آزمایش سوم، ثابت هابل:

بردیا عالیان، بردیا حسن پور
گروه سوم

۱. تئوری:

میخواهیم خطی به معادله خط $\mathrm{y} = \mathrm{mx} + \mathrm{b}$ را به داده های وزن دارمان برازش کنیم، ابتدا s را طوری تعریف میکنیم که اختلاف مقدار پیش بینی شده و مقدار واقعی را با احتساب وزن آن بدهد:

$$s_i = w_i(y_i - mx_i - b)$$

برای برازش بهترین خط باید جمع s ها کمینه شود:

$$\begin{cases} \dfrac{\partial S}{\partial m} = \dfrac{\partial}{\partial m}\sum_i s_i = 0 \\ \dfrac{\partial S}{\partial b} = \dfrac{\partial}{\partial b}\sum_i s_i = 0 \end{cases}$$

که جواب نهایی این دو معادله به روابط زیر می‌رسد:

$$\begin{cases} m = \dfrac{Cov(x,y)_w}{Var(x)_w} \\ b = \overline{y}_w - m\overline{x}_w \end{cases}$$

که تمامی روابط وایانس و کواریانس و میانگین ها با احتساب وزن می‌باشند.

برای برازش با معادله $y = mx$ نیز می‌توان به همین روش به جواب زیر رسید:

$$m = \frac{\sum_i w_i x_i y_i}{\sum_i w_i x_i^2}$$

حال برای حساب کردن وزن با استفاده از خطا برای هر داده می‌توان از رابطه نشر خطا کمک گرفت:

$$\delta_{s_i}^2 = \left(\frac{\partial s_i}{\partial y_i}\delta_{y_i}\right)^2 + \left(\frac{\partial s_i}{\partial x_i}\delta_{x_i}\right)^2 = \delta_{y_i}^2 + m^2\delta_{x_i}^2$$

با فرض اینکه وزن داده ها برابر معکوس مربع خطای آنهاست داریم:

$$w_i = \frac{1}{\delta_{y_i}^2 + m^2\delta_{x_i}^2}$$

در نهایت عدد رگرسیون به صورت زیر بدست می‌آید:

$$R^2 = 1 - \frac{\sum_i w_i s_i^2}{\sum_i w_i \left(y_i - \overline{y}_w\right)^2}$$

۲. پردازش داده ها:

۲.۱. cepheid_regression.py:

۲.۱.۱. تبدیل قدر ظاهری به مطلق:

```
def app_to_abs_table(app_mag, period, app_mag_err):
    distance = 2400000 / 3.26
    abs_mag = [round(i + 5 - 5  * np.log10(distance), ndigits = 2) for i in app_mag]
    abs_mag_error =  [i for i in app_mag_err]
    data = [['Mv', 'Period', 'Mv Error']]
    for i in range(len(app_mag)):
        data.append([abs_mag[i], period[i], abs_mag_error[i]])

    print(tabulate(data, headers='firstrow', tablefmt='fancy_grid'))

    return abs_mag, abs_mag_error

abs_mag, abs_mag_err = app_to_abs_table(app_mag, period, app_mag_err)
```

با استفاده از رابطه $m - M = 5\log(d) - 5$ قدر مطلق را بدست می‌آوریم. برای خطا روی این رابطه نشر خطا می‌زنیم که به رابطه $\delta_M = \delta_m$ می‌رسیم.

۲.۱.۲. برازش خط با خطا بروی y:

وقتی تنها خطا بروی y باشد خطاهای x را می‌توانیم صفر در نظر بگیریم:

$$\delta_{s_i} = \delta_{y_i} \Longrightarrow w = \frac{1}{\delta_{y_i}^2}$$

در برازش در زمانی که تنها خطا داده های y وجود دارند به مشکل حضور شیب خط در محاسبه وزن نمی‌خوریم و می‌توان به راحتی شیب و عرض از مبدا و خطا ها را به صورت زیر حساب کرد:

```
def reg_y_error(y, x, y_err):

    y = np.array(y)
    x = np.array(x)
    y_err = np.array(y_err)

    w_y = 1 / y_err ** 2

    x_wmean = np.average(x, weights=w_y)
    y_wmean = np.average(y, weights=w_y)

    covariance_w = np.average((x - x_wmean) * (y - y_wmean), weights=w_y)
    variance_w = np.average((x - x_wmean)**2, weights=w_y)

    slope = covariance_w / variance_w
    intercept = y_wmean - slope * x_wmean

    y_pred = intercept + slope * x
    residuals = y - y_pred

    N = len(x)
    slope_error = np.sqrt(np.sum(w_y * residuals**2) / ((N - 2) * np.sum(w_y * (x - x_wmean)**2)))
    intercept_error = np.sqrt(np.sum(w_y * residuals**2) / (N - 2)) * np.sqrt((1/N) + (x_wmean**2) / np.sum(w_y * (x - x_wmean)**2))

    R_2 = 1 - np.sum(w_y * residuals ** 2) / np.sum(w_y * (y - y_wmean) ** 2)

    return slope, intercept, R_2, slope_error, intercept_error
```

۲.۲. hubble_regression.py:

۲.۲.۱. پیدا کردن فاصله:

در برنامه ابتدا قدر مطلق را از روی دوره تناوب با توجه به رابطه بدست آمده برای قیفاووسی ها بدست می‌آوریم و با رابطه نشر خطا خطای قدر مطلق را بروی خطای ثوابت رابطه به صورت زیر بدست می‌آوریم:

$$\delta_M = \sqrt{\left(\log\left(P(day)\right)\delta_m\right)^2 + \delta_b^2}$$

در نهایت برای فاصله دوباره از رابطه بین قدر ظاهری و مطلق استفاده می‌کنیم. دوباره برای خطا از نشر خطا استفاده می‌کنیم و به رابطه زیر می‌رسیم:

$$\delta_{distance} = \frac{\ln(10)}{5} \times distance \times \sqrt{\delta_m^2 + \delta_M^2}$$

```python
def distance_calculator(app_mag, period, err_app_mag):
    # M_v = alog(p) + b
    a = -3.81
    err_a = 0.00403
    b = 1.47
    err_b = 0.00752

    M_v = [a * np.log10(i) + b for i in period]
    err_M_v = [np.sqrt((err_a * np.log10(i)) ** 2 + (err_b) ** 2) for i in period]

    # calculating distance in Mpc
    distance = [10 ** ((m - M) / 5 - 5) for m, M in zip(app_mag, M_v)]
    err_distance = [np.log(10) * d / 5 * np.sqrt((err_m) ** 2 + (err_M) ** 2) for d, err_m, err_M in zip(distance, err_app_mag, err_M_v)]

    return distance, err_distance

```

۲.۲.۲. برازش خط استاندارد:

در این بخش وزن به صورت زیر می‌باشد:

$$w_i = \frac{1}{\delta_{y_i}^2 + m^2\delta_{x_i}^2}$$

مشکلی که به آن در این بخش می‌خوریم اینست که برای بدست آوردن شیب به شیب نیاز داریم. برای اینکه این مشکل را برطرف کنیم از الگوریتم try and error استفاده می‌کنیم. ابتدا یک مقدار حدودی از شیب داخل وزن می‌گذاریم و سپس شیب جدید را در می‌اوریم و بجای شیب قبلی در وزن آن را جاگذاری می‌کنیم و تا دقت دلبخواب این کار را انجام می‌دهیم. بقیه پارامترها به راحتی بدست می‌آیند.

```python
def reg_xy_error(y, x, y_err, x_err):

    y = np.array(y)
    x = np.array(x)
    y_err = np.array(y_err)
    x_err = np.array(x_err)
    slope = 74
    tolerance = 0.001
    max_iterations = 10000000
    iteration = 0
    while True:
        combined_err = np.sqrt(y_err**2 + (slope * x_err)**2)
        w = 1 / combined_err**2

        x_wmean = np.average(x, weights=w)
        y_wmean = np.average(y, weights=w)

        covariance_w = np.average((x - x_wmean) * (y - y_wmean), weights=w)
        variance_w = np.average((x - x_wmean)**2, weights=w)

        slope_new = covariance_w / variance_w

        if np.abs(slope_new - slope) < tolerance or iteration > max_iterations:
            slope = slope_new
            break

        slope = slope_new
        iteration += 1

    intercept = y_wmean - slope * x_wmean

    y_pred = intercept + slope * x
    residuals = y - y_pred

    N = len(x)
    slope_error = np.sqrt(np.sum(w * residuals**2) / ((N - 2) * np.sum(w * (x - x_wmean)**2)))
    intercept_error = np.sqrt(np.sum(w * residuals**2) / (N - 2)) * np.sqrt((1/N) + (x_wmean**2) / np.sum(w * (x - x_wmean)**2))

    R_2 = 1 - np.sum(w * residuals ** 2) / np.sum(w * (y - y_wmean) ** 2)

    return slope, intercept, R_2, slope_error, intercept_error
```

۲.۲.۳. برازش مبدا گذر:

در اینجا نیز به مشکل بخش قبل می‌خوریم و همانند بخش قبل عمل می‌کنیم.

```python
def reg_xy_error_origin(y, x, y_err, x_err):
    y = np.array(y)
    x = np.array(x)
    y_err = np.array(y_err)
    x_err = np.array(x_err)
    slope = 74
    tolerance = 0.001
    max_iterations = 10000000
    iteration = 0
    while True:
        combined_err = np.sqrt(y_err**2 + (slope * x_err)**2)
        w = 1 / combined_err**2
        x_wmean = np.average(x, weights=w)
        y_wmean = np.average(y, weights=w)
        slope_new = np.sum(w * x * y) / np.sum(w * x**2)

        if np.abs(slope_new - slope) < tolerance or iteration > max_iterations:
            slope = slope_new
            break

        slope = slope_new
        iteration += 1

    y_pred = slope * x
    residuals = y - y_pred

    N = len(x)
    slope_error = np.sqrt(np.sum(w * residuals**2) / ((N - 1) * np.sum(w * (x - x_wmean)**2)))

    R_2 = 1 - np.sum(w * residuals ** 2) / np.sum(w * (y - y_wmean) ** 2)

    return slope, R_2, slope_error
```

۲.۲.۴. برازش استاندارد یا مبدا گذر؟:

به طور شهودی می‌توان گفت که به دلیل اینکه برازش استاندارد یک قید از برازش مبدا گذر کمتر دارد می‌تواند برازشی نزدیک تر به داده ها باشد. اما برای اینکه به طور دقیق این موضوع را بفهمیم هر دو برازش را انجام می‌دهیم و در نهایت برای داده ها از داده ای که رگرسیون بهتری دارد استفاده می‌کنیم.

```python
distance, err_distance = distance_calculator(app_mag, period, err_app_mag)

slope_1, intercept_1, R_2_1, slope_error_1, intercept_error_1 = reg_xy_error(v, distance, err_v, err_distance)
slope_2 , R_2_2 , slope_error_2 = reg_xy_error_origin(v, distance, err_v, err_distance)

print(f"""For not through origin:
Slope : {slope_1}
intercept : {intercept_1}
R_2 : {R_2_1}
slope_error : {slope_error_1}
intercept_error : {intercept_error_1}""")

print(f"""For through origin:
Slope : {slope_2}
R_2 : {R_2_2}
slope_error : {slope_error_2}""")

if R_2_1 > R_2_2:
    line_of_best_fit = slope_1 * np.array(distance) + intercept_1
    print("Not through origin is better")

else:
    line_of_best_fit = slope_2 * np.array(distance)
    print("Through origin is better")
```

۳. خواسته ها:

۳.۱.

با توجه به برنامه جدول قدر مطلق بر حسب دوره تناوب به صورت زیر می‌باشد:

Mv	Period	Mv Error
-2.79	13.15	0.12
-3.88	25.4	0.17
-3.08	15.62	0.13
-3.98	26.98	0.18
-1.96	7.97	0.1
-2.98	14.78	0.13
-2.24	9.41	0.11
-3.97	26.79	0.18
-3.35	18.39	0.14
-2.16	8.96	0.11
-4.16	30.08	0.19
-2.03	8.27	0.11
-3.99	27.14	0.18
-4	27.32	0.18
-3.15	16.29	0.14
-1.79	7.17	0.16

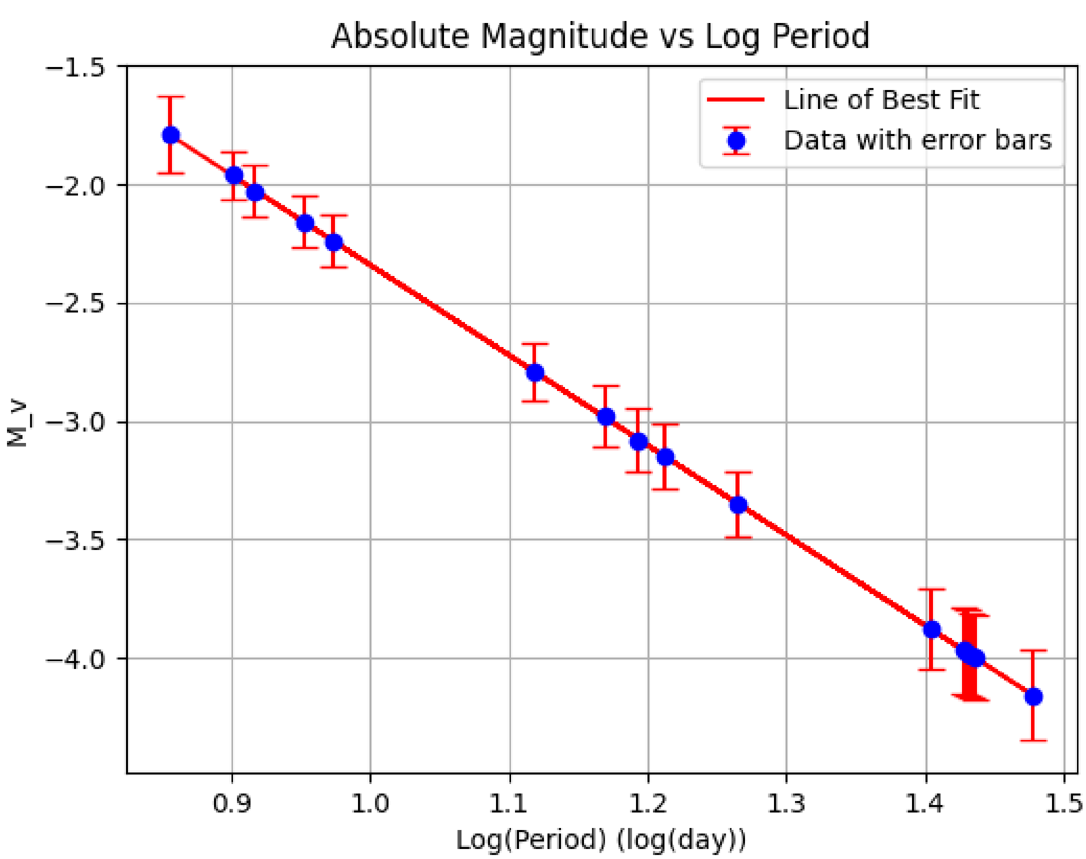
و داده های برازش به صورت زیر می‌باشد:

```
Slope : -3.806816989165384
intercept : 1.4670305811109166
R_2 : 0.9999842737689981
slope_error : 0.004034722974539847
intercept_error : 0.007517004304026486
```

در نتیجه رابطه قدر دوره تناوب به صورت زیر است:

$$M_v = -(3.81 \pm 0.01)\log\big(P(day)\big) + (1.47 \pm 0.01)$$

نمودار این داده ها نیز به صورت زیر است:



۲.۳.

```
For not through origin:
Slope : 64.12315934091276
intercept : -34.21336372440754
R_2 : 0.9943602715265559
slope_error : 1.7073679856720916
intercept_error : 13.384565364272902
For through origin:
Slope : 59.90011244950477
R_2 : 0.9898309890065261
slope_error : 2.1621272120684556
Not through origin is better
```

داده های هر دو نوع برازش به صورت بالا است. همانطور که از قبل با شهود گفتیم برازش استاندارد دقت بیشتری دارد. با توجه به این داده ها یک خطای سیستماتیک در سرعت وجود دارد که سرعت هایمان را به اندازه $-34.2 \pm 13.4 \frac{km}{s}$ شیفت داده.

مقدار ثابت هابل از داده های داده شده به اندازه زیر می‌باشد:

$$H_0 = 64.1 \pm 1.7 \frac{km}{s. Mpc}$$

نمودار سرعت بر حسب فاصله به صورت زیر می‌باشد: