Import libraries

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        import numpy.random as rnd
```

# Least-Squares Fit

Data obtained from experiments can contain a significant amount of random noise. We may wish to find a smooth curve that fits the data points on average. Ideally this curve should have a simple form (such as a low-order polynomial), so as to not reproduce the noise.

Consider the following function that will be used to fit a data set

$$\hat{y}(x) = \hat{y}(x; a_1, a_2, \ldots, a_m) = a_1 f_1(x) + a_2 f_2(x) + \ldots + a_m f_m(x)$$

where $a_1, a_2, \ldots, a_m$ are variable paramters. The data set consists of the points

$$(x_1, y_1),\ (x_2, y_2), \ldots (x_n, y_n)$$

where $m < n$. The form of $\hat{y}(x)$ is determined beforehand, ususally from the theory associated with the experiment from which the data was obtained. Therefore, the only means of adjusting the fit are the parameters themselves. The estimation of the data at each data point $x_i$ for is shown as

$$\begin{aligned}
\hat{y}(x_1) &= a_1 f_1(x_1) + a_2 f_2(x_1) + \ldots + a_m f_m(x_1) \\
\hat{y}(x_2) &= a_1 f_1(x_2) + a_2 f_2(x_2) + \ldots + a_m f_m(x_2) \\
&\vdots \\
\hat{y}(x_n) &= a_1 f_1(x_n) + a_2 f_2(x_n) + \ldots + a_m f_m(x_n)
\end{aligned}$$

In other words,

$$y \approx \hat{y} = Ax \quad \implies \quad \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \approx \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \dots & f_m(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_n) & f_2(x_n) & \dots & f_m(x_n) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}$$

The least-squares prolem seeks accomplish the following optimization problem

$$min \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

The solution to this least-squares problem is

$$\boxed{x = \left(A^T A\right)^{-1} A^T y}$$

## Linear Least-Squares with Exponential Functions

Suppose we want to fit data with an exponential curve. How does this change our approach compared to the simpler linear least-squares problem? Consider the following equation

$$y(x) = ae^{bx}$$

Recall the general form for a fitting function whose coefficients we wish to solve for. Notice that the exponential does not let us apply this form directly.

$$\hat{y}(x) = a_1 f_1(x) + a_2 f_2(x) + \ldots + a_m f_m(x)$$

We can use the properties of logarithms to our advantage and reshape this equation into a more suitable form. Start by taking the logarithm of both sides.

$$\begin{aligned} \ln y &= \ln ae^{bx} \\ &= \ln a + \ln e^{bx} \\ &= \ln a + bx \end{aligned}$$

Since $\ln a$ is a constant, we can treat this new equation as a monomial linear least-squares solution. It is important to note that when we fit this equation, we are looking for a solution that minimizes $(y - \hat{y})^2$ but rather $(\ln y - \ln \hat{y})^2$. A familiar equation emerges when we define $c = \ln a$ and treat it as one of our unknown coefficients. We can see this plainly when we set up our system of equations in matrix form.

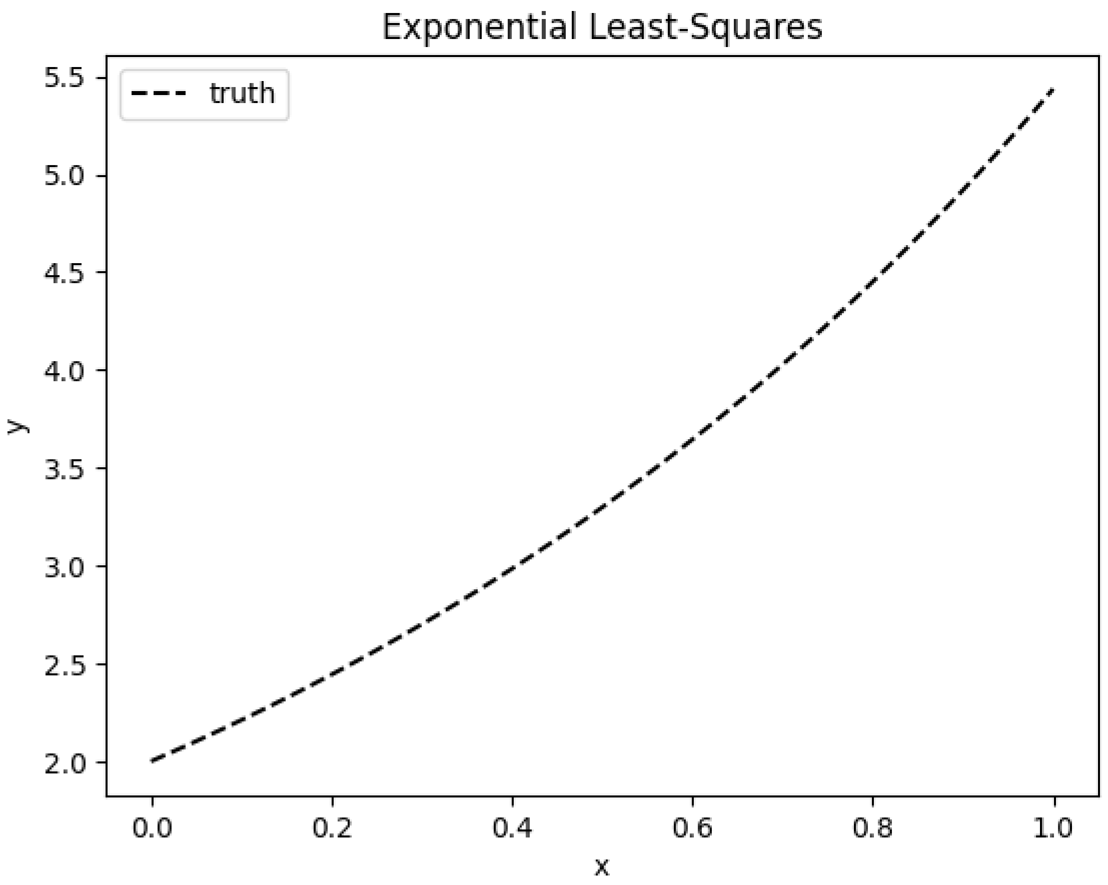
$$\ln \hat{y} = Ax$$

$$\begin{bmatrix} \ln y_1 \\ \ln y_2 \\ \vdots \\ \ln y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} c \\ b \end{bmatrix}$$

This now becomes a problem we can easily solve by taking the pseudoinverse of A and applying it to $\ln \hat{Y}$. Doing this will supply us with the optimal coefficient vector which contains $c$ and $b$. Recall how we defined $c$ and realize that we can use it to solve for $a$ with the equation $a = e^c$.

In [ ]:
```
x = np.linspace(0,1,50)
yTrue = 2*np.exp(x)

plt.plot(x,yTrue,'k--',label='truth')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Exponential Least-Squares')
plt.legend()
plt.show()
```



## Linear Least-Squares with Harmonic Functions

Suppose we also wish to fit data that would be best represented by harmonic behavior. This problem is similar in nature to that of the exponential least-squares as we cannot directly apply the general form of the fitting function.

$$\hat{y}(x) = A_1 \sin\left(\omega_1 x + \phi_1\right) + A_2 \sin\left(\omega_2 x + \phi_2\right) + \ldots + A_m \sin\left(\omega_m x + \phi_m\right)$$

Similar to taking advantage of logarithmic properties in the exponential case, we can take advantage of trigonometric identities to transform this equation into a form with linear constants that we can solve for. Recall the following trigonometric identities

$$\cos(a \pm b) = \cos a \cos b \mp \sin a \sin b$$
$$\sin(a \pm b) = \sin a \cos b \pm \cos a \sin b$$

We can use the second equation for the above example that uses sines. By applying this equation, we are eliminating the $\phi$ in within the trigonometric function and instead obtaining another constant that linearly scales a summed component.

$$\hat{y}(x) = [\alpha_1 \sin(\omega_1 x) + \beta_1 \cos(\omega_1 x)] + [\alpha_2 \sin(\omega_2 x) + \beta_2 \cos(\omega_2 x)] + \ldots + [\alpha_m \sin(\omega_m x) + \beta_m \cos(\omega_m x)]$$

where the following substitution was made

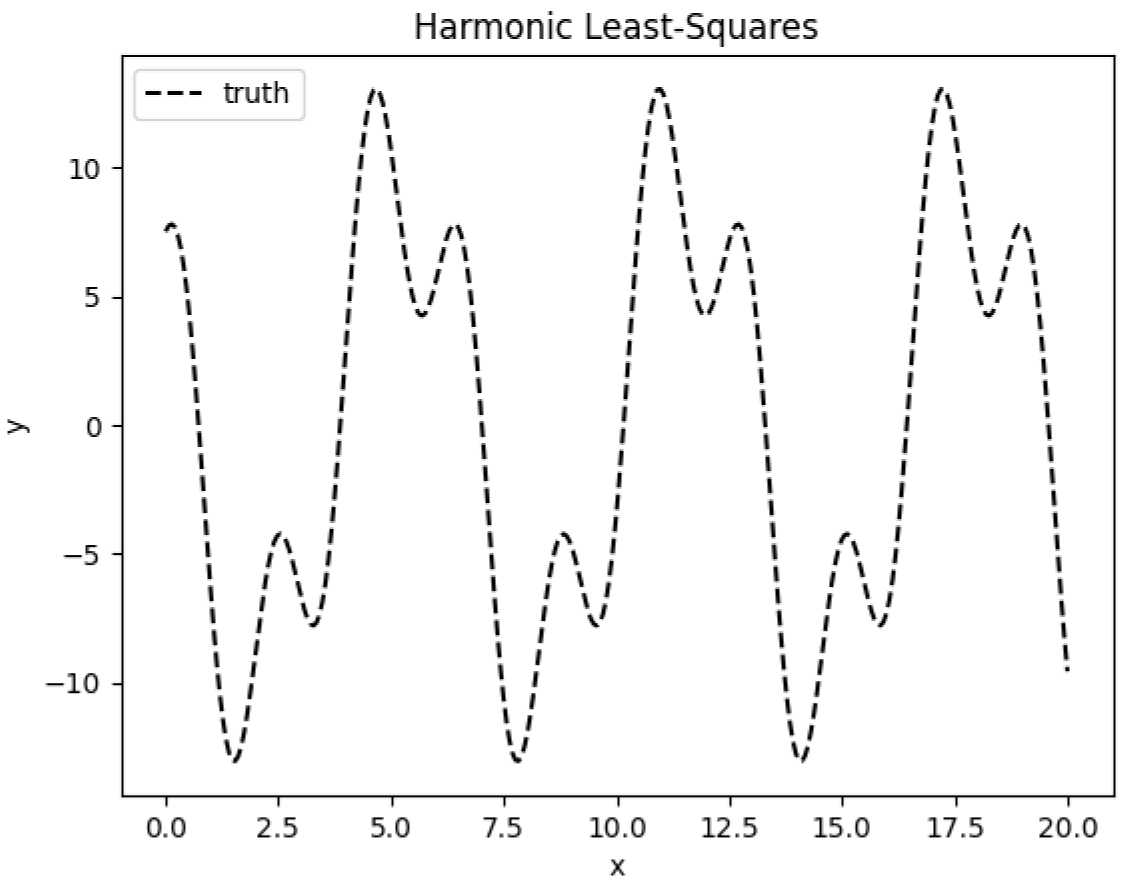
$$\alpha_i = A_i \cos\phi_i, \quad \beta_i = A_i \sin\phi_i$$

We can now convert this into the familiar matrix form of the least-squares problem.

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} \sin(\omega_1 x_1) & \cos(\omega_1 x_1) & \sin(\omega_2 x_1) & \cos(\omega_2 x_1) & \ldots & \sin(\omega_m x_1) & \cos(\omega_m x_1) \\ \sin(\omega_1 x_2) & \cos(\omega_1 x_2) & \sin(\omega_2 x_2) & \cos(\omega_2 x_2) & \ldots & \sin(\omega_m x_2) & \cos(\omega_m x_2) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \sin(\omega_1 x_n) & \cos(\omega_1 x_n) & \sin(\omega_2 x_n) & \cos(\omega_2 x_n) & \ldots & \sin(\omega_m x_n) & \cos(\omega_m x_n) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \beta_1 \\ \alpha_2 \\ \beta_2 \\ \vdots \\ \alpha_m \\ \beta_m \end{bmatrix}$$

In [ ]:
```python
x = np.linspace(0,20,300)
yTrue = 10*np.cos(x + np.pi/3) - 5*np.cos(3*x + 2*np.pi/3)

plt.plot(x,yTrue,'k--',label='truth')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Harmonic Least-Squares')
plt.legend()
plt.show()
```



## Practical Example: Mass-Spring-Damper System

The well-studied mass-spring-damper problem is governed by the following equation of motion for an unforced system

$$m\ddot{x} + c\dot{x} + kx = 0$$

and the has the general homogeneuos solution for the underdamped case

$$x(t) = C_1 e^{-\alpha t} \cos \omega t + C_2 e^{-\alpha t} \sin \omega t$$

where

$$\alpha = \frac{c}{2m}, \quad \omega = \sqrt{\frac{k}{m} - \left(\frac{c}{2m}\right)^2}$$

Let us assume that $m = 10$, $c = 5$, and $k = 20$ as well as the initial conditions $x(0) = 1$ and $\dot{x}(0) = 2$. Apply the following form

$$x(t) = x_0 e^{-\alpha t}\cos\omega t + \frac{\dot{x}_0 + \alpha x_0}{\omega} e^{-\alpha t}\cos\omega t$$

In [ ]:
```
# given values
m = 10
c = 5
k = 20

alpha = c / (2*m)
omega = np.sqrt(k/m - (c/(2*m))**2)

x0 = 1
dx0 = 2

##########
# SOLUTION
##########
```

In [ ]:
```
#############
# PLOT ERRORS
#############
```

# Orthogonality and Legendre & Chebychev Polynomials

## Orthogonality

Where Lagrange polynomials and monomials formed a linearly independent basis for a polynomial curve, Legendre and Chebychev polynomials have the additional benefit of being sets of orthogonal functions. By definition, an orthogonal polynomial sequence in a family of polynomials such that two different polynomials in the sequence are orthogonal to each other under some inner product. Two vectors $x$ and $y$ are said to be orthogonal if their inner product is zero, that is,

$$\langle x, y\rangle = 0$$

Both Legendre and Chebychev polynomials can be used to fit or interpolate data.

## Legendre Polynomials

The orthogonality of Legendre polynomials is defined with respect to the inner product

$$\langle P_m, P_n \rangle = \int_{-1}^{1} P_m(x) P_n(x) dx = \begin{cases} 0 & \text{if } m \neq n \\ \frac{2}{2n+1} & \text{if } m = n \end{cases}$$

The Legendre polynomials are defined as

$$P_0(x) = 1$$

$$P_1(x) = x$$

$$P_k(x) = \frac{2k-1}{k} x P_{k-1}(x) - \frac{k-1}{k} P_{k-2}(x), \qquad x \in [-1, 1]$$

## Chebychev Polynomials

The Chebychev polynomials are two sequences of polynomials related to the cosine and sine functions and are denoted by $T_n(x)$ and $U_n(x)$. These functions are given by the following equations

$$\begin{cases} \text{First kind} & T_0(x) = 1 & T_1(x) = x & T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x) \\ \text{Second kind} & U_0(x) = 1 & U_1(x) = 2x & T_{k+1}(x) = 2xU_k(x) - U_{k-1}(x) \end{cases}$$

and

$$T_k(\cos\theta) = \cos k\theta, \qquad U_k(\cos\theta)\sin\theta = \sin((k+1)\theta)$$

These functions are related by their derivatives and integrals such that

$$\frac{dT_k}{dx} = kU_{k-1}, \qquad \frac{dU_k}{dx} = \frac{(k+1)T_{k+1} - xU_k}{x^2 - 1}$$

and

$$\int T_k dx = \frac{kT_{k+1}}{k^2-1} - \frac{xT_k}{k-1}, \qquad \int U_k dx = \frac{T_{k+1}}{k+1}$$

The orthogonality of Chebychev polynomials is defined with respect to the inner product

$$\langle T_m, T_n \rangle = \int_{-1}^{1} T_m(x) T_n(x) \frac{dx}{\sqrt{1-x^2}} = \begin{cases} 0 & \text{if } m \neq n \\ \pi & \text{if } m = n = 0 \\ \pi/2 & \text{if } m = n \neq 0 \end{cases}$$