ProjectNotebook

June 7, 2022

1 Project Description

1.1 Quantifying Translation Rate of Mitochondrial mRNA

The goal of our project is to analyze raw data representing protein expression over time and infer translation elongation times of those genes of interest.

After DNA gets transcribed to mRNA in the cytoplasm, mRNA gets translated to protein. The translation elongation process has especially been found to be integral to mRNA localization to the mitochondria. Because the mitochondria is an important organelle for ATP production, our group wanted to calculate the rate of translation elongation of nuclear-encoded mitochondrial genes.

Our raw data contains measurements of luminescence from luciferase assays. This obtained by using an in-vivo elongation reporter containing luciferase to report the protein expression of the gene of interest (GOI). This reporter contains a tetracycline inducible promoter to govern transcription and translation of the GOI and luciferase. This data was imported in dataframe form. We know that protein expression is proportional to mRNA amounts and time, and mRNA amounts are proportional to DNA amounts and time. Knowing that DNA amounts are constant, nLuc expression can then be proportional to time. We can then take the square root of nLuc expression to produce a Schleif plot, which displays a nice linearization to further analyze the data (Schleif et al., 1973). After identifying the linear portion of the Schelif plot, we will produce a line of best fit to calculate the x-intercept, which represents the time it takes for the first protein to be produced. Then, we will take the difference between the x-intercepts of the control and the gene of interest in order to appreciate the elongation time of the gene of interest.

Works Cited:

Schleif, R., Hess, W., Finkelstein, S., & Ellis, D. (1973). Induction kinetics of the L-arabinose operon of Escherichia coli. Journal of bacteriology, 115(1), 9–14. https://doi.org/10.1128/jb.115.1.9-14.1973

Tatsuhisa Tsuboi, Matheus P Viana, Fan Xu, Jingwen Yu, Raghav Chanchani, Ximena G Arceo, Evelina Tutucci, Joonhyuk Choi, Yang S Chen, Robert H Singer, Susanne M Rafelski, Brian M Zid (2020) Mitochondrial volume fraction and translation duration impact mitochondrial mRNA localization and protein synthesis eLife 9:e57814. https://doi.org/10.7554/eLife.57814

Williams, C. C., Jan, C. H., & Weissman, J. S. (2014). Targeting and plasticity of mitochondrial proteins revealed by proximity-specific ribosome profiling. Science, 346(6210), 748-751.

1.1.1 Team Member Names and Contributions

Please specify who in your group worked on which parts of the project (1-2 sentences per team member).

- [redacted]): worked on dataframe manipulation, created the function of finding the linear regression line, defined the error handling for the negative value test, proposed on changing the functions into a more flexible way, visualized the original data and plotted the Schleif plot
- [redacted]provided background knowledge and raw data for analysis, created the function for calculating translation elongation rate, helped with initial calculation of Schlief plot data, checked for consistency of our program by comparing to original analysis, error handling for checking the initial raw data
- [redacted] created a colab file for easy communication and sharing of progress; wrote background information; added markdown cells throughout the notebook for organization and directionality of project

1.2 Project Code

If it makes sense for your project, you can have code and outputs here in the notebook as well.

-

```
[1]: # import packages needed for data manipulation and data visualization
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
```

```
[2]: # create the dataframe called lia_df from the csv file of luciferase induction␣
      ↪assays
     lia_df = pd.read_csv("luciferase induction assays.csv")
     lia_df
```

```
[2]:      Time [s]  CON (-ATC)  CON (+ATC)  HEM1 (-ATC)  HEM1 (+ATC)  COR1 (-ATC)  \
     0         0.0         339         361          212          222           90
     1        30.0         322         365          204          220           56
     2        60.0         329         338          223          238           72
     3        90.1         362         392          181          230           41
     4       120.0         386         404          209          247           48
     ..        …           …           …            …            …            …
     116    3480.4         679       40393          356        26195           76
     117    3510.7         650       40737          339        26573           59
     118    3540.4         662       41721          361        26957           86
     119    3570.5         676       42678          390        27226           59
     120    3600.5         703       42968          395        27512           75

          COR1 (+ATC)
     0             70
     1             63
```

```
2              71
3              56
4              63
..            …
116          4944
117          4987
118          5065
119          4964
120          5035

[121 rows x 7 columns]
```

- **Error Handling!** Before proceeding with data analysis, we need to ensure that the dataframe contains 121 rows. This indicates that the luciferase induction assay was done for 3600 seconds, or 1 hour.

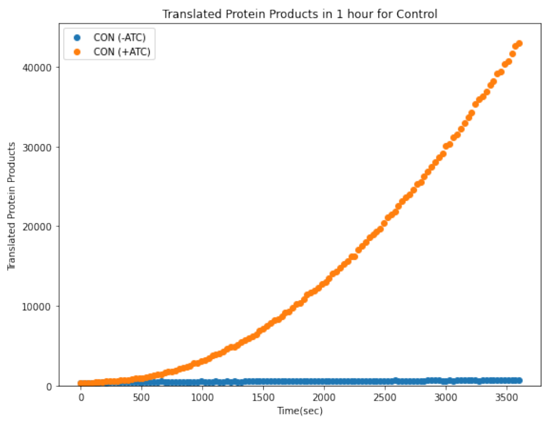
```
[3]: # test if the luciferase induction assay completes the recording for an hour
try:
    assert lia_df.shape[0] == 121, "Data set not complete."
except Exception as msg:
    print(msg)
```

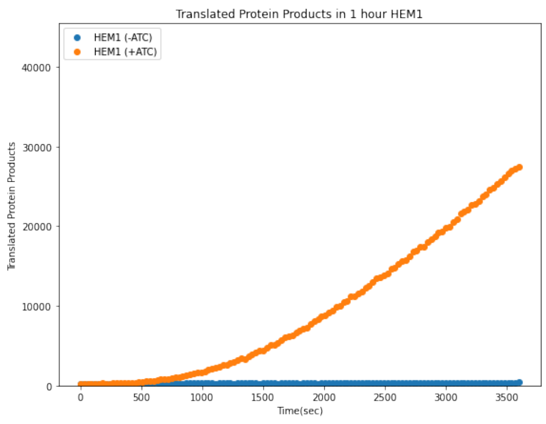
- **Data Visualization** Now, we will visualize translation rate of the control and experimental gene in the presence and absence of ATC. In the absence of ATC, also known as tetracycline, transcription is not induced. Therefore, we should expect to see very little expression in our plots. From this point, we will use HEM1 gene as a training template for establishing well-defined functions that can be used for determining the transcription elongation rate. However, the code should work for any gene in the dataset as long as it contains the required measurements.

```
[4]: def plot_translated_protein_products(gene_withoutATC, gene_withATC):
    '''
    Plot the control and any gene protein products overtime to visualize the␣
 ↪difference between two genes, taking the input of the name of columns of the␣
 ↪target gene in the dataframe.
    By default, the plot will produce 2 subplots, one of those is the control␣
 ↪gene, and the other is selected by user.

    :param gene_withoutATC: The name of the column for the target gene without␣
 ↪tetracycline
    :param gene_withATC: The name of the column for the target gene with␣
 ↪tetracycline
```

```
    '''

    # generate a figure with subplots
    fig, ax = plt.subplots(1,2,figsize=(20,7))

    # plot control(no ATC) and control(with ATC) on the first axis, [0]
    ax[0].scatter(lia_df.get('Time [s]'), lia_df.get('CON (-ATC)'))
    ax[0].scatter(lia_df.get('Time [s]'), lia_df.get('CON (+ATC)'))

    # plot control(no ATC) and control(with ATC) on the first axis, [1]
    ax[1].scatter(lia_df.get('Time [s]'), lia_df.get(gene_withoutATC))
    ax[1].scatter(lia_df.get('Time [s]'), lia_df.get(gene_withATC))

    # normalize the y range of the subplots to the largest number between the␣
↪translated protein products of selected gene and control, add 2500 to expand␣
↪the upper limit of y-axis
    ax[0].set_ylim(0, max(lia_df.get('CON (+ATC)').max(), lia_df.get('HEM1␣
↪(+ATC)').max()) + 2500)
    ax[1].set_ylim(0, max(lia_df.get('CON (+ATC)').max(), lia_df.get('HEM1␣
↪(+ATC)').max()) + 2500)

    # update axis parameters
    ax[0].set_ylabel('Translated Protein Products')
    ax[1].set_ylabel('Translated Protein Products')
    ax[0].set_xlabel('Time(sec)')
    ax[1].set_xlabel('Time(sec)')
    ax[0].set_title('Translated Protein Products in 1 hour for Control')
    ax[1].set_title('Translated Protein Products in 1 hour '+gene_withoutATC.
↪split(' ')[0])

    # display the legend of the plots
    ax[0].legend(['CON (-ATC)', 'CON (+ATC)'], loc="upper left")
    ax[1].legend([gene_withoutATC, gene_withATC], loc="upper left")

    plt.show()
```

[5]:
```
# plot the data for the control and target gene
plot_translated_protein_products('HEM1 (-ATC)', 'HEM1 (+ATC)')
```





•

Data Normalization After getting an idea of how our raw data looks like, we will normalize our data by doing the following calculation: (amount of protein produced in the presence of ATC) - (amount of protein produced in the absence of ATC). We will perform this calculation for both the control and experimental (HEM1) gene and create two new columns for these, called diff_CON and diff, respectively.

```
[6]: # add 2 columns containing the difference between the translated protein␣
     ↪products with and without tetracycline
     lia_df = lia_df.assign(diff_CON = lia_df.get('CON (+ATC)') - lia_df.get('CON␣
     ↪(-ATC)'),
                              diff_HEM1 = lia_df.get('HEM1 (+ATC)') - lia_df.get('HEM1␣
     ↪(-ATC)'))
     lia_df
```

```
[6]:     Time [s]  CON (-ATC)  CON (+ATC)  HEM1 (-ATC)  HEM1 (+ATC)  COR1 (-ATC)  \
     0         0.0         339         361          212          222           90
     1        30.0         322         365          204          220           56
     2        60.0         329         338          223          238           72
     3        90.1         362         392          181          230           41
     4       120.0         386         404          209          247           48
     ..        …           …           …            …            …            …
     116    3480.4         679       40393          356        26195           76
     117    3510.7         650       40737          339        26573           59
     118    3540.4         662       41721          361        26957           86
     119    3570.5         676       42678          390        27226           59
     120    3600.5         703       42968          395        27512           75

          COR1 (+ATC)  diff_CON  diff_HEM1
     0             70        22         10
     1             63        43         16
```

```
2               71         9         15
3               56        30         49
4               63        18         38
..             ...       ...        ...
116           4944     39714      25839
117           4987     40087      26234
118           5065     41059      26596
119           4964     42002      26836
120           5035     42265      27117

[121 rows x 9 columns]
```

-

Take the square root of difference As previously mentioned, we need to take the square root of protein expression in order to create a Schlief plot. Schleif plots are useful because it linearizes data, from which we can infer the x-intercept. We will add two additional columns called sqrt_CON and sqrt_HEM1 to our dataframe.

[7]:
```
# add 2 columns containing the square root of the difference
lia_df = lia_df.assign(sqrt_CON = np.sqrt(lia_df.get('diff_CON')),
                       sqrt_HEM1 = np.sqrt(lia_df.get('diff_HEM1')))
lia_df
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:397:
RuntimeWarning: invalid value encountered in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

[7]:
```
     Time [s]  CON (-ATC)  CON (+ATC)  HEM1 (-ATC)  HEM1 (+ATC)  COR1 (-ATC)  \
0         0.0         339         361          212          222           90
1        30.0         322         365          204          220           56
2        60.0         329         338          223          238           72
3        90.1         362         392          181          230           41
4       120.0         386         404          209          247           48
..        ...         ...         ...          ...          ...          ...
116    3480.4         679       40393          356        26195           76
117    3510.7         650       40737          339        26573           59
118    3540.4         662       41721          361        26957           86
119    3570.5         676       42678          390        27226           59
120    3600.5         703       42968          395        27512           75

     COR1 (+ATC)  diff_CON  diff_HEM1  sqrt_CON  sqrt_HEM1
0             70        22         10  4.690416   3.162278
1             63        43         16  6.557439   4.000000
2             71         9         15  3.000000   3.872983
3             56        30         49  5.477226   7.000000
4             63        18         38  4.242641   6.164414
```

```
..             …       …         …          …           …
116         4944   39714     25839  199.283717  160.745140
117         4987   40087     26234  200.217382  161.969133
118         5065   41059     26596  202.630205  163.082801
119         4964   42002     26836  204.943895  163.816971
120         5035   42265     27117  205.584532  164.672402

[121 rows x 11 columns]
```

•

[8]:
```
# user defined range for performing linear regression
lower_bound=input('What is the lower bound of your selection?')
upper_bound=input('What is the upper bound of your selection?')
```

```
What is the lower bound of your selection?1140
What is the upper bound of your selection?1800
```

[9]:
```
# create a subset of the dataframe containing only the data points within the␣
↪given interval
interval = lia_df[(lia_df.get('Time [s]')>=float(lower_bound))&(lia_df.
↪get('Time [s]')<=float(upper_bound))]
interval
```

[9]:
```
    Time [s]  CON (-ATC)  CON (+ATC)  HEM1 (-ATC)  HEM1 (+ATC)  COR1 (-ATC)  \
38    1140.4         483        4063          280         2371           55
39    1170.4         502        4272          293         2597           40
40    1200.4         525        4606          282         2620           48
41    1230.4         489        4837          284         2806           43
42    1260.5         522        4829          273         3006           47
43    1290.4         505        5148          292         3260           53
44    1320.2         503        5530          295         3383           33
45    1350.2         566        5684          291         3306           42
46    1380.2         558        5979          295         3726           57
47    1410.2         549        6235          294         3937           51
48    1440.2         541        6400          280         4146           56
49    1470.2         526        6853          260         4411           38
50    1500.2         536        7104          285         4432           49
51    1530.2         536        7477          310         4814           43
52    1560.2         530        7828          283         5122           38
53    1590.2         542        8216          293         5101           40
54    1620.2         524        8308          274         5350           40
55    1650.3         534        8723          312         5701           62
56    1680.2         546        9190          279         6071           41
57    1710.5         580        9309          288         6213           45
```

```
58    1740.5          517        9752          323          6364            54
59    1770.4          556       10205          299          6718            51

    COR1 (+ATC)  diff_CON  diff_HEM1   sqrt_CON  sqrt_HEM1
38          234      3580       2091  59.833101  45.727453
39          275      3770       2304  61.400326  48.000000
40          278      4081       2338  63.882705  48.352870
41          312      4348       2522  65.939366  50.219518
42          369      4307       2733  65.627738  52.278102
43          394      4643       2968  68.139563  54.479354
44          389      5027       3088  70.901340  55.569776
45          433      5118       3015  71.540198  54.909016
46          449      5421       3431  73.627441  58.574739
47          436      5686       3643  75.405570  60.357270
48          501      5859       3866  76.544105  62.177166
49          529      6327       4151  79.542442  64.428255
50          496      6568       4147  81.043198  64.397205
51          576      6941       4504  83.312664  67.111847
52          604      7298       4839  85.428333  69.562921
53          635      7674       4808  87.601370  69.339743
54          689      7784       5076  88.226980  71.246053
55          732      8189       5389  90.493094  73.409809
56          739      8644       5792  92.973114  76.105190
57          727      8729       5925  93.429118  76.974022
58          791      9235       6041  96.098907  77.723870
59          905      9649       6419  98.229324  80.118662
```

-

Error Handling! It wouldn't make sense to take the square root of a negative value. Therefore, negative_test and sqrt_validity_test check whether there is a negative value in our difference columns. If there is a negative value, it will give us an error message.

[10]:
```
def negative_test(lower_bound, upper_bound, diff_gene):
    '''
    This function takes the lower_bound and upper_bound of the user\'s␣
↪selection and test if there are negative values in the difference column for␣
↪the given gene.
    This function should report an error message if there is a negative value␣
↪in the selected column, otherwise, should return nothing.

    :param lower_bound: The lower bound string that is entered by user
    :param upper_bound: The upper bound string that is entered by user
    :param diff_gene: The name of the column in lia_df, which contains the␣
↪difference between the translated protein products with and without␣
↪tetracycline for the target gene
```

```
    '''

    # create a subset of the dataframe containing only the data points within␣
↪the given interval
    interval = lia_df[(lia_df.get('Time [s]')>=float(lower_bound))&(lia_df.
↪get('Time [s]')<=float(upper_bound))]

    # check every data points within the difference column, raise an error if␣
↪there is a negative value
    for i in interval.get('diff_HEM1'):
        if i<=0:
            raise ValueError('There are negative values in the difference, try␣
↪a different time interval.')
```

```
[11]: def sqrt_validity_test(lower_bound, upper_bound, diff_gene):
    '''
    This function takes the lower_bound and upper_bound of the user\'s␣
↪selection and test if there are negative values in the difference column for␣
↪the given gene.
    This function should report an error message if the try block fails,␣
↪otherwise, should return nothing.

    :param lower_bound: The lower bound string that is entered by user
    :param upper_bound: The upper bound string that is entered by user
    :param diff_gene: The name of the column in lia_df, which contains the␣
↪difference between the translated protein products with and without␣
↪tetracycline for the target gene
    '''

    # perform the negative_test with the given parameter to see if there is any␣
↪failed value in the difference column, if the negative_test fails, the code␣
↪should report an error message
    try:
        negative_test(lower_bound, upper_bound, diff_gene)
    except ValueError:
        print('There are negative values in the difference, try a different␣
↪time interval.')
        raise
```

```
[12]: # test if the selected interval is valid for linear regression
sqrt_validity_test(lower_bound, upper_bound, 'diff_HEM1')
```

There is a negative value in the 'diff_HEM1' column in the first 200 seconds. We will use our test function to test the validity of the function.

```
[25]: sqrt_validity_test(0, 200, 'diff_HEM1')
```

```
There are negative values in the difference, try a different time interval.
```

```
        ␣
 ↪---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call␣
 ↪last)

        /tmp/ipykernel_1251/1074555221.py in <module>
    ----> 1 sqrt_validity_test(0, 200, 'diff_HEM1')


        /tmp/ipykernel_1251/3541824904.py in sqrt_validity_test(lower_bound,␣
 ↪upper_bound, diff_gene)
         11     # perform the negative_test with the given parameter to see if␣
 ↪there is any failed value in the difference column, if the negative_test␣
 ↪fails, the code should report an error message
         12     try:
    ---> 13         negative_test(lower_bound, upper_bound, diff_gene)
         14     except ValueError:
         15         print('There are negative values in the difference, try a␣
 ↪different time interval.')


        /tmp/ipykernel_1251/144357414.py in negative_test(lower_bound,␣
 ↪upper_bound, diff_gene)
         15     for i in interval.get('diff_HEM1'):
         16         if i<=0:
    ---> 17             raise ValueError('There are negative values in the␣
 ↪difference, try a different time interval.')


        ValueError: There are negative values in the difference, try a different␣
 ↪time interval.
```

- **Find the equation of the line of best fit** Our function linear_regression will give us the equation of the line of best fit. We will then input sqrt_CON and sqrt_HEM1 to determine the equation for the control and experimental (HEM1) gene, respectively.

[14]:
```
def standard_unit(sqrt):
    '''
    This function converts the square root of a series/array into its standard␣
 ↪units
```

```
    :param sqrt: The name of the column in interval dataframe, which contains␣
↪the square root of the difference between the translated protein products␣
↪with and without tetracycline for the target gene
    '''

    mean = np.mean(sqrt) # calculate the mean of the square root of given gene
    std = np.std(sqrt) # calculare the standard deviation of the square root of␣
↪given gene
    return (sqrt-mean)/std # standard unit = (x-mean)/standard deviation
```

[15]:
```
def linear_regression(sqrt):
    '''
    This function finds the best fit linear regression line based on given␣
↪interval and the target gene.
    This function should return 3 parameters, including the slope of the␣
↪best-fit line, the intercept of the best-fit line, and a string containing␣
↪the equation in standard form.

    :param sqrt: The name of the column which contains the square root info for␣
↪the target gene
    '''

    # find the best-fit linear regression line
    r = np.mean(standard_unit(interval.get(sqrt))*standard_unit(interval.
↪get('Time [s]'))) # calculate the correlation coefficient
    slope = r*np.std(interval.get(sqrt))/np.std(interval.get('Time [s]')) #␣
↪calculate the slope based on the correlation coefficient
    y_intercept = np.mean(interval.get(sqrt))-slope*np.mean(interval.get('Time␣
↪[s]')) # calculate the y-intercept based on the slope and the mean of␣
↪independent and dependent variable
    equation = 'Linear Regression Line Equation: ' + 'y = '+str(y_intercept)+'␣
↪+ '+str(slope)+' * x'
    return slope, y_intercept, equation
```

[16]:
```
# calculate the linear regression for the control
linear_regression('sqrt_CON')
```

[16]:
```
(0.06086408218934878,
 -9.974589922964782,
 'Linear Regression Line Equation: y = -9.974589922964782 + 0.06086408218934878
* x')
```

[17]:
```
# calculate the linear regression for the target gene
linear_regression('sqrt_HEM1')
```

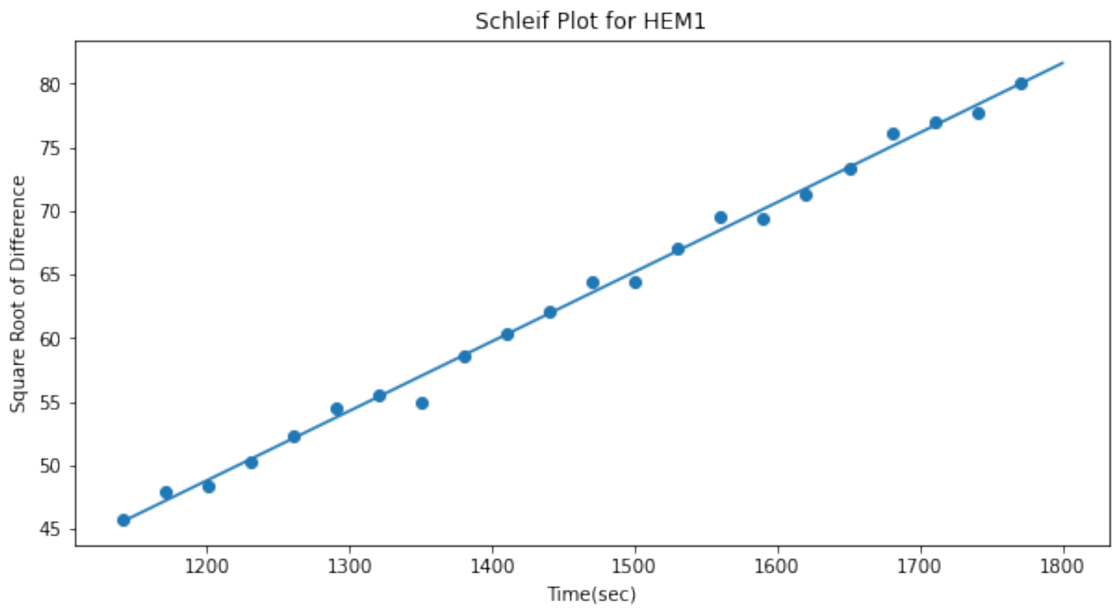
```
[17]: (0.054672718878843059,
       -16.789624151693353,
       'Linear Regression Line Equation: y = -16.789624151693353 + 0.054672718878843059
      * x')
```

-

Create Schleif Plot Now, we can create a Schleif plot for control and target gene!

```
[18]: def Schleif_Plot(lower_bound, upper_bound, sqrt):
          '''
          This function plots the Schleif_Plot and the scatter plot of the original␣
       ↪data between the interval.
          This function should produce a graph, otherwise, no other output.

          :param lower_bound: The lower bound string that is entered by user
          :param upper_bound: The upper bound string that is entered by user
          :param sqrt: The name of the column which contains the square root info for␣
       ↪the target gene
          '''

          # setup of the Schleif Plot
          x = np.linspace(float(lower_bound), float(upper_bound)) # set the range on␣
       ↪the x-axis which should be the range between the lower_bound and upper_bound
          y = linear_regression(sqrt)[1] + linear_regression(sqrt)[0]*x # use the␣
       ↪output of linear_regression(sqrt) function to draft the equation
          fig = plt.figure(figsize = (10, 5))

          # plot the graphs
          plt.plot(x, y) # plot the Schleif Plot
          plt.scatter(interval.get('Time [s]'), interval.get(sqrt)) # plot the␣
       ↪scatter plot

          # update axis parameters
          plt.title('Schleif Plot for '+ sqrt.split('_')[1])
          plt.ylabel('Square Root of Difference')
          plt.xlabel('Time(sec)')

          plt.show()
```

```
[19]: # plot the Schleif Plot and scatter plot in a single graph
      Schleif_Plot(lower_bound, upper_bound, 'sqrt_HEM1')
```



- **Determine x-intercept** As our ultimate goal is to determine elongation rate, we need to calculate time. The x-intercept of a Schelif plot represents the time it takes for the first protein to be produced.

[20]:
```
def x_intercept(lower_bound, upper_bound, sqrt):
    '''
    This function calculates the x-intercept based on the linear regression␣
↪line.

    :param lower_bound: The lower bound string that is entered by user
    :param upper_bound: The upper bound string that is entered by user
    :param sqrt: The name of the column which contains the square root info for␣
↪the target gene
    '''

    equation = linear_regression(sqrt)# find the linear equation
    x_intercept = -equation[1] / equation[0] # calcualte the x-intercept by␣
↪setting y=0
    return x_intercept
```

[21]:
```
# calculate the x_intercept for control
x_intercept(lower_bound, upper_bound, 'sqrt_CON')
```

[21]:
```
163.88302532737995
```

[22]:
```
# calculate the x_intercept for target gene
x_intercept(lower_bound, upper_bound, 'sqrt_HEM1')
```

[22]:
```
307.0932724056856
```

- **Calculate Translation Elongation Rate** Finally, we are ready to calculate the elongation rate of our control and target gene! Translation elongation rate is determined by dividing the number of proteins (peptide_num) by the translation elongation time (translation_elongation_time).

- **Error Handling!** After calculating the translation elongation rate, we also wanted to check that the rate was within a reasonable range. If the rate is outside of this range, it will raise an error.

[23]:
```
def elongation_rate(mrna, sqrt, lower_bound, upper_bound):
    '''
    This function calculates the elongation rate for the target gene.

    :param mrna: The length of the mrna for the target gene in interger
    :param sqrt: The name of the column which contains the square root info for␣
 ↪the target gene
    :param lower_bound: The lower bound string that is entered by user
    :param upper_bound: The upper bound string that is entered by user

    '''

    peptide_num = mrna/3 # determining polypeptide length
    transcription_elongation_time = mrna/25 # transcription elongation time in␣
 ↪seconds calculated from knowing transcription occurs at 25 amino acids/sec
    elongation_time = x_intercept(lower_bound, upper_bound,␣
 ↪sqrt)-x_intercept(lower_bound, upper_bound, 'sqrt_CON') # determining the␣
 ↪total elongation time of the gene of interest
    translation_elongation_time = elongation_time-transcription_elongation_time␣
 ↪# subtracts out the transcription elongation time from the total elongation␣
 ↪time to isolate the translation elongation time
    translation_elongation_rate = peptide_num/translation_elongation_time

    # test if the elongation rate is in a reasonable range
    if not translation_elongation_rate>0 and translation_elongation_rate<=30:
        raise ValueError('The translational elongation rate isn\'t within a␣
 ↪reasonable range, try a new time interval.')

    return translation_elongation_rate
```

```
[24]: # find the elongation rate for the target gene
      elongation_rate(1644, 'sqrt_HEM1', lower_bound, upper_bound)

[24]: 7.075510029631639
```

1.3 Reflection

[redacted]

At the beginning of this course, I had absolutely no knowledge about coding. My professor in the lab I work in wanted me to dabble in some bioinformatics, but it was really difficult due to my lack of knowledge. He actually heard about this new course and told me about it so I thought it would be a good opportunity to learn. This project definitely challanged us as we had to put together a lot of things we learned from class and make a program from scratch. We were able to incorporate concepts of dataframes, NumPy, plotting/visualization, functions, etc. I currently do a lot of research and experiments on my own, so I thought it would be cool to transfer some of the data analysis I do on Excel into a program. We used the data analysis I do by manually on Excel as a guide to help us create a program. Finishing this project really showed a lot about what we're able to code and figure out on our own, especially when wanting to use functions that we aren't familiar with. I think it's also fun that we got to apply some of the research that is happening here at UCSD.

[redacted]

BILD62 is not the very first coding class in my life. I started to learn a little bit coding last quarter in ECON5, which introduces R and STATA in social science. I was originally taking that class for an econ minor, but ending up finding that coding is quite interesting in general. I am working in a research lab and we work on the RNA splicing factor called SRSF3 and non-alcoholic steatohepatitis (NASH). As I start to work on the project, I found that sometimes I need to analyze the RNA-seq and DNA sequencing data, which motivates me to learn a little bit coding. BILD62 is my first python class (although I am taking DSC10 at the same time) and I learned a lot in manipulating dataset and use python to create customized tools to carry out functions needed for analysis. Comparing to my previous experience with R, I do prefer to use python to perform data analysis because it has a more comprehensive interface and well-developed packages to facilitate the process of analysis. When our group is working on the final project, I think there are multiple things that worth to mention here. First, I really appreciate the luciferase induction assay dataset that was provided by Vicky. Our group had a hard time in determine what topic we should do in the beginning. It is also a challenge for me to understand what is actually going on with the data and the purpose of performing the experiment.Vicky helped me a lot in getting familized with the background of the research topic and how the luciferase induction assay is performed.As we started to actually work on the project, the most challenging portion of this project is the error handling because it is hard to predict what kinds of error that can be encountered by users, in the perspective of a coder, when they are running the code. The biggest challenge that I met when I was coding is that I kept on failing to raise an error message using the try/except function. Therefore, I googled about how the try/except function block should be coded. But the tutorials were not that helpful in genneral. So I went to python's documentation to browse the logic behind the try/except function. It was quite informative, so I fixed the problem of having no error report when there is actually an error. Overall, I believe my coding ability is improved and I learned a lot of new stuffs when I was doing this final project. If there is any possibility, I would like to learn more about how to use python in data analysis in the future.

[redacted]

As many others, I had zero knowledge of Python or coding in general before I started this class. As a neurobiology major, I was mainly involved in my biology classes and wanted to step out of my comfort zone to explore something new. I feel that I learned a great deal from this class as I am now familiar with the coding language and can at least have a vague idea of what's going on when I look at a completely new script. Personally, the final project was definitely a great learning opportunity as we decided to work on existing data from one of the members' lab. Before getting started with coding, I had to familiarize myself with background information of the lab's study. Then, I also found the coding process to be difficult as I lacked the guidance we often had in assignments. For this reason, I frankly wasn't able to contribute much to the coding process but I tried my best to understand my teammate's code and provide supplemental help to make our notebook easier to navigate by adding markdown cells and comments. Overall, I am really glad that I took this class as I was able to get a taste of coding and see how it could be applicable to biology!

```
[ ]:
```