```
import matplotlib.pyplot as plt

import numpy as np



time_averaged = [3.006, 2.542, 2.112, 1.839, 2.773, 1.646, 2.328, 1.965,
    1.255]

delta_x = [0.4694, 0.3995, 0.3, 0.1994, 0.4338, 0.149, 0.3536, 0.2493,
    0.0975]

num_trials = [20, 20, 20, 20, 5, 5, 5, 5, 5]

time_squared = []

for value in time_averaged:

    time_squared.append(value ** 2)
```

To test whether or not the accelartion is independant on the distance traveled, we will plot a graph with Δx plotted on the x-axis (independant variable), and T^2 plotted on the y-axis (dependant variable). If the acceleration is constant, we expect to see linear trend, with the slope being the uniform acceleration. However, before we can plot, we must find the error in our measurements!

The error in any given averaged time measurment will simply be the standard error. To compute the standard error (α), we will exploit $\alpha = \frac{\sigma}{\sqrt{n}}$. So to obtain the standard error for an averaged time value, we will compute the standard deviation of the trial that obtained that average, and then divide that by $\sqrt{n}$.

```
standard_error_time = []



data_values1 = [

    2.998, 3.039, 3.025, 2.962, 2.98, 2.985, 3.01, 2.999, 3.13, 2.996,
        2.996, 3.01,

    3.098, 2.941, 3.033, 2.978, 2.971, 2.949, 2.997, 3.015, 2.516, 2.535,
        2.53, 2.5,

    2.515, 2.522, 2.447, 2.503, 2.543, 2.549, 2.55, 2.587, 2.532, 2.576,
        2.584, 2.57,

    2.588, 2.56, 2.545, 2.595, 2.108, 2.091, 2.069, 2.097, 2.094, 2.11,
        2.091, 2.099,
```

```
14
15      2.088, 2.067, 2.103, 2.111, 2.115, 2.163, 2.116, 2.109, 2.175, 2.096,
            2.126, 2.22,
16
17      1.812, 1.819, 1.796, 1.847, 1.811, 1.9, 1.826, 1.836, 1.848, 1.885,
            1.742, 1.853,
18
19      1.845, 1.833, 1.879, 1.866, 1.848, 1.877, 1.797, 1.86, 2.782, 2.764,
            2.761,
20
21      2.758, 2.802, 1.672, 1.617, 1.647, 1.646, 1.648, 2.371, 2.325, 2.3,
            2.296, 2.347,
22
23      1.969, 1.985, 1.955, 1.957, 1.961, 1.281, 1.331, 1.256, 1.234, 1.174
24
25  ]
26
27
28
29  data_values = np.array(data_values1)
30
31
32
33
34
35  trial_1 = data_values[:20]
36
37  trial_2 = data_values[20:40]
38
39  trial_3 = data_values[40:60]
40
41  trial_4 = data_values[60:80]
42
43  trial_5 = data_values[80:85]
44
45  trial_6 = data_values[85:90]
46
47  trial_7 = data_values[90:95]
48
49  trial_8 = data_values[95:100]
50
51  trial_9 = data_values[100:105]
52
53
54
55  for i in range(1, 10):
56
57      trial_key = f'trial_{i}'  # Construct the trial name
58
59      trial_data = locals()[trial_key]  # Access the trial list using locals()
60
61
62
63      # Calculate standard error and append to the list
64
65      standard_error_time.append((np.std(trial_data, ddof=1) / np.sqrt(len(
            trial_data))))
```

```
    #standard_error_time.append(.25 / np.sqrt(len(trial_data)))




print(f"our standard error values for each averaged time is: {
    standard_error_time}")
```

```
our standard error values for each averaged time is: [0.010138358633895013,
    0.00819283223311695, 0.008089759608685273, 0.008217119485049426,
    0.00828009661779379, 0.008723531395025744, 0.014189432687743404,
    0.005455272678794347, 0.025937231926325525]
```

Now that we have obtained our error values for each averaged time value, we must propograte this error to the T^2 values. To do this we will use the error propogration formula defined below:

$$\delta Z = |A^n \times n \times \frac{\delta A}{A}|$$

```
def PropPower(A,dA,n):

  Z = A**n

  return abs(n*Z*(dA/A))

standard_error_time_squared = PropPower(np.array(time_averaged), np.array(
    standard_error_time),2)

print(standard_error_time_squared)
```

```
[0.06095181 0.04165236 0.03417114 0.03022257 0.04592142 0.02871787
 0.066066   0.02143922 0.06510245]
```

Now we have obtained the standard error in our T^2 values! The way we obtained the error in our Δx values is as follows:
1) We guessed the last digit of our measurement (ie. the value inbetween the smallest gradation)
2) We set the uncertainty as ± 0.5 the smallest gradation
Therefore the uncertainty in Δx is 0.0005 meters.

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

```
4
5  plt.rcParams['font.family'] = 'serif'
6  plt.rcParams['text.usetex'] = True
7  plt.rcParams['font.size'] = 14  # Set overall font size to 14
8
9  # Fit the data
10 slope, intercept = np.polyfit(delta_x, time_squared, 1)
11
12 # Generate best-fit line data
13 x_best_fit = np.linspace(min(delta_x), max(delta_x), 100)
14 y_best_fit = intercept + x_best_fit * slope
15
16 # Calculate residuals
17 residuals = np.array(time_squared) - (intercept + np.array(delta_x) * slope)
18
19 # Create figure and subplots
20 fig, (ax1, ax2) = plt.subplots(2, 1, gridspec_kw={'height_ratios': [3, 1]},
       figsize=(8, 6))
21
22 # First plot: Best-fit line and data points
23 ax1.errorbar(delta_x, time_squared, xerr=0.0005, yerr=
       standard_error_time_squared,
24              fmt="x", ecolor="blue", label="Data with error bars", capsize
                    =5, color="red")
25 ax1.plot(x_best_fit, y_best_fit, label="Line of Best Fit", color="orange")
26 ax2.set_xlabel("Distance Traveled (m)")
27
28 ax1.set_ylabel("Change in Time Squared (s$^2$)")
29 ax1.set_title("Time Squared vs Distance Travelled")
30 ax1.legend()
31
32 # Second plot: Residuals
33 ax2.scatter(delta_x, residuals-0.04, color='red', label='Residuals')
34 ax2.axhline(0, color='gray', linestyle='--', linewidth=0.8)  # Add
       horizontal line at zero
35 ax2.set_ylabel("Residuals")
36
37 ax2.errorbar(delta_x, residuals-0.04, xerr=0.0005, yerr=
       standard_error_time_squared,
38   fmt="x", ecolor="blue", label="Data with error bars", capsize=5, color="
       red")
39 #ax2.legend(loc="upper left")
40
41 # Adjust spacing between subplots
42 plt.tight_layout()
43
44 # Show the plot
45 plt.show()
46
47 # Output the slope and intercept
48 print(f"The slope is: {slope:.5f} and the intercept is: {intercept:.4f}")
49
50
51
52 from scipy.stats import chi2
53 # Degrees of freedom
54 degrees_of_freedom = len(delta_x) - 2  # 2 parameters estimated (slope and
```

```
    intercept)

# Calculate chi-squared statistic
chi_squared = np.sum((residuals / (5* standard_error_time_squared)) ** 2)

# Calculate p-value
p_value = 1 - chi2.cdf(chi_squared, degrees_of_freedom)

print(f"Chi-squared statistic: {chi_squared:.3f}")
print(f"P-value: {p_value:.921f}")

# Interpretation:
if p_value > 0.05:
    print("The fit is considered good. No reason to reject the model.")
elif p_value < 0.05 and p_value > 1e-3:
    print("The fit is questionable. Further investigation might be needed.")
else:  # p_value < 1e-3
    print("The fit is considered poor. The model is likely incorrect.")
```

```
The slope is: 18.20886 and the intercept is: -0.4095
Chi-squared statistic: 44.439
P-value:
    0.000000175672338342813816325360676273703575134277343750000000000000000000000000000000000000000000000000000
The fit is considered poor. The model is likely incorrect.
```

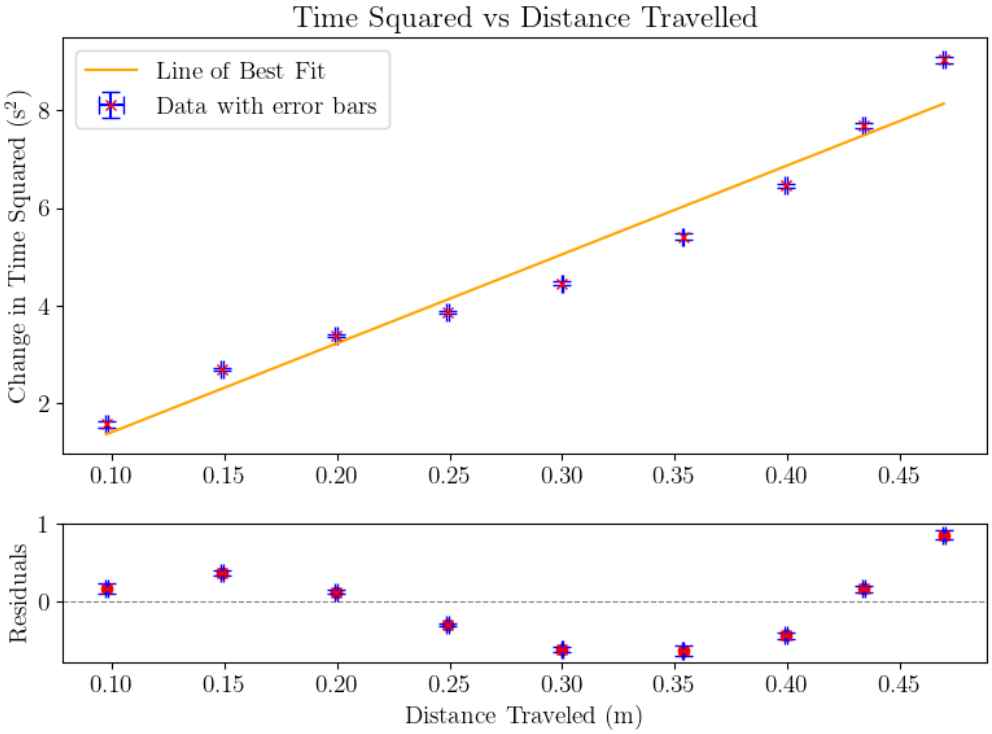

Figure 1:

Notice we seen a clear sinusodal trend in our residual plot. This indicates a systematic error which is reinforced by the absence of our error bars crossing the line of best fit (yes they are there, just very small). However, we have reviewed our error calculations and have come to the conclusion that they are correct. That leaves a question begging to be answered, namely, what is causing this sinusodal trend in our residual plot? We discussed a few possible reasons for this effect, but the one we are most confident is: a non-uniform density of the cylinder (ie. mass is not evenely distributed). This non-uniform distribution of mass would cause a constantly changing acceleration; this is due to the center of mass not being in the center of the cylinder. Consequently, the cylinder has intervals of its rotation where the excess mass is aiding in rotating (when the center of mass is moving downwards), and intervals of its rotation where the excess mass is opposing rotation (when the

center of mass is moving upwards). This would cause our line of best fit to be overestimating the time taken for a specific displacement to be traversed when the cylinder's center of mass spent more time opposing roation, and underestiming the time taken when the cylinder's center of mass spent more time aiding rotation. Now that we have convicned ourselves that this is a pluasible explanation, what evidence do we have that this may be the exact explanation? A strong indicatator of this explanation is that the period of the sine wave is 42.5 cm. This number may seem random, but since the radius of the cylinder is around 7.6cm (obtained from Mr. Ruffolo), the circumference is around 47.5cm. The circumference and the period being in agreement suggest that this trend repeats after one full revoltuion (which is predicted by our hypothesis). Unfortunately, since we were not expecting this sinusodal trend we did not take the measurements to provide further support of this hypothesis. Although, given more time/resources, to further investigate this hypothesis we could attempt to isolate which interval of rotation aids motion, and which opposes. We would then mark the cylinder accordingly, and then test if these intervals still aid/oppose motion when expected given different enviornments (ie. different ramp, angle of inclination, etc.). We could also test longer displacements to see if this trend continous indefintly. All in all, this is a very curious explanation, and one which needs a bit more testing before it can be verfified.