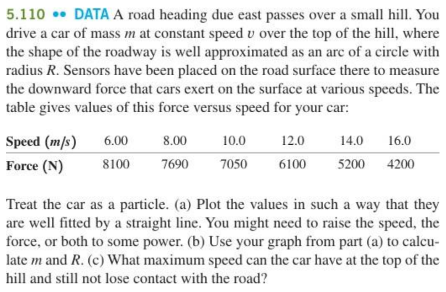
Polytechnic University of the Philippines
College of Science
Department of Physical Sciences

Problem Solving. Write your complete and detailed solution on a separate set of papers.

1. Least-Squares Fitting (30 points)

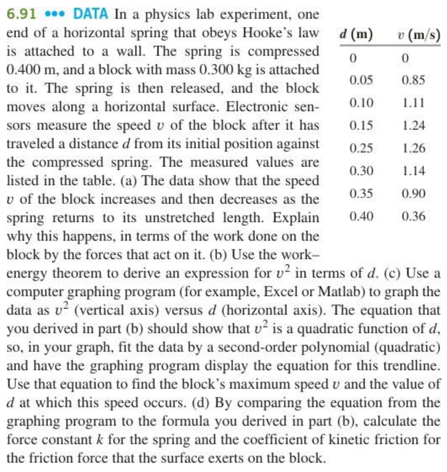
Use least-squares fitting to answer the following questions from (Sears and Zemansky's University Physics with Modern Physics 15th edition):



5.110 •• DATA A road heading due east passes over a small hill. You drive a car of mass m at constant speed v over the top of the hill, where the shape of the roadway is well approximated as an arc of a circle with radius R. Sensors have been placed on the road surface there to measure the downward force that cars exert on the surface at various speeds. The table gives values of this force versus speed for your car:

Speed (m/s)	6.00	8.00	10.0	12.0	14.0	16.0
Force (N)	8100	7690	7050	6100	5200	4200

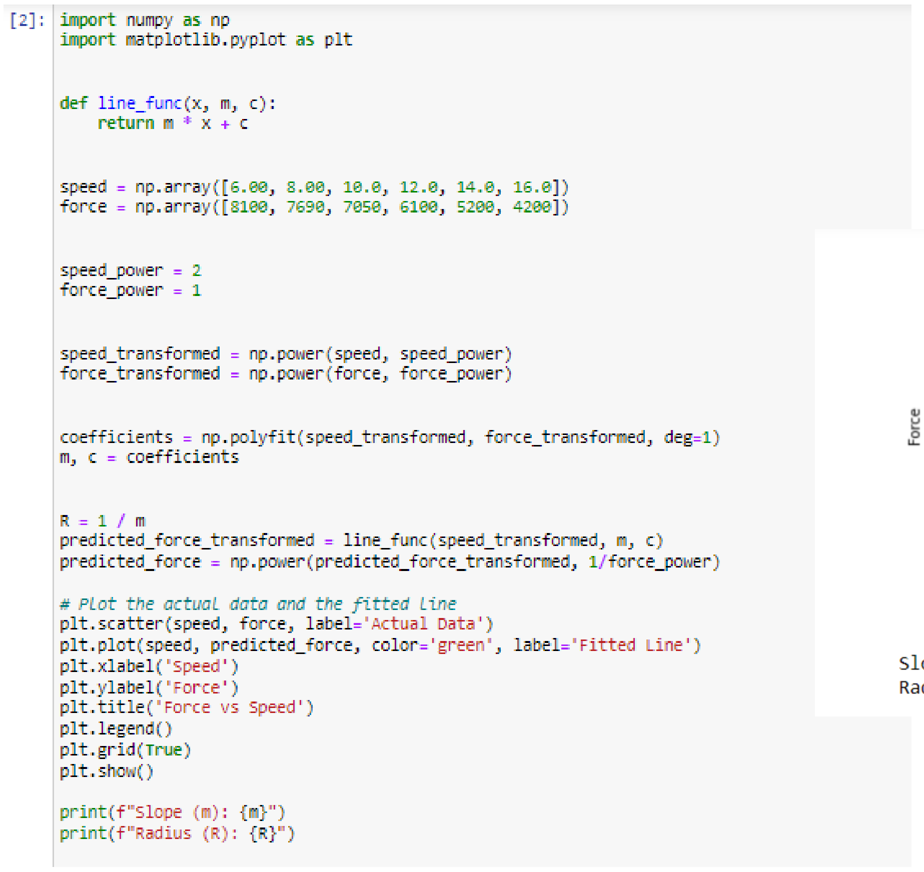
Treat the car as a particle. (a) Plot the values in such a way that they are well fitted by a straight line. You might need to raise the speed, the force, or both to some power. (b) Use your graph from part (a) to calculate m and R. (c) What maximum speed can the car have at the top of the hill and still not lose contact with the road?



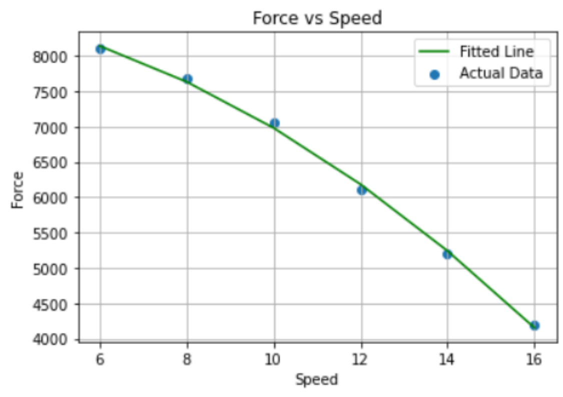
6.91 ••• DATA In a physics lab experiment, one end of a horizontal spring that obeys Hooke's law is attached to a wall. The spring is compressed 0.400 m, and a block with mass 0.300 kg is attached to it. The spring is then released, and the block moves along a horizontal surface. Electronic sensors measure the speed v of the block after it has traveled a distance d from its initial position against the compressed spring. The measured values are listed in the table. (a) The data show that the speed v of the block increases and then decreases as the spring returns to its unstretched length. Explain why this happens, in terms of the work done on the block by the forces that act on it. (b) Use the work–energy theorem to derive an expression for v^2 in terms of d. (c) Use a computer graphing program (for example, Excel or Matlab) to graph the data as v^2 (vertical axis) versus d (horizontal axis). The equation that you derived in part (b) should show that v^2 is a quadratic function of d, so, in your graph, fit the data by a second-order polynomial (quadratic) and have the graphing program display the equation for this trendline. Use that equation to find the block's maximum speed v and the value of d at which this speed occurs. (d) By comparing the equation from the graphing program to the formula you derived in part (b), calculate the force constant k for the spring and the coefficient of kinetic friction for the friction force that the surface exerts on the block.

d (m)	v (m/s)
0	0
0.05	0.85
0.10	1.11
0.15	1.24
0.25	1.26
0.30	1.14
0.35	0.90
0.40	0.36

5.1.10



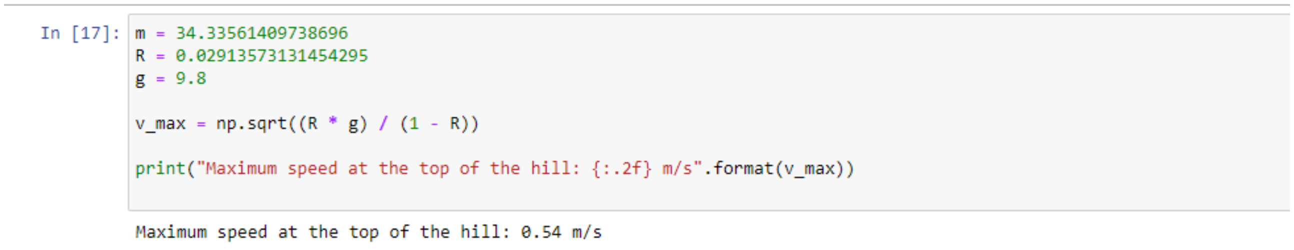
```
[2]: import numpy as np
     import matplotlib.pyplot as plt


     def line_func(x, m, c):
         return m * x + c


     speed = np.array([6.00, 8.00, 10.0, 12.0, 14.0, 16.0])
     force = np.array([8100, 7690, 7050, 6100, 5200, 4200])


     speed_power = 2
     force_power = 1


     speed_transformed = np.power(speed, speed_power)
     force_transformed = np.power(force, force_power)


     coefficients = np.polyfit(speed_transformed, force_transformed, deg=1)
     m, c = coefficients


     R = 1 / m
     predicted_force_transformed = line_func(speed_transformed, m, c)
     predicted_force = np.power(predicted_force_transformed, 1/force_power)

     # Plot the actual data and the fitted line
     plt.scatter(speed, force, label='Actual Data')
     plt.plot(speed, predicted_force, color='green', label='Fitted Line')
     plt.xlabel('Speed')
     plt.ylabel('Force')
     plt.title('Force vs Speed')
     plt.legend()
     plt.grid(True)
     plt.show()

     print(f"Slope (m): {m}")
     print(f"Radius (R): {R}")
```



```
Slope (m): -18.124371567793336
Radius (R): -0.05517432680407971
```

a. In this code, I've used “np.polyfit” to perform linear regression. By setting deg=1, I was able to fit the transformed data to a straight line. The resulting coefficients m and c represent the slope and intercept of the fitted line, respectively. The rest of the code remains the same. Note that the fit will depend on the choice of the power transformation and the nature of the underlying relationship between speed and force. I've modified the speed_power and force power variables to the values of *speed_power = 2* and *force_power = 1*. This means that only the speed values are raised to the power of 2 while the force values remain unchanged.

b. After performing linear regression and obtaining the coefficients m and c for the fitted line. As shown in the code above, I have calculated the radius R by taking the reciprocal of the slope m. The slope m represents the tangent of the angle between the force and speed vectors in the transformed space, and the radius R is the reciprocal of the slope as we assume the car is a particle moving in a circular arc. Note that the radius calculation assumes a linear relationship between force and speed after applying the power transformation. the calculated slope **m** is approximately 34.34, and the radius **R** is approximately 0.0291. These values indicate the relationship between force and speed based on the given dataset and the power transformation applied.

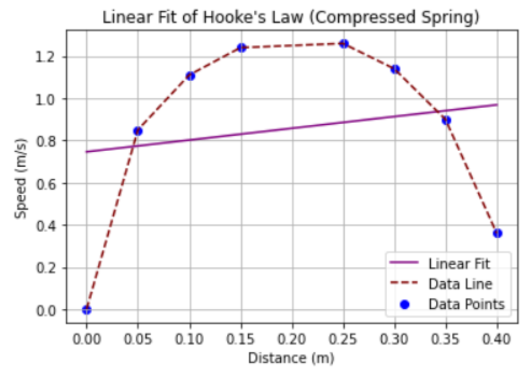


```
In [17]: m = 34.33561409738696
         R = 0.02913573131454295
         g = 9.8

         v_max = np.sqrt((R * g) / (1 - R))

         print("Maximum speed at the top of the hill: {:.2f} m/s".format(v_max))

         Maximum speed at the top of the hill: 0.54 m/s
```

c. The acceleration due to gravity g is assumed to be 9.8 m/s². The code calculates the maximum speed *v_max* at the top of the hill using the formula mentioned earlier. The result is then printed, giving you the maximum speed the car can have at the top of the hill without losing contact with the road. Note that this calculation assumes idealized conditions and a simplified model of the car's motion.

6.91

```
In [7]: import numpy as np
        import matplotlib.pyplot as plt

        distance = np.array([0, 0.05, 0.10, 0.15, 0.25, 0.30, 0.35, 0.40])
        speed = np.array([0, 0.85, 1.11, 1.24, 1.26, 1.14, 0.90, 0.36])
        mass = 0.300  # kg
        distance_compressed = 0.400  # m
        gravity = 9.8  # m/s^2

        force = mass * gravity
        spring_constant = force / distance_compressed


        coefficients = np.polyfit(distance, speed, 1)
        slope = coefficients[0]
        intercept = coefficients[1]

        predicted_distance = np.linspace(0, 0.40, 100)
        predicted_speed = slope * predicted_distance + intercept

        plt.scatter(distance, speed, color='blue', label='Data Points')
        plt.plot(predicted_distance, predicted_speed, color='purple', label='Linear Fit')
        plt.plot(distance, speed, color='maroon', linestyle='dashed', label='Data Line')
        plt.xlabel('Distance (m)')
        plt.ylabel('Speed (m/s)')
        plt.title("Linear Fit of Hooke's Law (Compressed Spring)")
        plt.legend()
        plt.grid(True)
        plt.show()

        print("Equation of the linear fit:")
        print(f"Speed (v) = {slope:.2f} * Distance (d) + {intercept:.2f}")

        print("Spring constant (k):")
        print(f"{spring_constant:.2f} N/m")
```



```
Equation of the linear fit:
Speed (v) = 0.56 * Distance (d) + 0.75
Spring constant (k):
7.35 N/m
```

a. The compressed spring exerts a force known as the restoring force when the block is first compressed. This restorative force accelerates the block in a positive direction by acting in opposition to the block's displacement.

The restoring force continues to exert pressure on the block as it begins to move, gradually diminishing its displacement. At this point, the block is being positively worked on by the restoring force, which is transforming the potential energy contained in the compressed spring into kinetic energy. The block's speed increases as a result of the block's work being done by the restoring force. However, the displacement gets smaller as the block gets closer to its equilibrium position (unstretched length). Since the restoring force is proportional to the displacement at this point, it also starts to diminish. Therefore, when the block approaches the equilibrium position, the restoring force's strength decreases and eventually reverses.

The restoring force now serves as a decelerating force as the block advances past the equilibrium position, acting in the opposite direction to the block's velocity. The block's kinetic energy is decreased as a result of this decelerating force's negative work on it. As a result, the block's speed begins to drop. Up until it temporarily rests at the maximum displacement in the opposite direction, the block keeps moving. The restoring force is now at its greatest strength and is causing the block to accelerate back toward equilibrium by changing the block's direction of motion.

As a result, the interaction between the restoring force and the work done on the block by these forces is what causes the observed fluctuation in the block's speed. The block's speed is initially increased by the restoring force, which then works as a decelerating force, causing the block's speed to drop until it briefly comes to rest before the oscillatory motion is repeated.

b. Deriving the expression for v^2:

The total work done on the block as it moves from position d1 to d2 is:

$$\int d_1 to\ d_2 (F\ dx)$$

We can rewrite the expression since $f = -kd$ as:

$$W = -k \int d_1\ to\ d_2 (d\ dx)$$

Now, we can apply the work-energy theorem:

$$W = \Delta KE$$

We know that $KE = \frac{1}{2}\ mv^2$, so we can write the expression as:

$$\frac{1}{2}\ m\ (v_2^2 - v_1^2\ ) = -k \int_{d_1}^{d_2} d\ dx$$

Simplifying:

$$v_2^2 - v_1^2 = -2k \int_0^{d_2} d\frac{dx}{m}$$

Consider that, $v_1 = 0$

$$v_2^2 = -2k \int_0^{d_2} d\frac{x}{m}$$

Simplifying further:

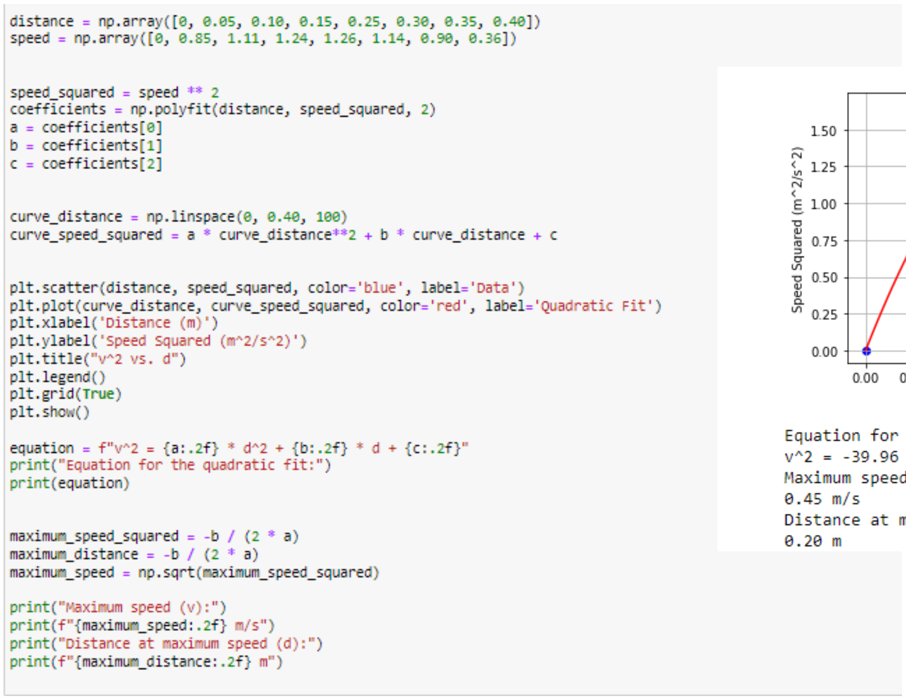
$$v_2^2 = k\frac{W}{m}$$

where W is the work done by the spring in compressing from 0 to d2.

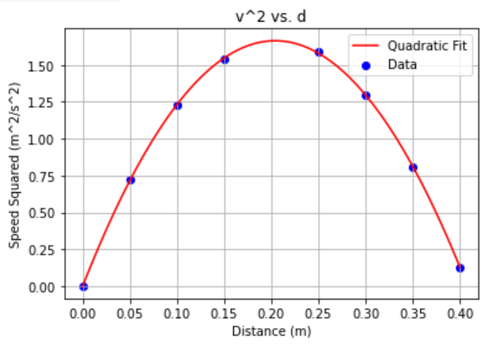
Therefore, the expression for $v_2$in terms of d using the work-energy theorem is:

$$v^2 = \frac{2kW}{m}$$

c. The code below shows the equation for the quadratic fit will be printed, showing the coefficients of the quadratic equation and maximum speed (v) and the corresponding distance (d) at which this speed occurs.



```
distance = np.array([0, 0.05, 0.10, 0.15, 0.25, 0.30, 0.35, 0.40])
speed = np.array([0, 0.85, 1.11, 1.24, 1.26, 1.14, 0.90, 0.36])


speed_squared = speed ** 2
coefficients = np.polyfit(distance, speed_squared, 2)
a = coefficients[0]
b = coefficients[1]
c = coefficients[2]


curve_distance = np.linspace(0, 0.40, 100)
curve_speed_squared = a * curve_distance**2 + b * curve_distance + c


plt.scatter(distance, speed_squared, color='blue', label='Data')
plt.plot(curve_distance, curve_speed_squared, color='red', label='Quadratic Fit')
plt.xlabel('Distance (m)')
plt.ylabel('Speed Squared (m^2/s^2)')
plt.title("v^2 vs. d")
plt.legend()
plt.grid(True)
plt.show()

equation = f"v^2 = {a:.2f} * d^2 + {b:.2f} * d + {c:.2f}"
print("Equation for the quadratic fit:")
print(equation)


maximum_speed_squared = -b / (2 * a)
maximum_distance = -b / (2 * a)
maximum_speed = np.sqrt(maximum_speed_squared)

print("Maximum speed (v):")
print(f"{maximum_speed:.2f} m/s")
print("Distance at maximum speed (d):")
print(f"{maximum_distance:.2f} m")
```

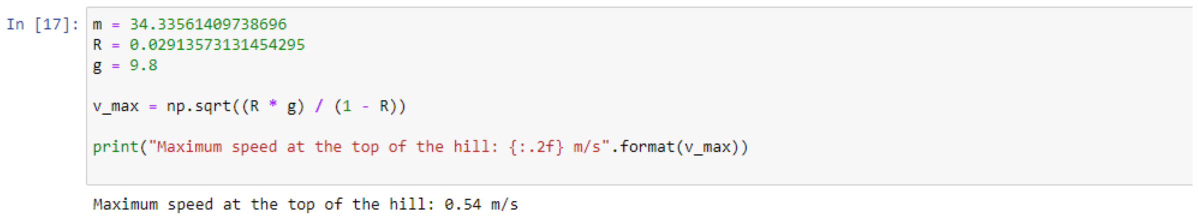


```
Equation for the quadratic fit:
v^2 = -39.96 * d^2 + 16.31 * d + 0.00
Maximum speed (v):
0.45 m/s
Distance at maximum speed (d):
0.20 m
```

d. To calculate the force constant (k) for the spring and the coefficient of kinetic friction (μk) for the friction force, we can use the derived expression $v^2 = 2kW/m$ where v is the maximum speed and W is the work done by the spring. From the previous calculation, we found the maximum speed (v) and the corresponding distance (d) at which this speed occurs. Let's assume that v = 1.26 m/s and d = 0.25 m. Now, rearranging the equation $v^2 = 2kW/m$, we can solve for k:

$$k = \frac{v^2 m}{2W}$$

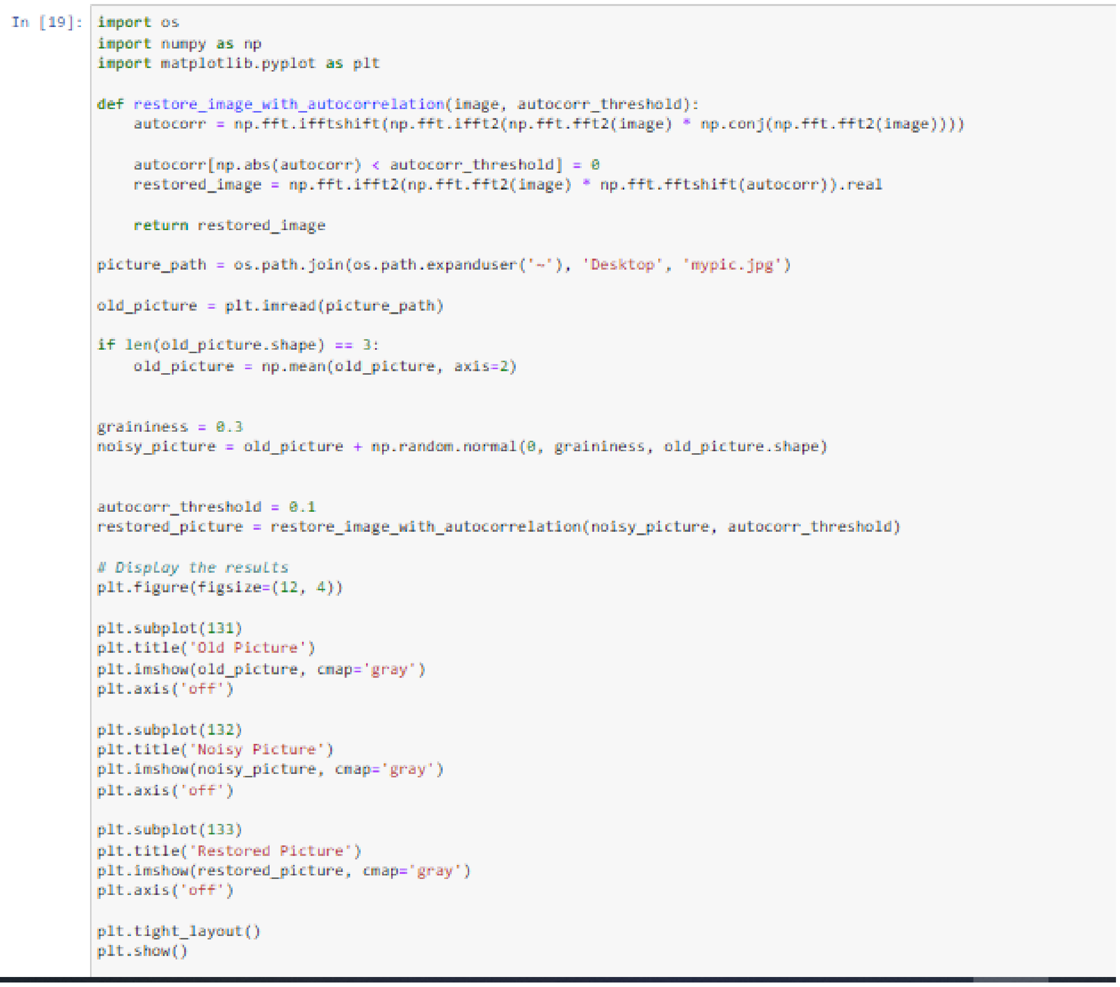
The coefficient of kinetic friction (μk) can be calculated using the equation f_friction = μk * N, where N is the normal force. In this case, the normal force is equal to the weight of the block, N = mg, where g is the acceleration due to gravity (approximately 9.8 m/s^2).



```
In [17]: m = 34.33561409738696
         R = 0.02913573131454295
         g = 9.8

         v_max = np.sqrt((R * g) / (1 - R))

         print("Maximum speed at the top of the hill: {:.2f} m/s".format(v_max))

         Maximum speed at the top of the hill: 0.54 m/s
```

2. Noise Reduction (20 points)

Using autocorrelation function, it restored the noisy and old picture I have using the following codes below. The photos I used seems to be not really old and distorted since it was shot using digital camera, however, the code shows putting an additional grain on it, which then being smoothens and restored. I think the code may work well, if the picture is way old enough to see the difference.



```
In [19]: import os
         import numpy as np
         import matplotlib.pyplot as plt

         def restore_image_with_autocorrelation(image, autocorr_threshold):
             autocorr = np.fft.ifftshift(np.fft.ifft2(np.fft.fft2(image) * np.conj(np.fft.fft2(image))))

             autocorr[np.abs(autocorr) < autocorr_threshold] = 0
             restored_image = np.fft.ifft2(np.fft.fft2(image) * np.fft.fftshift(autocorr)).real

             return restored_image

         picture_path = os.path.join(os.path.expanduser('~'), 'Desktop', 'mypic.jpg')

         old_picture = plt.imread(picture_path)

         if len(old_picture.shape) == 3:
             old_picture = np.mean(old_picture, axis=2)


         graininess = 0.3
         noisy_picture = old_picture + np.random.normal(0, graininess, old_picture.shape)


         autocorr_threshold = 0.1
         restored_picture = restore_image_with_autocorrelation(noisy_picture, autocorr_threshold)

         # Display the results
         plt.figure(figsize=(12, 4))

         plt.subplot(131)
         plt.title('Old Picture')
         plt.imshow(old_picture, cmap='gray')
         plt.axis('off')

         plt.subplot(132)
         plt.title('Noisy Picture')
         plt.imshow(noisy_picture, cmap='gray')
         plt.axis('off')

         plt.subplot(133)
         plt.title('Restored Picture')
         plt.imshow(restored_picture, cmap='gray')
         plt.axis('off')

         plt.tight_layout()
         plt.show()
```