2.3 Homework #2 (10 points)

1- For a quadratic linear regression model $y = ax^2 + bx + c$, analytically find a, b and c by minimizing the residual cost function $J(a, b, c)$. ***(3 points).***

2- Let's assume $y = 1.5x^2 + 2x + 1 + \epsilon_n$. where $\epsilon_n \sim N(0,1)$. ϵ is an additive noise. Write a Python script to calculate true values of y for x, which is an array of 50 equally spaced values between 0 and 10. Use np.random.seed(0). a) Define the cost function for the quadratic regression and minimize it to find the best estimate of a, b and c. ***(4 points)***. And then b) use the Negative Log-Likelihood function as the cost function and minimize it to estimate a, b and c. Compare the results of a) and b). ***(3 points)***

Bonus- Find/design an example with dataset in your area/research field that can be represented by Poisson regression model. Then derive the Likelihood Function and maximize it (or minimize Negative Log-Likelihood) to estimate the model parameters. ***(2 points)***

In [7]:
```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import subplots
import seaborn as sns
import numpy as np
import scipy.integrate as spi
import scipy.stats as stats
import scipy.optimize as opt
```

1: For a quadratic linear regression model $y = ax^2 + bx + c$, analytically find a, b and c by minimizing the residual cost function $J(a, b, c)$. (3 points).

CE410: MSTA	HW2	Augeung, Brandon

Given: $y = ax^2 + bx + c$

Find: The least squares regression estimates

solution:

$$J(a,b,c) = \sum_{i=1}^{n} \varepsilon_i^2 = \sum \left(y_i - (ax_i^2 + bx_i + c)\right)^2 \qquad * \sum_{i=1}^{n} = \sum$$

$$J(a,b,c) = \sum \Big[a^2 x_i^4 + 2abx_i^3 + 2acx_i^2 - 2ax_i^2 y_i + b^2 x_i^2 + \cdots$$
$$\cdots 2bcx_i - 2bx_i y_i + c^2 + 2cy_i + y_i^2 \Big]$$

PARTIALS:

$$\frac{\partial J(a,b,c)}{\partial a} = \sum \left[2ax_i^4 + 2bx_i^3 + 2cx_i^2 - 2x_i^2 y_i \right] = 0 \qquad (1)$$

$$\frac{\partial J(a,b,c)}{\partial b} = \sum \left[2ax_i^3 + 2bx_i^2 + 2cx_i - 2x_i y_i \right] = 0 \qquad (2)$$

$$\frac{\partial J(a,b,c)}{\partial c} = \sum \left[2ax_i^2 + 2bx_i + 2c - 2y_i \right] = 0 \qquad (3)$$

call (3): $0 = a\sum x_i^2 + b\sum x_i + cn - \sum y_i$

$$-cn = a\sum x_i^2 + b\sum x_i - \sum y_i$$

$$c = -n^{-1}\left[a\sum x_i^2 + b\sum x_i - \sum y_i\right]$$

☆ $$c = \bar{y} - a\overline{x^2} - b\bar{x}$$

call ①: $0 = \sum a x_i^4 + b x_i^3 + c x_i^2 - x_i^2 y_i$

$$0 = a\sum x_i^4 + b\sum x_i^3 + c\sum x_i^2 - \sum x_i^2 y_i$$

$$0 = a\sum x_i^4 + b\sum x_i^3 + \left(\frac{\sum y_i}{n} - a\frac{\sum x_i^2}{n} - b\frac{\sum x_i}{n}\right)\sum x_i^2 - \sum x_i^2 y_i$$

$$0 = a\sum x_i^4 + b\sum x^3 + \frac{\sum y_i \sum x_i^2}{n} - a\frac{[\sum x_i^2]^2}{n} - b\frac{\sum x_i^2 \sum x_i}{n} - \sum x_i^2 y_i$$

$$0 = a\underbrace{\left(\sum x_i^4 - \frac{[\sum x_i^2]^2}{n}\right)}_{S(x^2x^2)} + b\underbrace{\left(\sum x_i^3 - \frac{\sum x_i^2 \sum x_i}{n}\right)}_{S(xx^2)} + \frac{\sum y_i \sum x_i^2}{n} - \sum x_i^2 y_i$$

$$0 = a\,S(x^2x^2) + b\,S(xx^2) + \frac{\sum y_i \sum x_i^2}{n} - \sum x_i^2 y_i$$

$$0 = a\,S(x^2x^2) + \left(\frac{S(xy) - aS(xx^2)}{S(xx)}\right)S(xx^2) + \frac{\sum y_i \sum x_i^2}{n} - \sum x_i^2 y_i$$

$$0 = a\,S(x^2x^2) + \frac{S(xy)S(xx^2) - a(S(xx^2))^2}{S(xx)} + \frac{\sum y_i \sum x_i^2}{n} - \sum x_i^2 y_i$$

$$0 = a\left(S(x^2x^2) - \frac{(S(xx^2))^2}{S(xx)}\right) + \frac{S(xy)S(xx^2)}{S(xx)} + \frac{\sum y_i \sum x_i^2}{n} - \sum x_i^2 y_i$$

$$-a\left(S(x^2x^2) - \frac{(S(xx^2))^2}{S(xx)}\right) = \frac{S(xy)S(xx^2)}{S(xx)} + \frac{\sum y_i \sum x_i^2}{n} - \sum x_i^2 y_i$$

$$-a = \frac{\frac{S(xy)S(xx^2)}{S(xx)} + \frac{\sum y_i \sum x_i^2}{n} - \sum x_i^2 y_i}{S(x^2x^2) - \frac{(S(xx^2))^2}{S(xx)}}$$

$$a = \frac{\overbrace{\sum x_i^2 y_i - \frac{\sum y_i \sum x_i^2}{n}}^{S(x^2y)} - \frac{S(xy)S(xx^2)}{S(xx)}}{S(x^2x^2) - \frac{(S(xx^2))^2}{S(xx)}} = \frac{S(x^2y) - \frac{S(xy)S(xx^2)}{S(xx)}}{S(x^2x^2) - \frac{(S(xx^2))^2}{S(xx)}}$$

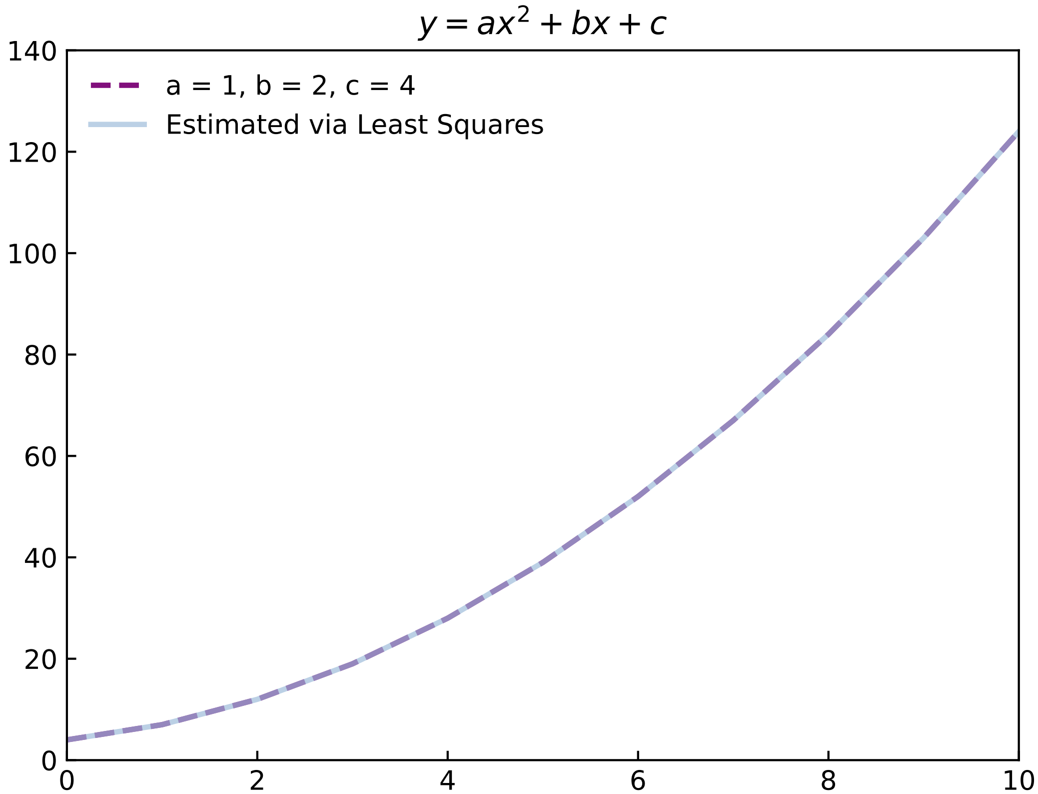
$$a = \frac{S(x^2y)S(xx) - S(xy)S(xx^2)}{S(x^2x^2)S(xx) - (S(xx^2))^2}$$

$$b = \frac{S(xy) - aS(xx^2)}{S(xx)} \qquad c = \frac{\sum y_i}{n} - a\frac{\sum x_i^2}{n} - b\frac{\sum x_i}{n}$$

In [8]:
```
# check analytical solution using a known parameter set
a_set = 1
b_set = 2
c_set = 4
x_set = np.linspace(0,10,11)
y_set = a_set*x_set**2 + b_set*x_set + c_set
n = len(x_set)

# analytical solution
Sxx = np.sum(x_set**2) - (np.sum(x_set)**2/n)
Sxy = np.sum(x_set*y_set) - ((np.sum(x_set)*np.sum(y_set))/n)
Sxx2 = np.sum(x_set**3) - ((np.sum(x_set)*np.sum(x_set**2))/n)
```

```
Sx2y = np.sum((x_set**2)*y_set) - ((np.sum(x_set**2)*np.sum(y_set))/n)
Sx2x2 = np.sum(x_set**4) - ((np.sum(x_set**2)**2)/n)

a_mod = ((Sx2y * Sxx) - (Sxy * Sxx2)) / ((Sxx * Sx2x2) - (Sxx2**2))
b_mod = ((Sxy * Sx2x2) - (Sx2y * Sxx2))/((Sxx * Sx2x2) - (Sxx2**2))
c_mod = (np.sum(y_set) - a_mod*np.sum(x_set**2) - b_mod*np.sum(x_set)) / n

print("a = ", a_mod)
print("b = ", b_mod)
print("c = ", c_mod)

y_mod = a_mod*x_set**2 + b_mod*x_set + c_mod

# plot
fig, ax = subplots(1,1)
ax.plot(x_set,y_set, '--', label = "a = 1, b = 2, c = 4", color ="#810f7c", linewidth = 2)
ax.plot(x_set,y_mod, label = "Estimated via Least Squares", color = "#9ebcda", linewidth = 2, alpha = 0.7)
# plot your estimates from the least squares fit here. this should match perfectly, since the data has no noise.
ax.set_title('$y = ax^2 + bx + c$')
ax.tick_params(which="both", direction = "in")
ax.legend(frameon = False)
ax.set_xbound(0,10)
ax.set_ybound(0,140)
```

```
a =  1.0
b =  2.0
c =  4.0
```



2: Let's assume $y = 1.5x^2 + 2x + 1 + \epsilon; \epsilon \sim \mathcal{N}(0, 1)$. ϵ is an additive noise. Write a Python script to calculate true values of y for x, which is an array of 50 equally spaced values between 0 and 10. Use np.random.seed(0).

a) Define the cost function for the quadratic regression and minimize it to find the best estimate of a, b and c. (4 points).

b) use the Negative Log-Likelihood function as the cost function and minimize it to estimate a, b and c. Compare the results of a) and b). (3 points)

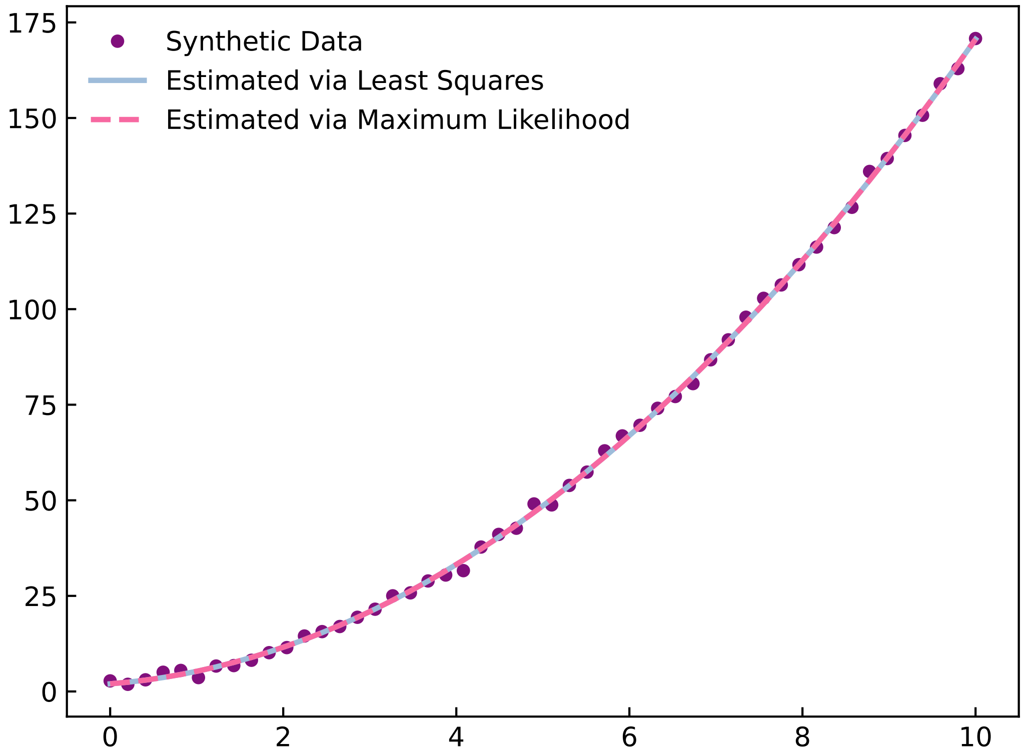
In [ ]:

```
# create synthetic data set to fit curve to
np.random.seed(0) # set seed for reproducibility
a = 1.5
b = 2
c = 1
epsilon = np.random.normal(0, 1, len(x)) # Additive noise
x = np.linspace(0,10,50)
y = a*x**2 + b*x + c + epsilon
```

```python
# plot
fig, ax = subplots(1,1)
ax.plot(x,y, 'o', label = "Synthetic Data", color ="#810f7c", markersize = 4)
ax.tick_params(which="both", direction = "in")
ax.set_xbound(0,10)
ax.set_ybound(0,140)

# Define the cost function to be minimized
def cost_function(beta2, beta1, beta0):
 predictions = beta2*x**2 + beta1*x + beta0
 residuals = y - predictions
 return np.sum(residuals**2)

# Use an optimization method to minimize the cost function
from scipy.optimize import minimize
initial_guess = [0, 0, 0] # Initial guess for a, b, and c
result = minimize(lambda params: cost_function(params[0], params[1], params[2]), initial_guess)
a_est, b_est, c_est = result.x
# Display the estimated parameters
print("Via Least Squares:")
print(f" Estimated a: {a_est:.3f}")
print(f" Estimated b: {b_est:.3f}")
print(f" Estimated c: {c_est:.3f}")

y_pred = a_est*x**2 + b_est*x + c_est
ax.plot(x,y_pred, label = "Estimated via Least Squares", color = "#9ebcda", linewidth = 2)
ax.legend(frameon = False)

# use negative log-likelihood to determine goodness of fit
def negative_log_likelihood(theta, y_data, error):
    y_model = theta[0]*x**2 + theta[1]*x + theta[2]
    return -1 * -0.5 * np.sum(np.log(2*np.pi*error**2) + ((y_data - y_model)**2)/(error**2))

theta_est = opt.fmin(negative_log_likelihood, [0, 0, 0], args=(y, 1)) # ask Peyman abt whether to use 1 or epsilon
print("Via MLE:")
print(f" Estimated a: {theta_est[0]:.3f}")
print(f" Estimated b: {theta_est[1]:.3f}")
print(f" Estimated c: {theta_est[2]:.3f}")

y_pred = theta_est[0]*x**2 + theta_est[1]*x + theta_est[2]
ax.plot(x,y_pred, label = "Estimated via Maximum Likelihood", color = "#f768a1", linewidth = 2, linestyle= '--')
ax.legend(frameon = False)
```

```
Via Least Squares:
 Estimated a: 1.509
 Estimated b: 1.767
 Estimated c: 1.997
Optimization terminated successfully.
         Current function value: 73.168419
         Iterations: 269
         Function evaluations: 488
Via MLE:
 Estimated a: 1.509
 Estimated b: 1.767
 Estimated c: 1.997
```

```
Out[ ]: <matplotlib.legend.Legend at 0x7a90b5d85160>
```



Least Squares and MLE provide the same answer when using (y, 1) as the arguments. when using (y, epsilon) as the arguments, the answer is different; MLE is more accurate than Least Squares.