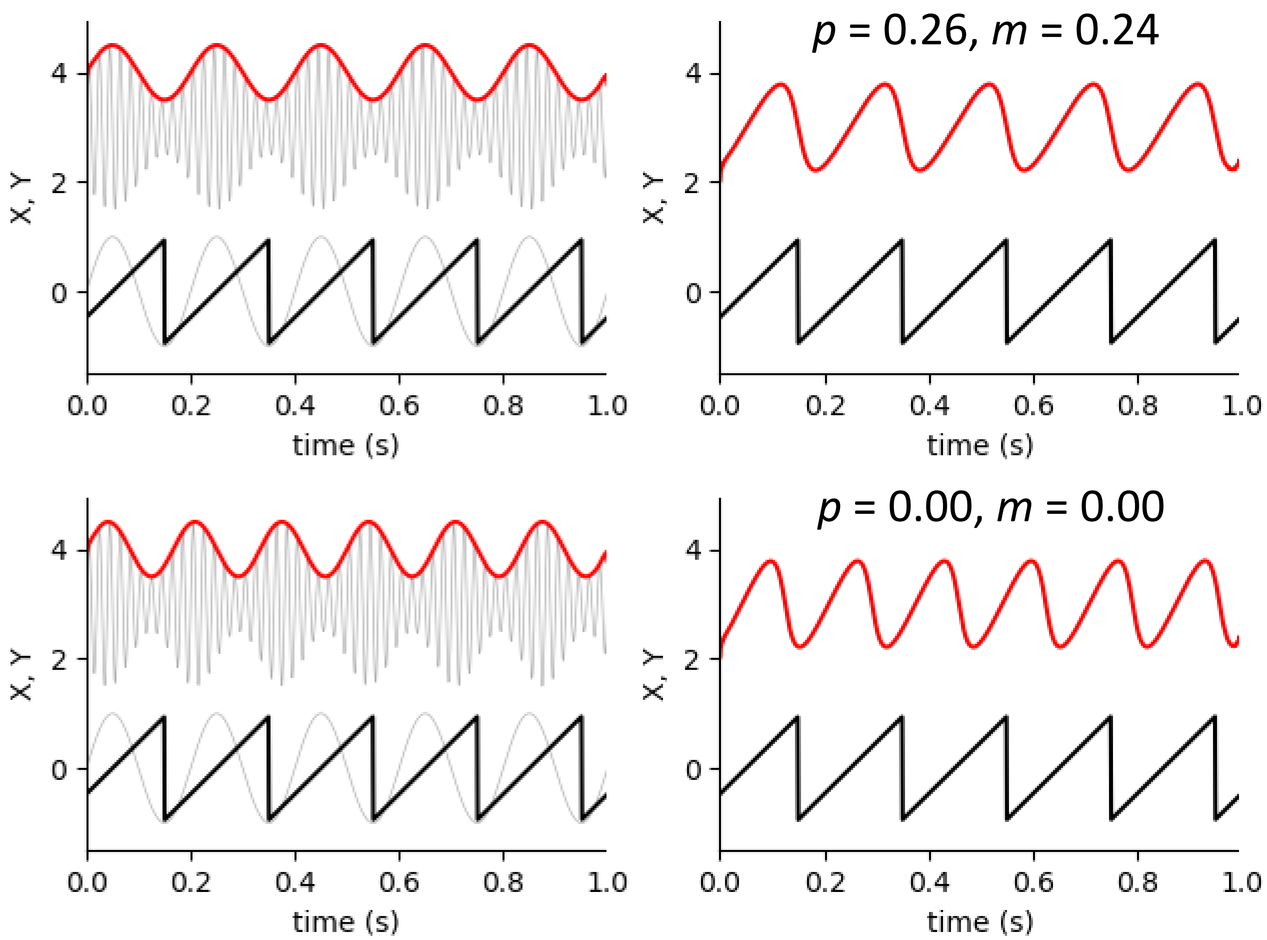


See, “L08_zero_lag_interactions_pac.py”

IV. Phase-amplitude modulation (2/2)

How could we detect phase-amplitude modulation?

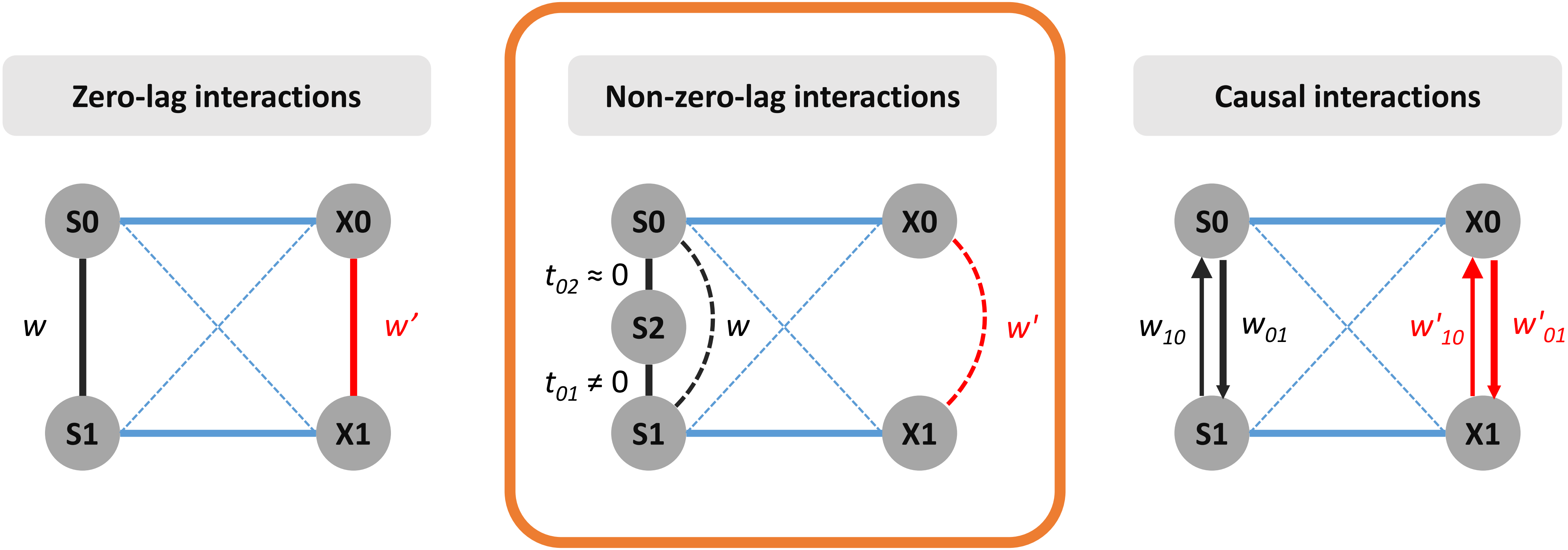
```
# phase of slow and amplitude of fast
PX = np.angle(signal.hilbert(X))
AY = np.abs(signal.hilbert(Y))

# * approach 1: intuitive
PA = np.angle(signal.hilbert(AY))
p = np.abs(np.sum(np.exp(1j * (PX - PA))) / N)

# * approach 2: modulation index
Z = AY * np.exp(1j * PX)
m = np.abs(np.mean(Z)) / np.sqrt(np.mean(AY**2))
```



See, “L08_zero_lag_interactions_pac.py”

Section 6. Non-zero-lag interactions

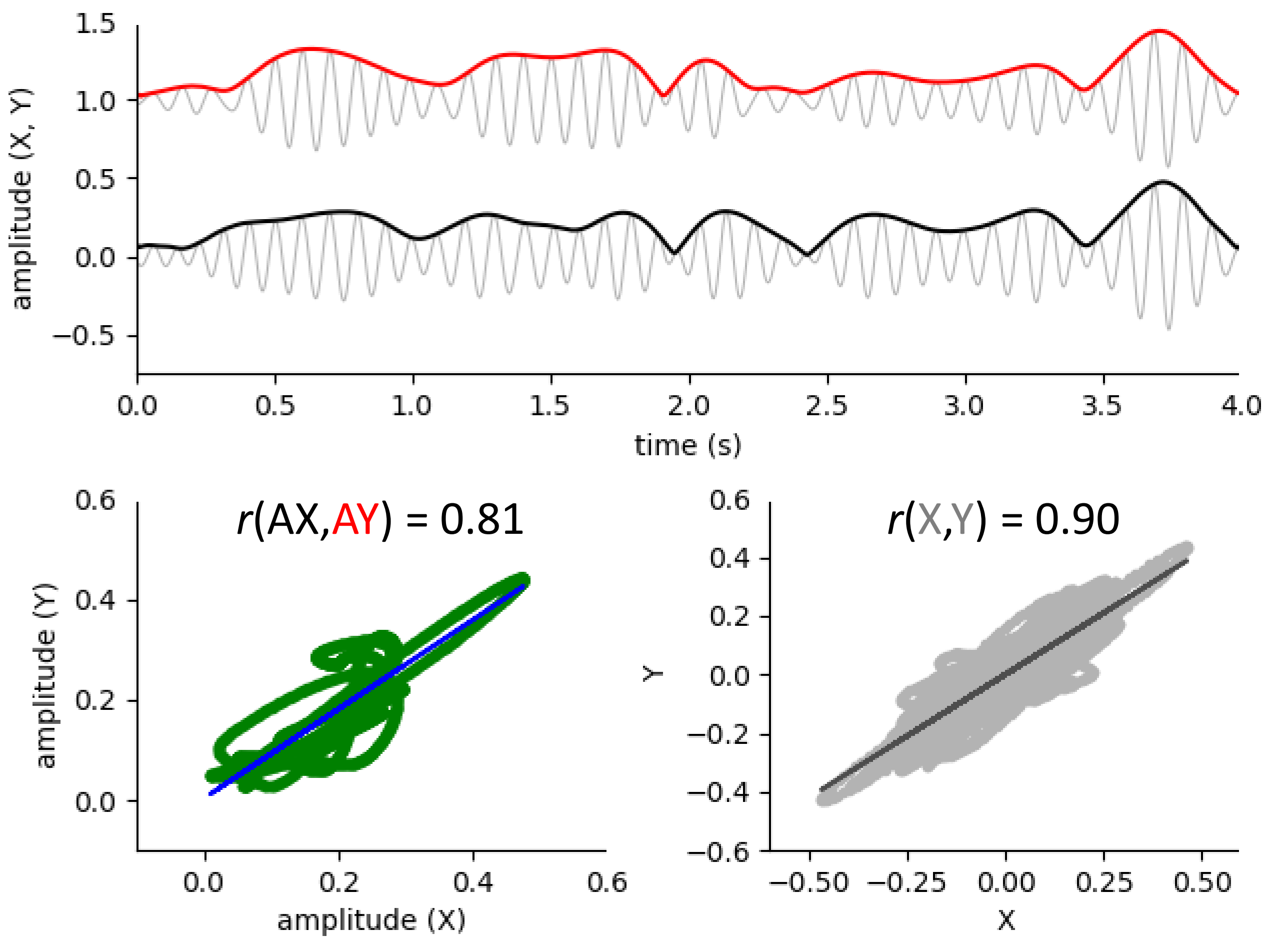
Interactions



I. Amplitude-amplitude correlations (1/2)

Correlation between amplitudes of two signals via third source.

```
# mixing matrix
A = np.array([[1.0, 0.0, 2.0], \
              [0.0, 1.0, 2.0], \
              [0.0, 0.0, 1.0]])

# amplitude
AX = np.abs(signal.hilbert(X))
AY = np.abs(signal.hilbert(Y))

# linear fit
p = np.polyfit(AX, AY, 1)
fAY = p[0] * AX + p[1]
p = np.polyfit(X, Y, 1)
fY = p[0] * X + p[1]

# correlation
rAA = np.corrcoef(AX, AY)[0, 1]
rXY = np.corrcoef(X, Y)[0, 1]
```

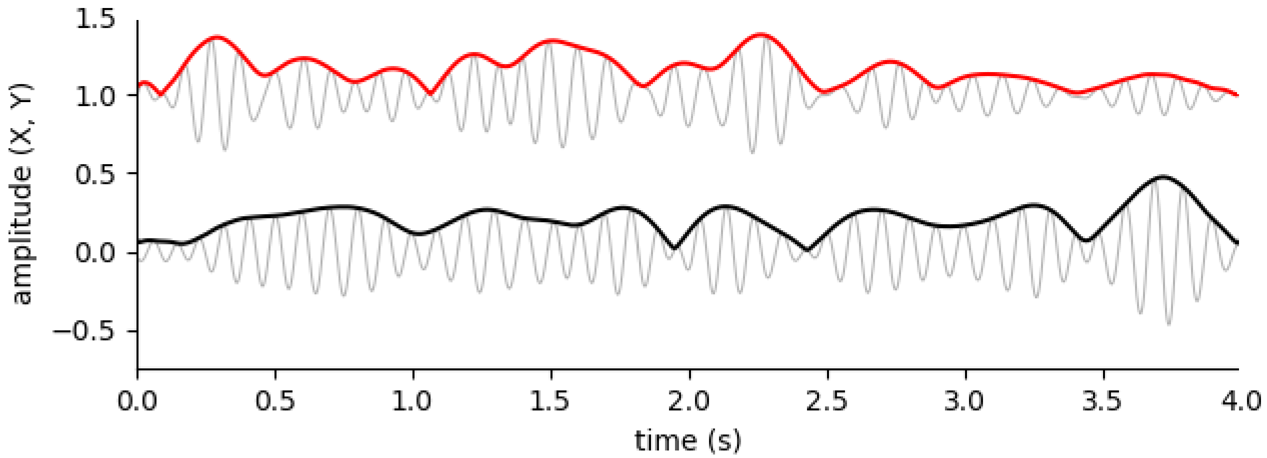


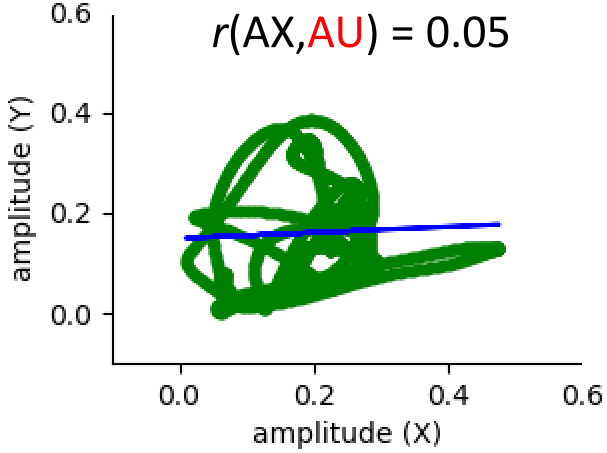
See, “L08_non_zero_lag_interactions_amplitude.py”

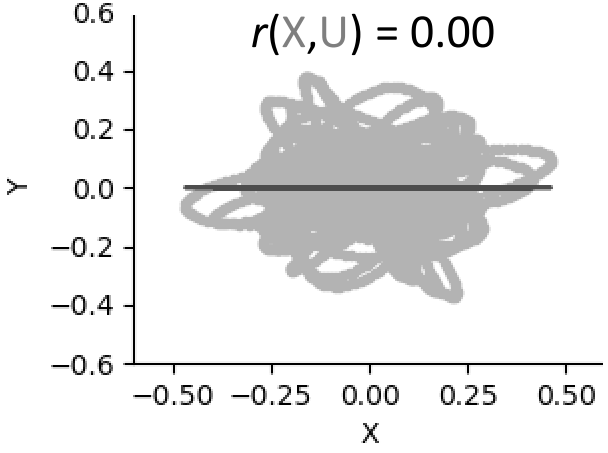
I. Amplitude-amplitude correlations (2/2)

Correlation between amplitudes of two signals with regressed out a common source.

```
# regression
b = np.sum((X / np.sum(X ** 2)) * Y)
U = Y - X * b

# amplitude
AX = np.abs(signal.hilbert(X))
AU = np.abs(signal.hilbert(U))

# linear fit
p = np.polyfit(AX, AU, 1)
AU = p[0] * AX + p[1]
p = np.polyfit(X, U, 1)
fU = p[0] * X + p[1]

# correlation
rAA = np.corrcoef(AX, AU)[0, 1]
rXU = np.corrcoef(X, U)[0, 1]
```





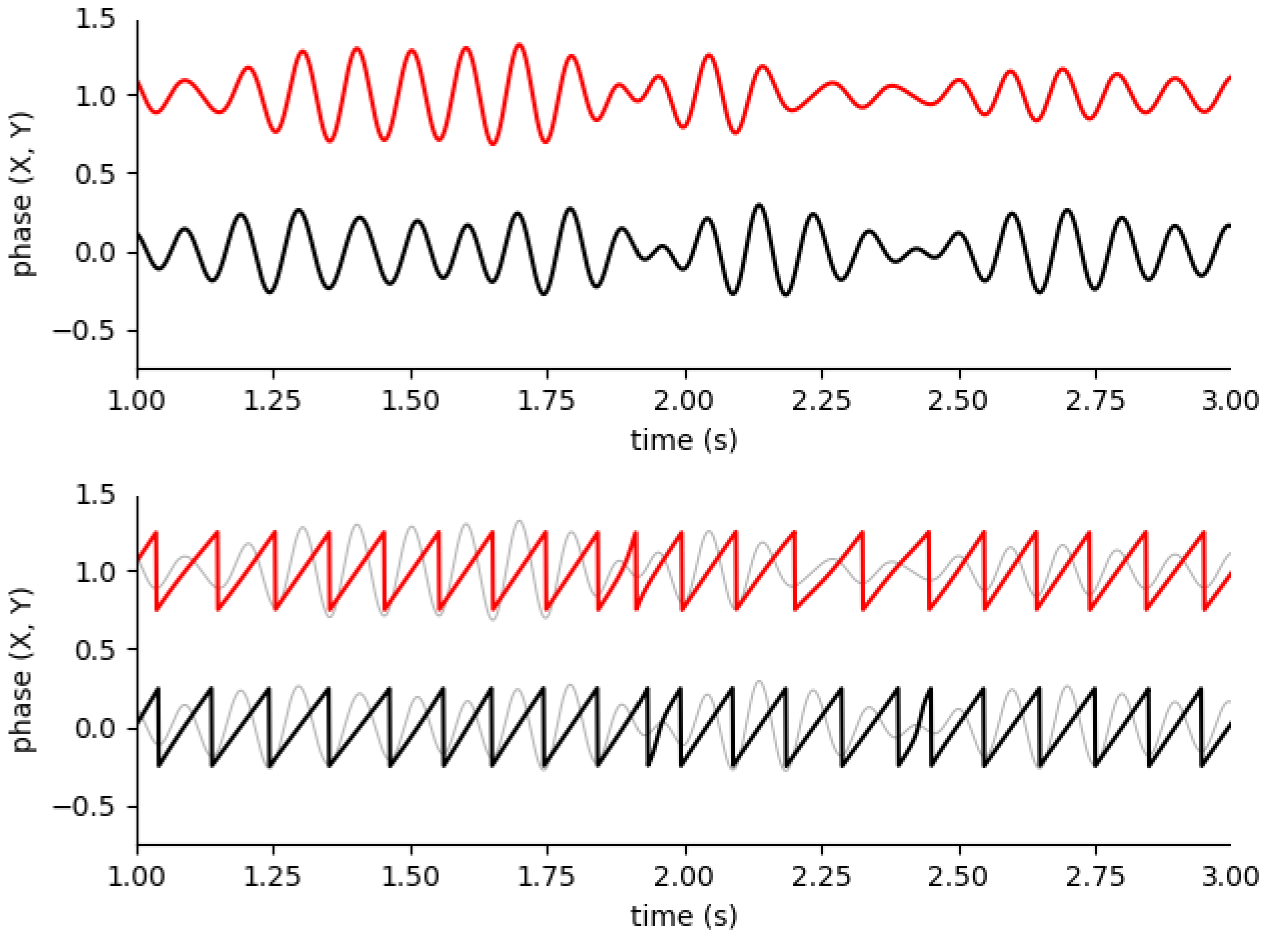


See, “L08_non_zero_lag_interactions_amplitude.py”

II. Phase-phase coupling (1/2)

Coupling between phases of two signals can be assessed via phase-locking value.

```
# phase
PX = np.angle(signal.hilbert(X))
PY = np.angle(signal.hilbert(Y))
```

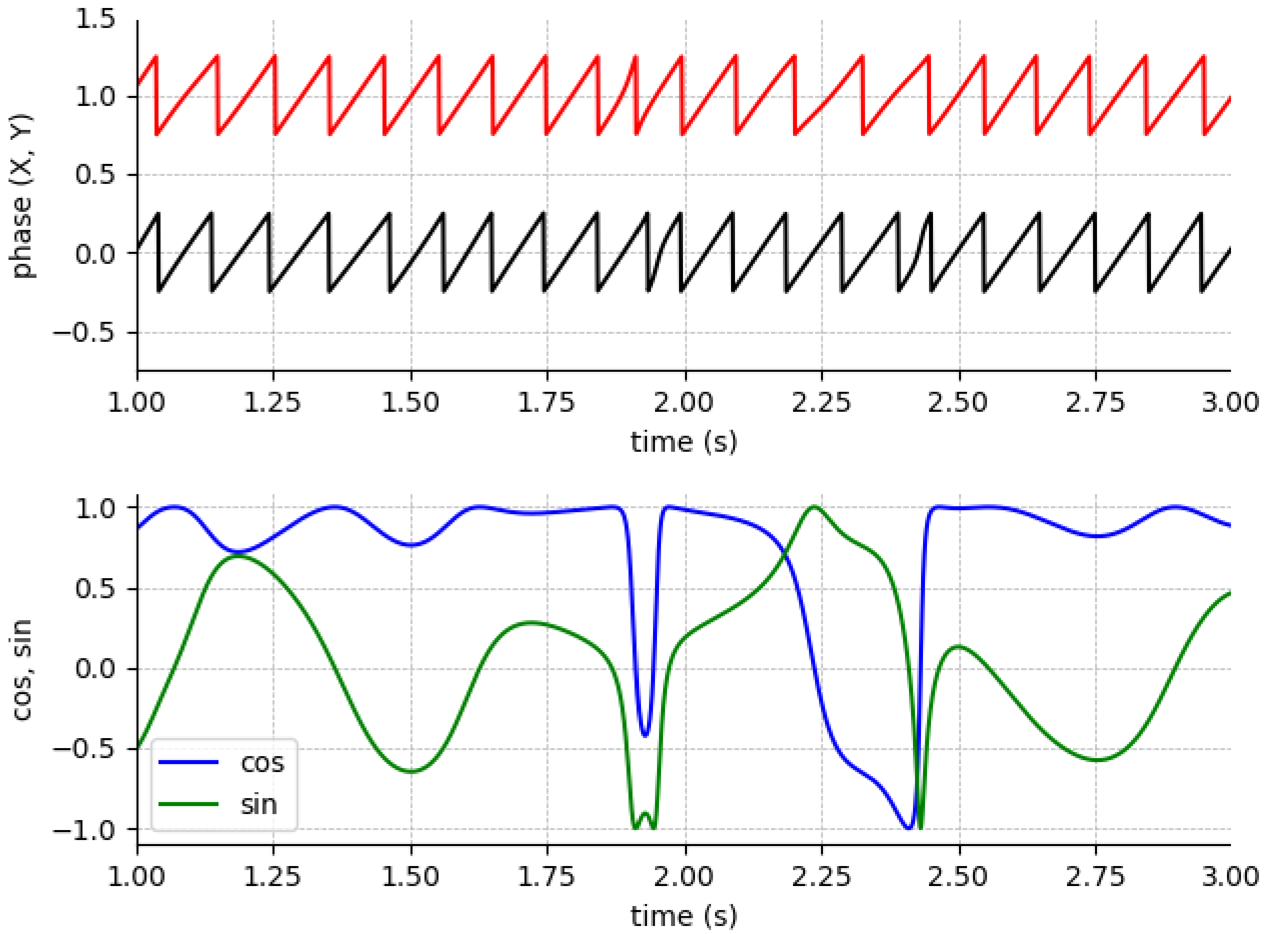


See, “L08_non_zero_lag_interactions_phase.py”

II. Phase-phase coupling (2/2)

Coupling between phases of two signals can be assessed via phase-locking value.

```
# phase
PX = np.angle(signal.hilbert(X))
PY = np.angle(signal.hilbert(Y))

# phase-locking value
p = np.abs(np.sum(np.exp(1j * (PX - PY))) / N)


# phase-locking value
pr = np.real(np.sum(np.exp(1j * (PX - PY)))
       / N)
pi = np.imag(np.sum(np.exp(1j * (PX - PY)))
       / N))
```



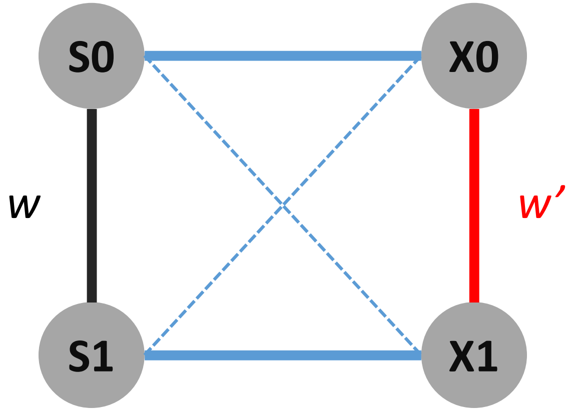
p(PX,PY) = 0.75 *p_re*(PX,PY) = 0.75 *p_im*(PX,PY) = 0.00

See, "L08_non_zero_lag_interactions_phase.py"

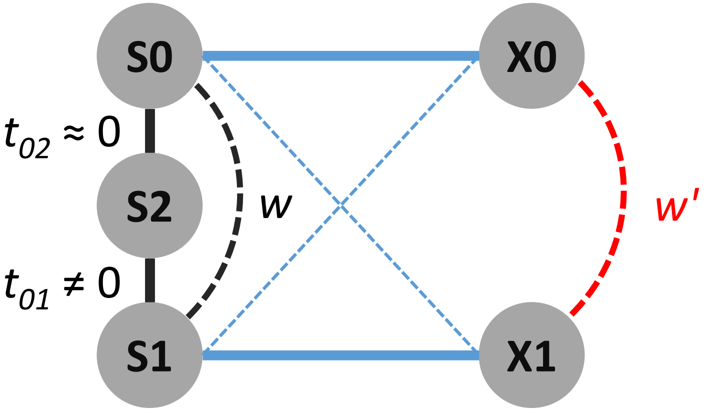
Section 7. Causal interactions

Interactions

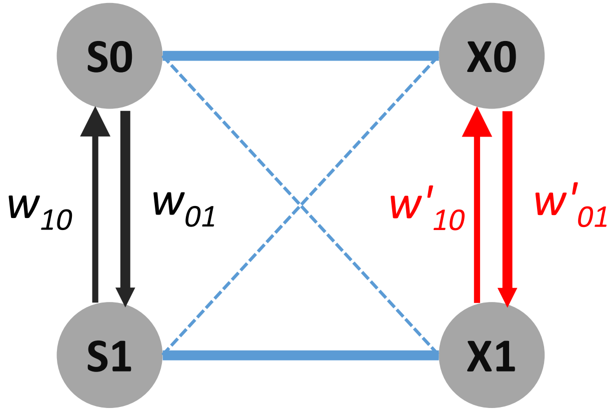
Zero-lag interactions



Non-zero-lag interactions



Causal interactions



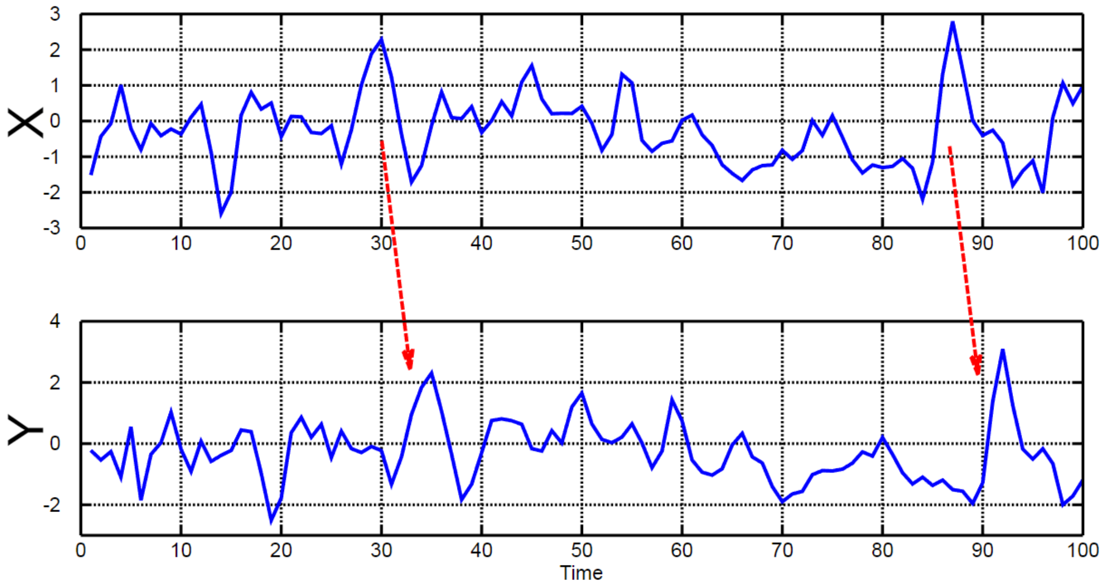
Granger causality introduction

The Granger causality test is a statistical hypothesis test for determining whether one time series is useful in **forecasting** another.

Intuition

We say that a variable **X** that evolves over time Granger-causes another evolving variable **Y**

if predictions of the value of **Y** based on its own past values and on the past values of X **are better** than predictions of Y based only on its own past values.

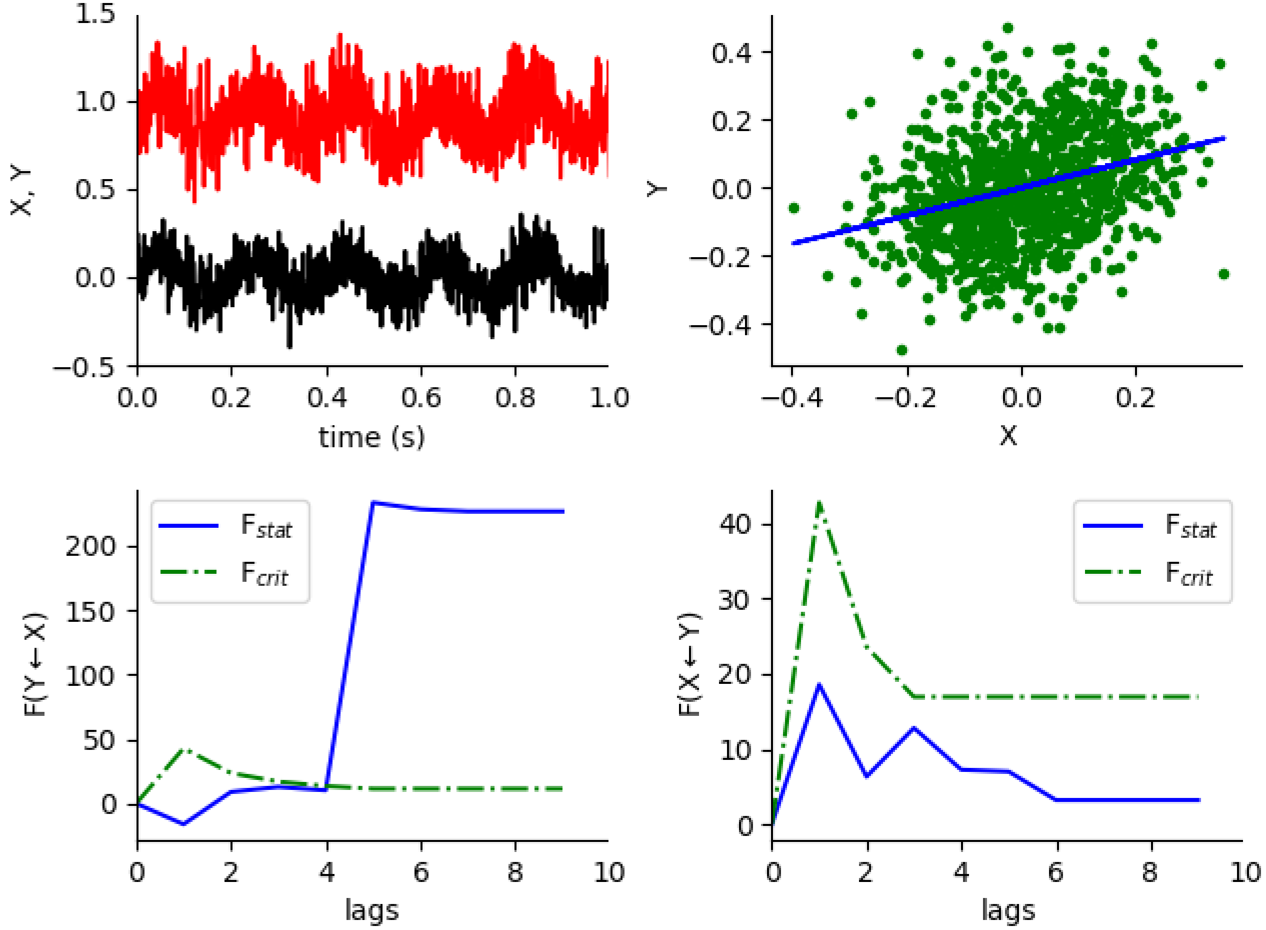


https://en.m.wikipedia.org/wiki/Granger_causality
http://www.scholarpedia.org/article/Granger_causality

Granger causality (1/2)

Does Y granger cause X?

```
# signal
lag = 5
X = 0.1 * np.sin(2 * np.pi * 5 * t) +
    0.1 * np.random.randn(N)
Y = np.concatenate((X[lag:], X[:lag])) +
    0.1 * np.random.randn(N)


# Granger causality
max_lags = 10

for max_lag in range(1, max_lags):
  # Y << X
  F_stat, F_crit = granger_causality(X, Y,
                   1e-10, max_lag)
  # X << Y
  F_stat, F_crit = granger_causality(Y, X,
                   1e-10, max_lag)
```

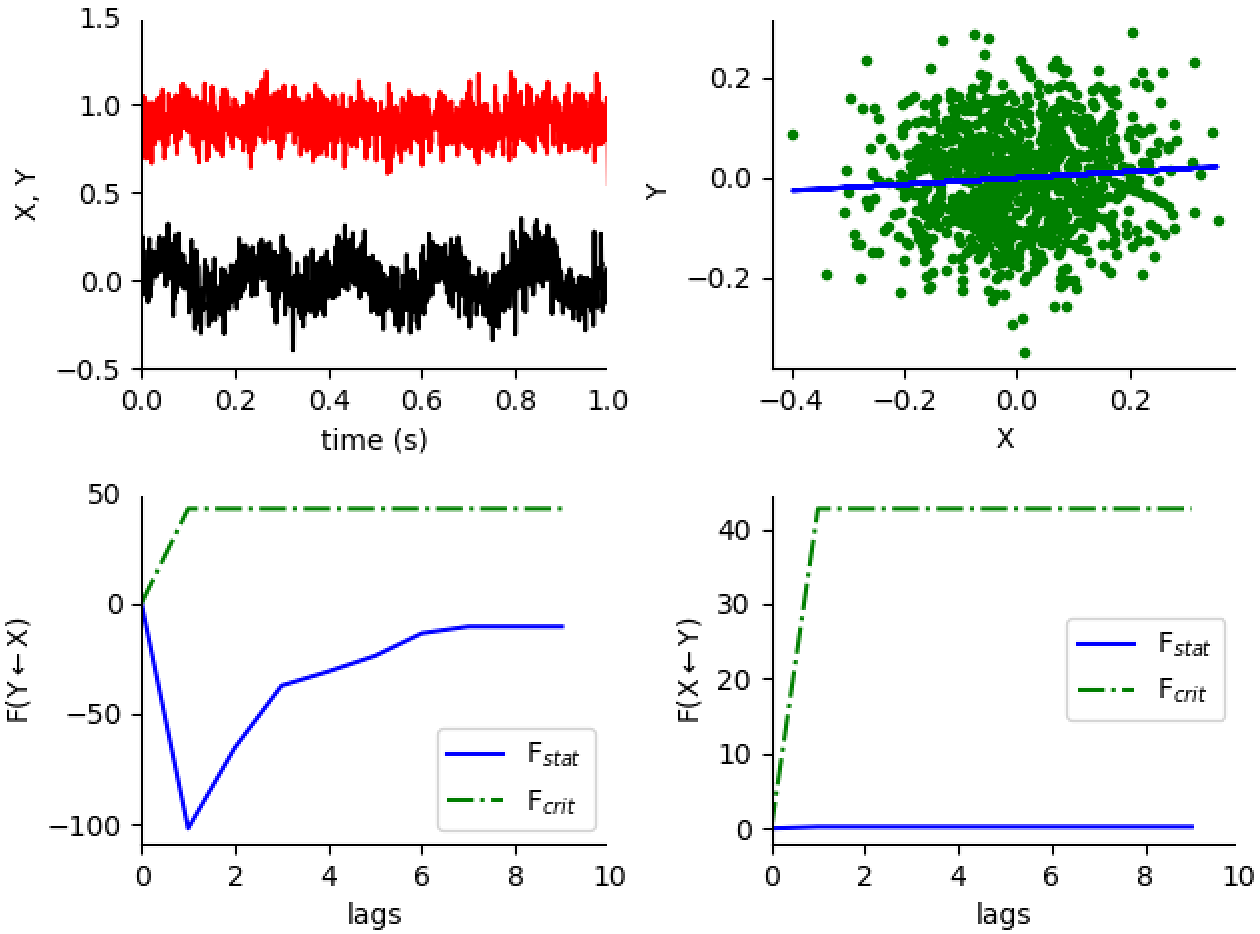


See, “L08_causal_interactions.py”

Granger causality (2/2)

Does Y granger cause X?

```
# signal
X = 0.1 * np.sin(2 * np.pi * 5 * t) +
    0.1 * np.random.randn(N)
Y = 0.1 * np.random.randn(N)


# Granger causality
max_lags = 10

for max_lag in range(1, max_lags):
  # Y << X
  F_stat, F_crit = granger_causality(X, Y,
                   1e-10, max_lag)
  # X << Y
  F_stat, F_crit = granger_causality(Y, X,
                   1e-10, max_lag)
```



See, “L08_causal_interactions.py”

Granger algorithm (1/3)

How does it work?

```
def granger_causality(x, y, alpha, max_lag):

  # fit restricted RSS model
  x_lag, RSS_R = fit_restricted_rss_model(x, max_lag)

  # fit full RSS model
  y_lag, RSS_F = fit_full_rss_model(x, y, x_lag, max_lag)

  # compare models
  F_stat, F_crit = compare_rss_models(x_lag, y_lag, RSS_R, RSS_F, alpha)
```

See, “L08_causal_interactions.py”

Granger algorithm (2/3)

How does it work?

```
def fit_restricted_rss_model(x, max_lag):

  # over lags
  for i in range(1, (max_lag+1)):
    Y = x[i:T]
    X = [(np.ones(T-i, 1), np.zeros(T-i, i)] # T = len(x)
    for j in range(1, (i+1)):
       X[:, j] = x[(i-j):(T-j)] # lags x


    # compute residuals
    b = np.linalg.lstsq(X, Y) # regression coefficients
    r = Y - np.dot(X, b[0])   # Y – X*b

    # compute the bayesian information criterion and init model
    BIC[i-1] = T*np.log(np.cov(r)*((T-2)/T)) + (i+1)*np.log(T)
    RSS_R[i-1] = np.cov(r)*(T-2) # error covariance

  # get best model
  x_lag = np.argmin(BIC)
```

See, “L08_causal_interactions.py”

Granger algorithm (3/3)

How does it work?

```
def fit_full_rss_model(x, y, x_lag, max_lag):

  # over lags
  for i in range(1, (max_lag+1)):
    Y = x[(i+x_lag):T]
    X = [(np.ones(T-(i+x_lag), 1), np.zeros(T-(i+x_lag), x_lag+i)]
    for j in range(1, (x_lag+1)):
      X[:, j] =         x[(i+x_lag-j):(T-j)] # lags x
    for j in range(1, (i+1)):
      X[:, (x_lag+j)] = y[(i+x_lag-j):(T-j)] # lags y
    # compute residuals
    b = np.linalg.lstsq(X, Y) # regression coefficients
    r = Y - np.dot(X, b[0])   # Y – X*b

    # compute the bayesian information criterion and init model
    BIC[i-1] = T*np.log(np.cov(r)*((T-2)/T)) + (i+1)*np.log(T)
    RSS_F[i-1] = np.cov(r)*(T-2) # error covariance

  # get best model
  y_lag = np.argmin(BIC)
```

See, “L08_causal_interactions.py”

VAR models

VAR(1) in two variables can be written in matrix form as,

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \begin{bmatrix} y_{1,t-1} \\ y_{2,t-1} \end{bmatrix} + \begin{bmatrix} e_{1,t} \\ e_{2,t} \end{bmatrix}$$

or, equivalently, as the following system of two equations

$$\begin{aligned} y_{1,t} &= c_1 + A_{1,1} y_{1,t-1} + A_{1,2} y_{2,t-1} + e_{1,t} \\ y_{2,t} &= c_2 + A_{2,1} y_{1,t-1} + A_{2,2} y_{2,t-1} + e_{2,t} \end{aligned}$$

Properties of the **VAR model** are usually summarized using Granger causality, impulse responses, and forecast error variance decompositions.

https://en.m.wikipedia.org/wiki/Vector_autoregression

Literature

- **Python programming language**

- http://www.scipy-lectures.org/, see "materials/L02_ScipyLectures.pdf"

- **Data analysis**

- Cohen M., "**Analyzing Neural Time Series Data: Theory and Practice**"