



수치해석

(2019학년도 1학기)

[12주/1차시 학습내용]: Ch. 16 Fourier Series (푸리에 급수)
를 통해 일반 선형 회귀를 학습한다.

Course 스케줄

- Ch 7: Optimization (최적화)
- Ch 14: Linear Regression (선형 회귀)
- Ch 15: Polynomial Regression (다항 회귀)
- Ch 14: Statistics Review (정규분포와 균등분포)
- **Ch 16: Fourier Series (일반 선형 회귀)**
- Ch 17: Polynomial Interpolation (다항식 보간법)
- Ch 19, 20, 21: Numerical Integration and Differentiation (수치 적분과 미분)
 - Ordinary Differential Equation, Newton-Cotes Formulas, The Trapezoidal Rule, Simpson's Rule, The Composite Trapezoidal Rule

Linear Regression과 Polynomial Regression의 비교

- Linear Regression: x 가 있는 직선 방정식을 사용하여 곡선 접합 (Curve Fitting) 수행
- Polynomial Regression: x^2, x^3 이 있는 곡선 방정식을 사용하여 곡선 접합 (Curve Fitting) 수행

14.3 LINEAR LEAST-SQUARES REGRESSION

Where substantial error is associated with data, the best curve-fitting strategy is to derive an approximating function that fits the shape or general trend of the data without necessarily matching the individual points. One approach to do this is to visually inspect the plotted data and then sketch a “best” line through the points. Although such “eyeball” approaches have commonsense appeal and are valid for “back-of-the-envelope” calculations, they are deficient because they are arbitrary. That is, unless the points define a perfect straight line (in which case, interpolation would be appropriate), different analysts would draw different lines.

To remove this subjectivity, some criterion must be devised to establish a basis for the fit. One way to do this is to derive a curve that minimizes the discrepancy between the data points and the curve. To do this, we must first quantify the discrepancy. The simplest example is fitting a straight line to a set of paired observations: $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$. The mathematical expression for the straight line is

$$y = a_0 + a_1x + e \quad (14.8)$$

where a_0 and a_1 are coefficients representing the intercept and the slope, respectively, and e is the error, or *residual*, between the model and the observations, which can be represented by rearranging Eq. (14.8) as

$$e = y - a_0 - a_1x \quad (14.9)$$

Thus, the residual is the discrepancy between the true value of y and the approximate value, $a_0 + a_1x$, predicted by the linear equation.

$$\min \sum_{i=1}^n (y_i - a_0 - a_1x_i)^2$$

$$\text{Slope } (b) = \frac{n \times \sum xy - (\sum x) \times (\sum y)}{n \times \sum x^2 - (\sum x)^2}$$

$$\text{Intercept } (a) = \frac{\sum y - b(\sum x)}{n}$$

$$\text{Regression } (y) = a + bx$$

국민대학교 소프트웨어학부

15.1 POLYNOMIAL REGRESSION

In Chap.14, a procedure was developed to derive the equation of a straight line using the least-squares criterion. Some data, although exhibiting a marked pattern such as seen in Fig. 15.1, are poorly represented by a straight line. For these cases, a curve would be better suited to fit the data. As discussed in Chap. 14, one method to accomplish this objective is to use transformations. Another alternative is to fit polynomials to the data using *polynomial regression*.

The least-squares procedure can be readily extended to fit the data to a higher-order polynomial. For example, suppose that we fit a second-order polynomial or quadratic:

$$y = a_0 + a_1x + a_2x^2 + e \quad (15.1)$$

361

$$\min \sum_{i=1}^n (y_i - a_0 - a_1x_i - a_2x_i^2)^2$$

Linear Regression과 Polynomial Regression의 비교

- Polynomial Regression (다항 회귀)를 유도하기

Least Squares Regression

$$S = \min \sum_{i=1}^n e_i^2 = \min \sum_{i=1}^n (y_i - a_0 - a_1 x_i)^2 \quad \text{Least Square Form}$$

Minimize the sum of the squares of the errors → optimization
Find min, max → differentiation

Calculate slope (a_1) and intercept (a_0) in order to minimize the sum of error
Multi variables (a_0, a_1) → Partial Derivative

$$\frac{\partial S}{\partial a_0} = \frac{\partial}{\partial a_0} \sum_{i=1}^n (y_i - a_0 - a_1 x_i)^2 = 0$$

$$\frac{\partial S}{\partial a_1} = \frac{\partial}{\partial a_1} \sum_{i=1}^n (y_i - a_0 - a_1 x_i)^2 = 0$$

$$n \left(\sum x_i \right) a_0 + n \left(\sum x_i^2 \right) a_1 = n \sum x_i y_i \quad \text{--- ③}$$

$$n \left(\sum x_i \right) a_0 + \left(\sum x_i \right)^2 a_1 = \sum x_i \sum y_i \quad \text{--- ④}$$

$$\text{③} - \text{④} \quad \left(n \left(\sum x_i^2 \right) - \left(\sum x_i \right)^2 \right) a_1 = n \sum x_i y_i - \sum x_i \sum y_i$$

$$a_1 = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \left(\sum x_i^2 \right) - \left(\sum x_i \right)^2}$$

$$a_0 = \frac{\sum y_i}{n} - \frac{\sum x_i}{n} a_1$$

선형 회귀(linear Regression)의 전개 과정

$$\min \sum_{i=1}^n (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2$$

For this case the sum of the squares of the residuals is

$$S_r = \sum_{i=1}^n (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2 \quad (15.2)$$

To generate the least-squares fit, we take the derivative of Eq. (15.2) with respect to each of the unknown coefficients of the polynomial, as in

$$\frac{\partial S_r}{\partial a_0} = -2 \sum (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

$$\frac{\partial S_r}{\partial a_1} = -2 \sum x_i (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

$$\frac{\partial S_r}{\partial a_2} = -2 \sum x_i^2 (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

These equations can be set equal to zero and rearranged to develop the following set of normal equations:

$$\begin{aligned} (n) a_0 + \left(\sum x_i \right) a_1 + \left(\sum x_i^2 \right) a_2 &= \sum y_i \\ \left(\sum x_i \right) a_0 + \left(\sum x_i^2 \right) a_1 + \left(\sum x_i^3 \right) a_2 &= \sum x_i y_i \\ \left(\sum x_i^2 \right) a_0 + \left(\sum x_i^3 \right) a_1 + \left(\sum x_i^4 \right) a_2 &= \sum x_i^2 y_i \end{aligned}$$

다항 회귀
(Polynomial Regression)의 전개 과정

Polynomial Regression (행렬 전개)

- Polynomial Regression (다항 회귀)를 유도하기
- 다항 회귀(Polynomial Regression)의 전개 과정에서 필요한 행렬 전개

For this case the sum of the squares of the residuals is

$$S_r = \sum_{i=1}^n (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2 \quad (15.2)$$

To generate the least-squares fit, we take the derivative of Eq. (15.2) with respect to each of the unknown coefficients of the polynomial, as in

$$\frac{\partial S_r}{\partial a_0} = -2 \sum (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

$$\frac{\partial S_r}{\partial a_1} = -2 \sum x_i (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

$$\frac{\partial S_r}{\partial a_2} = -2 \sum x_i^2 (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

These equations can be set equal to zero and rearranged to develop the following set of normal equations:

$$\begin{aligned} (n)a_0 + (\sum x_i) a_1 + (\sum x_i^2) a_2 &= \sum y_i \\ (\sum x_i) a_0 + (\sum x_i^2) a_1 + (\sum x_i^3) a_2 &= \sum x_i y_i \\ (\sum x_i^2) a_0 + (\sum x_i^3) a_1 + (\sum x_i^4) a_2 &= \sum x_i^2 y_i \end{aligned}$$



$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{bmatrix}$$

다변수 선형 회귀 (Multiple Linear Regression)

- 두 개 변수(x, y 또는 x_1, x_2)의 1차 방정식을 사용하여 곡선 적합 수행

15.2 MULTIPLE LINEAR REGRESSION

Another useful extension of linear regression is the case where y is a linear function of two or more independent variables. For example, y might be a linear function of x_1 and x_2 , as in

$$y = a_0 + a_1x_1 + a_2x_2 + e$$

Such an equation is particularly useful when fitting experimental data where the variable being studied is often a function of two other variables. For this two-dimensional case, the regression “line” becomes a “plane” (Fig. 15.3).

As with the previous cases, the “best” values of the coefficients are determined by formulating the sum of the squares of the residuals:

$$S_r = \sum_{i=1}^n (y_i - a_0 - a_1x_{1,i} - a_2x_{2,i})^2 \quad (15.4)$$

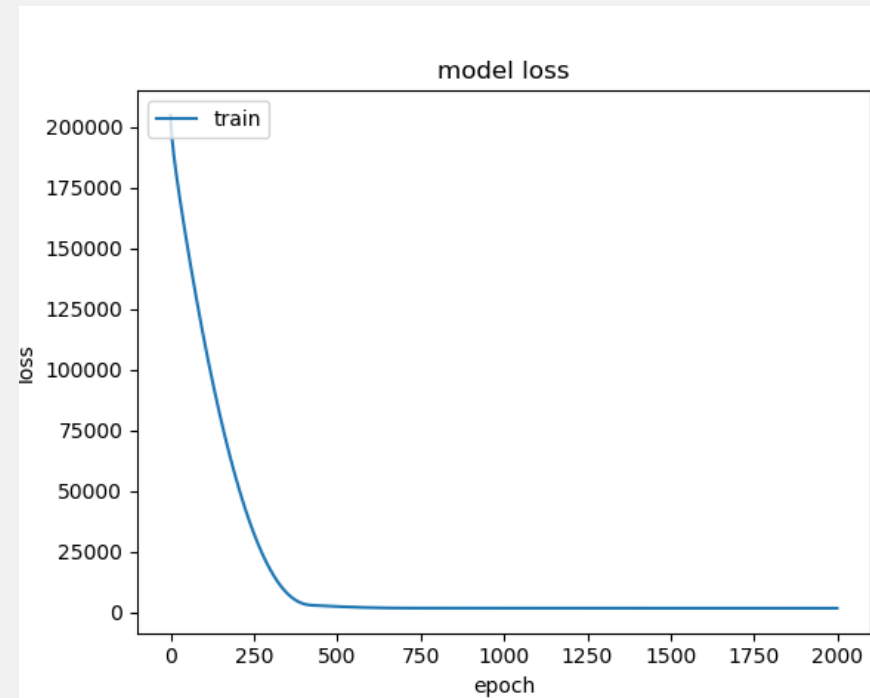
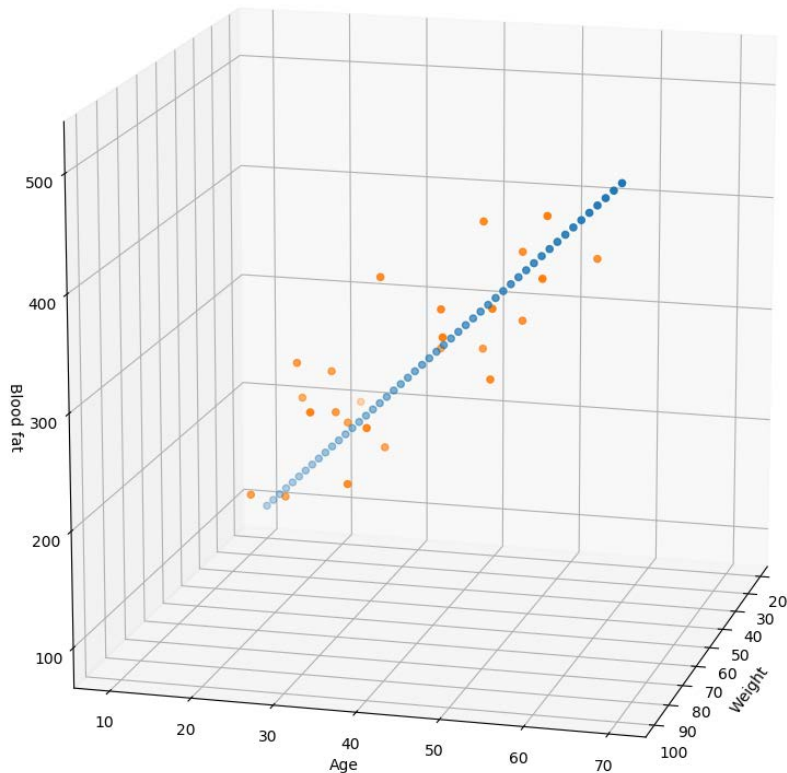
and differentiating with respect to each of the unknown coefficients:

$$\begin{aligned} \frac{\partial S_r}{\partial a_0} &= -2 \sum (y_i - a_0 - a_1x_{1,i} - a_2x_{2,i}) \\ \frac{\partial S_r}{\partial a_1} &= -2 \sum x_{1,i} (y_i - a_0 - a_1x_{1,i} - a_2x_{2,i}) \\ \frac{\partial S_r}{\partial a_2} &= -2 \sum x_{2,i} (y_i - a_0 - a_1x_{1,i} - a_2x_{2,i}) \end{aligned}$$

$$y = a_0 + a_1x_1 + a_2x_2 + e$$

다변수 선형 회귀 (Multiple Linear Regression)

- 몸무게 (변수 x_s), 100kg, 나이 (변수 y_s), 40세인 사람의 혈중 지방(변수 z_s)을 예측을 실시한다



General Linear Regression (일반 선형 회귀)

- z_1, z_2, \dots, z_n 의 n 개 함수로 곡선 접합 수행

15.3 GENERAL LINEAR LEAST SQUARES

In the preceding pages, we have introduced three types of regression: simple linear, polynomial, and multiple linear. In fact, all three belong to the following general linear least-squares model:

$$y = a_0 z_0 + a_1 z_1 + a_2 z_2 + \cdots + a_m z_m + e \quad (15.7)$$

where z_0, z_1, \dots, z_m are $m + 1$ basis functions. It can easily be seen how simple linear and multiple linear regression fall within this model—that is, $z_0 = 1, z_1 = x_1, z_2 = x_2, \dots, z_m = x_m$. Further, polynomial regression is also included if the basis functions are simple monomials as in $z_0 = 1, z_1 = x, z_2 = x^2, \dots, z_m = x^m$.

Note that the terminology “linear” refers only to the model’s dependence on its parameters—that is, the a ’s. As in the case of polynomial regression, the functions themselves can be highly nonlinear. For example, the z ’s can be sinusoids, as in

$$y = a_0 + a_1 \cos(\omega x) + a_2 \sin(\omega x)$$

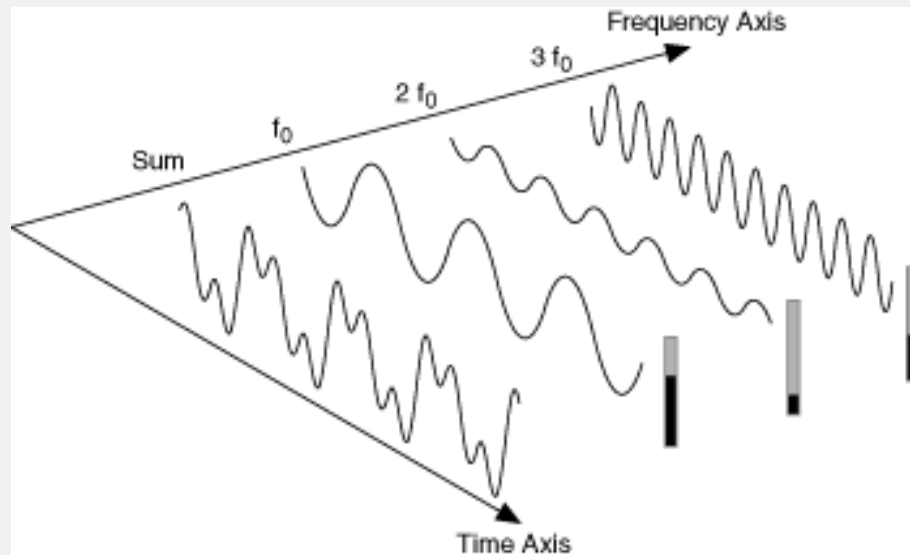
Curve Fitting을 위한 Regression 방법 비교

- Linear Regression (선형 회귀)
 - 1개 변수 x 의 일차 방정식을 사용하여 곡선 접합 수행
 - Numpy, scikit-learn, tensorflow 메소드 이용
- Polynomial Regression (다항 회귀)
 - 1개 변수의 x 의 2차, 3차 방정식(x^2, x^3)을 사용하여 곡선 접합 수행
 - Numpy, scikit-learn, tensorflow 메소드 이용
- Multiple Linear Regression (다변수 선형 회귀)
 - 두 개 이상의 변수 x, y 또는 x_1, x_2 의 일차 방정식을 사용하여 곡선 접합 수행
 - scikit-learn, tensorflow 메소드 이용
- General Linear Regression (일반 선형 회귀)
 - 두 개 이상의 함수 z_1, z_2, \dots, z_n 의 n 개 일차 함수로 곡선 접합 수행

General Linear Regression:

두 개 이상의 함수, 각 함수는 1차원 변수 ($z_1(x^1), z_2(x^1), \dots, z_n(x^1)$)를 사용하여 곡선 접합 수행

- Fourier Analysis는 General Linear Regression 중의 하나다.
- Sinusoids 함수로 이진 사각형파(Square Wave)를 곡선접합을 할 수 있다.
- Sinusoids 함수 여러 개를 조합하면 이진 사각형파를 만들어 낼 수 있음을 보여주는 것이 Fourier Analysis이다.



15.3 GENERAL LINEAR LEAST SQUARES

In the preceding pages, we have introduced three types of regression: simple linear, polynomial, and multiple linear. In fact, all three belong to the following general linear least-squares model:

$$y = a_0 z_0 + a_1 z_1 + a_2 z_2 + \dots + a_m z_m + e \quad (15.7)$$

where z_0, z_1, \dots, z_m are $m+1$ basis functions. It can easily be seen how simple linear and multiple linear regression fall within this model—that is, $z_0 = 1, z_1 = x_1, z_2 = x_2, \dots, z_m = x_m$. Further, polynomial regression is also included if the basis functions are simple monomials as in $z_0 = 1, z_1 = x, z_2 = x^2, \dots, z_m = x^m$.

Note that the terminology “linear” refers only to the model’s dependence on its parameters—that is, the a ’s. As in the case of polynomial regression, the functions themselves can be highly nonlinear. For example, the z ’s can be sinusoids, as in

$$y = a_0 + a_1 \cos(\omega x) + a_2 \sin(\omega x)$$

Fourier Series

Fourier Analysis는 General Linear Regression 중
의 하나다.

Fourier Series

- 주기 함수, **사각형파**를 삼각함수로 분해한 시리즈
- 프랑스의 과학자이자 수학자인 조제프 푸리에가 도입함
- 천문학에서는 분광기를 통해 별빛의 주파수를 분해하여 별을 이루는 화학 물질을 알아내는 데 쓰임
- 네트워크에서는 전송해야 하는 데이터 신호의 스펙트럼을 이용하여 네트워크 시스템 설계를 최적화하는 데 쓰임
- 진동 해석, 음향학, 광학, 신호 처리와 화상처리, 데이터 압축에 사용됨

Fourier Series

- Function $s(x)$ (in red) is a sum of six sine functions of different amplitudes and harmonically related frequencies.
- Their summation is called a Fourier series.
- The Fourier transform, $S(f)$ (in blue), which depicts amplitude vs frequency, reveals the 6 frequencies and their amplitudes.



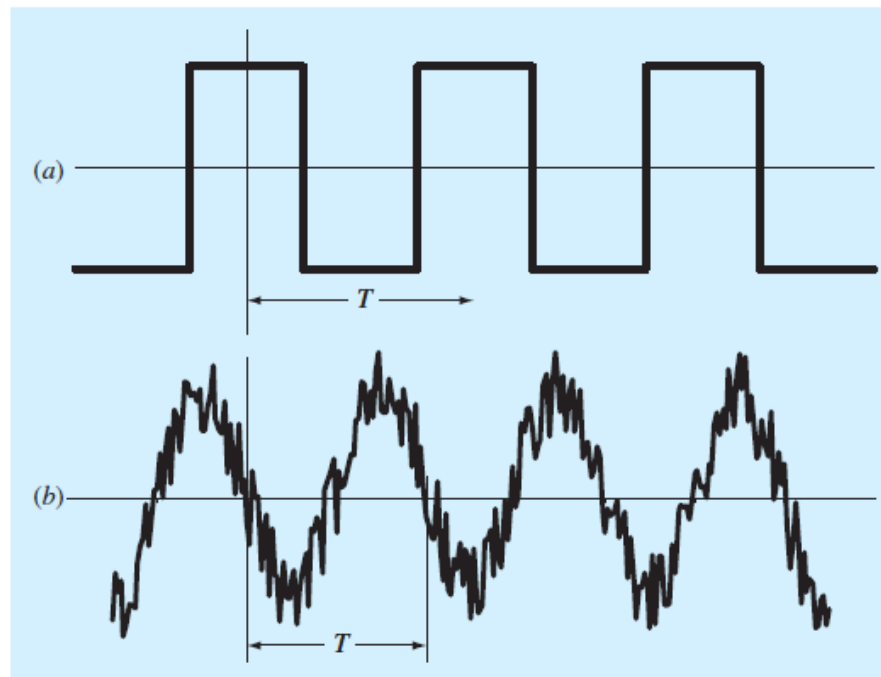
https://en.wikipedia.org/wiki/Fourier_series

Nature Signal and Computer Signal

- Analog Signal: Trigonometric function (Sine, Cosine)
- Digital Signal: Square Waves, bits

FIGURE 16.1

Aside from trigonometric functions such as sines and cosines, periodic functions include idealized waveforms like the square wave depicted in (a). Beyond such artificial forms, periodic signals in nature can be contaminated by noise like the air temperatures shown in (b).

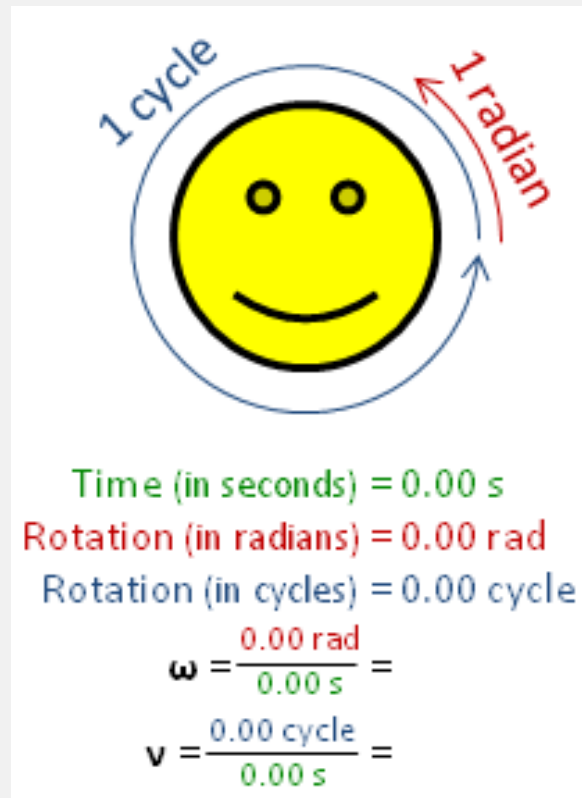


Sinusoidal (정현파)

- Any waveform that can be described as a sine or cosine
- $f(t) = A_0 + C_1 \sin(\omega_0 t + \theta) = A_0 + C_1 \sin(2\pi f_0 t + \theta)$
- C_1 = The amplitude (진폭)
 - the peak deviation of the function from zero.
- $\omega_0 = 2\pi f_0$ = Angular frequency (각 주파수)
 - the rate of change of the function argument in units of radians per second
 - 2π [radians] f_0 [one per second] = [radians per second] = **rad/s**
- f_0 = Ordinary Frequency (일반 주파수)
 - the number of oscillations (cycles) that occur each second of time, in unit of hertz [Hz] = $1/T$ [one per second]
- θ = Phase (위상)
 - specifies (in radians) where in its cycle the oscillation is at $t = 0$.

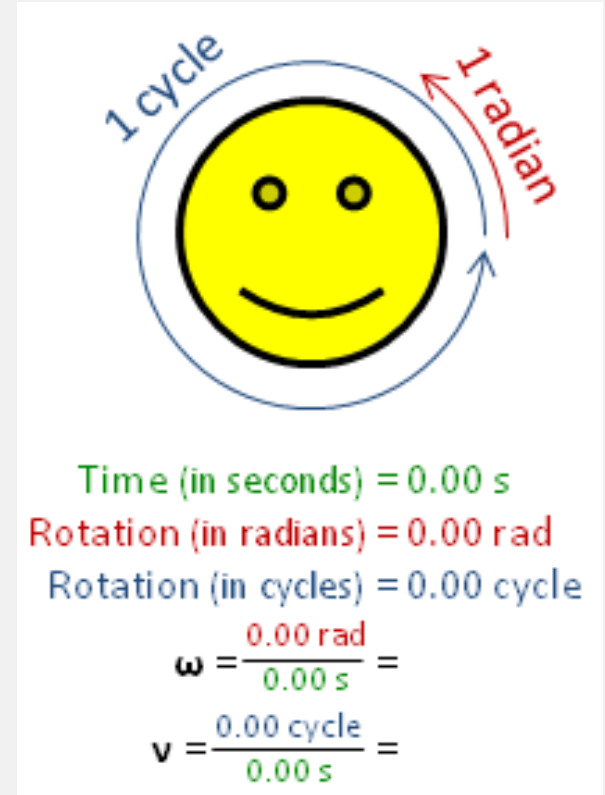
Radian per second ω_0

- Angular frequency ω_0 (in radians per second), is larger than frequency f_0 (ν , in cycles per second, also called Hz), by a factor of 2π , because 2π rad/s corresponds to 1 Hz



[1rad/sec] 각주파수 ω_0 와 [1Hz] 일반주파수 f_0

- 원 한 바퀴는 2π *rad, 또는 1cycle 로 표현할 수 있다.
- 각속도로 원 한 바퀴를 1초에 간다고 생각하면, 2π *rad/1초로 표현할 수 있고,
- 일반 주파수로는 원 한 바퀴를 1초에 간다고 생각하면 1cycle/1초로 표시할 수 있다.
- 이때 1cycle/1초는 일반 주파수 단위로는 1 [1 Hz] 이며, 2π *rad/1초는 각속도 단위로는 2π [1 rad/sec] 가 된다.
- 따라서, 1 [1Hz]= 2π [1rad/sec]가 되며, [1Hz]는 [1rad/sec]의 2π 배 크다



https://en.wikipedia.org/wiki/Radian_per_second

Draw Sinusoidal

정현파를 시각화한다.

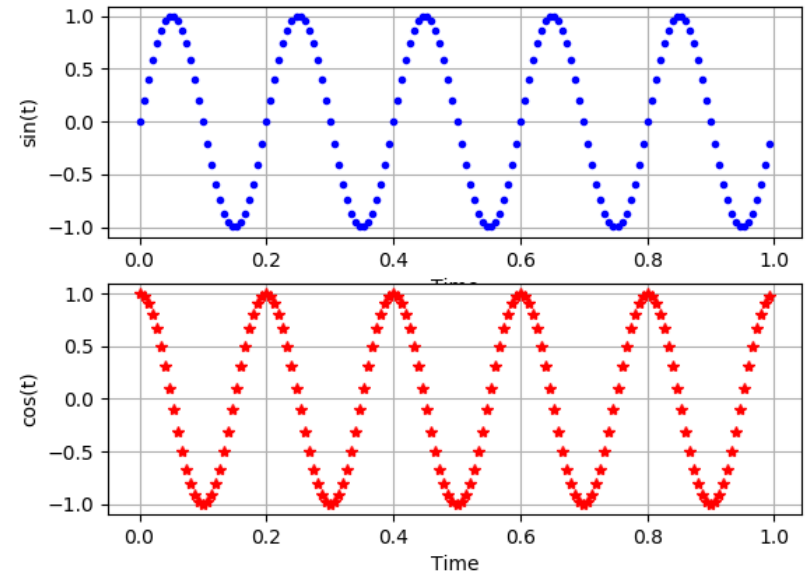
<https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/fourier.py>

Draw Sinusoidal (sin(t) and cos(t))

- 정현파 (Sinusoidal) 그리기

```
import numpy as np
import matplotlib.pyplot as plt
Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval or sampling time
# 0.006666666666666667
t = np.arange(0,1,Ts) # time vector
# 0에서 1초 사이에 0.006 초 간격의 (Ts개) 시간을
# 만들어라

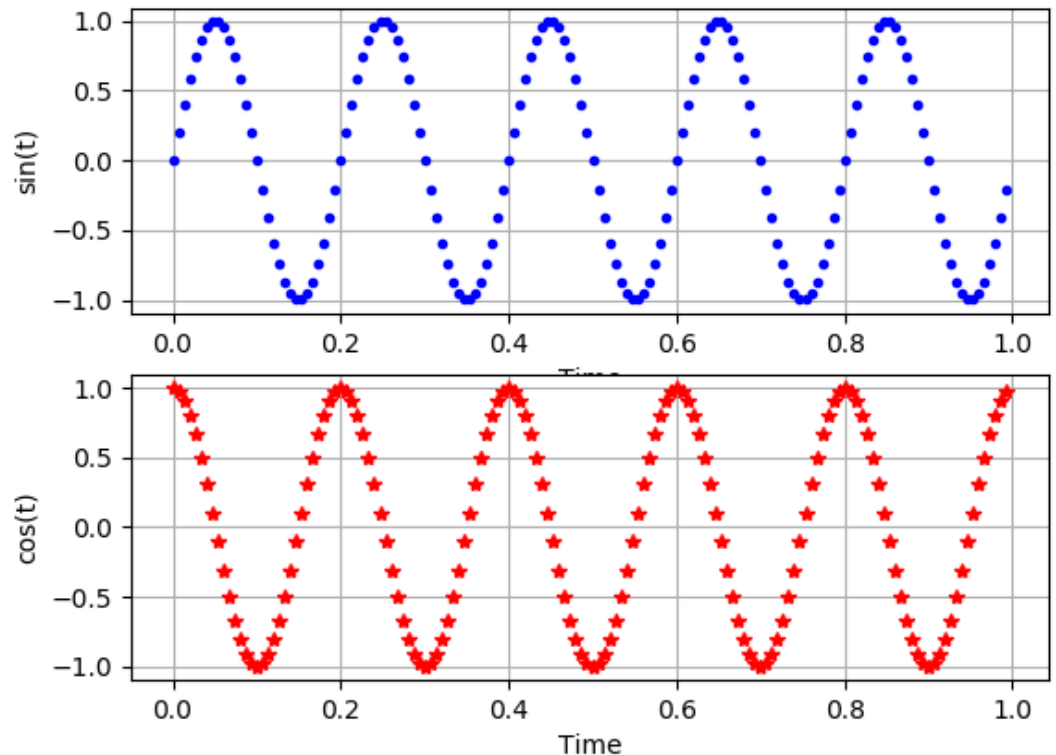
# np.size(t)=150
ff = 5 # frequency of the signal
y = np.sin(2*np.pi*ff*t)
# sin(2*np.pi)를 1초(t) 안에 5개 (ff)의 2*pi 주기의
# sin 파를
# 만들어라. 그런데, 1초 안에는 150개의 점을 찍어라.
# np.size(y)=150
# compare sine and cosine
y1= np.cos(2*np.pi*ff*t)
```



Draw Sinusoidal (sin(t) and cos(t))

- 정현파 (Sinusoidal) 그리기
- The graphs of the sine and cosine functions are **sinusoids of different phases**.

```
plt.figure(100)
plt.subplot(2,1,1)
plt.plot(t,y, 'b.')
# np.size(t) = 150
# np.size(y) = 150
plt.xlabel('Time')
plt.ylabel('sin(t)')
plt.grid()
plt.subplot(2,1,2)
plt.plot(t,y1, 'r*')
plt.xlabel('Time')
plt.ylabel('cos(t)')
plt.grid()
```

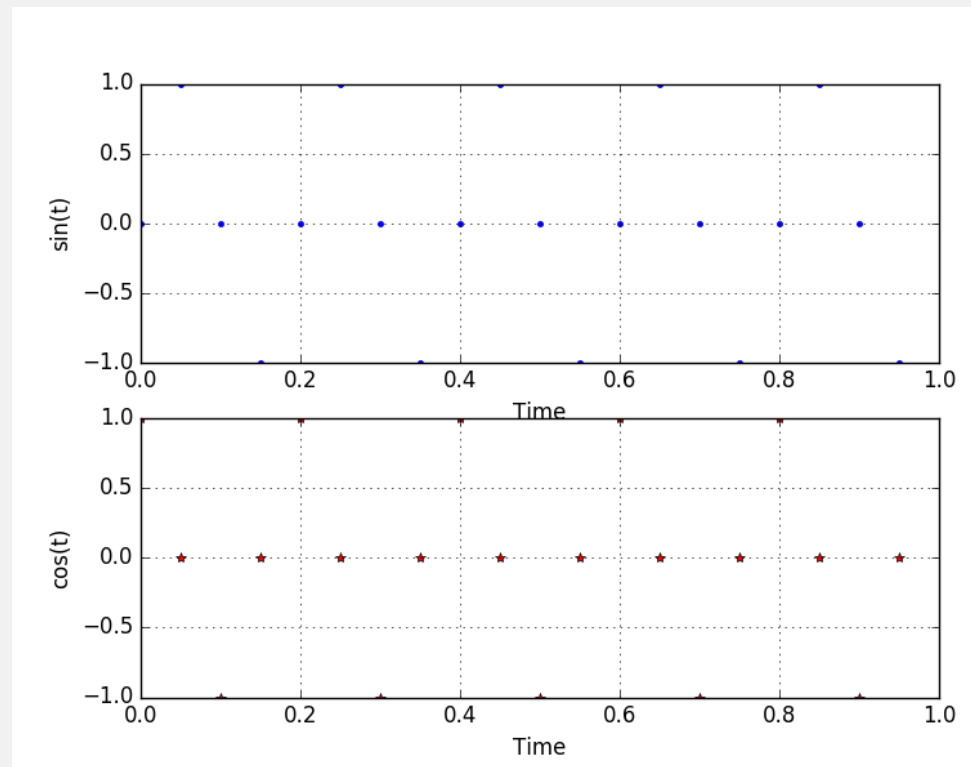


Sampling Rate

샘플링 비율 (Sampling rate)

- 샘플링은 연속 신호를 이산 신호로 만들 때 사용함
- 연속 신호가 샘플링되어 이산 신호로 축소됨

```
Fs = 20.0 # sampling rate
Ts = 1.0/Fs # sampling interval
# 0.05
t = np.arange(0,1,Ts) # time vector
ff = 5 # frequency of the signal
y = np.sin(2*np.pi*ff*t)
# compare sin and cos
y1= np.cos(2*np.pi*ff*t)
##### draw sin and cos
plt.figure(100)
plt.subplot(2,1,1)
plt.plot(t,y, 'b.')
# np.size(t) = 150
# np.size(y) = 150
plt.xlabel('Time')
plt.ylabel('sin(t)')
plt.grid()
plt.subplot(2,1,2)
plt.plot(t,y1,'r*')
plt.xlabel('Time')
plt.ylabel('cos(t)')
plt.grid()
```

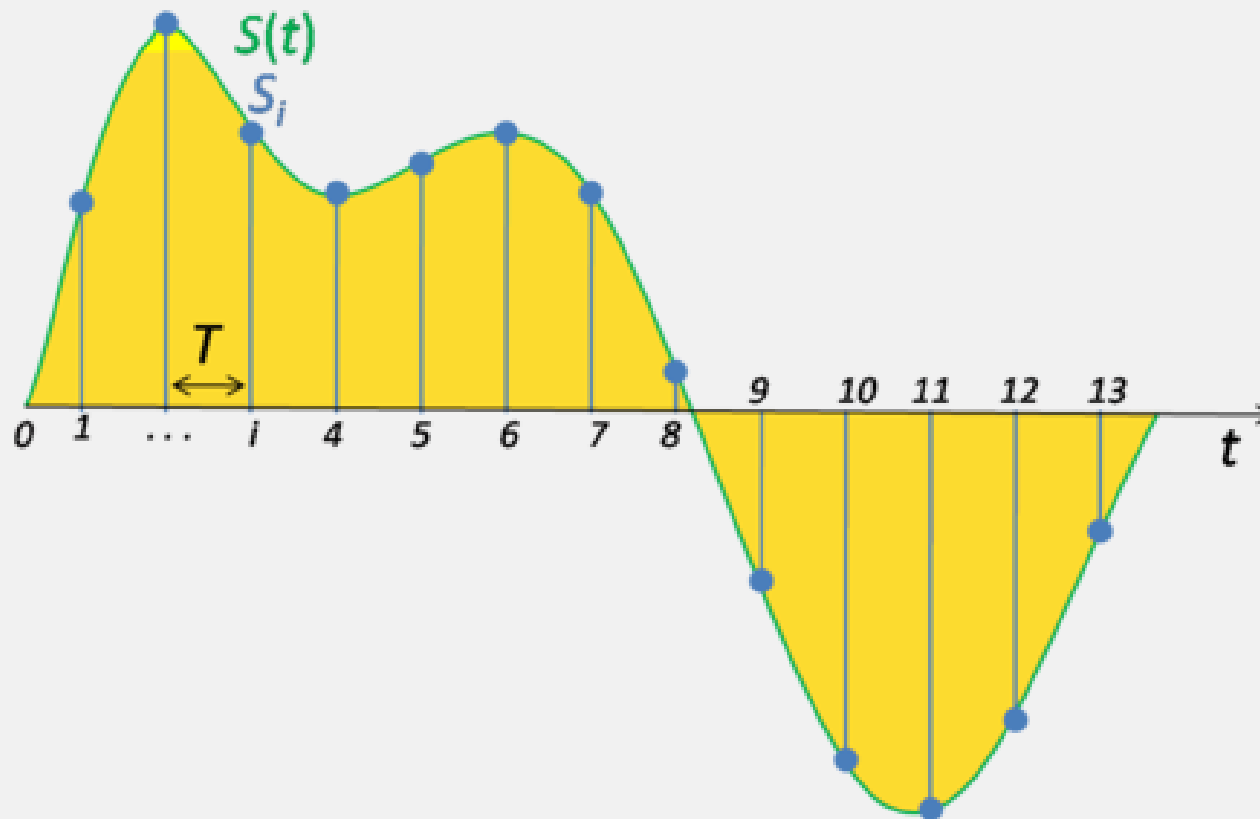


Sampling Rate (or Sampling Frequency)

- 주파수라는 개념은 1초에 어떤 것이 몇 개 있느냐를 측정한 것이다.
- 주파수의 단위는 Hz이다
- 일반 주파수(Ordinary Frequency) 계산에서 그 어떤 것은 Cycle 이 되며, Cycle이 1초에 5개 있으면 Frequency는 5Hz이다.
- 샘플링 주파수(Sampling Frequency) 계산에서 그 어떤 것은 Sampling Point가 되며, Sampling Point(Cycle 이 아니라)가 1초에 50개 있으면, Sampling 주파수는 50Hz이다
- Rate는 비율이다. 나누기이다.
 - 1초에 Sampling Point가 50개이면, 비율(rate)은 50 Sampling Point/1Sec 이다.
 - 50 [Sampling Point/Sec]이다.

Sampling Rate (or Sampling Frequency)

- The continuous signal is represented with a green colored line while the discrete samples are indicated by the blue vertical lines.



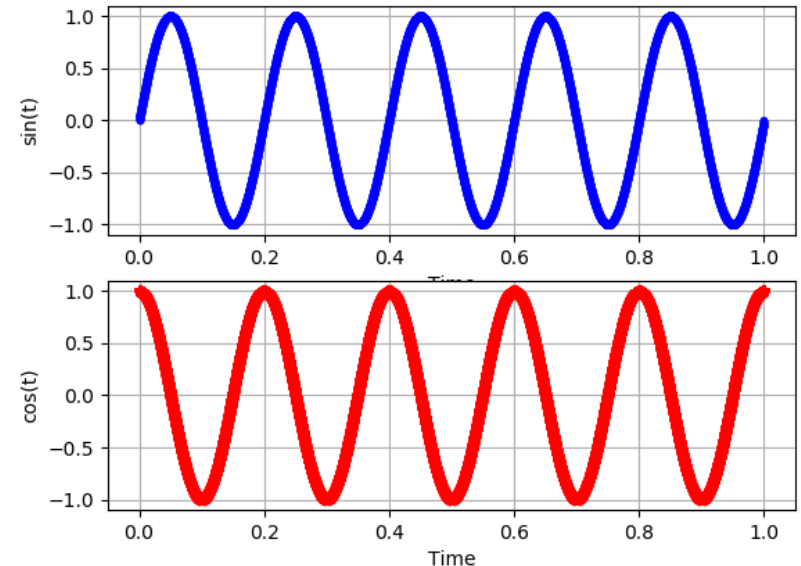
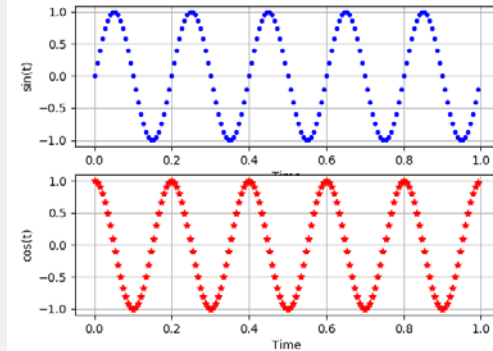
Sampling Rate (or Sampling Frequency)

- When it is necessary to capture audio covering the entire 20–20,000 (20k) Hz range of human hearing, such as when recording music or many types of acoustic events, audio waveforms are typically sampled at 44.1 kHz (CD), 48 kHz, 88.2 kHz, or 96 kHz.
- The approximately double-rate requirement is a consequence of the Nyquist theorem. ($20\text{kHz} \times 2 = 40\text{kHz}$)
- Sampling rates higher than about 50 kHz to 60 kHz cannot supply more usable information for human listeners.
- Early professional audio equipment manufacturers chose sampling rates in the region of 50 kHz for this reason.

Increase Sampling Rate (or Sampling Frequency)

```
Fs = 10000.0 # sampling rate
Ts = 1.0/Fs # sampling interval or
sampling time
# 0.05
t = np.arange(0,1,Ts) # time vector
ff = 5 # frequency of the signal
y = np.sin(2*np.pi*ff*t)
y1= np.cos(2*np.pi*ff*t)
```

```
plt.figure(102)
plt.subplot(2,1,1)
plt.plot(t,y, 'b.')
# np.size(t) = 150
# np.size(y) = 150
plt.xlabel('Time')
plt.ylabel('sin(t)')
plt.grid()
plt.subplot(2,1,2)
plt.plot(t,y1, 'r*')
plt.xlabel('Time')
plt.ylabel('cos(t)')
plt.grid()
```



Sinusoidal Characteristics

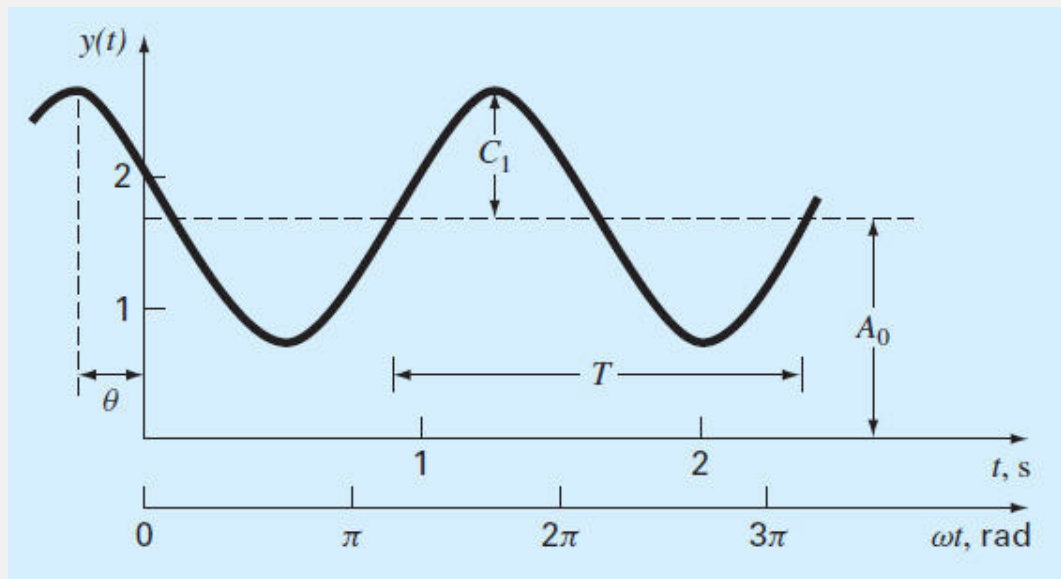
모든 파형은 정현파를 통해 표현될 수 있다

Sinusoids (정현파)의 특징

- Any waveform that can be described as a sine or cosine.
- 모든 파형은 sine과 cosine의 정현파로 만들어질 수 있다

$$f(t) = A_0 + C_1 \sin(\omega_0 t + \theta)$$

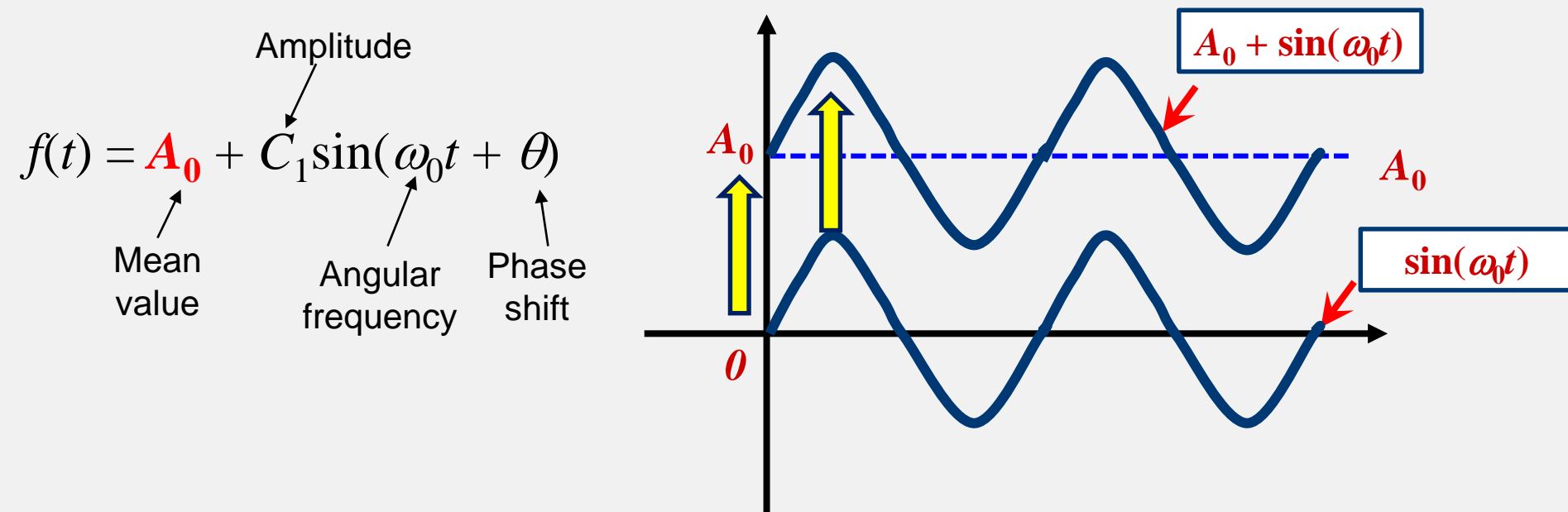
Mean value Amplitude Angular frequency Phase shift



$$\omega_0 = 2\pi f = \frac{2\pi}{T}$$

Sinusoids (정현파)의 평균값 (mean value) 바꾸기

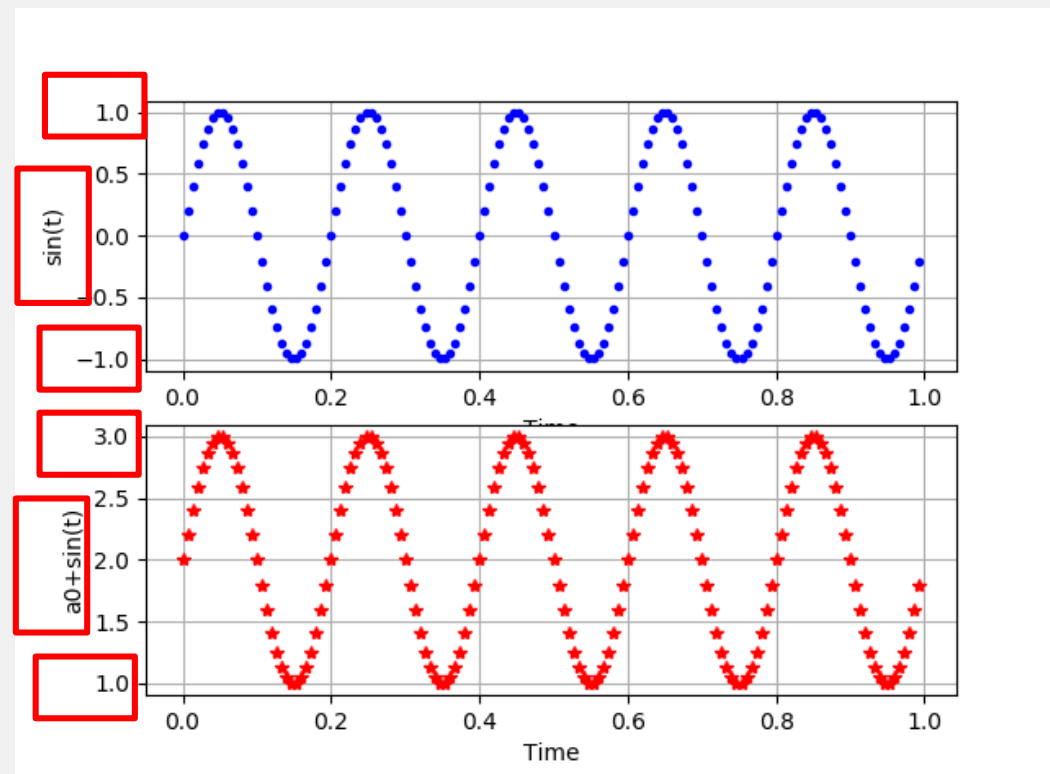
- 고유 sin(사인파)는 $0 + \sin(\omega_0 t)$ 이다
- 바꿀 수 있는 것은 $A_0 + \sin(\omega_0 t)$ 에서 평균값(mean value)을 0 에서 A_0 로 바꿀 수 있다



Sinusoids (정현파)의 평균값 (mean value) 바꾸기

```
Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval
t = np.arange(0,1,Ts) # time vector
ff = 5 # frequency of the signal
y = np.sin(2*np.pi*ff*t)
a0=2
y1= a0+np.sin(2*np.pi*ff*t)
```

```
#####
plt.figure(103)
plt.subplot(2,1,1)
plt.plot(t,y, 'b.')
# np.size(t) = 150
# np.size(y) = 150
plt.xlabel('Time')
plt.ylabel('sin(t)')
plt.grid()
plt.subplot(2,1,2)
plt.plot(t,y1,'r*')
plt.xlabel('Time')
plt.ylabel('a0+sin(t)')
plt.grid()
```

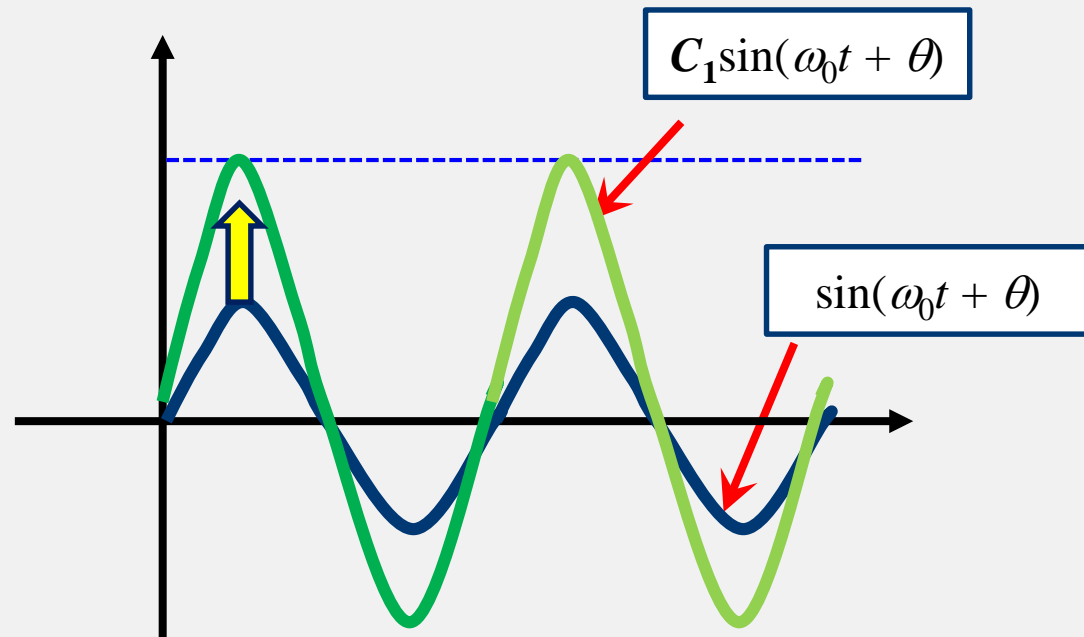


Sinusoids (정현파)의 진폭 (Amplitude) 바꾸기

- 고유 sin(사인파)는 $0+1*\sin(\omega_0 t)$ 이다.
- 바꿀 수 있는 것은 $0+ \mathbf{C_1}*\sin(\omega_0 t)$ 에서 진폭(Amplitude)을 **1**에서 **C_1** 로 바꿀 수 있다

$$f(t) = A_0 + \mathbf{C_1} \sin(\omega_0 t + \theta)$$

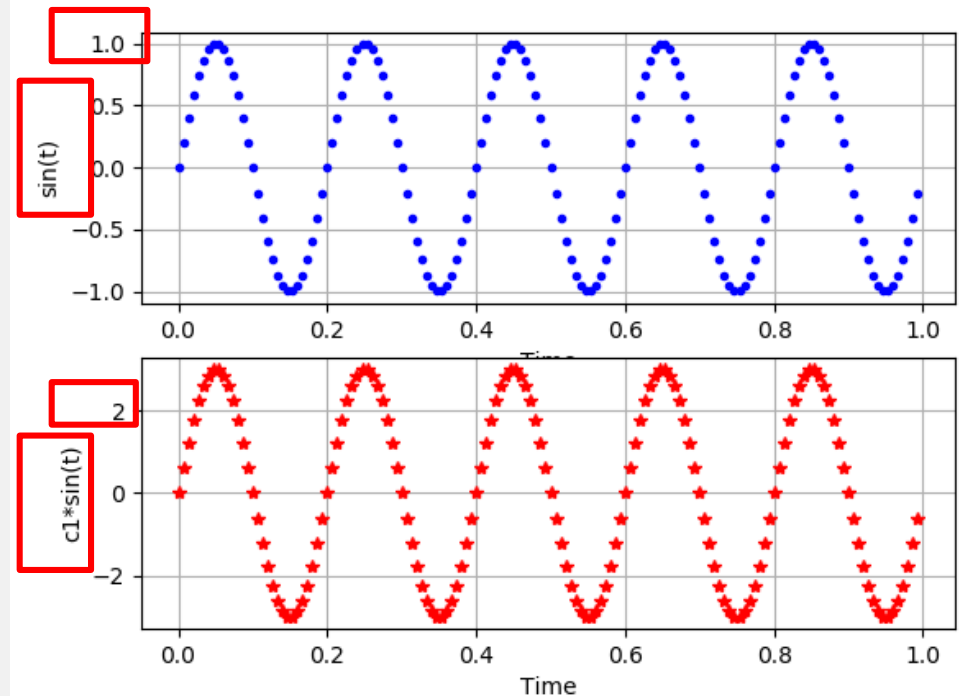
Mean value Amplitude Angular frequency Phase shift



Sinusoids (정현파)의 진폭 (Amplitude) 바꾸기

```
Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval
t = np.arange(0,1,Ts) # time vector
ff = 5 # frequency of the signal
y = np.sin(2*np.pi*ff*t)
c1=3
y1= c1*np.sin(2*np.pi*ff*t)
```

```
#####
plt.figure(104)
plt.subplot(2,1,1)
plt.plot(t,y, 'b.')
# np.size(t) = 150
# np.size(y) = 150
plt.xlabel('Time')
plt.ylabel('sin(t)')
plt.grid()
plt.subplot(2,1,2)
plt.plot(t,y1,'r*')
plt.xlabel('Time')
plt.ylabel('a0+sin(t)')
plt.grid()
```



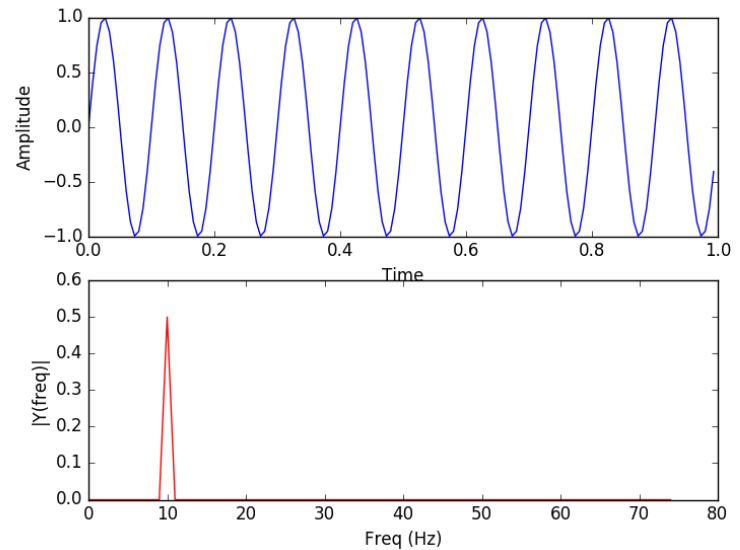
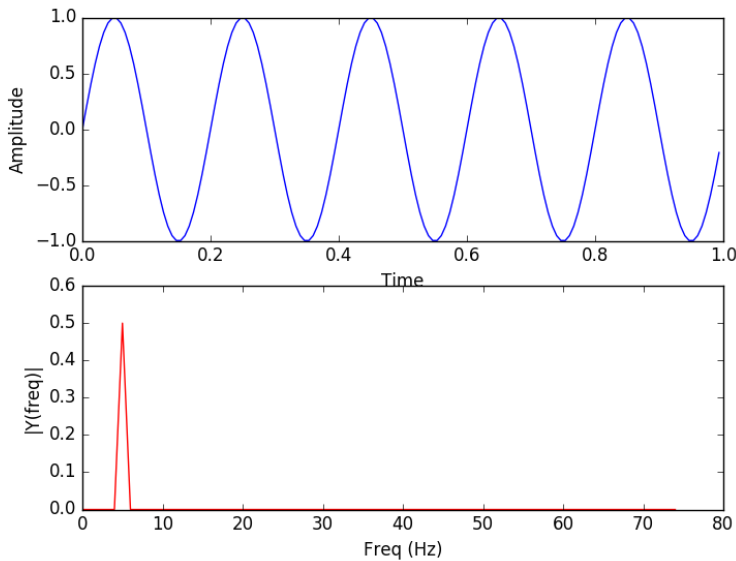
<https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/fourier.py>

Sinusoids (정현파)의 주파수 바꾸기

- 고유 sin(사인파)는 $0+1*\sin(\omega_0 t)$
- 바꿀 수 있는 것은 $\omega_0 = 2*\pi*f_0$ 에서 주파수 f_0 를 f_1 으로 바꿀 수 있다. ($0+1*\sin(\omega_1 t)$)

$$f(t) = A_0 + C_1 \sin(\omega_0 t + \theta)$$

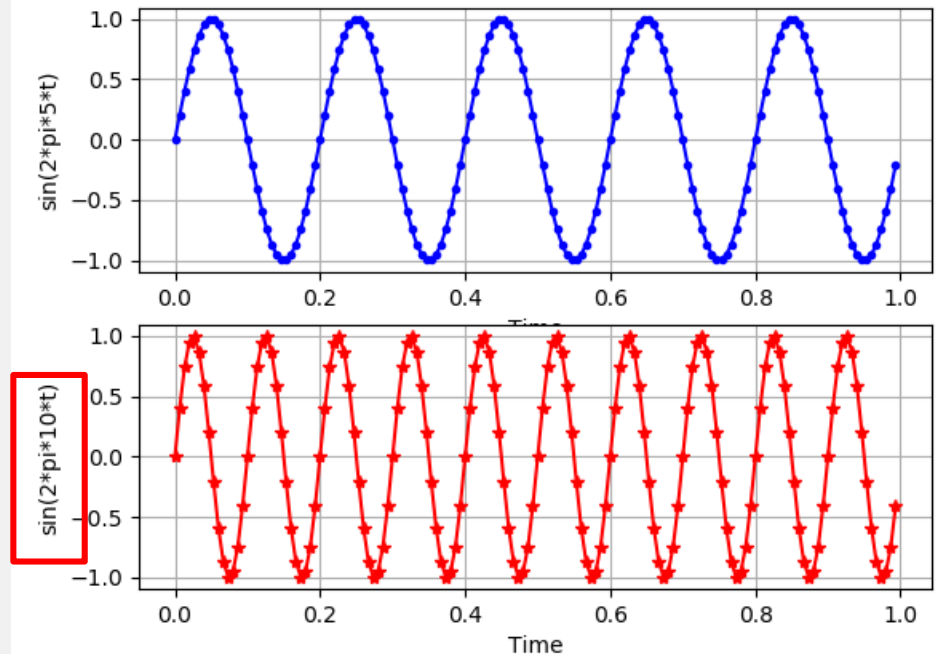
Mean value Amplitude Angular frequency Phase shift



Sinusoids (정현파)의 주파수 바꾸기

```
Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval
t = np.arange(0,1,Ts) # time vector
ff = 5 # frequency of the signal
y = np.sin(2*np.pi*ff*t)
ff1=10
y1= np.sin(2*np.pi*ff1*t)
```

```
#####
plt.figure(105)
plt.subplot(2,1,1)
plt.plot(t,y, 'b.-')
# np.size(t) = 150
# np.size(y) = 150
plt.xlabel('Time')
plt.ylabel('sin(2*pi*5*t)')
plt.grid()
plt.subplot(2,1,2)
plt.plot(t,y1, 'r*-')
plt.xlabel('Time')
plt.ylabel('sin(2*pi*10*t)')
plt.grid()
```



Sinusoids (정현파)의 위상 바꾸기

- 고유 sin(사인파)는 $0+1*\sin(\mathbf{w_0t})$ 이다
- 바꿀 수 있는 것은 $\mathbf{w_0t}$ 에서 $0+1*\sin(\mathbf{w_0t+ \theta})$ 로 위상 θ 를 바꿀 수 있다

$$f(t) = A_0 + C_1 \sin(\omega_0 t + \theta)$$

Mean value Amplitude Angular frequency Phase shift

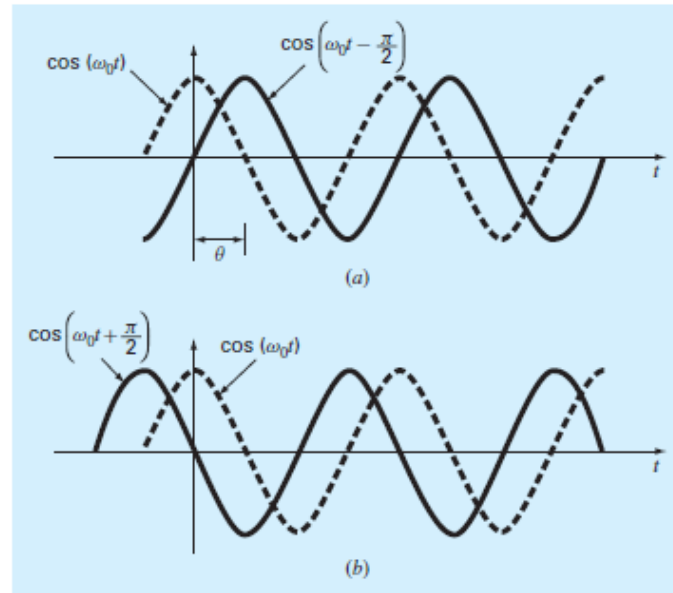


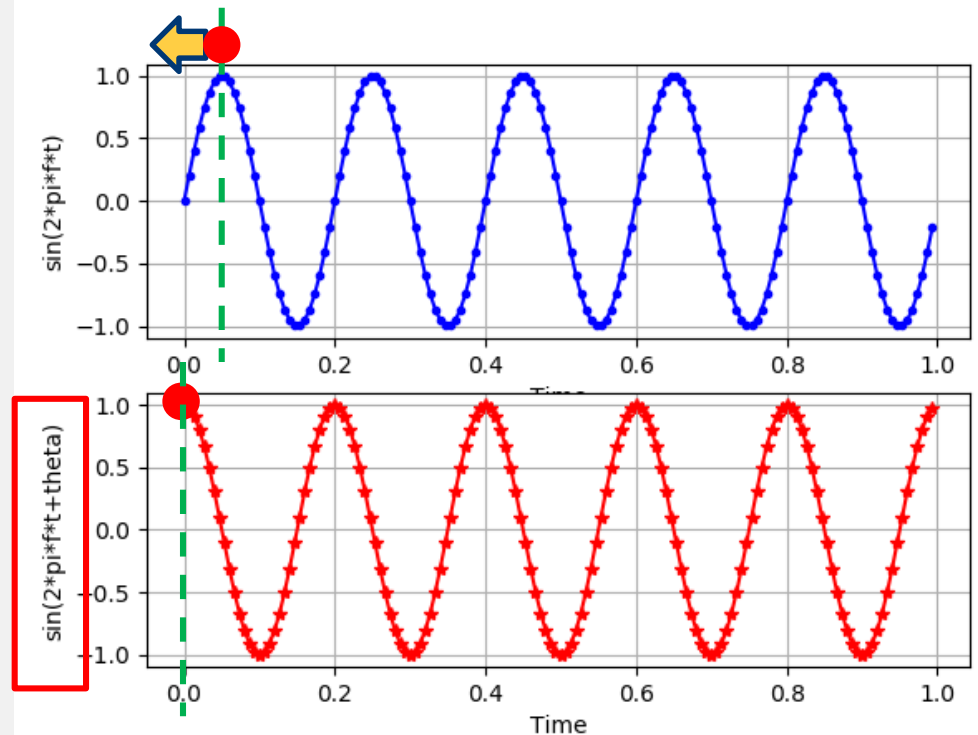
FIGURE 16.3

Graphical depictions of (a) a lagging phase angle and (b) a leading phase angle. Note that the lagging curve in (a) can be alternatively described as $\cos(\omega_0 t + 3\pi/2)$. In other words, if a curve lags by an angle of α , it can also be represented as leading by $2\pi - \alpha$.

Sinusoids (정현파)의 위상 바꾸기

```
Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval
t = np.arange(0,1,Ts) # time vector
ff = 5 # frequency of the signal
y = np.sin(2*np.pi*ff*t)
theta=np.pi/2
y1= np.sin(2*np.pi*ff*t+theta)
```

```
#####
plt.figure(106)
plt.subplot(2,1,1)
plt.plot(t,y, 'b.-')
# np.size(t) = 150
# np.size(y) = 150
plt.xlabel('Time')
plt.ylabel('sin(2*pi*5*t)')
plt.grid()
plt.subplot(2,1,2)
plt.plot(t,y1, 'r*-')
plt.xlabel('Time')
plt.ylabel('sin(2*pi*10*t)')
plt.grid()
```



<https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/fourier.py>

Sinusoidal 응용

일반선형회귀 전에 사인과 코사인을 이용한 다양한 응용 분야
를 살펴보자

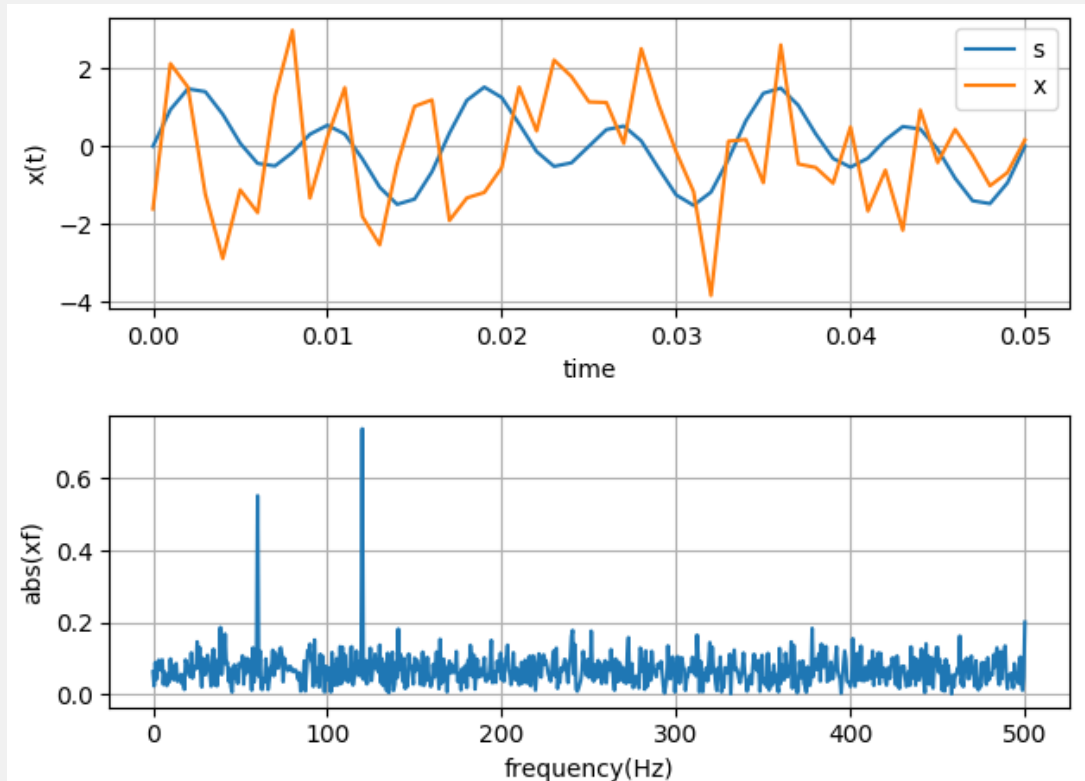
정현파와 주파수

- 다음은 60 Hz 와 120 Hz의 사인 곡선이 중첩된 신호에 $N(\mu=0, \sigma=2)$ 를 따르는 노이즈가 포함된 신호에 대해 주파수 분석을 한 예이다.

```
import numpy as np
import matplotlib.pyplot as plt
fs = 1000      # sampling frequency 1000 Hz
dt = 1/fs      # sampling period
N = 1500       # length of signal
t = np.arange(0,N)*dt # time = [0, dt, ..., (N-1)*dt]
s = 0.7*np.sin(2*np.pi*60*t) + np.sin(2*np.pi*120*t)
x = s+2*np.random.randn(N) # random number Normal distn, N(0,2)... N(0,2*2)
plt.subplot(2,1,1)
plt.plot(t[0:51],s[0:51],label='s')
plt.plot(t[0:51],x[0:51],label='x')
plt.legend()
plt.xlabel('time'); plt.ylabel('x(t)'); plt.grid()
# Fourier spectrum
df = fs/N      # df = 1/N = fmax/N
f = np.arange(0,N)*df # frq = [0, df, ..., (N-1)*df]
xf = np.fft.fft(x)*dt
plt.subplot(2,1,2)
plt.plot(f[0:int(N/2+1)],np.abs(xf[0:int(N/2+1)]))
plt.xlabel('frequency(Hz)'); plt.ylabel('abs(xf)'); plt.grid()
plt.tight_layout()
```

정현파에 실려 있는 주파수를 찾아보자

- 샘플링 주파수는 100 Hz이고 1500개의 데이터가 있다고 가정하기로 한다. 주파수 분석결과 60 Hz 및 120 Hz에서 피크를 보임을 알 수 있다.



<https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/frequency.py>

국민대학교 소프트웨어학부

사인과 코사인을 이용한 3차원 데이터 시각화

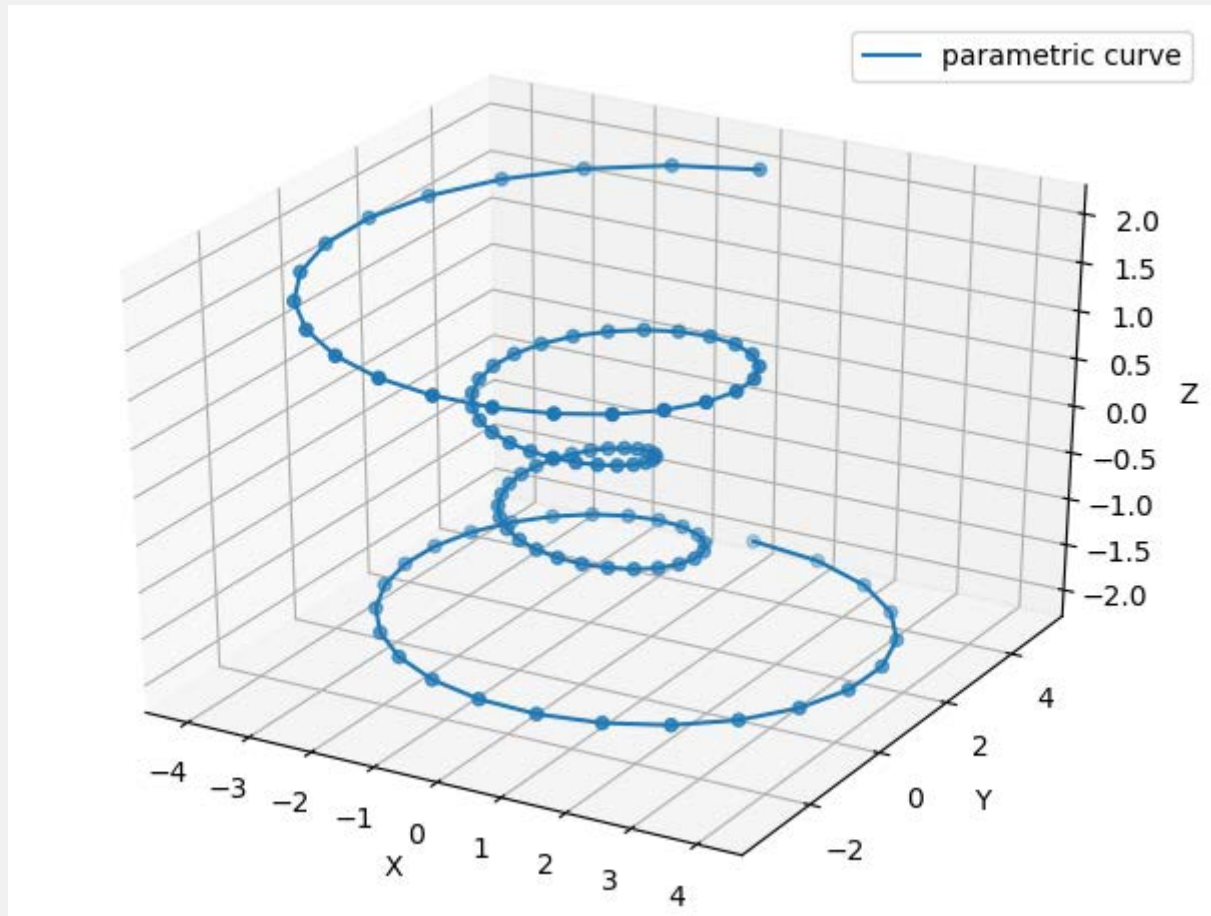
- 다음은 line plot과 scatter plot의 예이다.
- θ 는 사인과 코사인에 들어갈 각도이다
- 두 변수 r , θ 가 생성된 후, 두 변수 r , θ 가 x 값과 y 값을 계산할 때 사용된다.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
theta = np.linspace(-4*np.pi, 4*np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z**2+1
x = r*np.sin(theta)
y = r*np.cos(theta)
ax.plot(x,y,z, label='parametric curve')
ax.scatter(x,y,z)
ax.legend()
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
fig.tight_layout()
plt.show()
```

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/3D_Sine.py

사인과 코사인을 이용한 데이터 시각화

- 사인과 코사인을 이용한 3차원 데이터 시각화



surface plot과 wireframe plot

- 사인을 이용한 surface plot과 wireframe plot의 예이다.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

fig = plt.figure()
ax = fig.gca(projection='3d')

X=np.arange(-5,5,0.25)
Y=np.arange(-5,5,0.25)
X,Y = np.meshgrid(X,Y)
R = np.sqrt(X**2+Y**2)
Z = np.sin(R)

surf = ax.plot_surface(X,Y,Z,cmap='coolwarm',linewidth=0,antialiased=False)
wire = ax.plot_wireframe(X,Y,Z,color='r',linewidth=0.1)
fig.colorbar(surf,shrink=0.5,aspect=5)
fig.tight_layout()
plt.show()
```

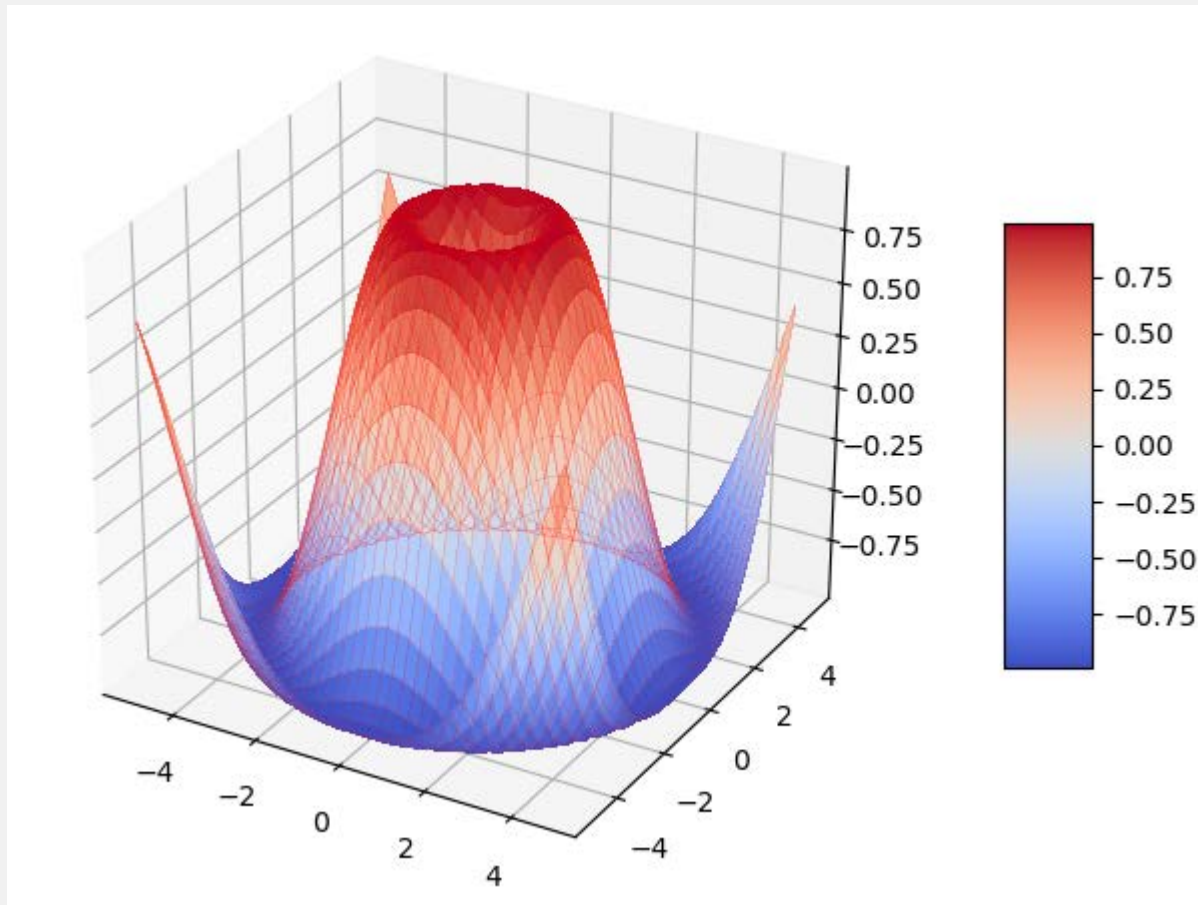
<https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/mesh.py>

surface plot과 wireframe plot

- 메쉬 그리드를 하고 난 후의 X, Y array 형태
- X
- `array([[-5. , -4.75, -4.5 , ..., 4.25, 4.5 , 4.75],`
- `[-5. , -4.75, -4.5 , ..., 4.25, 4.5 , 4.75],`
- `[-5. , -4.75, -4.5 , ..., 4.25, 4.5 , 4.75],`
- `...,`
- `[-5. , -4.75, -4.5 , ..., 4.25, 4.5 , 4.75],`
- `[-5. , -4.75, -4.5 , ..., 4.25, 4.5 , 4.75],`
- `[-5. , -4.75, -4.5 , ..., 4.25, 4.5 , 4.75]])`
- Y
- `array([[-5. , -5. , -5. , ..., -5. , -5. , -5.],`
- `[-4.75, -4.75, -4.75, ..., -4.75, -4.75, -4.75],`
- `[-4.5 , -4.5 , -4.5 , ..., -4.5 , -4.5 , -4.5],`
- `...,`
- `[4.25, 4.25, 4.25, ..., 4.25, 4.25, 4.25],`

surface plot과 wireframe plot 데이터 시각화

- surface plot과 wireframe plot



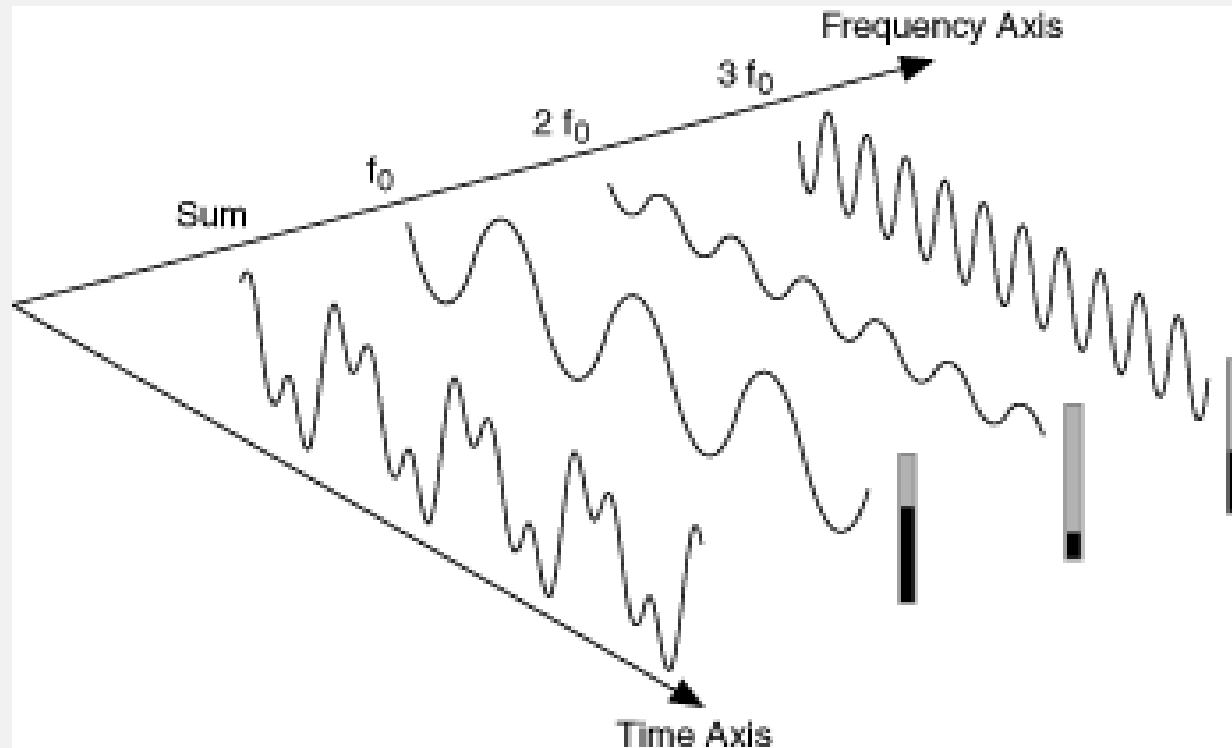
Fourier Series

Curve Fitting by Sinusoidal

두 개 이상의 함수, 각 함수는 1차원 변수 ($z_1(x^1), z_2(x^1), \dots, z_n(x^1)$)
를 사용하여 곡선 접합 수행

General Linear Regression

- $f(t) = 1 \cdot \sin(2\pi (1 \cdot f_0)t) + 1/3 \cdot \sin(2\pi(3 \cdot f_0)t) + 1/5 \cdot \sin(2\pi(5 \cdot f_0)t) + 1/7 \cdot \sin(2\pi(7 \cdot f_0)t) + 1/9 \cdot \sin(2\pi(9 \cdot f_0)t) + 1/11 \cdot \sin(2\pi(11 \cdot f_0)t) + 1/13 \cdot \sin(2\pi(13 \cdot f_0)t) + 1/15 \cdot \sin(2\pi(15 \cdot f_0)t) + 1/17 \cdot \sin(2\pi(17 \cdot f_0)t)$
- 두 개 이상의 함수, 각 함수는 1차원 변수 ($z_1(x^1), z_2(x^1), \dots, z_n(x^1)$)를 사용하여 곡선 접합 수행

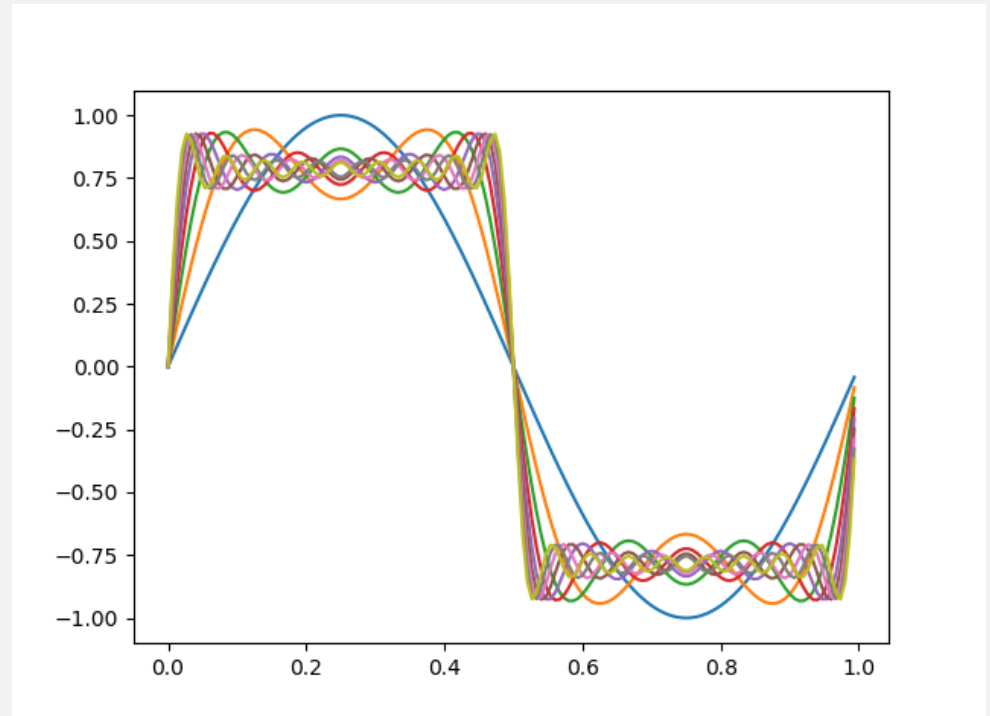


Fourier Series

- 두 개 이상의 정현파 더하기 → General Linear Regression
- $f(t) = 1 \cdot \sin(2\pi (1 \cdot f_0)t) + 1/3 \cdot \sin(2\pi(3 \cdot f_0)t) + 1/5 \cdot \sin(2\pi(5 \cdot f_0)t)$

```
import numpy as np
import matplotlib.pyplot as plt
Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval
t = np.arange(0, 1, Ts) # time vector

ff1 = 1 # frequency of the signal
sqwave=0
c1=1
plt.figure(201)
for loop in np.arange(1, 10, 1):
    sqwave = sqwave +
(1/c1)*np.sin(c1*2*np.pi*ff1*t)
    # fourier series
    plt.plot(t, sqwave)
    c1=c1+2
    plt.pause(3)
```



<https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/fourier.py>

Fourier Series

- Fourier showed that an arbitrary periodic function can be represented by an infinite series of sinusoids of harmonically related frequencies.


```
import numpy as np
import matplotlib.pyplot as plt

Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval

t = np.arange(0, 1, Ts) # time vector

ff1 = 1 # frequency of the signal
sqwave=0
c1=1
plt.figure(201)

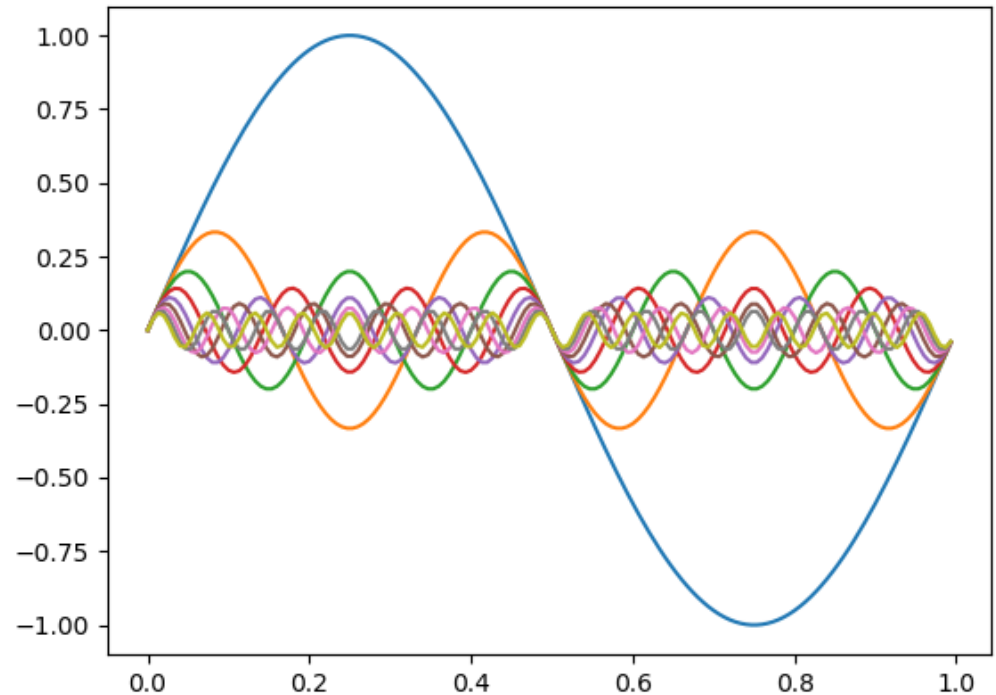
for loop in np.arange(1, 10, 1):
    sqwave = sqwave + (1/c1)*np.sin(c1*2 * np.pi * ff1 * t)
    # fourier series
    plt.plot(t, sqwave)
    c1=c1+2
    plt.pause(3)
```


$$\begin{aligned} f(t) = & 1 \cdot \sin(2\pi(1 \cdot f_0)t) + 1/3 \cdot \sin(2\pi(3 \cdot f_0)t) \\ & + 1/5 \cdot \sin(2\pi(5 \cdot f_0)t) + 1/7 \cdot \sin(2\pi(7 \cdot f_0)t) \\ & + 1/9 \cdot \sin(2\pi(9 \cdot f_0)t) + 1/11 \cdot \sin(2\pi(11 \cdot f_0)t) \\ & + 1/13 \cdot \sin(2\pi(13 \cdot f_0)t) + 1/15 \cdot \sin(2\pi(15 \cdot f_0)t) \\ & + 1/17 \cdot \sin(2\pi(17 \cdot f_0)t) \end{aligned}$$

Fourier Series

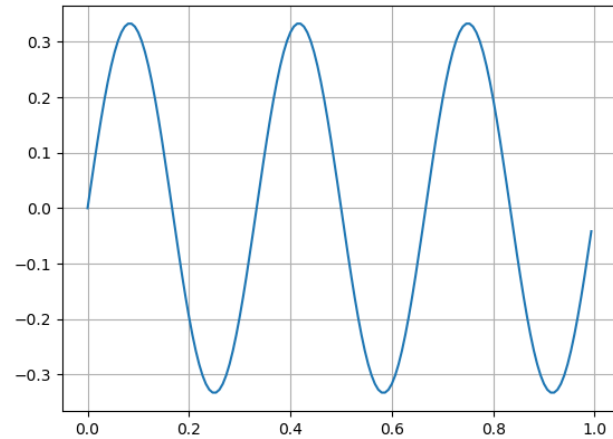
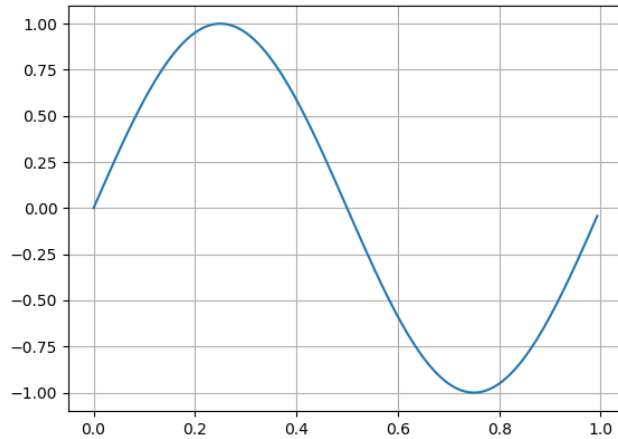
$$f(t) = 1 \cdot \sin(2\pi (1 \cdot f_0)t) + 1/3 \cdot \sin(2\pi(3 \cdot f_0)t) + 1/5 \cdot \sin(2\pi(5 \cdot f_0)t) + 1/7 \cdot \sin(2\pi(7 \cdot f_0)t) + 1/9 \cdot \sin(2\pi(9 \cdot f_0)t) + 1/11 \cdot \sin(2\pi(11 \cdot f_0)t) + 1/13 \cdot \sin(2\pi(13 \cdot f_0)t) + 1/15 \cdot \sin(2\pi(15 \cdot f_0)t) + 1/17 \cdot \sin(2\pi(17 \cdot f_0)t)$$

```
plt.figure(202)
f1=(1/1)*np.sin(1*2*np.pi*ff1*t)
plt.plot(t, f1)
f3=(1/3)*np.sin(3*2*np.pi*ff1*t)
plt.plot(t, f3)
f5=(1/5)*np.sin(5*2*np.pi*ff1*t)
plt.plot(t, f5)
f7=(1/7)*np.sin(7*2*np.pi*ff1*t)
plt.plot(t, f7)
f9=(1/9)*np.sin(9*2*np.pi*ff1*t)
plt.plot(t, f9)
f11=(1/11)*np.sin(11*2*np.pi*ff1*t)
plt.plot(t, f11)
f13=(1/13)*np.sin(13*2*np.pi*ff1*t)
plt.plot(t, f13)
f15=(1/15)*np.sin(15*2*np.pi*ff1*t)
plt.plot(t, f15)
f17=(1/17)*np.sin(17*2*np.pi*ff1*t)
plt.plot(t, f17)
```

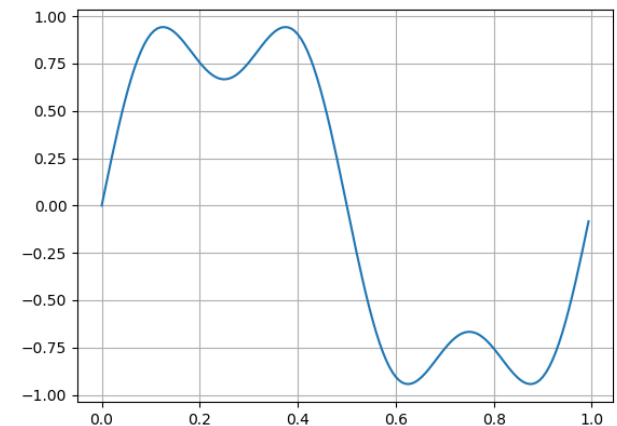


<https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/fourier.py>

$$f(t) = 1 \cdot \sin(2\pi (1 \cdot f_0)t) + 1/3 \cdot \sin(2\pi(3 \cdot f_0)t)$$

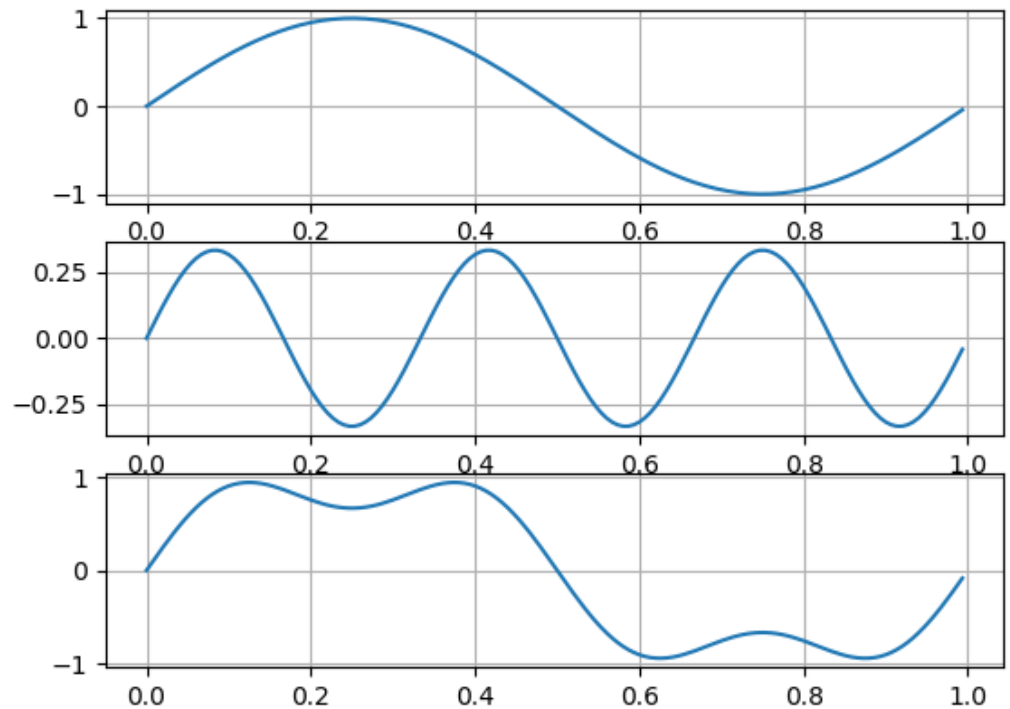
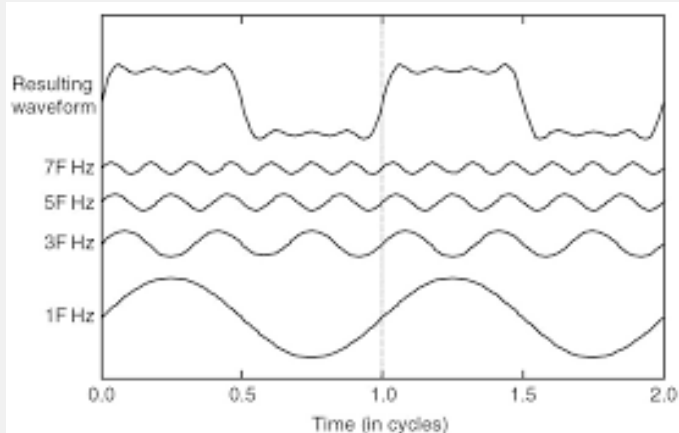


```
plt.figure(203)
f1=(1/1)*np.sin(1*2*np.pi*ff1*t)
plt.plot(t, f1)
plt.grid()
plt.figure(204)
f3=(1/3)*np.sin(3*2*np.pi*ff1*t)
plt.plot(t, f3)
plt.grid()
plt.figure(205)
plt.plot(t, f1+f3)
plt.grid()
```



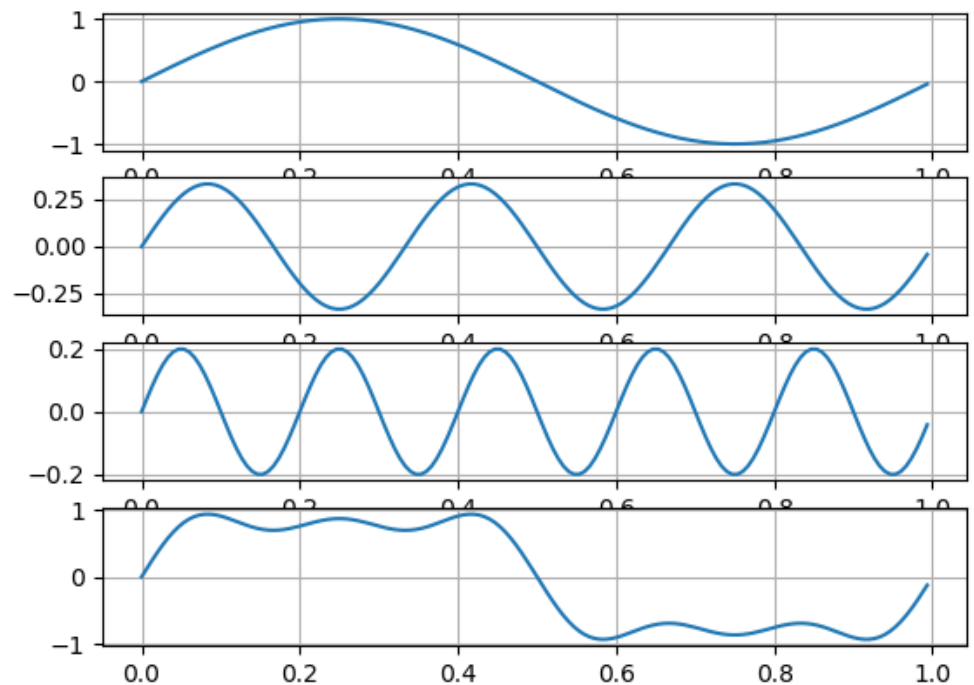
$$f(t) = 1 \cdot \sin(2\pi (1 \cdot f_0)t) + 1/3 \cdot \sin(2\pi(3 \cdot f_0)t)$$

```
plt.figure(206)
f1=(1/1)*np.sin(1*2*np.pi*ff1*t)
plt.subplot(3, 1, 1)
plt.plot(t, f1)
plt.grid()
f3=(1/3)*np.sin(3*2*np.pi*ff1*t)
plt.subplot(3, 1, 2)
plt.plot(t, f3)
plt.grid()
plt.subplot(3, 1, 3)
plt.plot(t, f1+f3)
plt.grid()
```



$$f(t) = 1 \cdot \sin(2\pi (1 \cdot f_0)t) + 1/3 \cdot \sin(2\pi(3 \cdot f_0)t) + 1/5 \cdot \sin(2\pi(5 \cdot f_0)t)$$

```
plt.figure(207)
f1=(1/1)*np.sin(1*2*np.pi*ff1*t)
plt.subplot(4, 1, 1)
plt.plot(t, f1)
plt.grid()
f3=(1/3)*np.sin(3*2*np.pi*ff1*t)
plt.subplot(4, 1, 2)
plt.plot(t, f3)
plt.grid()
f5=(1/5)*np.sin(5*2*np.pi*ff1*t)
plt.subplot(4, 1, 3)
plt.plot(t, f5)
plt.grid()
plt.subplot(4, 1, 4)
plt.plot(t, f1+f3+f5)
plt.grid()
```

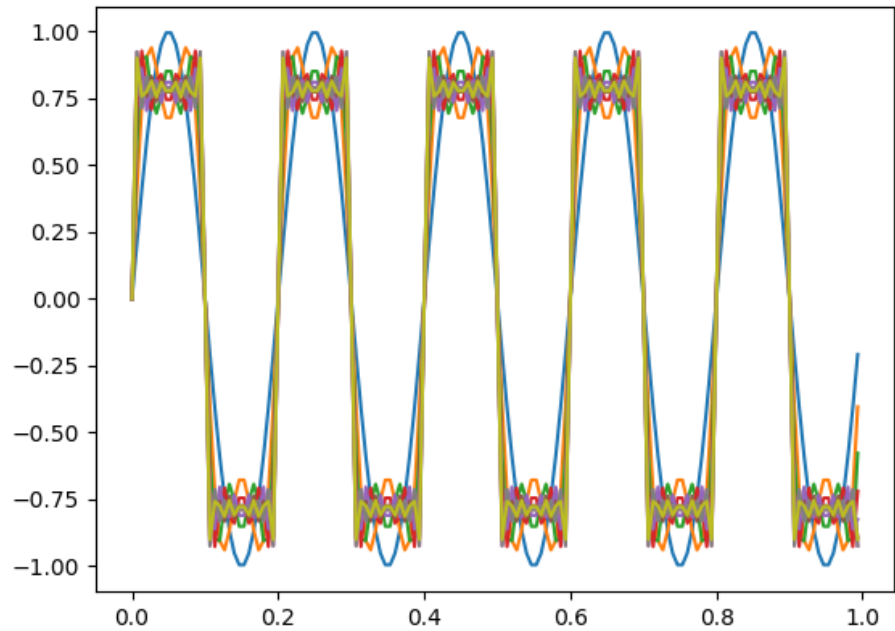


두 개 이상의 정현파로 사각형파,
삼각형파, 톱니파 만들기

sin() 함수 정현파 더하기

- sin() 함수 정현파 더하기 → 사각형파 만들기
- $f(t) = 1 \cdot \sin(2\pi (1 \cdot f_0)t) + 1/3 \cdot \sin(2\pi(3 \cdot f_0)t) + 1/5 \cdot \sin(2\pi(5 \cdot f_0)t) + \dots$

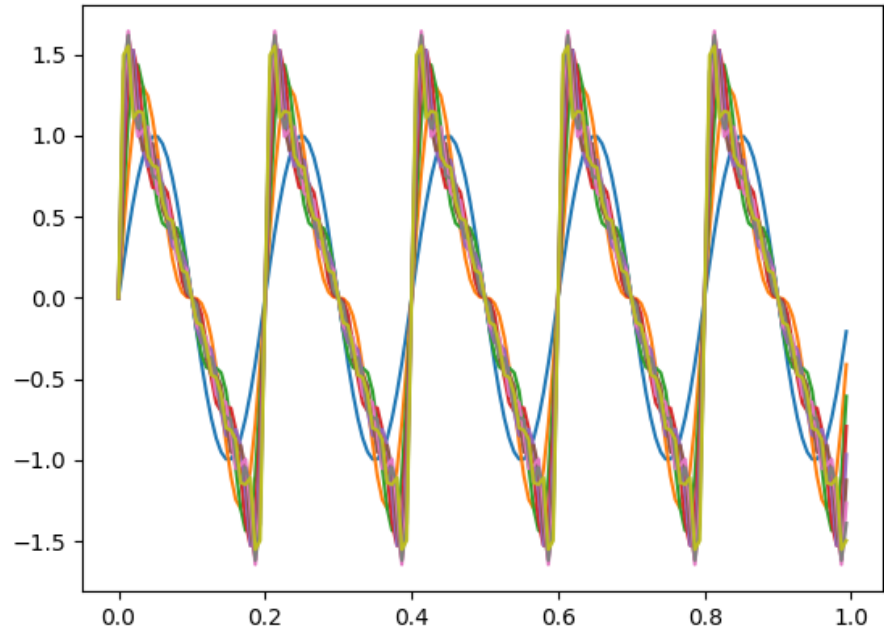
```
import numpy as np
import matplotlib.pyplot as plt
Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval
t = np.arange(0, 1, Ts) # time vector
ff1 = 5
sqwave=0
c1=1
plt.figure(301)
for loop in np.arange(1, 10, 1):
    sqwave =
sqwave+(1/c1)*np.sin(c1*2*np.pi*
ff1*t)
    # fourier series
    plt.plot(t, sqwave)
    c1=c1+2
plt.pause(3)
```



sin() 홀수+짝수 정현파 더하기

- sin() 홀수+짝수 정현파 더하기 → 톱니파 만들기
- $f(t) = 1 \cdot \sin(2\pi (1 \cdot f_0)t) + 1/2 \cdot \sin(2\pi (2 \cdot f_0)t) + 1/3 \cdot \sin(2\pi (3 \cdot f_0)t) + \dots$

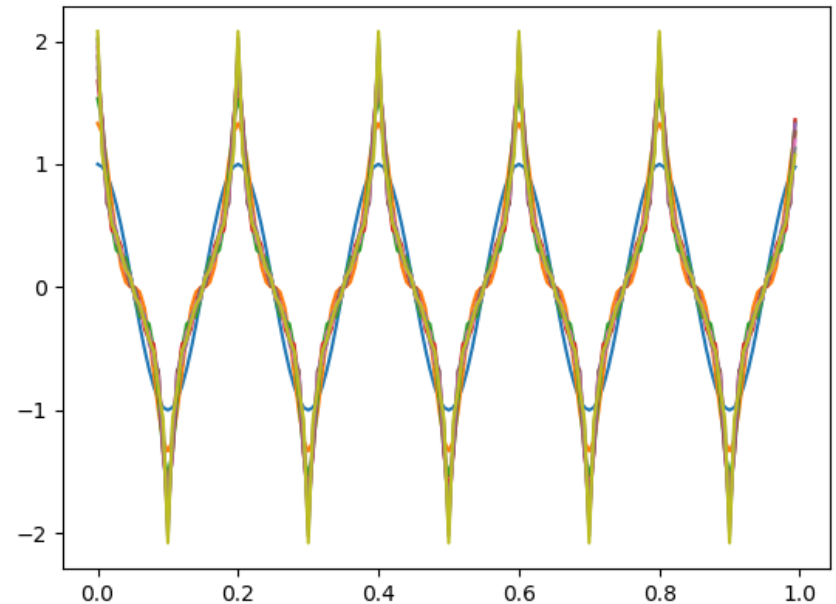
```
Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval
t = np.arange(0, 1, Ts) # time vector
ff1 = 5 # frequency of the signal
sqwave=0
c1=1
plt.figure(302)
for loop in np.arange(1, 10, 1):
    sqwave =
sqwave+(1/c1)*np.sin(c1*2*np.pi*
ff1*t)
    # fourier series
    plt.plot(t, sqwave)
    c1=c1+1
    plt.pause(3)
```



cos() 함수 정현파 더하기

- cos() 함수 정현파 더하기 → 넓은 삼각형파 만들기
- $f(t) = 1 \cdot \cos(2\pi (1 \cdot f_0)t) + 1/3 \cdot \cos(2\pi(3 \cdot f_0)t) + 1/5 \cdot \cos(2\pi(5 \cdot f_0)t) + \dots$

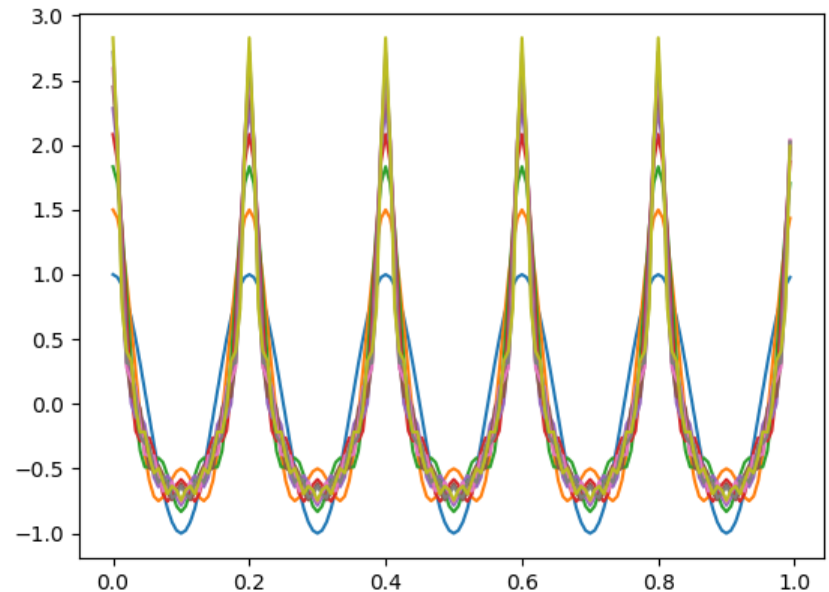
```
Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval
t = np.arange(0, 1, Ts) # time vector
ff1 = 5
sqwave=0
c1=1
plt.figure(303)
for loop in np.arange(1, 10, 1):
    sqwave =
sqwave+(1/c1)*np.cos(c1*2*np.pi*ff1*t
)
    # fourier series
    plt.plot(t, sqwave)
    c1=c1+2
    plt.pause(3)
```



cos() 홀수+짝수 정현파 더하기

- cos() 홀수+짝수 정현파 더하기 → 삼각형파 만들기
- $f(t) = 1 \cdot \cos(2\pi (1 \cdot f_0)t) + 1/2 \cdot \cos(2\pi (2 \cdot f_0)t) + 1/3 \cdot \cos(2\pi (3 \cdot f_0)t) + \dots$

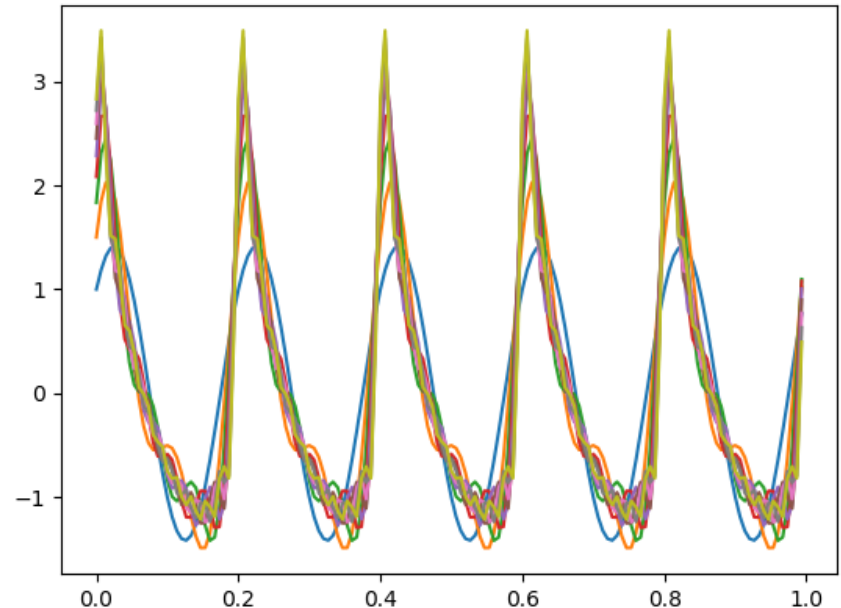
```
Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval
t = np.arange(0, 1, Ts) # time vector
ff1 = 5 # frequency of the signal
sqwave=0
c1=1
plt.figure(304)
for loop in np.arange(1, 10, 1):
    sqwave =
sqwave+(1/c1)*np.cos(c1*2*np.pi*ff1*t)
    # fourier series
    plt.plot(t, sqwave)
    c1=c1+1
    plt.pause(3)
```



cos() 홀수+짝수 정현파와 sin() 홀수+짝수 정현파 더하기

- cos() 홀수+짝수 정현파와 sin() 홀수+짝수 정현파 더하기 → 톱니같은(sin) 삼각형파(cos) 만들기
- $f(t) = 1 \cdot \cos(2\pi (1 \cdot f_0)t) + 1 \cdot \sin(2\pi (1 \cdot f_0)t) + 1/2 \cdot \cos(2\pi (2 \cdot f_0)t) + 1/2 \cdot \sin(2\pi (2 \cdot f_0)t) + 1/3 \cdot \cos(2\pi (3 \cdot f_0)t) + 1/3 \cdot \sin(2\pi (3 \cdot f_0)t) + \dots$

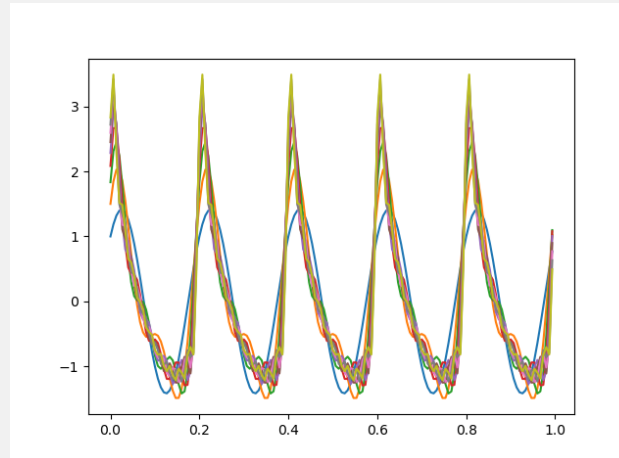
```
Fs = 150.0 # sampling rate
Ts = 1.0/Fs # sampling interval
t = np.arange(0, 1, Ts) # time vector
ff1 = 5 # frequency of the signal
sqwave=0
c1=1
plt.figure(305)
for loop in np.arange(1, 10, 1):
    sqwave =
sqwave+(1/c1)*np.cos(c1*2*np.pi*ff1*
t)+(1/c1)*np.sin(c1*2*np.pi*ff1*t)
    # fourier series
    plt.plot(t, sqwave)
    c1=c1+1
plt.pause(3)
```



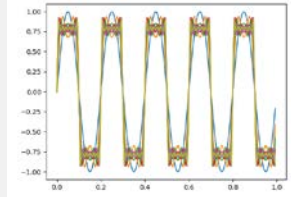
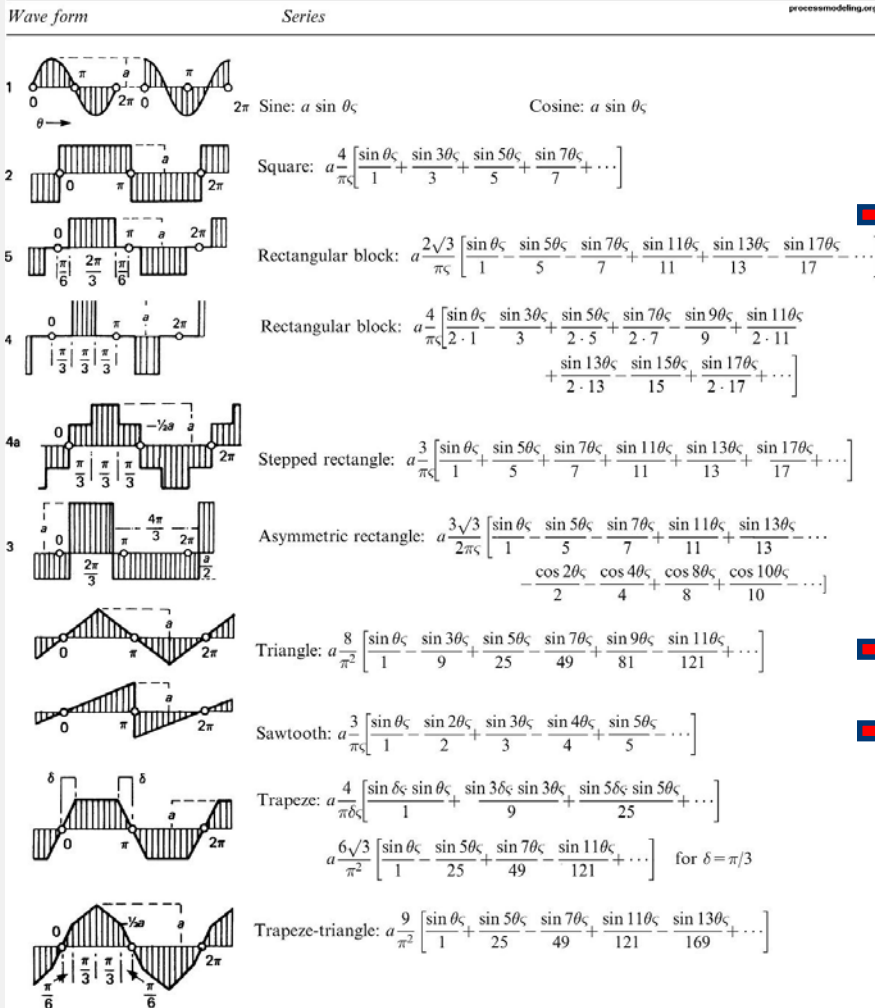
Fourier Series

- 두 개 이상의 정현파 더하기 → $\cos()$ 홀수+짝수 정현파와 $\sin()$ 홀수+짝수 정현파 더하기 → 톱니같은(sin) 삼각형파(cos) 만 들기

- $$f(t) = 1 \cdot \cos(2\pi (1 \cdot f_0)t) + 1 \cdot \sin(2\pi (1 \cdot f_0)t) + \frac{1}{2} \cdot \cos(2\pi (2 \cdot f_0)t) + \frac{1}{2} \cdot \sin(2\pi (2 \cdot f_0)t) + \frac{1}{3} \cdot \cos(2\pi (3 \cdot f_0)t) + \frac{1}{3} \cdot \sin(2\pi (3 \cdot f_0)t) + \dots$$
$$f(t) = a_0 + \sum_{n=1}^{\infty} (a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t)) = a_0 + a_1 \cos(1\omega_0 t) + a_2 \cos(2\omega_0 t) + a_3 \cos(3\omega_0 t) + \dots$$
$$+ b_1 \sin(1\omega_0 t) + b_2 \sin(2\omega_0 t) + a_3 \sin(3\omega_0 t) + \dots$$

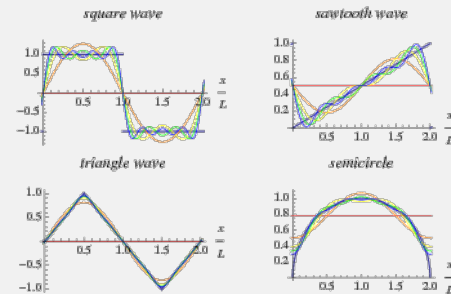


Fourier Series



sin() 홀수 정현파 더하기 → **사각형파 만들기**

$$f(t) = 1 \cdot \sin(2\pi (1 \cdot f_0)t) + 1/3 \cdot \sin(2\pi (3 \cdot f_0)t) + 1/5 \cdot \sin(2\pi (5 \cdot f_0)t) + \dots$$



cos() 홀수 정현파 더하기 → **삼각형파 만들기**

$$f(t) = 1 \cdot \cos(2\pi (1 \cdot f_0)t) + 1/3 \cdot \cos(2\pi (3 \cdot f_0)t) + 1/5 \cdot \cos(2\pi (5 \cdot f_0)t) + \dots$$

sin() 홀수+짝수 정현파 더하기 → **톱니파 만들기**

$$f(t) = 1 \cdot \sin(2\pi (1 \cdot f_0)t) + 1/2 \cdot \sin(2\pi (2 \cdot f_0)t) + 1/3 \cdot \sin(2\pi (3 \cdot f_0)t) + \dots$$

