Use of beam loss monitor to compute lifetime at SLS

Yacine El Yamani

August 2023

Contents

1 Introduction

In this document I will explain how to use beam loss monitors (BLM) and how to compute lifetime with their values. To do this I will use Fast beam loss monitors (Libera BLM) and slow beam loss monitors (CMOS BLM). I will also explain how to compute Touschek lifetime during user operation at SLS

2 Libera BLM

2.1 Working modes of Libera BLM

The first sensors we will discuss are the Libera BLM. The Libera BLM is a beam loss monitor that can work in two modes [1]:

- The slow mode: used during user operation and for which a 1MΩ impedance is needed
- The fast mode: used during the injection and for which a 50Ω impedance is needed

In order to command these modes a program is necessary to control the impedance of the sensors at each time. The general method is to get the information if an injection occurs or not. This is given by the EPICS channel ALIRF-GUN:TRIG-FLIP.OVAL (0 means there is no injection, 1 means there is an injection). According to this PV values we need to change the values of the EPICS channels controlling the impedance of the sensors:
ARIDI-BLM10:TerA_s / :TerB_s / :TerC_s / :TerD_s 0=50Ω (low impedance) 1=1MΩ (high impedance). The control of these EPICS channel is done with the Python library PYCAFE [2]

2.2 Loss, lifetime and beam current

With the Libera BLM I gathered in the same graph the evolution of the loss and of the lifetime versus time and also the loss and PCT reading versus time. To create this graphs, the EPICS channels for all these parameters are needed:

- For the losses, look at the channels:ARIDI-BLM10:SigSa.A,B, C and D which are the four channels where the losses are measured
- For the lifetime reading, use the EPICS channel: ARIDI-PCT:TAU-HOUR
- For the PCT reading , use the EPICS channel:ARIDI-PCT:beam current

Knowing these channels, a plot of lifetime, loss and PCT over time is then possible:

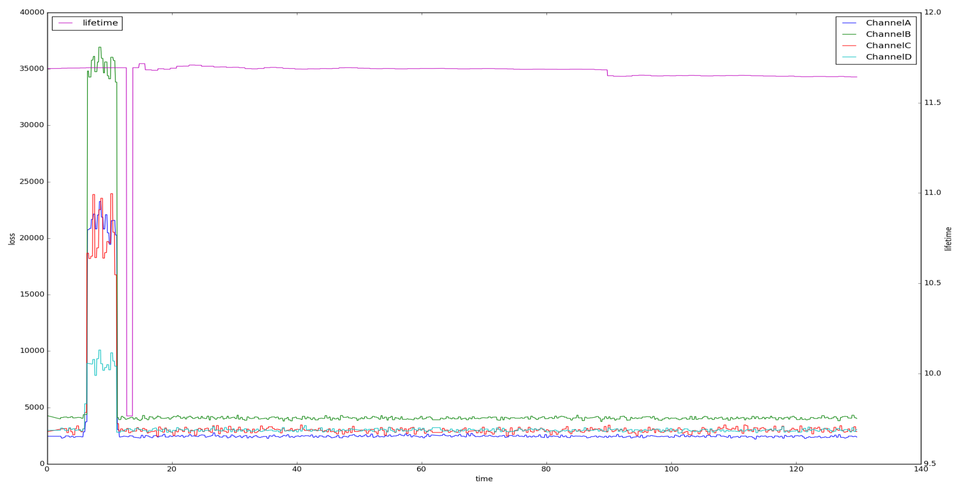


Figure 1: Lifetime and loss VS time over 130s (blue lines depicts the loss measured in channel A, green are the losses measured in channel B, red are the losses measured in channel C, cyan are the losses measured in channel, D and magenta is the lifetime in hour)

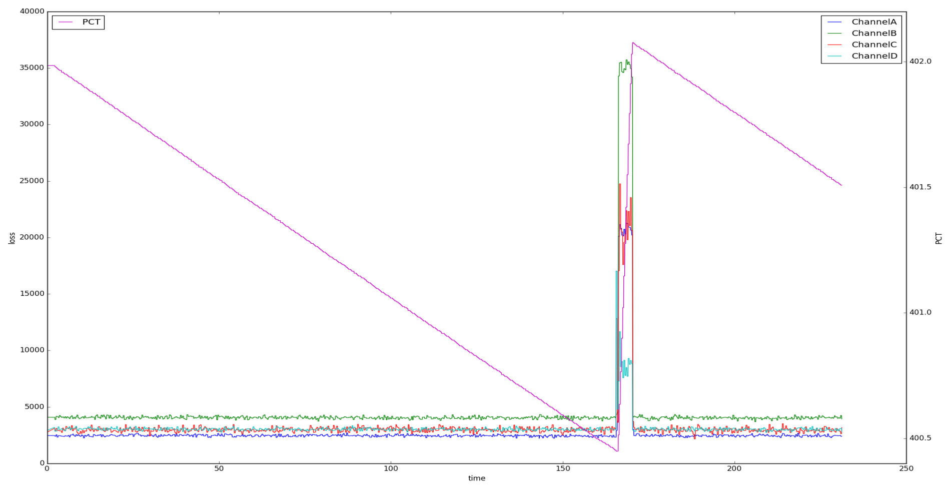


Figure 2: PCT and loss VS time over 250s (Blue are the losses measured in channel A, green are the losses measured in channel B, red are the losses measured in channel C, cyan are the losses measured in channel, D and magenta is the PCT readings)

Looking at fig. 1 and fig. 2 we can conclude that:

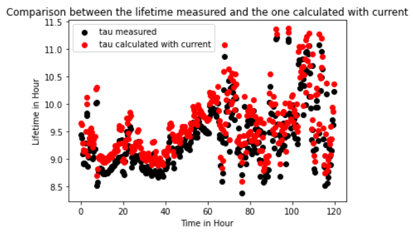
- To see a real correlation between lifetime loss and PCT reading we need to discount injection which distorts the results. Thus, from now on I did not measure during injection.
- To see a better correlation, a measurement over a longer period of time is necessary

3 How the lifetime is beam currently calculated

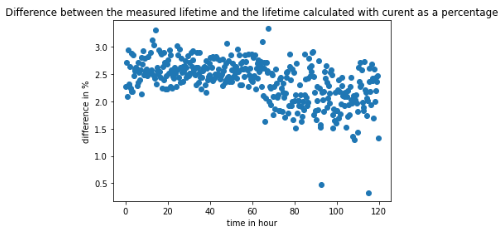
The relationship between lifetime and beam current is [6]:

$$\tau = -\frac{I}{\frac{dI}{dt}} \tag{1}$$

The lifetime calculated using Eq. (1) is compared to the lifetime computed on the EPICS channels ARIDI-PCT:TAU-HOUR. A moving average was used to smooth the lifetime calculated with the beam current.



(a)



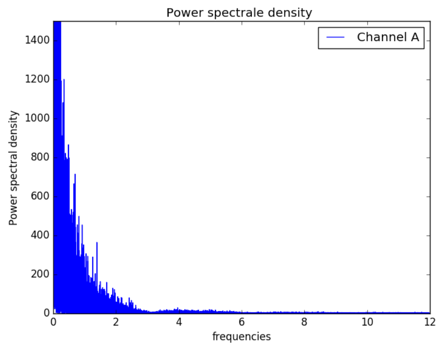
(b)

Figure 3: Figures of lifetime calculated with current of the stored beam as measured by PCT reading

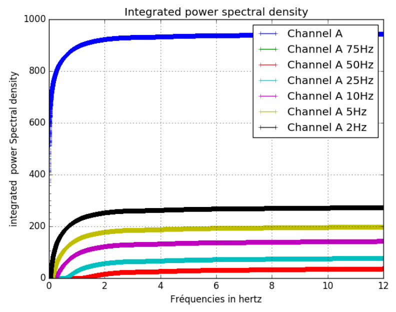
Figure 3a shows that the lifetime given by the EPICS channel is almost the same as the one we calculated with the PCT readings. Figure 3b shows that the difference between the EPICS channel and the lifetime calculated, without any injection, is constant around 2.5%

4 Noise measurement

We measured the background noise of the sensors without any beam in the accelerator and a gain of 0.4. An analysis of its spectral content and an attempt to filter the signal were done by calculating the PSD and the integrated PSD and finding a good filter. The fast loss monitor were the only beam loss monitor studied since the slow ones have too low sampling frequency (3 Hz) to have a good spectral analysis. The EPICS channel and the data taking code were



(a) Power spectral density



(b) Integrated PSD channel A for different filters

Figure 4: Noise analysis

limiting the sampling rate to a maximum of 24 Hz.

Fig. 4a and fig. 4b reveal that most of the noise is within 0 Hz to 2 Hz . Furthermore ,fig. 4b shows that by simply cutting the frequencies under 2 Hz more than 75% of the noise has been deleted. Nevertheless since the measurement frequency was only 24 Hz we cannot see the behavior of the sensor at high frequencies. To check if the noise is effectively at low frequencies, we will filter the signal with a cutting frequency fc=10 Hz

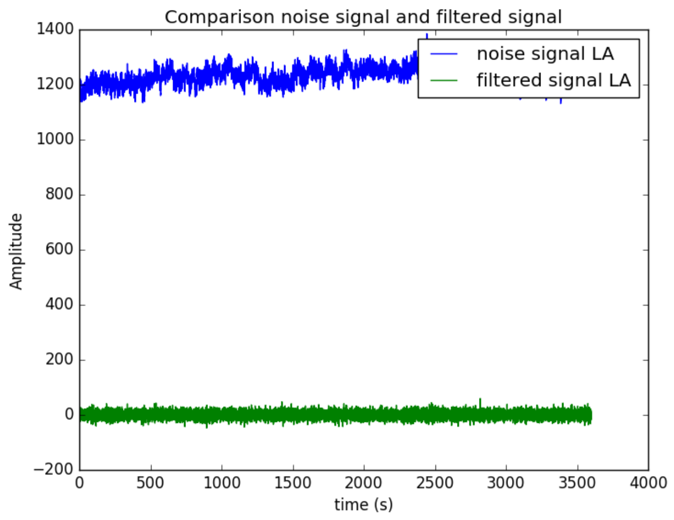


Figure 5: Noise signal and filtered signal

Figure 5 shows that almost all the noise has been filtered and evidently the background noise for the fast loss monitor is at low frequencies only.

5 First attempt to compute the lifetime with beam loss monitor

In order to compute the lifetime with the beam loss monitor, a measurement session was organised. We change the lifetime in different ways and observed the behavior of the slow and fast loss monitors:

- First measurement: Changing horizontal chromaticity from 6 to 1 gradually to change the lifetime of the machine
- Second measurement: Changing the vertical chromaticity from 3.5 to 6 gradually (the measurement width is smaller for reasons of machine stability)
- Third measurement: Changing the horizontal tune from 0.38 to 0.46 gradually to reach the half integer resonance
- Fourth measurement: Changing the horizontal tune from 0.38 to 0.26 gradually to reach the third order resonance

5.1 First measurement: Changing horizontal chromaticity

This measurement involves changing horizontal chromaticity from 6 to 1 to change the lifetime of the machine.
To change horizontal chromaticity: ARIMA-OPTIC:CX-SHIFT
To record data from loss monitors: ARIDI-BLM10:SigSa.A-D and ARIDI-BLM01-4:LOSS1-11
A record of lifetime, beam current and its derivative and chromaticity is also necessary.

5.1.1 Impact of the change of horizontal chromaticity

The change of horizontal chromaticity has a large impact on loss and lifetime. Figure 6 shows that between a chromaticity of 2 and 5 the behaviors of loss and lifetime are linear with chromaticity. Where outside this linear range, instabilities begin. A linear fit [3] of this range shows that the linear behavior of fast loss monitor and lifetime with horizontal chromaticity is linked to the stability of the machine. Furthermore fig. 6 reveals that the slow loss monitor are not sensitive to these changes. This graph also shows that the nominal chromaticity of 5 is not optimizing lifetime and loss and that with a lower chromaticity a significant increase of lifetime would be possible. The table in fig. 7 demonstrate that it is possible, for a beam current of 150 mA, to increase the lifetime by 4 hours with a chromaticity of 3.11 and avoid the instabilities of a too low chromaticity. The

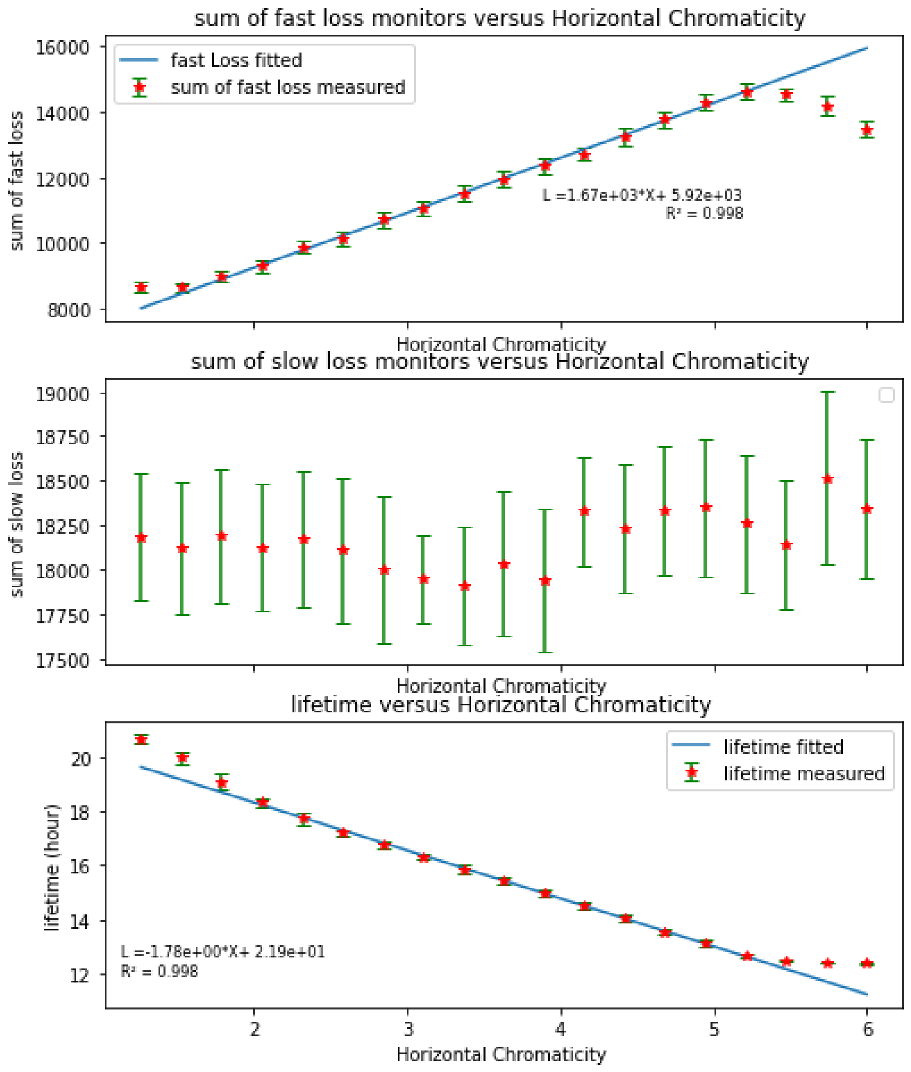
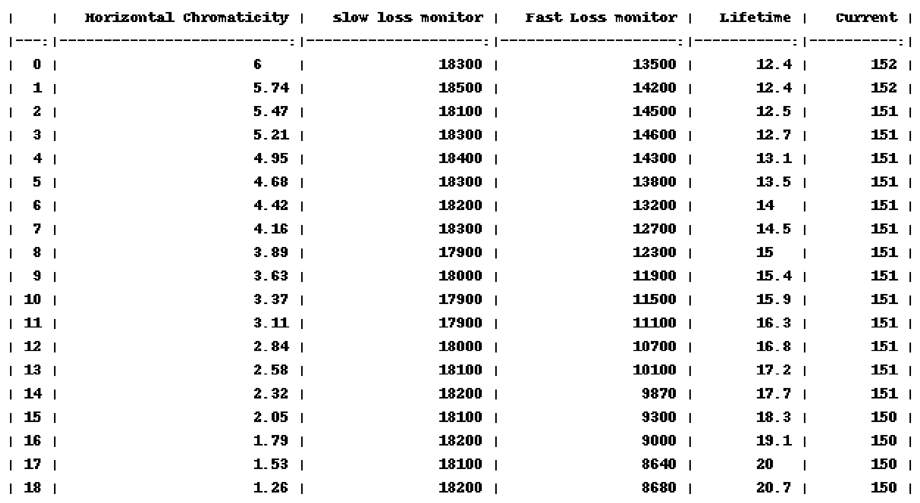


Figure 6: Impact of the change of horizontal chromaticity

main problem is that these experiments were done at $\approx 150\,\text{mA}$. Therefore we cannot conclude that at $\approx 400\,\text{mA}$ the instabilities will be absent.



	Horizontal Chromaticity	Slow loss monitor	Fast Loss monitor	Lifetime	Current
0	6	18300	13500	12.4	152
1	5.74	18500	14200	12.4	152
2	5.47	18100	14500	12.5	151
3	5.21	18300	14600	12.7	151
4	4.95	18400	14300	13.1	151
5	4.68	18300	13800	13.5	151
6	4.42	18200	13200	14	151
7	4.16	18300	12700	14.5	151
8	3.89	17900	12300	15	151
9	3.63	18000	11900	15.4	151
10	3.37	17900	11500	15.9	151
11	3.11	17900	11100	16.3	151
12	2.84	18000	10700	16.8	151
13	2.58	18100	10100	17.2	151
14	2.32	18200	9870	17.7	151
15	2.05	18100	9300	18.3	150
16	1.79	18200	9000	19.1	150
17	1.53	18100	8640	20	150
18	1.26	18200	8680	20.7	150

Figure 7: Table of values for a change of horizontal chromaticity

5.1.2 Fit of lifetime with loss

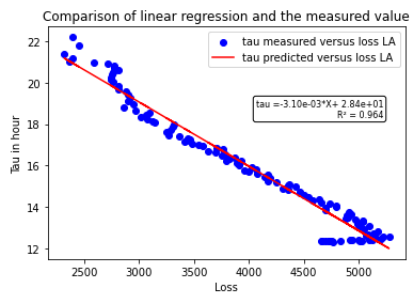
After a data analysis (moving average, deleting the peaks that were not relevant, computing the correlation between lifetime and all loss monitors) a fit of the lifetime with the loss is then possible. The most correlated loss monitors are the fast monitors and especially the channel A. A fit for the channel A of fast loss monitor (LA) gives the following result:

$$\tau_{fit} = 28.4 \pm 0.2 - (3.10 \times 10^{-3} \pm 7.2 \times 10^{-5}) * LA \tag{2}$$

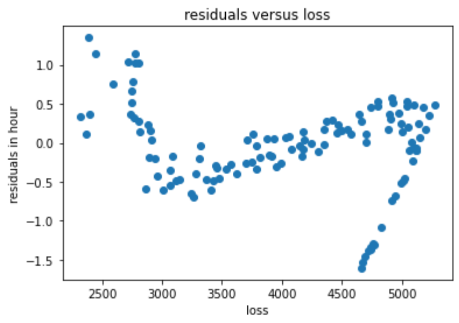
With a R^2=0.964, a mean squared error MSE=0.20 and a mean absolute error MAE=0.34 The fig. 8b shows the residuals of the fit. To compute it the following formula is used:

$$\Delta = \tau_{Epics} - \tau_{fit.} \tag{3}$$

The more the curve is near 0 and evenly distributed around zero the more it is accurate. Figure 8b illustrate that at low and high loss the fit is less accurate. In fig. 8 it can be seen that the for a specific change of horizontal chromaticity it is possible to compute the lifetime with a beam loss monitor with a small uncertainty on the fit, and that there is a linear relationship between the two of them. The linear relationship only work on a certain range where loss are not to high.



(a) Comparison of simple linear regression with LA and value measured



(b) Residuals versus loss

Figure 8: Fit of lifetime with loss

5.2 Second measurement: Changing vertical chromaticity

This measurement involves changing the vertical chromaticity from 3.5 to 6 to change the lifetime of the machine.
To change horizontal chromaticity: ARIMA-OPTIC:CY-SHIFT
To record data from loss monitors: ARIDI-BLM10:SigSa.A-D and ARIDI-BLM01-4:LOSS1-11
A record of lifetime, beam current and its derivative and chromaticity is also necessary.

5.2.1 Impact of the change of vertical chromaticity

The change of vertical chromaticity has a large impact on loss and lifetime. In fig. 9 it can be seen that the change of vertical chromaticity doesn't influence loss and lifetime in the same way as the changes of the horizontal one. The stability range for chromaticity is narrower.

A linear fit is performed using the linear part of the curve. It shows that the part where instabilities occurs is the non-linear part of the curves. Again, the slow loss monitor are not sensitive to these changes. An optimisation of chromaticity that would increase lifetime and reduce losses is harder for vertical chromaticity, since the system is more sensitive to a small change of vertical chromaticity. The table of values in fig. 10 shows that it is possible to change the chromaticity to have a better lifetime. However since the machine is more sensitive to changes of vertical chromaticity than to change of horizontal chromaticity at ($\approx 150\,\mathrm{mA}$), it is highly probable that at a higher beam current ($\approx 400\,\mathrm{mA}$), it won't be possible to change the vertical chromaticity and keep the machine stability.

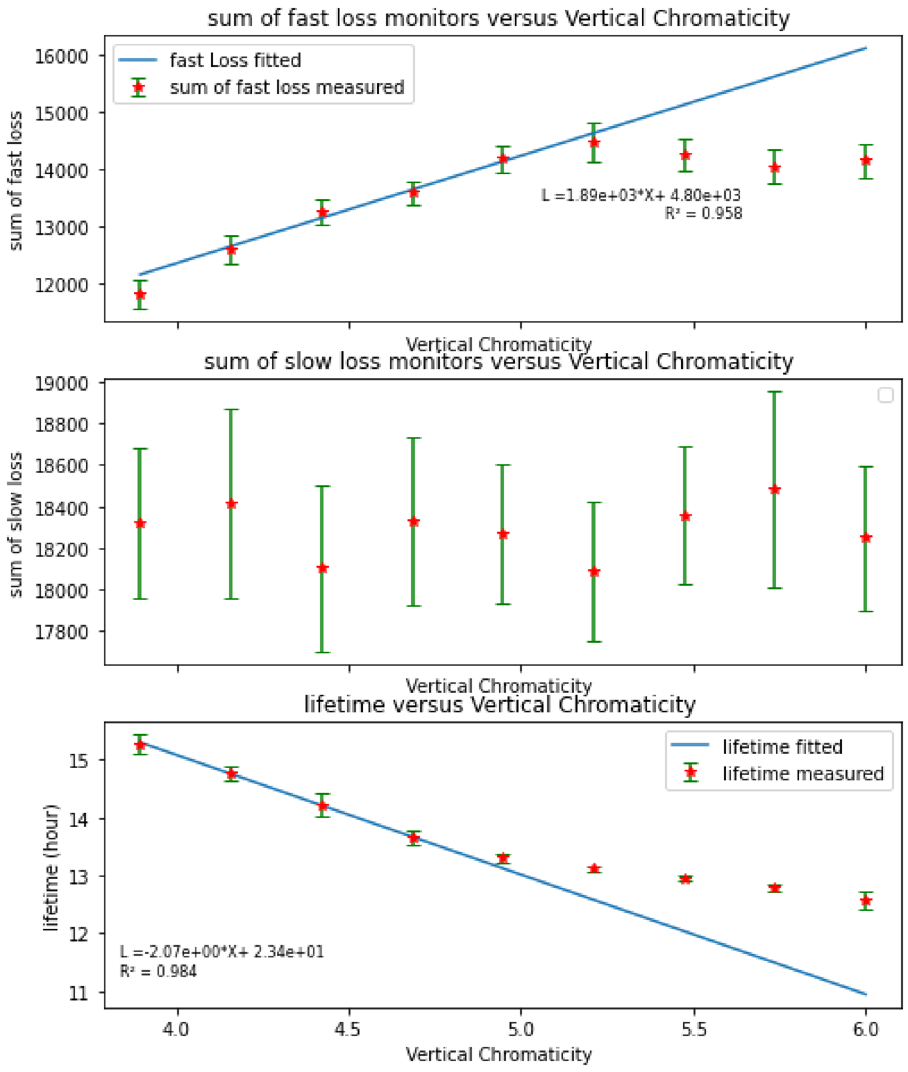
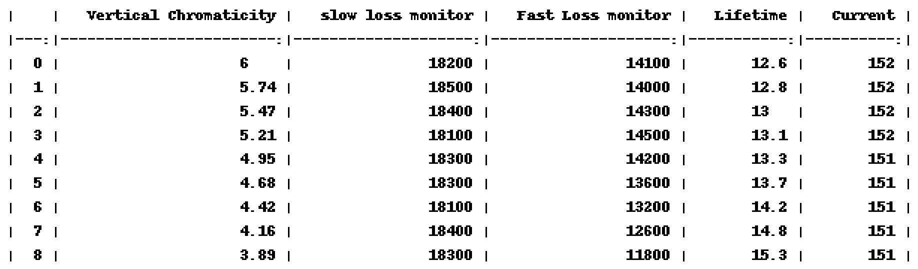


Figure 9: Impact of the change of vertical chromaticity



	Vertical Chromaticity	slow loss monitor	Fast Loss monitor	Lifetime	Current
0	6	18200	14100	12.6	152
1	5.74	18500	14000	12.8	152
2	5.47	18400	14300	13	152
3	5.21	18100	14500	13.1	152
4	4.95	18300	14200	13.3	151
5	4.68	18300	13600	13.7	151
6	4.42	18100	13200	14.2	151
7	4.16	18400	12600	14.8	151
8	3.89	18300	11800	15.3	151

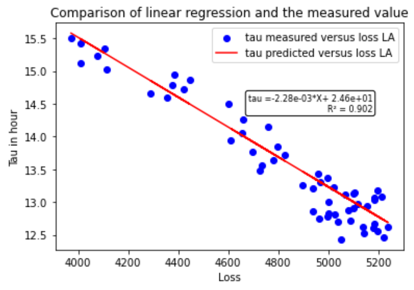
Figure 10: Table of values for a change of vertical chromaticity

5.2.2 Fit of lifetime with loss

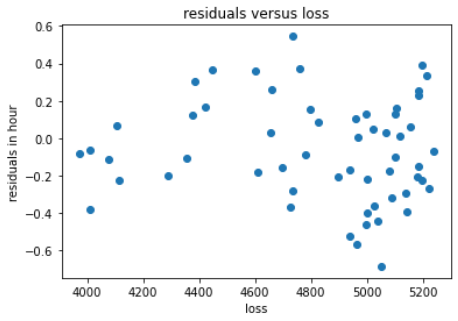
The most correlated loss monitors are the fast monitors and especially the channel A. A fit for the channel A of fast loss monitor (LA) gives the following result:

$$\tau = 24.5 \pm 0.5 - (2.28 * 10^{-3} \pm 2.1 * 10^{-4}) * LA \tag{4}$$

With a R^2=0.90 a mean squared error MSE=0.07 and a mean absolute error MAE=0.21



(a) Comparison of simple linear regression with LA and value measured



(b) Residuals versus loss

Figure 11: Fit of lifetime with loss

Figure 11a for a specific change in vertical chromaticity, it is possible to compute the lifetime with a beam loss monitor and that there is a linear relationship between them. The residuals on fig. 11b illustrate that linear relationship is a relevant model since they seem evenly distributed around zero Nevertheless, the fit for a change of vertical chromaticity is less precise than the one for the horizontal chromaticity. It is certainly due to the fact that the range of stability for the vertical chromaticity is narrower than the horizontal one.

5.3 Third measurement: Changing the horizontal tune

This measurement involves changing the Horizontal tune from 0.38 to 0.46 to reach the half-integer resonance.
To record data from loss monitors: ARIDI-BLM10:SigSa.A-D and ARIDI-BLM01-4:LOSS1-11.
A record of lifetime, beam current and its derivative and of the tune is also necessary.

5.3.1 Impact of the change of horizontal tune

The change of tune has an impact on loss ,lifetime and beam current. We tried to approach the half integer resonance to reduce the lifetime and thus increase the loss. Figure 12 and fig. 13 show a plot of lifetime, beam current, and the values of the sum of all the fast loss monitors over the ring and the value of the sum of all the slow loss monitors over the ring with respect to tune. It is necessary to separate the resonant and non-resonant part of the curves to see what happens during these two moments. Figure 12 shows the behavior of the measured parameters when the resonance is not yet reached. Where fig. 13 shows the change of these parameters when the resonance is reached.

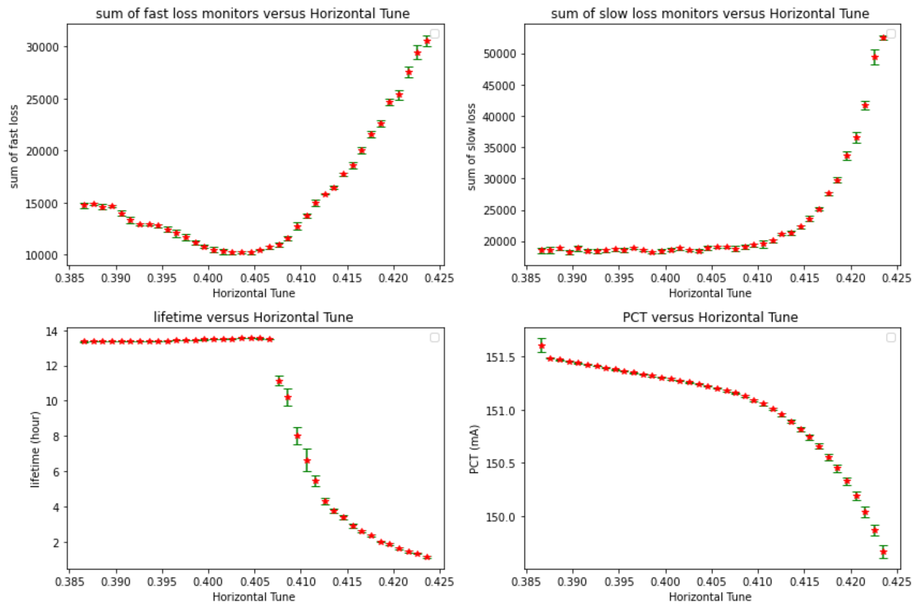


Figure 12: Impact of the change of horizontal tune

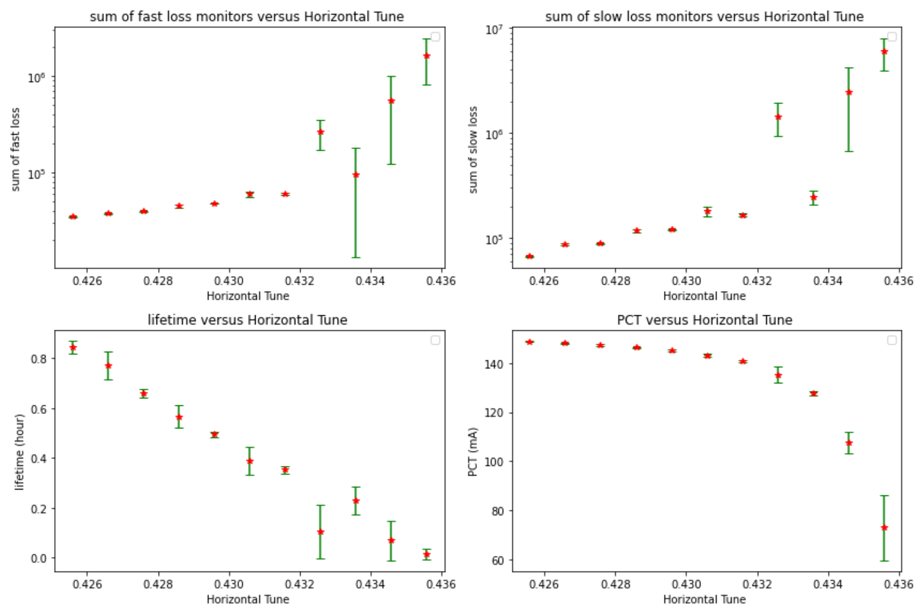


Figure 13: Impact of the change of horizontal tune during resonance

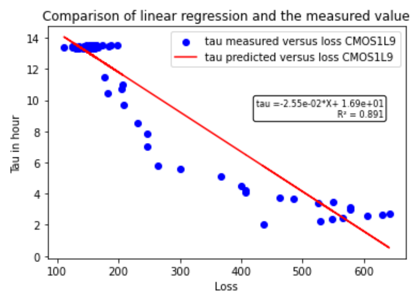
Figure 12 demonstrates that the behavior of the slow and fast monitor isn't the same with the tune. It shows that the loss are indeed localised and that a change of tune will affect at first the location of the fast sensor. Indeed it shows that an increase of loss for a sensor doesn't directly mean a decrease of lifetime, because another loss monitor could decrease at the same time. Here the slow loss monitors are also sensitive to this changes of horizontal tune which was not the case during the two previous measurements. The change of lifetime during the resonance is not due to Touschek scattering. Thus, it is not possible to compute simply the Touschek lifetime during resonance.

5.3.2 Fit of lifetime with loss

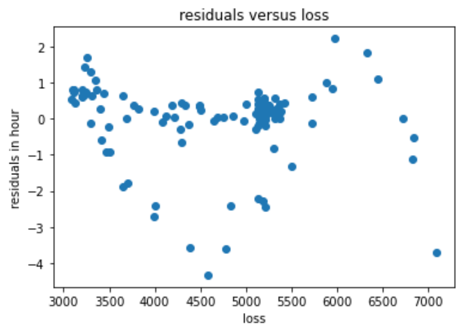
The most correlated loss monitors are this time the slow monitors and especially the one in arc 3 (CMOS1L9). A fit for the most correlated slow monitor (CMOS1L9) gives the following result:

$$\tau = 16.9 \pm 0.21 - (2.55 * 10^{-2} \pm 1.23 * 10^{-3}) * CMOS1L9 \tag{5}$$

With R^2=0.89, a mean absolute error MAE=0.65 and a mean squared error MSE=1.12
To perform this fit the resonant part has been ignored. The fit was only performed on the non-resonant part. Thus the Touschek scattering has more effect on the change of lifetime than during resonance (we still cannot conclude that the lifetime is mostly Touschek). The fit for a change of horizontal tune are less precise than the ones on chromaticity and especially in this experiment when we



(a) Comparison of linear regression and value measured



(b) Residuals versus loss

Figure 14: Fit of lifetime with loss

approach a tune of 0.46. The relationship between this loss monitor and lifetime cannot be model by a linear formula. Figure 14b shows residuals far from 0 and unequally distributed around zero. Figure 14a demonstrate the relationship is not linear between this sensor and lifetime for a change of tune.

5.4 Fourth measurement: Changing the horizontal tune

This measurement involves the horizontal tune from 0.38 to 0.26 to reach the sum resonance. Indeed the vertical tune is around 0.73.
To record data from loss monitors: ARIDI-BLM10:SigSa.A-D and ARIDI-BLM01-4:LOSS1-11
A record of lifetime, beam current and its derivative and of the tune is also necessary.

5.4.1 Impact of the change of horizontal tune

The change of tune has impact on loss, lifetime and PCT reading. Since we try to approach the sum resonance the losses increase and the lifetime consequently decrease. Figure 15 and fig. 16 show a plot of lifetime, beam current, and the values of the sum of all the fast loss monitors over the ring and the value of the sum of all the slow loss monitors over the ring with respect to tune. It is necessary to separate the resonant and non-resonant part of the curve to understand the behavior of these variables. Figure 15 shows the behavior of these parameters when the resonance is not yet reached. Where fig. 16 shows the change of these parameters the resonance when the resonance is reached.

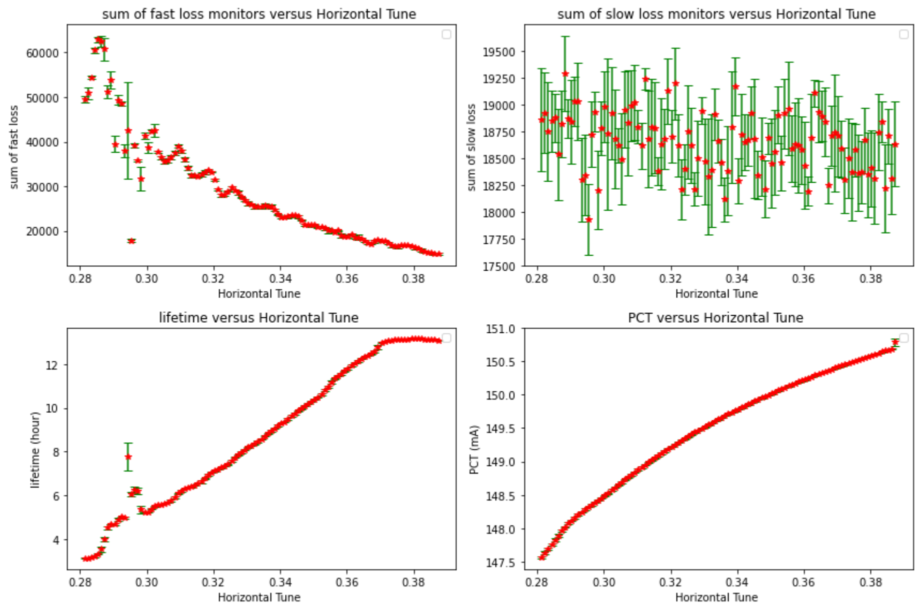


Figure 15: Impact of the change of horizontal tune

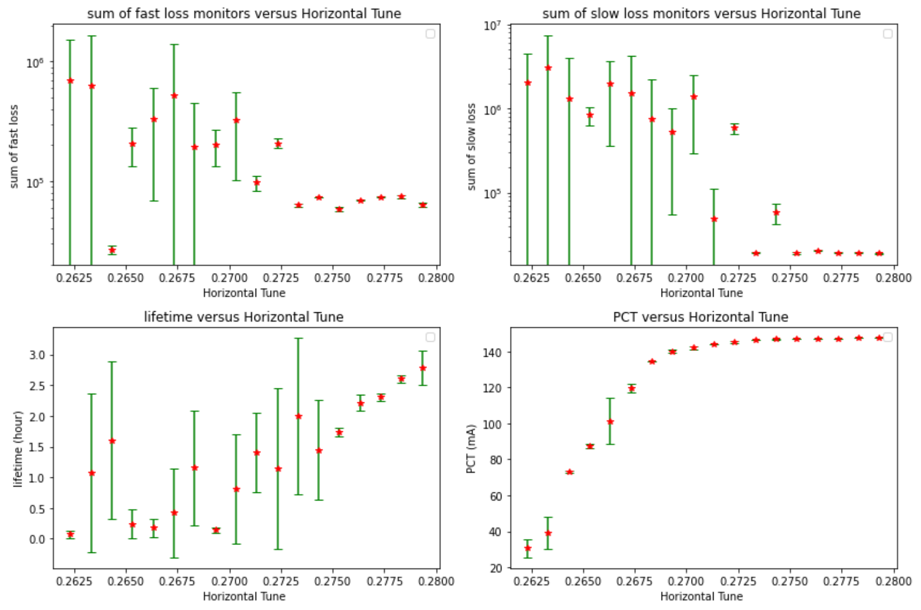


Figure 16: Impact of the change of horizontal tune during resonance

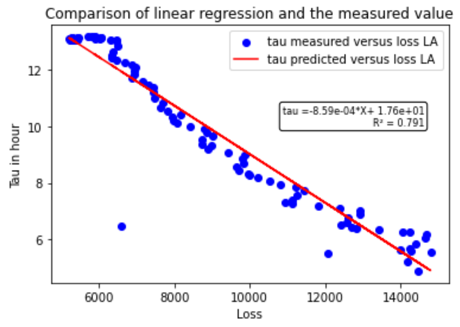
Figure 15 demonstrate that the behavior of the slow and fast monitor isn't the same with the tune. The fast loss monitors seems to have a linear relationship with the change of the tune when we avoid the resonance while the slow monitor seem to stay constant until the resonance. However the behavior of the lifetime with tune shows directly that the fast loss monitor is better correlated than the slow ones. During resonance in fig. 16 the correlation between fast losses and lifetime is still clear. But the slow loss monitor is also becoming more correlated to lifetime. It shows that even for fast changes in losses, if these loss are high enough the slow loss monitor will become correlated to the lifetime.

5.4.2 Fit of lifetime with loss

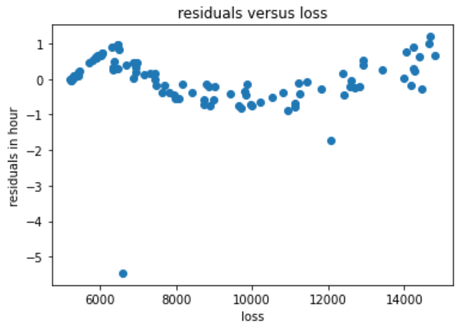
The most correlated loss monitors are the fast monitors and especially the channel A. A fit for the channel A of fast loss monitor (LA) gives the following result:

$$\tau = 17.6 \pm 0.17 - (8.59 * 10^{-4} \pm 2.06 * 10^{-5}) * LA \tag{6}$$

With R^2=0.79, a mean absolute error MAE=0.62 and a mean squared error MSE=1.26 To perform this fit we also ignored the resonant part and we can see



(a) Comparison of linear regression and value measured



(b) Residuals versus loss

Figure 17: Fit of lifetime with loss

that for this experiment the fit is also less accurate than the ones on chromaticity. The unequally distributed residuals in fig. 17b and the comparison between the fit and the EPICS channel lifetime also illustrate that the relationship loss monitors and lifetime is not linear during a change of tune.

6 Fit of the lifetime on nominal values

6.1 Test of the fits done during the shift

Let's test the models that we have determined during the four measurement from section 5, on data of loss and lifetime acquired passively on the machine during user operation. The objective is to compare the lifetime given by these fits to the lifetime given by the machine.

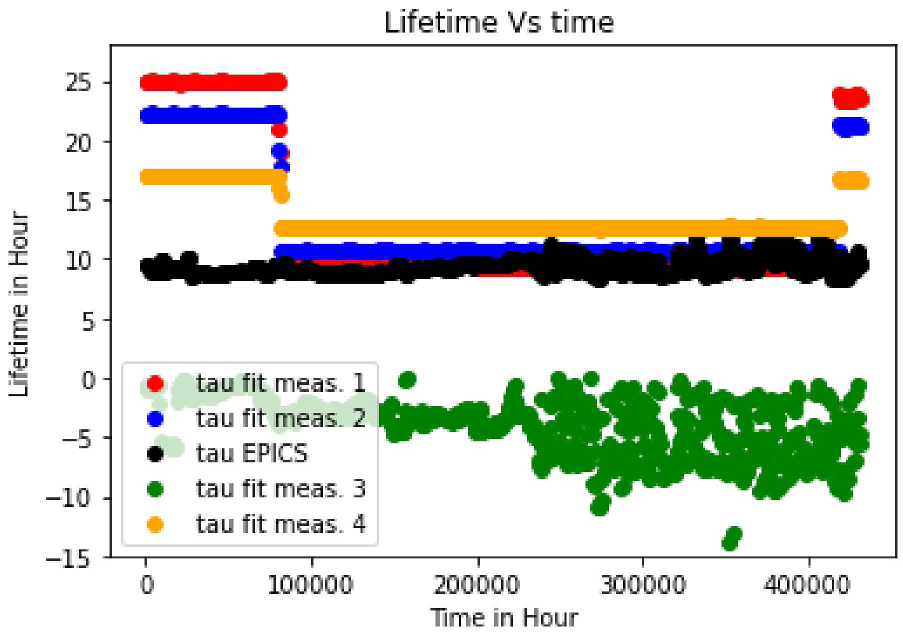


Figure 18: Comparison between the models from the shift and nominal data

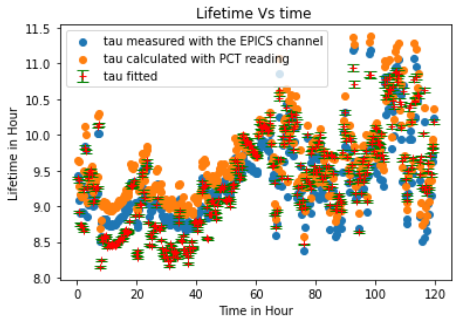
Figure 18 shows that these models are too sensitive and most of the fits give always wrong values for the lifetime during user operation.

6.2 Using data acquired passively during user operation

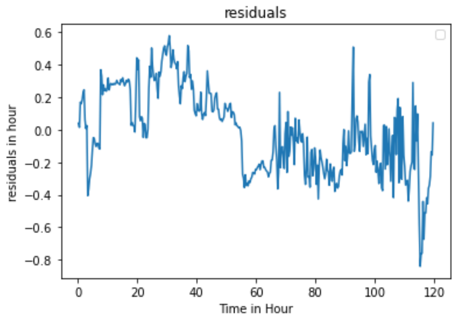
The fits on the lifetime with the experimental data don't work on nominal data. This is due to two things: the beam current for the experiment was lower than the nominal beam current, and moreover losses generated by the change of tune or chromaticity are not the same as losses during normal operation. A fit on nominal data acquired passively in the control room over several hours was performed, and to ensure the independence of the beam current value on the fit it was necessary to normalize each loss by the beam current. By looking at the most correlated loss monitors with respect to lifetime, we see that the slow loss monitors are more correlated and especially the sensors near the undulator like CMOS1L11 or CMOS2L1 thus by performing a fit with CMOS1L11 we have:

$$\tau = 12.39 \pm 0.005 - (0.163 \pm 0.001) * \frac{CMOS1L11}{I} \tag{7}$$

This fit permits to compute the Touschek lifetime during normal operation in faster way than the lifetime directly computed by the EPICS channel. Indeed, during normal operation the Touschek lifetime is dominant and we can state that $\tau_{tousch} \approx \tau$. A proof of this will be given at the end of section 7.



(a) Lifetime versus time



(b) Residuals versus time

Figure 19: Fit of lifetime from loss

6.3 How the data is processed ?

The dataset used for this fit contained 785586 elements. A data processing is, therefore, important to eliminate outliers that could distort the results of the regression. Four methods were used on after the other to process the data:

- The fact that values are not measured during injection doesn't insure that the sensors had the time just before an injection to go back to normal values. It is then important to delete every value for which the derivative of PCT reading is positive.
- It is important to delete all the values for which the lifetime calculated by the EPICS channel is the same. To make sure that for two same values of lifetime we don't have two different values of loss.
- A moving average is used to smooth the signal of the sensors.
- Finally a normalisation by PCT reading was done to insure the independence of the loss monitor with respect to beam current

At the end of this data processing the dataset contained 737 elements and the fit was computed from these values. The model was then used on a different set of raw data, acquired after the fitting, to test its accuracy fig. 19a.

7 Fifth measurement: Change of the beam size

A last experiment was done to measure Touschek lifetime and gas scattering lifetime. To measure Touschek lifetime we excited the beam at the tune frequency with a signal generator linked to the multi-bunch feedback. We changed the gain of the multi bunch feedback from 100% to 0% in several steps. This changed the beam size from 40 μm to 170 μm. This same measurement was done three times, because a linear change of gain didn't imply a linear change of forward power. Thus different ways to change the gain were tried.

- First, scanning the gain linearly from 100% to 0% with 80 points
- Second, scanning the gain linearly from 80% to 0% with 80 points because the forward power didn't change from a gain of 100% to 80%
- Third, scanning the gain on a logarithmic scale from 80% to 0% with 80 points to counter the non-linear change of forward power

7.1 Measurement of gas scattering lifetime and Touschek lifetime

We state that gas scattering lifetime is constant during the measurement. When the beam size is really large the space between all particles is larger and the scattering between particles has less importance on the loss,thus $\tau_{tousch} \gg \tau_{gas}$. Therefore most of the lifetime would be gas scattering lifetime for an infinite beam size. It is then possible to compute the Touschek lifetime with this equation [6]:

$$\frac{1}{\tau} = \frac{1}{\tau_{Tousch}} + \frac{1}{\tau_{gas}} \tag{8}$$

And consequently:

$$\tau_{Tousch} = \frac{1}{\frac{1}{\tau} - \frac{1}{\tau_{gas}}} \tag{9}$$

This measurement involves a change of beam size from 40 μm to 170 μm. However when the beam size is small between 40μm to 60 μm. It is not possible to be sure that the beam size measurement is right and not modified by the oscillation of the beam. Thus all the fits were performed from 60 μm to 170 μm. To know the lifetime for an infinite beam size, fit the inverse of lifetime with respect to the beam size. For each measurement two methods to determine gas scattering lifetime and Touschek lifetime were used:

- Doing a fit on the lifetime given by the EPICS channel
- Doing a fit on a lifetime calculated directly with beam current using Eq. (1)

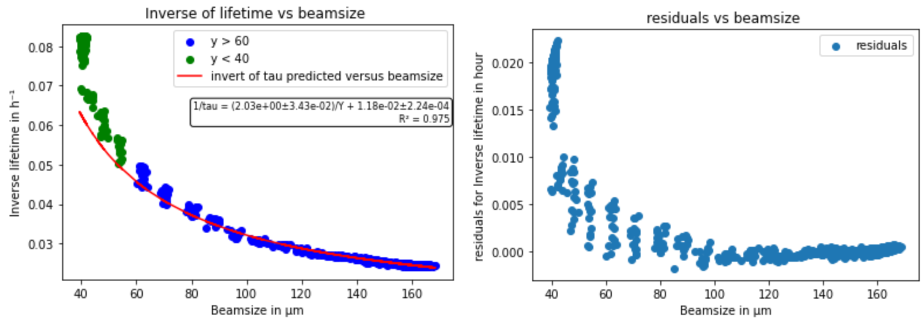
7.2 Using the lifetime given by the EPICS channel

7.2.1 First measurement

The fit for this measurement shows the following relationship:

$$\frac{1}{\tau} = 1.18 * 10^{-2} \pm 2.24 * 10^{-4} + \frac{2.03 \pm 3.43 * 10^{-2}}{Y} \tag{10}$$

With a R^2=0.975, a mean absolute error MAE= 3.79*10^{-4} and a mean squared error MSE=6.65*10^{-7}



(a) Inverse lifetime vs beam size (b) Residuals of lifetime versus beamsize

Figure 20: Fit of lifetime with respect to beam size

As the residuals in fig. 20b show, ignoring small beam size does change the fit that approximate the data and thus the gas scattering lifetime. The gas scattering lifetime is then:

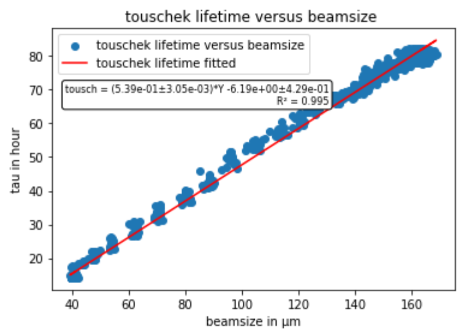
$$\tau_{gas} = \frac{1}{1.18 * 10^{-2}} = 84.7h \pm 3.3h \tag{11}$$

It is then possible to compute the Touschek lifetime using Eq.(8):

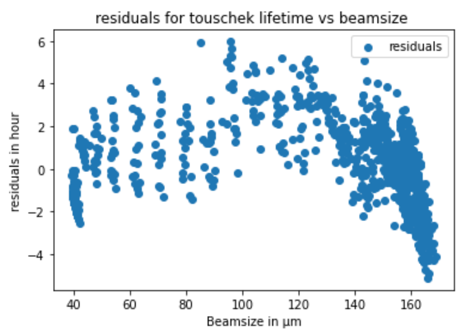
$$\tau_{tousch} = (5.39 * 10^{-1} \pm 3.05 * 10^{-3}) * Y - (6.19 \pm 4.29 * 10^{-1}) \tag{12}$$

With a R^2=0.995, a mean absolute error MAE= 1.48 and a mean squared error MSE=3.29.
As expected there is almost a linear relationship between the Touschek lifetime and the beam size fig. 21a. The residuals fig. 21b shows that the fit is well approximating the data, except for large beam size. Furthermore by looking at the Touschek lifetime for a forward power P=0 it is possible to determine the Touschek lifetime during normal operations.

$$\tau_{Tousch_{nominal}} = 14.08h \pm 0.09h \tag{13}$$



(a) Touschek lifetime vs beam size



(b) Residuals of Touschek lifetime versus beam size

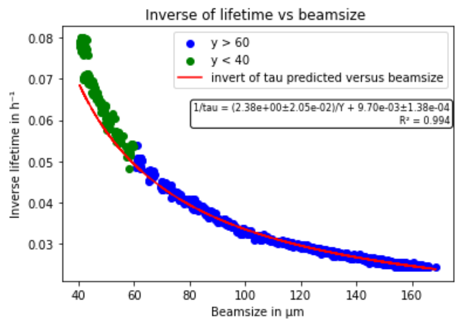
Figure 21: Fit of Touschek lifetime with respect to beam size

7.2.2 Second measurement

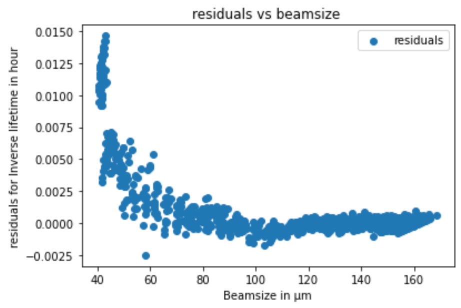
The fit for this measurement shows the following relationship:

$$\frac{1}{\tau} = 9.70 * 10^{-3} \pm 1.38 * 10^{-4} + \frac{2.38 \pm 2.05 * 10^{-2}}{Y} \tag{14}$$

With a R^2=0.994, a mean absolute error MAE= $3.03*10^{-4}$ and a mean squared error MSE=$1.83*10^{-7}$



(a) Inverse lifetime vs beam size



(b) Residuals of lifetime versus beam size

Figure 22: Fit of lifetime with respect to beam size

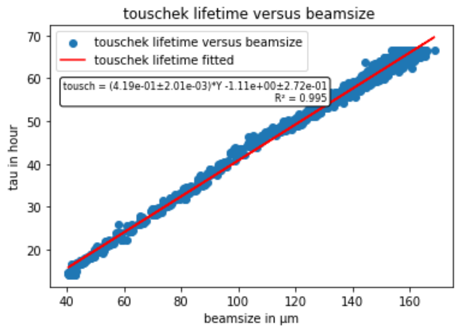
As the residuals fig. 22b show, ignoring small beam size does change the fit that approximate the data and thus the gas scattering lifetime. The gas scattering lifetime is then:

$$\tau_{gas} = \frac{1}{9.70 * 10^{-3}} = 103.1h \pm 3.09h \tag{15}$$

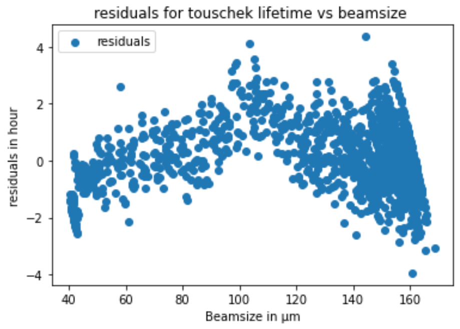
It is then possible to compute the Touschek lifetime using Eq.(8):

$$\tau_{tousch} = (4.19 * 10^{-1} \pm 2.01 * 10^{-3}) * Y - 1.11 \pm 2.72 * 10^{-1} \tag{16}$$

With a R^2=0.995, a mean absolute error MAE=0.98 and a mean squared error MSE=1.51



(a) Touschek lifetime vs beam size



(b) Residuals of Touschek lifetime versus beam size

Figure 23: Fit of Touschek lifetime with respect to beam size

As expected there is almost a linear relationship between the Touschek lifetime and the beam size fig. 23a. The residuals fig. 23b shows that the fit is well approximating the data, except for large beam size. Furthermore by looking at the Touschek lifetime for a forward power P=0 it is possible to determine the Touschek lifetime during normal operations.

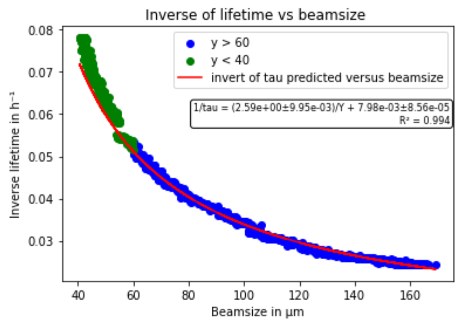
$$\tau_{Tousch_{nominal}} = 14.18h \pm 0.06h \tag{17}$$

7.2.3 Third measurement

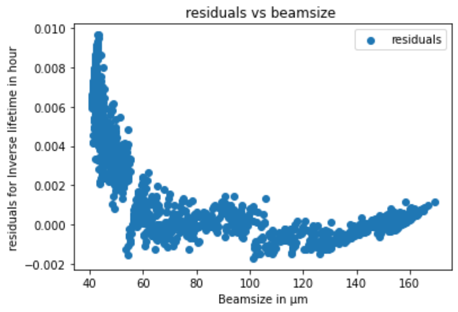
The fit for this measurement shows the following relationship:

$$\frac{1}{\tau} = 7.98 * 10^{-3} \pm 8.56 * 10^{-5} + \frac{2.59 \pm 9.95 * 10^{-3}}{Y} \tag{18}$$

With a R^2=0.994, a mean absolute error MAE= $5.03*10^{-4}$ and a mean squared error MSE=$4.20*10^{-7}$.



(a) Inverse lifetime vs beam size



(b) Residuals of lifetime versus beam size

Figure 24: Fit of lifetime with respect to beam size

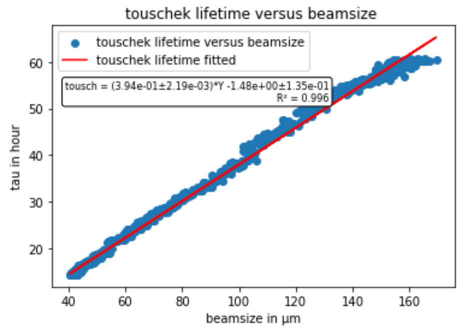
As the residuals fig. 24b show, ignoring small beam size does change the fit that approximate the data and thus the gas scattering lifetime. The gas scattering lifetime is then:

$$\tau_{gas} = \frac{1}{7.98 * 10^{-3}} = 125.3h \pm 2.6h \tag{19}$$

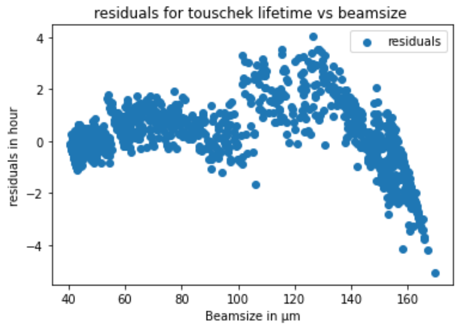
It is then possible to compute the Touschek lifetime using Eq.(8):

$$\tau_{tousch} = (3.94 * 10^{-1} \pm 2.19 * 10^{-3}) * Y - 1.48 \pm 1.35 * 10^{-1} \tag{20}$$

With a R^2=0.996, a mean absolute error MAE=0.75 and a mean squared error MSE=1.14



(a) Touschek lifetime vs beam size



(b) Residuals of Touschek lifetime versus beam size

Figure 25: Fit of Touschek lifetime with respect to beam size

As expected there is almost a linear relationship between the Touschek lifetime and the beam size fig. 25a. The residuals fig. 25b show that the fit is well approximating the data, except for large beam size. Furthermore by looking at the Touschek lifetime for a forward power P=0 it is possible to determine the Touschek lifetime during normal operations.

$$\tau_{Tousch_{nominal}} = 14.27h \pm 0.03h \tag{21}$$

7.2.4 Conclusion on these three fits

Using a logarithmic scale for the change of gain on the multi-bunch feedback permits to have a well distributed beam size for the fit. The values of Touschek lifetime with there error seem similar for each measurement. However the gas scattering lifetime has the same magnitude but the value seem far from each other. Nevertheless it doesn't modify the total lifetime significantly since $\frac{1}{tau_{gas}} \approx 1 * 10^{-2}$ for all of the measurements.

7.3 Using the lifetime calculated with the beam current

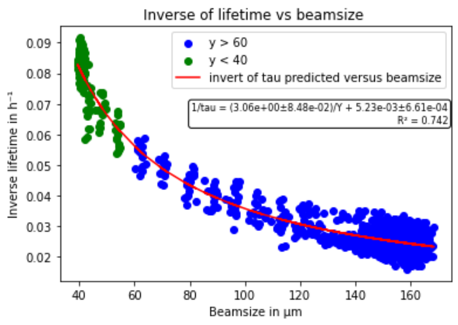
The fit is done with Eq.(1) on the raw data. Indeed the only advantage of using PCT reading for the computation of lifetime is the high sampling rate of this channel, which insure the independence of each values. Thus it wouldn't be useful to do a moving average on those data, because it would only give the same result as EPICS lifetime. We would loose the advantage of the PCT reading.

7.3.1 First Measurement

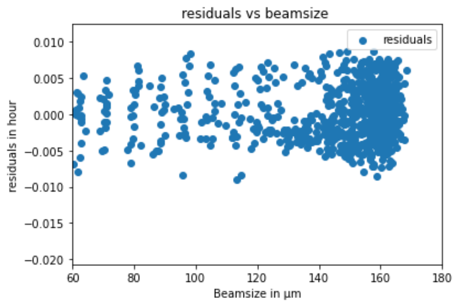
The fit for this measurement shows the following relationship:

$$\frac{1}{\tau} = 5.23 * 10^{-3} \pm 6.61 * 10^{-4} + \frac{3.06 \pm 8.42 * 10^{-2}}{Y} \tag{22}$$

With a R^2=0.74, a mean absolute error MAE= 3.31*10^{-3} and a mean squared error MSE=1.54*10^{-5}



(a) Inverse lifetime versus beam size



(b) Residuals of lifetime versus beam size

Figure 26: Fit of lifetime with respect to beam size

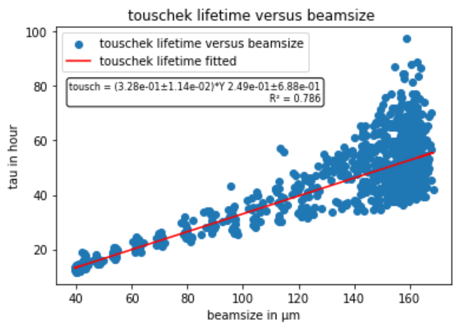
Figure 26b show that even by only taking data from 60μm to 170μm the fit is giving good and evenly distributed residuals around zero for all beam sizes. Thus the gas scattering lifetime has not been changed by the fact that we ignored small beam size. The gas scattering lifetime is then:

$$\tau_{gas} = \frac{1}{5.23 * 10^{-3}} = 191.2h \pm 39.4h \tag{23}$$

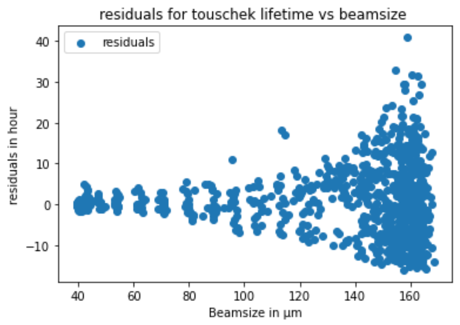
It is then possible to compute the Touschek lifetime using Eq.(8):

$$\tau_{tousch} = (3.28 * 10^{-1} \pm 1.13 * 10^{-2}) * Y + 2.49 * 10^{-1} \pm 6.88 * 10^{-1} \tag{24}$$

With a R^2=0.0.786, a mean absolute error MAE= 6.0 and mean squared error MSE=74.0



(a) Touschek lifetime vs beam size



(b) Residuals of Touschek lifetime versus beam size

Figure 27: Fit of Touschek lifetime with respect to beam size

As expected there is almost a linear relationship between the Touschek lifetime and the beam size fig. 27a. Nevertheless as seen with the values of uncertainties on the intercept of the fit and at MAE and MSE the fit isn't giving usable data. Indeed since the standard deviation of the Touschek lifetime especially for long beam size is big it is hard for the script to find a fit minimising the uncertainties. Furthermore by looking at the Touschek lifetime for a forward power P=0 it is possible to determine the Touschek lifetime during normal operations.

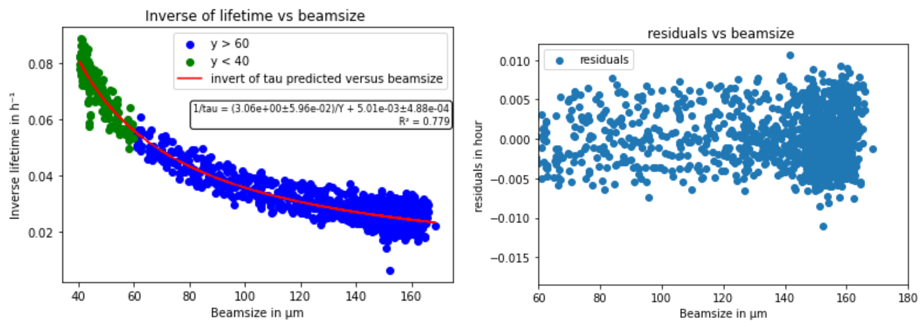
$$\tau_{Tousch_{nominal}} = 13.4h \pm 0.2h \tag{25}$$

7.3.2 Second Measurement

The fit for this measurement shows the following relationship:

$$\frac{1}{\tau} = 5.01 * 10^{-3} \pm 4.88 * 10^{-4} + \frac{3.06 \pm 5.96 * 10^{-2}}{Y} \tag{26}$$

With a R^2=0.779, a mean absolute error MAE= 2.86*10^{-3} and a mean squared error MSE=1.30*10^{-5}



(a) Inverse lifetime versus beam size (b) Residuals of lifetime versus beam size

Figure 28: Fit of lifetime with respect to beam size

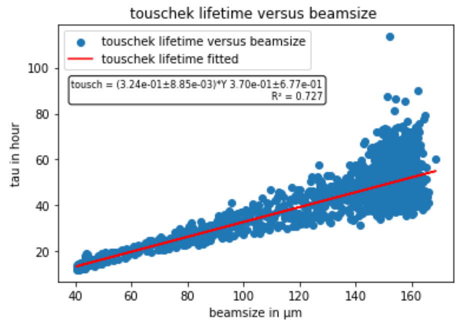
Figure 28b show that even by only taking data from 60μm to 170μm the fit is giving good and evenly distributed residuals around zero for all beam sizes. Thus the gas scattering lifetime has not been changed by the fact that we ignored small beam size. The gas scattering lifetime is then:

$$\tau_{gas} = \frac{1}{5.01 * 10^{-3}} = 199.7h \pm 35.8h \tag{27}$$

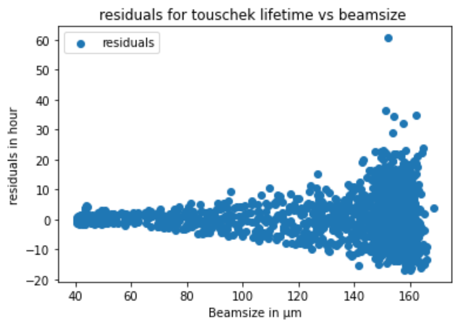
It is then possible to compute the Touschek lifetime using Eq.(8):

$$\tau_{tousch} = (3.23 * 10^{-1} \pm 8.84 * 10^{-3}) * Y + 3.7 * 10^{-1} \pm 6.8 * 10^{-1} \tag{28}$$

With a R^2=0.727, a mean absolute error MAE= 5.7 and a mean squared error MSE=71.6



(a) Touschek lifetime vs beam size



(b) Residuals of Touschek lifetime versus beam size

Figure 29: Fit of Touschek lifetime with respect to beam size

As expected there is almost a linear relationship between the Touschek lifetime and the beam size fig. 29a. Nevertheless as seen with the values of uncertainties on the intercept of the fit and at MAE and MSE the fit isn't giving usable data. Indeed since the standard deviation of the Touschek lifetime especially for long beam size is big it is hard for the script to find a fit minimising the uncertainties. Furthermore by looking at the Touschek lifetime for a forward power P=0 it is possible to determine the Touschek lifetime during normal operations.

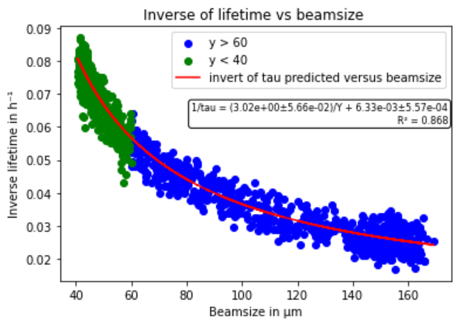
$$\tau_{Tousch_{nominal}} = 13.9 \pm 0.2 \tag{29}$$

7.3.3 Third Measurement

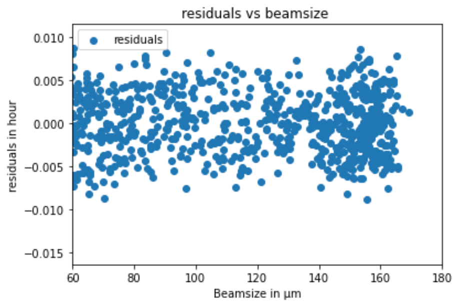
The fit for this measurement shows the following relationship:

$$\frac{1}{\tau} = 6.33 * 10^{-3} \pm 5.57 * 10^{-4} + \frac{3.02 \pm 5.66 * 10^{-2}}{Y} \tag{30}$$

With a R^2=0.87, a mean absolute error MAE= 2.90*10^{-3} and a mean squared error MSE=1.27*10^{-5}



(a) Inverse lifetime versus beam size



(b) Residuals of lifetime versus beam size

Figure 30: Fit of lifetime with respect to beam size

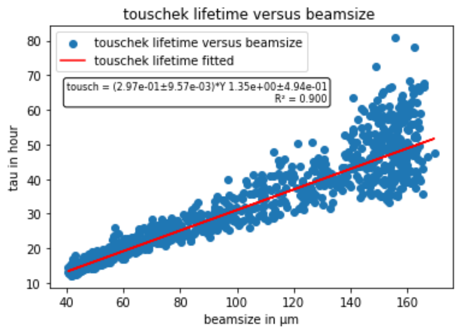
Figure 30b show that even by only taking data from 60μm to 170μm the fit is giving good and evenly distributed residuals around zero for all beam sizes. Thus the gas scattering lifetime has not been changed by the fact that we ignored small beam size. The gas scattering lifetime is then:

$$\tau_{gas} = \frac{1}{6.33 * 10^{-3}} = 158.0h \pm 29.7h \tag{31}$$

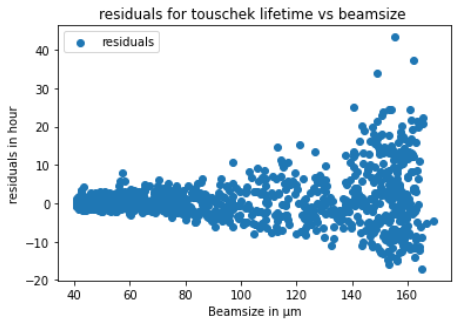
It is then possible to compute the Touschek lifetime using Eq.(8):

$$\tau_{tousch} = (2.97 * 10^{-1} \pm 9.57 * 10^{-3}) * Y + 1.35 \pm 4.94 * 10^{-1} \tag{32}$$

With a R^2=0.9, a mean absolute error MAE= 2.5 and a mean squared error MSE=19.8



(a) Touschek lifetime vs beam size



(b) Residuals of Touschek lifetime versus beam size

Figure 31: Fit of Touschek lifetime with respect to beam size

As expected there is almost a linear relationship between the Touschek lifetime and the beam size fig. 31a. This time the error on the fit is significantly smaller and the R^2 bigger. It is certainly due to the way we changed the gain on this third experiment.Scanning the gain on logarithmic scale has certainly countered the non-linear changes of forward power with respect to gain. And distributed the PCT reading in a better way. Indeed since the sampling rate of the PCT reading is bigger than the one for the lifetime channel, the way we scan may seems to have more importance on how the fit is computed. Furthermore by looking at the Touschek lifetime for a forward power P=0 it is possible to determine the Touschek lifetime during normal operations.

$$\tau_{Tousch_{nominal}} = 13.7h \pm 0.2h \tag{33}$$

7.3.4 Conclusion on these three fits

Using the lifetime calculated with PCT reading is useful because the sampling rate of the beam current is way higher than the one on the lifetime channel. Consequently, we are confident that each values are independent of each others. However, the lifetime calculated with the beam current is fluctuating more which implies a higher error on every fit and every measurement. Which leads to values for gas scattering lifetime and touschek lifetime that are more fluctuating. Finally the way we change the gain affects more the results when the lifetime is calculated with beam current. Scanning the gain on logarithmic scale is probably a good way to reduce these fluctuation of lifetime during measurement.

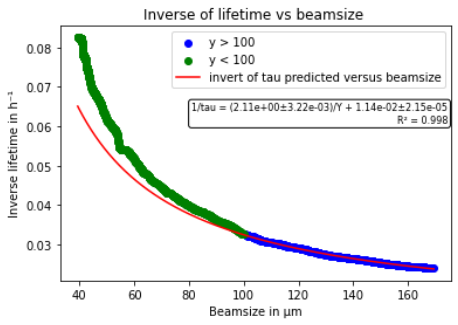
7.4 Using the three measurement merged for a fit

In this section, the data from the three experiments are merged to perform a linear regression to compute Touschek lifetime. The sections before showed that to compute with a fit more accurately the gas scattering lifetime, only the large beam sizes are important. Thus the gas scattering lifetime was computed by fitting the inverse lifetime with beam size for a beam size from 100μm to 140μm. After a test it shows that when the fit is done with the PCT reading the regression is not possible to compute. The way the gain was changed did affect to much the data when merged. Nevertheless for a fit on the lifetime given by the EPICS channel results where better than the the measurement separately.

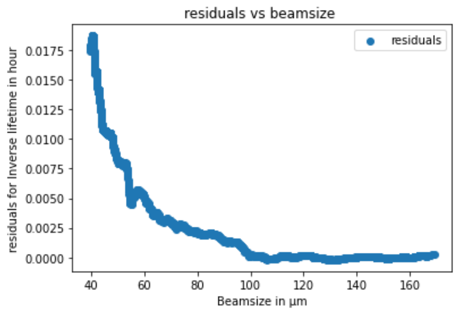
The fit for this measurement shows the following relationship:

$$\frac{1}{\tau} = 1.13 * 10^{-2} \pm 2.14 * 10^{-5} + \frac{2.113 \pm 3.22 * 10^{-3}}{Y} \tag{34}$$

With a R^2=0.998, a mean absolute error MAE= 5.74*10^{-5} and a mean squared error MSE=5.44*10^{-9}



(a) Inverse lifetime vs beam size



(b) Residuals of lifetime versus beam size

Figure 32: Fit of lifetime with respect to beam size

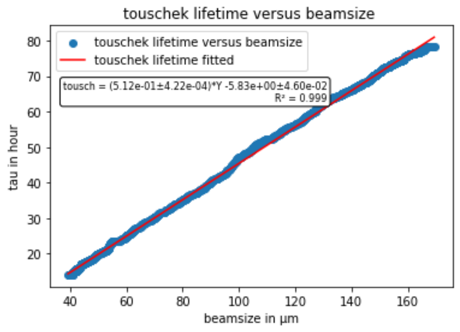
As the residuals fig. 32b shows it, ignoring small beam size does change the fit that approximate the data and thus the gas scattering lifetime. The MSE and MAE and the uncertainties on the fit are lower than on the previous fits. The gas scattering lifetime is then:

$$\tau_{gas} = \frac{1}{9.70 * 10^{-3}} = 88.1h \pm 0.3h \tag{35}$$

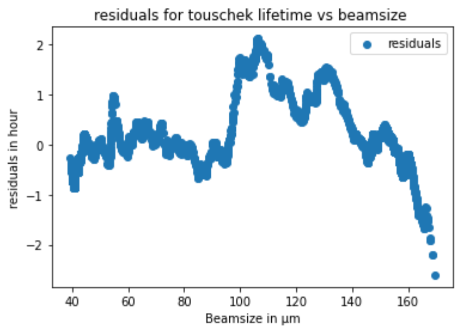
The gas scattering lifetime is given with a lower uncertainty which shows that looking only at large beam size improve the accuracy of the result. It is then possible to compute the Touschek lifetime using Eq.(8):

$$\tau_{tousch} = (5.12 * 10^{-1} \pm 4.22 * 10^{-4}) * Y - 5.83 \pm 4.60 * 10^{-2} \tag{36}$$

With a R^2=0.999, a mean absolute error MAE=0.452 and a mean squared error MSE=0.387



(a) Touschek lifetime vs beam size



(b) Residuals of Touschek lifetime versus beam size

Figure 33: Fit of Touschek lifetime with respect to beam size

As expected there is almost a linear relationship between the Touschek lifetime and the beam size fig. 33a. The residuals fig. 33b shows that the fit is well approximating the data, except for large beam size. The error on the fit is lower and R^2 is bigger. Furthermore by looking at the Touschek lifetime for a forward power P=0 it is possible to determine the Touschek lifetime during normal operations.

$$\tau_{Tousch_{nominal}} = 14.052h \pm 0.008h \tag{37}$$

To conclude, the gas scattering lifetime is then, when using the merged data 88 hours which is a significant change. To improve this measurement it is necessary to look at a larger beam size and compute all the fits on a short range around the maximum beam size measured. Moreover the Touschek lifetime remains the same around 14 hours for each measurements. It shows that that the Touschek lifetime is indeed around 14 hours at SLS during user operation. Fits on Touschek lifetime with respect to loss Touschek lifetime was done during the data analysis but in the case of nominal data, the test did not yield conclusive results. All the figure are presented in appendix.

8 Uncertainty on the fits

The bootstrap method was used to get the uncertainties on the intercepts and coefficients of the regression because the library used for the regression couldn't give directly these values. The core of the uncertainty calculation lies in the bootstrap resampling loop [4, 5]. This loop is designed to estimate the uncertainties associated with the regression coefficients and intercept. It involves several important steps.

During each iteration of the loop, a new training dataset is created by resampling the original training data with replacement. This means that observations are randomly selected from the original dataset, and some observations may be chosen multiple times while others may not be included at all. In our case we test bootstrapping with 1000, 10 000 and 1 000 000 iterations. What we can conclude is that the result was roughly the same after a number of iteration bigger than 10 000. Thus for script efficiency reasons we limited ourselves to 10 000 iterations

Subsequently, a linear regression model is fitted on the bootstrapped training dataset. This means that a regression line is calculated based on the relationship between the independent variable (CMOS1L11 in this case) and the dependent variable (tau). The coefficients of the regression line, as well as the intercept, are recorded for each iteration.

After all the iterations are completed, the standard deviations of the coefficient samples and intercept samples are computed. These standard deviations represent the uncertainties associated with the regression coefficients and intercept.

Then we use the formula of the propagation of uncertainties to have the uncertainty on the predicted lifetime. If we note the linear equation followed by our model $\tau = a \times \frac{L}{I} + b$ and there uncertainties with Δ The error on the fit is:

$$\Delta\tau_{predicted} = \tau_{predicted} \times \sqrt{(\frac{\Delta a}{a})^2 + (\frac{\Delta L}{L})^2 + (\frac{\Delta I}{I})^2} + \Delta b \qquad (38)$$

9 Conclusion

With these measurement we can conclude that:

- The fast loss monitor are better to compute lifetime and beam current during experiments and the slow ones during user operation
- The chromaticity may be not optimized at SLS and could be lowered to increase lifetime if we stay in the stable range of chromaticity
- The loss monitors and lifetime are less correlated during a change of tune and especially near the resonance. And the relationship between the two seems non-linear for a change of horizontal tune from 0.38 to 0.46
- It is possible to compute lifetime and beam current with loss monitors, however the experiment shows that loss monitors don't behave in the same way during normal operation and experiments
- To compute lifetime with a loss monitor more precisely it is necessary to normalize loss by beam current.
- Finally at SLS, the Touschek lifetime during normal operation is 14h and the gas scattering lifetime is 88h

Further steps to improve this study are:

- Repeat measurement of lifetime and loss while changing horizontal chromaticity at nominal beam current to find the optimized chromaticity of the machine
- Repeat measurement of lifetime versus beam size with a wider range of beam sizes to have a more precise value of gas scattering lifetime.

References

[1] L. Torino K.B. Scheidt, New beam loss detector system for EBS-ESRF, Grenoble, https://accelconf.web.cern.ch/ibic2018/papers/weob01.pdf

[2] Jan Chrin tutorial for pycafe, SLS, http://cafe.psi.ch/cython.html

[3] Wei-Meng Lee, Python Machine Learning April, 2019

[4] Ethan wicker, Bootstrap Resampling, CERN Feb 2021, https://ethanwicker.com/2021-02-23-bootstrap-resampling-001/

[5] Evaluating statistical uncertainties and correlations using the bootstrap method ,September 2021, http://cds.cern.ch/record/2759945/files/ATL-PHYS-PUB-2021-011.pdf?version=2

[6] B. Nash, F. Ewald, L. Farvacque, J. Jacob, E. Plouviez, J.L. Revol, K. Scheidt, Touschek lifetime and momentum acceptance measurements for ESRF, San Sebastian 2011 https://accelconf.web.cern.ch/ipac2011/papers/thpc008.pdf

10 Appendix

10.1 Figures of Tosuchek lifetime vs Loss

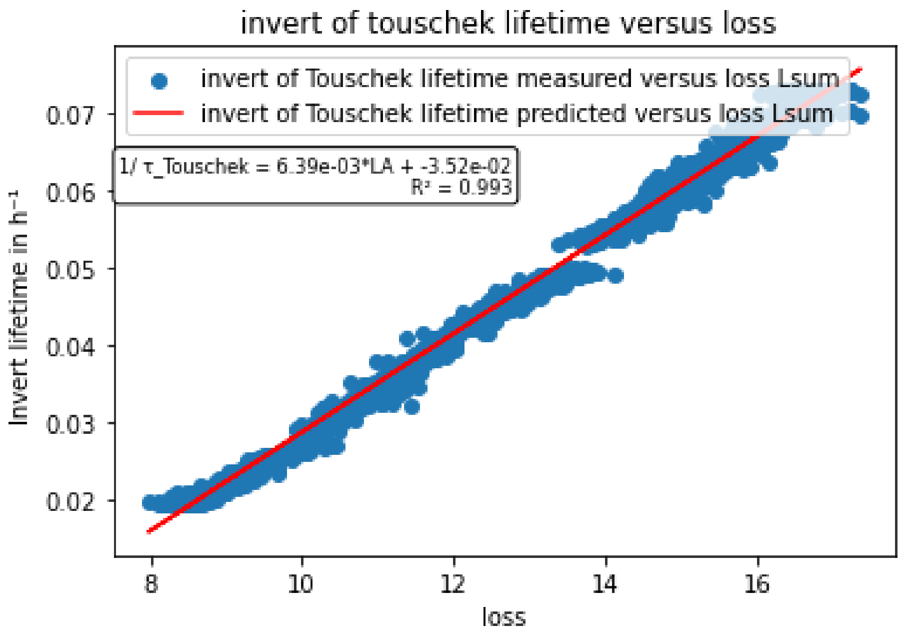


Figure 34: inverse of Touschek lifetime vs sum of fast loss

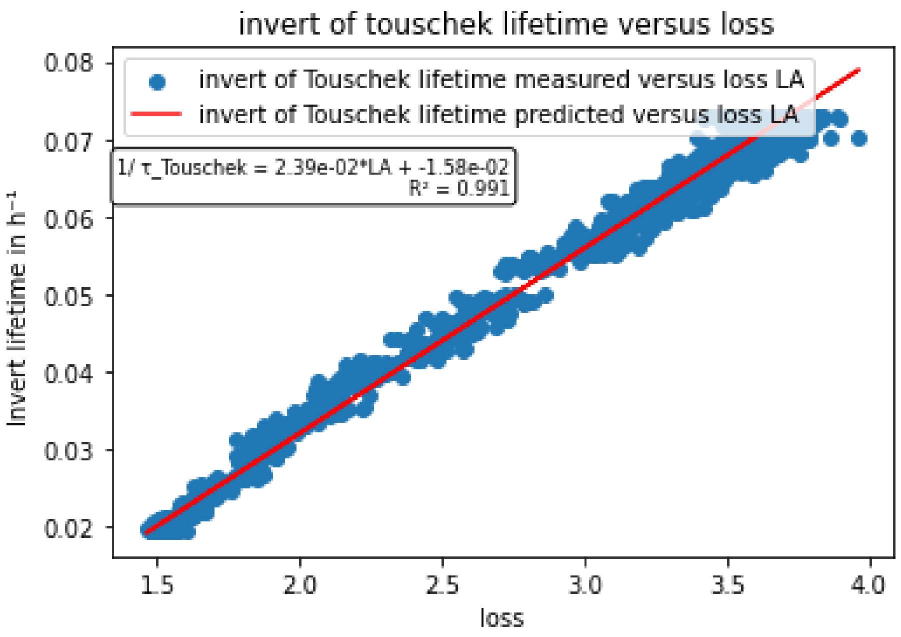


Figure 35: inverse of Touschek lifetime vs LA

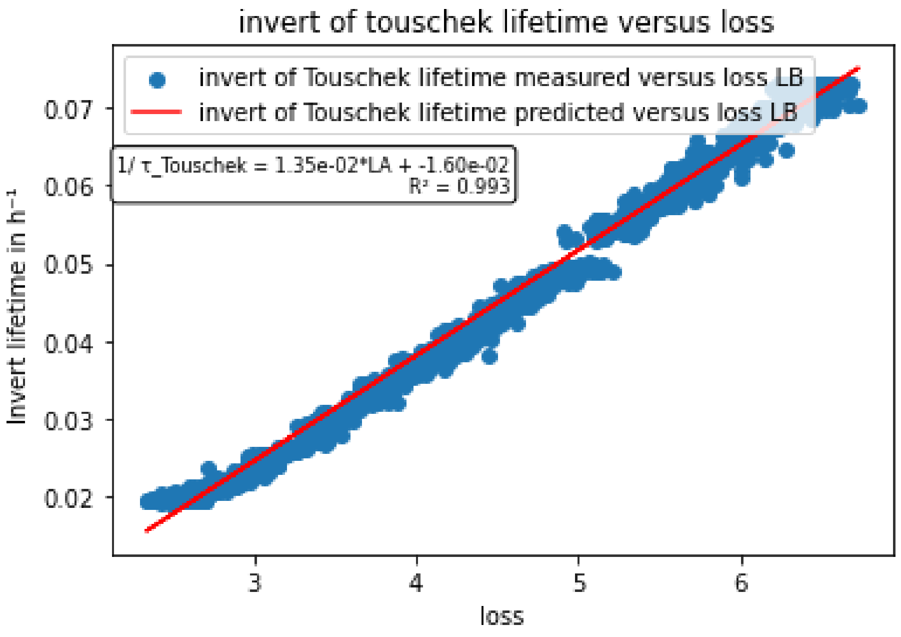


Figure 36: inverse of Touschek lifetime vs LB

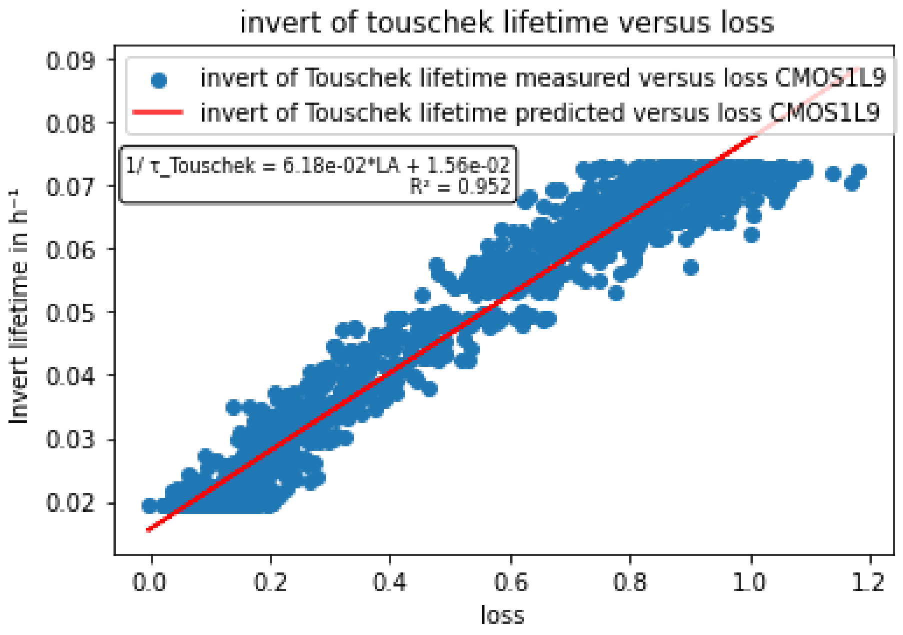


Figure 37: inverse of Touschek lifetime vs CMOS1L9

10.2 Scripts

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 15 16:15:54 2023
@author: elyama_y

This script processes the data collected to improve the
    results of a fit.
"""

import numpy as np
import os

# ==================================
# extract the data from files
# ==================================
# Define the list of file names to read
files_names = [
    "lifetime.txt", "dPCT.txt", "beam current.txt", "LA.txt"
        , "LB.txt", "LC.txt", "LD.txt",
    "CMOS1L1.txt", "CMOS1L2.txt", "CMOS1L3.txt", "CMOS1L4.
        txt", "CMOS1L5.txt", "CMOS1L6.txt",
    "CMOS1L7.txt", "CMOS1L8.txt", "CMOS1L9.txt", "CMOS1L10.
        txt", "CMOS1L11.txt", "CMOS2L1.txt",
    "CMOS2L2.txt", "CMOS2L3.txt", "CMOS2L4.txt", "CMOS2L5.
        txt", "CMOS2L6.txt", "CMOS2L7.txt",
    "CMOS2L8.txt", "CMOS3L1.txt", "CMOS3L2.txt", "CMOS3L3.
        txt", "CMOS3L4.txt", "CMOS3L5.txt",
    "CMOS3L6.txt", "CMOS3L7.txt", "CMOS4L1.txt", "CMOS4L2.
        txt", "CMOS4L3.txt", "CMOS4L4.txt",
    "CMOS4L5.txt", "CMOS4L6.txt", "CMOS4L7.txt", "CMOS4L8.
        txt", "CMOS4L9.txt", "CMOS4L10.txt",
    "CMOS4L11.txt", "tchroma_vs_lifetime.txt"
]  # files_names should always respect 2 rules for the
    program to work
# the last file should always be time
# put all the parameters that are not loss monitors at the
    beginning and then loss monitors

data_liste = [[] for _ in range(len(files_names))]
# Initialize the directory path
directory = "C:/Users/elyama_y/Desktop/Data/tmp"

# Get the folder name from user input
folder_name = input("folder name : ")

# Create the full directory path
directory = os.path.join(directory, folder_name)

# Read data from files and store in respective lists
for i, file_name in enumerate(files_names):
    chemin_fichier = os.path.join(directory, file_name)
    with open(chemin_fichier, "r") as fichier:
        contenu = fichier.readlines()
    data_liste[i].extend([float(ligne.strip()) for ligne in
        contenu])



# ===================================
# deleting injection
# ===================================

# Get the length of the LA list
n = len(data_liste[0])

# Generate a list of indices in reverse order
indices = list(range(n - 1, 0. -1))

# Filter indices to remove based on the sign of the
    derivative of beam current
index_dPCT = files_names.index("dPCT.txt")
```

```python
indices_to_remove = [i for i in indices if data_liste[
    index_dPCT] >= 0]


# remove values during injection
def remove_injec(A):
    for i in indices_to_remove:
        A.pop(i)


for i in data_liste:
    remove_injec(i)

# ==================================
# remove the same values of lifetime
# ==================================

index_tau = files_names.index("lifetime.txt")


def index_deleted(liste):
    unique_elements = []
    deleted_index = []

    for index, element in enumerate(liste):
        if element not in unique_elements:
            unique_elements.append(element)
        else:
            deleted_index.append(index)
    deleted_index.sort(reverse=True)
    return deleted_index


def delete_duplicate(A, B):
    for i in A:
        B.pop(i)


A = index_deleted(data_liste[index_tau])

for k in range(len(data_liste)):
    delete_duplicate(A, data_liste[k])

# ==================================
# Mean over p values
# ==================================

data_listeM = [[] for _ in range(len(data_liste))]
p = 4
```

```
def mean(p, A, B):
    if len(A) % p == 0:
        for i in range(0. len(A), p):
            B.append(np.mean(A[i:i + p]))
    else:
        for i in range(len(A) - 1, len(A) - len(A) % p - 1,
            -1):
            A.pop(i)
            print(len(A))
        for i in range(0. len(A), p):
            B.append(np.mean(A[i:i + p]))
    return B


for k in range(len(data_liste)):
    mean(p, data_liste[k], data_listeM[k])

index_LA = files_names.index("LA.txt")
index_t = files_names.index("tchroma_vs_lifetime.txt")
index_PCT = files_names.index("beam current.txt")

# ==================================
# normalize by beam current
# ==================================


def normalize(A, B):
    for i in range(len(A)):
        A[i] = A[i] / B[i]
    return A


for k in range(index_LA, index_t):
    normalize(data_listeM[k], data_listeM[index_PCT])

# ==================================
# rewrite the data on files
# ==================================

# Define the directory to save the files
directory = "C:/Users/elyama_y/Desktop/Data/tmp"

# Prompt the user for the folder name
folder_name = input("Folder name : ")

# Create the folder path
folder_path = os.path.join(directory, folder_name)

# Create the folder
```

```
os.makedirs(folder_path)

# Write the data arrays to individual text files
for i in range(len(files_names)):
    file_path = os.path.join(folder_path, files_names[i])
    with open(file_path, "w") as file:
        for item in data_listeM[i]:
            file.write(str(item) + "\n")
```

Listing 1: Data processing script

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 24 09:52:31 2023

@author: elyama_y

This code takes data every sample_period and save each
    sensor data on one file
"""


# ==================================
# Libraries
# ===================================
import time
import PyCafe
import os
import asyncio

cafe = PyCafe.CyCafe()
cyca = PyCafe.CyCa()

# ==================================
# Parameters
# ==================================
minute = 60
hour = 3600
total_time = 1 * hour + 30 * minute #Choose the duration of
    the datataking
sensors_names = ['ARIDI-TU2:AMP-FWDPWR', 'ARIDI-TU2:AMP-
    REVPWR', 'ARIDI-TU2:RFGAIN-GET', 'ARIMA-OPTIC:CY-NOM', '
    ARIMA-OPTIC:CX-NOM',"ARIMA-OPTIC:CX-SHIFT", "ARIMA-OPTIC:
    CY-SHIFT", 'ARIDI-PCT:TAU-HOUR', 'ARIDI-PCT:CUR-ROCS', '
    ARIDI-PCT:beam current','ARIDI-BLM10:SigSa.A'] # put the
    channel of the sensors you want
files_names = ["Fwd_power.txt", "Rev_power.txt", "Gain.txt",
     "Ver_Chroma.txt", "Hor_Chroma.txt", "Hor_Chromashift.txt
    ",
"Ver_Chromashift.txt", "lifetime.txt", "dPCT.txt", "beam
    current.txt", "LA.txt", "tchroma_vs_lifetime.txt"] #
    choose there names and add the name of the time variable
    put always the time at the end
directory = "/sls/bd/exchange/bd/home/elyama_y/tmp"
folder_name = input("folder name please:") # choose the name
     of the folder
directory = directory + "/" + folder_name

sample_period = 1
initial_time = time.time()
```

```
t = 0

# ==================================
# Coroutines
# ==================================


async def get_time(initial_time, directory, file_name):
    global t
    t = time.time() - initial_time
    await write_value(directory, file_name, t)


async def get_value(sensor_name, directory, file_name):
    value = cafe.get(sensor_name)
    await write_value(directory, file_name, value)


async def write_value(directory, file_name, value):
    if not os.path.exists(directory):
        os.mkdir(directory)
        print("The following directory was successfully
            created:", directory, "\n")
    file_path = directory + "/" + file_name
    if os.path.exists(file_path):
        with open(file_path, "a") as file:
            file.write(str(value) + "\n")
    else:
        with open(file_path, "w") as file:
            file.write(str(value) + "\n")
    file_path = os.path.join(directory, file_name)


async def main_task():
    tasks = []
    for n in range(len(sensors_names)):
        task = get_value(sensors_names[n], directory,
            files_names[n])
        tasks.append(task)
    await asyncio.gather(get_time(initial_time, directory,
        files_names[len(sensors_names)]), *tasks)


# ==================================
# Main()
# ==================================


async def main():
    global t
    while t < total_time:
        await asyncio.gather(main_task(), asyncio.sleep(
            sample_period))


if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    try:
        loop.run_until_complete(main())
    finally:
        loop.close()
```

Listing 2: Data taking script

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 24 10:11:04 2023

Author: elyama_y

This script permit to compute a fit on processed data
"""

import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error,
    mean_absolute_error
import numpy as np
import pandas as pd
from sklearn.linear_model import HuberRegressor
import os
from sklearn.linear_model import LinearRegression
from sklearn.utils import resample


# ==================================
# # parameters
# ==================================
Vtune = []
Htuneshift = []
Htune = []
tau = []
dPCT = []
PCT = []
t = []

# ==================================
# # Slow BLM
# ==================================
CMOS1L1 = []
CMOS1L2 = []
```

```
CMOS1L3 = []
CMOS1L4 = []
CMOS1L5 = []
CMOS1L6 = []
CMOS1L7 = []
CMOS1L8 = []
CMOS1L9 = []
CMOS1L10 = []
CMOS1L11 = []

CMOS2L1 = []
CMOS2L2 = []
CMOS2L3 = []
CMOS2L4 = []
CMOS2L5 = []
CMOS2L6 = []
CMOS2L7 = []
CMOS2L8 = []

CMOS3L1 = []
CMOS3L2 = []
CMOS3L3 = []
CMOS3L4 = []
CMOS3L5 = []
CMOS3L6 = []
CMOS3L7 = []

CMOS4L1 = []
CMOS4L2 = []
CMOS4L3 = []
CMOS4L4 = []
CMOS4L5 = []
CMOS4L6 = []
CMOS4L7 = []
CMOS4L8 = []
CMOS4L9 = []
CMOS4L10 = []
CMOS4L11 = []
CMOSsum = []

# ==================================
# # fast BLM
# ==================================
LA = []
LB = []
LC = []
LD = []
Lsum = []

files_names = ["lifetime.txt", "dPCT.txt", "beam current.txt
```

```
    ", "LA.txt", "LB.txt", "LC.txt", "LD.txt", "CMOS1L1.txt",
     "CMOS1L2.txt", "CMOS1L3.txt", "CMOS1L4.txt", "CMOS1L5.
    txt", "CMOS1L6.txt", "CMOS1L7.txt", "CMOS1L8.txt", "
    CMOS1L9.txt", "CMOS1L10.txt", "CMOS1L11.txt", "CMOS2L1.
    txt", "CMOS2L2.txt", "CMOS2L3.txt", "CMOS2L4.txt", "
    CMOS2L5.txt", "CMOS2L6.txt", "CMOS2L7.txt", "CMOS2L8.txt"
    , "CMOS3L1.txt", "CMOS3L2.txt", "CMOS3L3.txt", "CMOS3L4.
    txt", "CMOS3L5.txt", "CMOS3L6.txt", "CMOS3L7.txt", "
    CMOS4L1.txt", "CMOS4L2.txt", "CMOS4L3.txt", "CMOS4L4.txt"
    , "CMOS4L5.txt", "CMOS4L6.txt", "CMOS4L7.txt", "CMOS4L8.
    txt", "CMOS4L9.txt", "CMOS4L10.txt", "CMOS4L11.txt", "
    tchroma_vs_lifetime.txt"]
data_liste = [tau, dPCT, PCT, LA, LB, LC, LD, CMOS1L1,
    CMOS1L2, CMOS1L3, CMOS1L4, CMOS1L5, CMOS1L6, CMOS1L7,
    CMOS1L8, CMOS1L9, CMOS1L10. CMOS1L11, CMOS2L1, CMOS2L2,
    CMOS2L3, CMOS2L4, CMOS2L5, CMOS2L6, CMOS2L7, CMOS2L8,
    CMOS3L1, CMOS3L2, CMOS3L3, CMOS3L4, CMOS3L5, CMOS3L6,
    CMOS3L7, CMOS4L1, CMOS4L2, CMOS4L3, CMOS4L4, CMOS4L5,
    CMOS4L6, CMOS4L7, CMOS4L8, CMOS4L9, CMOS4L10. CMOS4L11, t
    ]

directory = "C:/Users/elyama_y/Desktop/Data/tmp"
folder_name = input("Folder name: ")
directory = os.path.join(directory, folder_name)

for i in range(len(files_names)):
    chemin_fichier = os.path.join(directory, files_names[i])
    fichier = open(chemin_fichier, "r")
    contenu = fichier.readlines()
    fichier.close()
    data_liste[i].extend([float(line.strip()) for line in
        contenu])


class Bunch(dict):
    def __init__(self, **kwargs):
        super().__init__(kwargs)
        self.__dict__ = self


# Calculate the sum of CMOS values for each index position
    using zip and list comprehension
CMOSsum = [sum(values) for values in zip(CMOS1L1, CMOS1L2,
    CMOS1L3, CMOS1L4, CMOS1L5, CMOS1L6, CMOS1L7, CMOS1L8,
    CMOS1L9, CMOS1L10. CMOS1L11, CMOS2L1, CMOS2L2, CMOS2L3,
    CMOS2L4, CMOS2L5, CMOS2L6, CMOS2L7, CMOS2L8, CMOS3L1,
    CMOS3L2, CMOS3L3, CMOS3L4, CMOS3L5, CMOS3L6, CMOS3L7,
    CMOS4L1, CMOS4L2, CMOS4L3, CMOS4L4, CMOS4L5, CMOS4L6,
    CMOS4L7, CMOS4L8, CMOS4L9, CMOS4L10. CMOS4L11)]
```

```
# Calculate the sum of L values for each index position
    using zip and list comprehension
Lsum = [La + Lb + Lc + Ld for La, Lb, Lc, Ld in zip(LA, LB,
    LC, LD)]

features = np.column_stack((LA, LB, LC, LD, CMOS1L1, CMOS1L2
    , CMOS1L3, CMOS1L4, CMOS1L5, CMOS1L6, CMOS1L7, CMOS1L8,
    CMOS1L9, CMOS1L10. CMOS1L11, CMOS2L1, CMOS2L2, CMOS2L3,
    CMOS2L4, CMOS2L5, CMOS2L6, CMOS2L7, CMOS2L8, CMOS3L1,
    CMOS3L2, CMOS3L3, CMOS3L4, CMOS3L5, CMOS3L6, CMOS3L7,
    CMOS4L1, CMOS4L2, CMOS4L3, CMOS4L4, CMOS4L5, CMOS4L6,
    CMOS4L7, CMOS4L8, CMOS4L9, CMOS4L10. CMOS4L11, CMOSsum,
    Lsum))

dataset = Bunch(data=features)
feature_names = ['LA', 'LB', 'LC', 'LD', 'CMOS1L1', 'CMOS1L2
    ', 'CMOS1L3', 'CMOS1L4', 'CMOS1L5', 'CMOS1L6', 'CMOS1L7',
     'CMOS1L8', 'CMOS1L9', 'CMOS1L10', 'CMOS1L11', 'CMOS2L1',
     'CMOS2L2', 'CMOS2L3', 'CMOS2L4', 'CMOS2L5', 'CMOS2L6', '
    CMOS2L7', 'CMOS2L8', 'CMOS3L1', 'CMOS3L2', 'CMOS3L3', '
    CMOS3L4', 'CMOS3L5', 'CMOS3L6', 'CMOS3L7', 'CMOS4L1', '
    CMOS4L2', 'CMOS4L3', 'CMOS4L4', 'CMOS4L5', 'CMOS4L6', '
    CMOS4L7', 'CMOS4L8', 'CMOS4L9', 'CMOS4L10', 'CMOS4L11', '
    CMOSsum', 'Lsum']
df = pd.DataFrame(dataset.data, columns=feature_names)
dataset.target = tau #choose the variable you want to fit

df['tau'] = dataset.target
df.head()

corr = df.corr()
print(corr)
print(df.corr().abs().nlargest(6, 'tau').index)

columns = ['LA'] # choose the variable you use to compute
    your fit
data = {'LA': df['LA']} # choose the variable you use to
    compute your fit
x = pd.DataFrame(data, columns=columns)
Y = df['tau']
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, Y,
    test_size=0.3, random_state=5)

n_iterations = 1000000  # Number of bootstrap iterations
x['intercept'] = 1
coef_samples = np.zeros((n_iterations, x.shape[1]))  # x
    represents your independent variables
y_pred_samples = np.zeros((n_iterations, len(y_test)))
intercept_samples = np.zeros(n_iterations)  # y_test
```

```
    represents your test dependent values

for i in range(n_iterations):
    # Bootstrap sampling of data
    x_boot, y_boot = resample(x_train, y_train)
    # Create a linear regression model and train it on the
        sampled data
    model = LinearRegression()
    model.fit(x_boot, y_boot)
    # Save the sampled regression coefficients
    coef_samples[i] = model.coef_
    # Make predictions on the sampled test data
    y_pred_samples[i] = model.predict(x_test)
    intercept_samples[i] = model.intercept_

# Calculate uncertainty on regression coefficients
coef_std = 2 * np.std(coef_samples, axis=0)
intercept_std = 2 * np.std(intercept_samples)
# Calculate error bars on predicted values
y_pred_mean = np.mean(y_pred_samples, axis=0)  # Mean of
    sampled predictions
y_pred_std = np.std(y_pred_samples, axis=0)  # Standard
    deviation of sampled predictions

# Calculate mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred_mean)
print("MSE:", mse)

# Calculate mean absolute error (MAE)
mae = mean_absolute_error(y_test, y_pred_mean)
print("MAE:", mae)

# Display regression coefficients with their uncertainty
for i, coef in enumerate(model.coef_):
    print("Coefficient", i, ":", coef, "  ", coef_std[i])
print("intercept:", model.intercept_, "  ", intercept_std)

tau_pred = model.predict(x_test)

print('R-squared:%4f' % model.score(x_test, y_test))

plt.figure(4)
plt.scatter(y_test, tau_pred)
plt.xlabel("Measured tau")
plt.ylabel('Predicted tau')
plt.title('Predicted tau vs Measured tau for linear
    regression on LA')

a = model.intercept_
b = model.coef_
```

```
Z = []

for i in range(len(LA)):
    k = a + b * LA[i]
    Z.append(k)

plt.figure(6)
plt.scatter(LA, tau, label='Measured tau versus LA loss')
plt.plot(LA, Z, label='Predicted tau versus LA loss')
plt.xlabel("Loss")
plt.ylabel('Tau in hours')
plt.legend()
plt.title('Comparison of linear regression and the measured
    value')
plt.show()
```

Listing 3: fitting script

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jun  2 14:56:20 2023

This script collects data from sensors, performs
    measurements, and fits the data
to compute Touschek and gas lifetimes.

Author: elyama_y
"""


# ===================================
# # libraries :
# ===================================
from sklearn.utils import resample
from sklearn.metrics import mean_squared_error,
    mean_absolute_error
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import numpy as np
import time
import PyCafe
from sklearn.linear_model import HuberRegressor
import os
cafe = PyCafe.CyCafe()
cyca = PyCafe.CyCa()
import asyncio
import pandas as pd
from sklearn.model_selection import train_test_split
# ===================================
# # parameters
# ==================================

minute=60
hour=3600
init_time = 25
N = 100
T=20
total_time = init_time + N*T+20

# Define gain values
gain = np.round(np.logspace(0,1.903,N)[::-1],1)[:74]


# List of sensor names and file names to store data
sensors_names = ['X09DA-FE-CCD1:Y-SIG-2','ARIDI-TU2:AMP-
    FWDPWR','ARIDI-TU2:AMP-REVPWR','ARIDI-TU2:RFGAIN-GET','
    ARIMA-OPTIC:CY-NOM','ARIMA-OPTIC:CX-NOM',"ARIMA-OPTIC:CX-
    SHIFT","ARIMA-OPTIC:CY-SHIFT",'ARIDI-PCT:TAU-HOUR','ARIDI
    -PCT:CUR-ROCS','ARIDI-PCT:CURRENT','ARIDI-BLM10:SigSa.A',
    'ARIDI-BLM10:SigSa.B','ARIDI-BLM10:SigSa.C','ARIDI-BLM10:
    SigSa.D','ARIDI-BLM01:LOSS1','ARIDI-BLM01:LOSS2','ARIDI-
    BLM01:LOSS3','ARIDI-BLM01:LOSS4','ARIDI-BLM01:LOSS5','
    ARIDI-BLM01:LOSS6','ARIDI-BLM01:LOSS7','ARIDI-BLM01:LOSS8
    ','ARIDI-BLM01:LOSS9','ARIDI-BLM01:LOSS10','ARIDI-BLM01:
    LOSS11','ARIDI-BLM02:LOSS1','ARIDI-BLM02:LOSS2','ARIDI-
    BLM02:LOSS3','ARIDI-BLM02:LOSS4','ARIDI-BLM02:LOSS5','
    ARIDI-BLM02:LOSS6','ARIDI-BLM02:LOSS7','ARIDI-BLM02:LOSS8
    ','ARIDI-BLM03:LOSS1','ARIDI-BLM03:LOSS2','ARIDI-BLM03:
    LOSS3','ARIDI-BLM03:LOSS4','ARIDI-BLM03:LOSS5','ARIDI-
    BLM03:LOSS6','ARIDI-BLM03:LOSS7','ARIDI-BLM04:LOSS1','
    ARIDI-BLM04:LOSS2','ARIDI-BLM04:LOSS3','ARIDI-BLM04:LOSS4
    ','ARIDI-BLM04:LOSS5','ARIDI-BLM04:LOSS6','ARIDI-BLM04:
    LOSS7','ARIDI-BLM04:LOSS8','ARIDI-BLM04:LOSS9','ARIDI-
    BLM04:LOSS10','ARIDI-BLM04:LOSS11']
files_names = ["Y_beamsize.txt","Fwd_power.txt","Rev_power.
    txt","Gain.txt","Ver_Chroma.txt", "Hor_Chroma.txt","
    Hor_Chromashift.txt","Ver_Chromashift.txt", "lifetime.txt
    ","dPCT.txt","current.txt","LA.txt", "LB.txt", "LC.txt",
    "LD.txt","CMOS1L1.txt", "CMOS1L2.txt", "CMOS1L3.txt", "
    CMOS1L4.txt","CMOS1L5.txt","CMOS1L6.txt","CMOS1L7.txt","
    CMOS1L8.txt","CMOS1L9.txt","CMOS1L10.txt","CMOS1L11.txt",
    "CMOS2L1.txt", "CMOS2L2.txt", "CMOS2L3.txt", "CMOS2L4.txt
    ", "CMOS2L5.txt", "CMOS2L6.txt", "CMOS2L7.txt", "CMOS2L8.
    txt","CMOS3L1.txt", "CMOS3L2.txt", "CMOS3L3.txt", "
    CMOS3L4.txt", "CMOS3L5.txt", "CMOS3L6.txt", "CMOS3L7.txt"
    ,"CMOS4L1.txt", "CMOS4L2.txt", "CMOS4L3.txt", "CMOS4L4.
    txt","CMOS4L5.txt","CMOS4L6.txt","CMOS4L7.txt","CMOS4L8.
    txt","CMOS4L9.txt","CMOS4L10.txt","CMOS4L11.txt","
    tchroma_vs_lifetime.txt"]
```

```
# Define the directory for data storage
directory = "/sls/bd/exchange/bd/home/elyama_y/tmp"
folder_name = input("folder name please :")
directory = directory +"/" + folder_name

# Set the sample period and initialize time
sample_period=1.2
initial_time = time.time()
t = 0




# ===================================
# # coroutines
# ===================================

# Function to get the current time and write it to a file
async def get_time(initial_time, directory, file_name):
    global t
    t = time.time() - initial_time
    await write_value(
                    directory,
                    file_name,
                    t
                    )

# Function to get sensor values and write them to a file
async def get_value(sensor_name, directory, file_name):
    value = cafe.get(sensor_name)
    await write_value(
                    directory,
                    file_name,
                    value
                    )

# Function to write a value to a file
async def write_value(directory, file_name, value):
    if not os.path.exists(directory) :
        os.mkdir(directory)
        print("The following directory was successly created
             : ",
              directory,
              "\n"
              )
    file_path = directory + "/" + file_name
    if os.path.exists(file_path):
        with open(file_path, "a") as file:
            file.write(str(value) + "\n")
    else:
        with open(file_path, "w") as file:
            file.write(str(value) + "\n")
    file_path = os.path.join(directory, file_name)


# Function to set the gain for the sensors
async def set_gain(i, T):
    cafe.set('ARIDI-TU2:RFGAIN-SET',i)
    await asyncio.sleep(T)

# Function to perform measurements using coroutines
async def do_measurements():
    tasks = []
    for n in range(len(sensors_names)):
        task = get_value(sensors_names[n], directory,
            files_names[n])
        tasks.append(task)
    await asyncio.gather(get_time(initial_time,
                                  directory,
                                  files_names[len(
                                      sensors_names)]),
                         *tasks)

# Coroutines to handle sensor measurements and gain setting
async def sensors_task():
    global t
    while t < total_time:
        await asyncio.gather(do_measurements(), asyncio.
            sleep(sample_period))


async def gain_task():
    await asyncio.sleep(init_time)
    for i in gain:
        await set_gain(i,T)


# ==================================
# # main()
# ==================================


async def main():

    await asyncio.gather(sensors_task(),
                         gain_task())
```

```
if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    try:
        loop.run_until_complete(main())
    finally:
        loop.close()



# ==================================
# # Analysis for the touschek lifetime  Load data from files
     into lists
# ==================================



# ==================================
# # parameters
# ==================================
tau = []
dPCT = []
PCT = []
t = []
y=[]
fwd_pow=[]
# ==================================
# # Slow BLM
# ==================================
CMOS1L1 = []
CMOS1L2 = []
CMOS1L3 = []
CMOS1L4 = []
CMOS1L5 = []
CMOS1L6 = []
CMOS1L7 = []
CMOS1L8 = []
CMOS1L9 = []
CMOS1L10 = []
CMOS1L11 = []

CMOS2L1 = []
CMOS2L2 = []
CMOS2L3 = []
CMOS2L4 = []
CMOS2L5 = []
CMOS2L6 = []
CMOS2L7 = []
CMOS2L8 = []

CMOS3L1 = []
```

```
CMOS3L2 = []
CMOS3L3 = []
CMOS3L4 = []
CMOS3L5 = []
CMOS3L6 = []
CMOS3L7 = []

CMOS4L1 = []
CMOS4L2 = []
CMOS4L3 = []
CMOS4L4 = []
CMOS4L5 = []
CMOS4L6 = []
CMOS4L7 = []
CMOS4L8 = []
CMOS4L9 = []
CMOS4L10 = []
CMOS4L11 = []
CMOSsum=[]
# ==================================
# # fast BLM
# ==================================
LA=[]
LB=[]
LC=[]
LD=[]
Lsum=[]

files_names = ["Fwd_power.txt", "Y_beamsize.txt","lifetime.
    txt","dPCT.txt" ,"current.txt","LA.txt", "LB.txt", "LC.
    txt", "LD.txt","CMOS1L1.txt", "CMOS1L2.txt", "CMOS1L3.txt
    ", "CMOS1L4.txt","CMOS1L5.txt","CMOS1L6.txt","CMOS1L7.txt
    ","CMOS1L8.txt","CMOS1L9.txt","CMOS1L10.txt","CMOS1L11.
    txt","CMOS2L1.txt", "CMOS2L2.txt", "CMOS2L3.txt", "
    CMOS2L4.txt", "CMOS2L5.txt", "CMOS2L6.txt", "CMOS2L7.txt"
    , "CMOS2L8.txt","CMOS3L1.txt", "CMOS3L2.txt", "CMOS3L3.
    txt", "CMOS3L4.txt", "CMOS3L5.txt", "CMOS3L6.txt", "
    CMOS3L7.txt","CMOS4L1.txt", "CMOS4L2.txt", "CMOS4L3.txt",
     "CMOS4L4.txt","CMOS4L5.txt","CMOS4L6.txt","CMOS4L7.txt",
    "CMOS4L8.txt","CMOS4L9.txt","CMOS4L10.txt","CMOS4L11.txt"
    ,"tchroma_vs_lifetime.txt"]
data_liste = [fwd_pow,y, tau, dPCT ,PCT,LA,LB,LC,LD,CMOS1L1,
    CMOS1L2,CMOS1L3,CMOS1L4,CMOS1L5,CMOS1L6,CMOS1L7,CMOS1L8,
    CMOS1L9,CMOS1L10,CMOS1L11,CMOS2L1,CMOS2L2,CMOS2L3,CMOS2L4
    ,CMOS2L5,CMOS2L6,CMOS2L7,CMOS2L8,CMOS3L1,CMOS3L2,CMOS3L3,
    CMOS3L4,CMOS3L5,CMOS3L6,CMOS3L7,CMOS4L1,CMOS4L2,CMOS4L3,
    CMOS4L4,CMOS4L5,CMOS4L6,CMOS4L7,CMOS4L8,CMOS4L9,CMOS4L10,
    CMOS4L11,t]

# Loop through the files and extract data
```

```
for i in range(len(files_names)):
    chemin_fichier = os.path.join(directory, files_names[i])
    fichier = open(chemin_fichier, "r")
    contenu = fichier.readlines()
    fichier.close()
    data_liste[i].extend([float(ligne.strip()) for ligne in
        contenu])



class Bunch(dict):
    def __init__(self, **kwargs):
        super().__init__(kwargs)
        self.__dict__ = self

tauinv=[]
taupctinv=[]
yinv=[]

# Get the length of the LA list
n = len(LA)

# Generate a list of indices in reverse order
indices = list(range(n-1, 0, -1))

# Filter indices to remove based on a condition
indices_to_remove = [i for i in indices if dPCT[i] >= 0]

# Remove corresponding elements from all lists during
    injection
for i in indices_to_remove:
    LA.pop(i)
    LB.pop(i)
    LC.pop(i)
    LD.pop(i)
    tau.pop(i)
    t.pop(i)
    dPCT.pop(i)
    PCT.pop(i)
    CMOS1L1.pop(i)
    CMOS1L2.pop(i)
    CMOS1L3.pop(i)
    CMOS1L4.pop(i)
    CMOS1L5.pop(i)
    CMOS1L6.pop(i)
    CMOS1L7.pop(i)
    CMOS1L8.pop(i)
    CMOS1L9.pop(i)
    CMOS1L10.pop(i)
    CMOS1L11.pop(i)
    CMOS2L1.pop(i)
    CMOS2L2.pop(i)
    CMOS2L3.pop(i)
    CMOS2L4.pop(i)
    CMOS2L5.pop(i)
    CMOS2L6.pop(i)
    CMOS2L7.pop(i)
    CMOS2L8.pop(i)
    CMOS3L1.pop(i)
    CMOS3L2.pop(i)
    CMOS3L3.pop(i)
    CMOS3L4.pop(i)
    CMOS3L5.pop(i)
    CMOS3L6.pop(i)
    CMOS3L7.pop(i)
    CMOS4L1.pop(i)
    CMOS4L2.pop(i)
    CMOS4L3.pop(i)
    CMOS4L4.pop(i)
    CMOS4L5.pop(i)
    CMOS4L6.pop(i)
    CMOS4L7.pop(i)
    CMOS4L8.pop(i)
    CMOS4L9.pop(i)
    CMOS4L10.pop(i)
    CMOS4L11.pop(i)
    y.pop(i)
    fwd_pow.pop(i)


taupct=[]
for i in range(len(t)):
    tauinv.append(1/tau[i])
    k=(-PCT[i]/dPCT[i])/60
    taupct.append(1/k)
    yinv.append(1/y[i])

tauinv_copy = tauinv.copy()
taupct_copy = taupct.copy()
y_copy = y.copy()
yinv_copy = yinv.copy()
for i in range(len(y)-1,-1,-1):
    if y[i] <=80:
        yinv_copy.pop(i)
        y_copy.pop(i)
        taupct_copy.pop(i)
        tauinv_copy.pop(i)


features = yinv_copy
```

```
dataset = Bunch(data=features)
feature_names = ['y']
df = pd.DataFrame(dataset.data, columns=feature_names)
dataset.target=tauinv_copy

df['tau'] = dataset.target
df.head()

corr=df.corr()
print(corr)
print(df.corr().abs().nlargest(6,'tau').index)

columns = ['y']
data = {'y':df['y']}
x = pd.DataFrame(data, columns=columns)
Y=df['tau']
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,Y,test_size
    =0.3,random_state=5)




n_iterations = 10000  # Number of bootstrap iterations
x['intercept'] = 1
coef_samples = np.zeros((n_iterations, x.shape[1]))  # x
    represents your independent variables
y_pred_samples = np.zeros((n_iterations, len(y_test)))
intercept_samples = np.zeros(n_iterations)  # y_test
    represents your dependent test values

p = 1000
gas = []

touschek = []
for i in range(n_iterations):
    tauousch = []
    # Bootstrap sampling of the data
    x_boot, y_boot = resample(x_train, y_train)
    # Create a linear regression model and train it on the
        bootstrapped data
    model = LinearRegression()

    model.fit(x_boot, y_boot)
    # Save the sampled regression coefficients
    coef_samples[i] = model.coef_
    gas.append(1/model.intercept_)
    for j in range(len(tau)):
        k = tauinv[j] - 1/gas[i]
        tauousch.append(1/k)
    touschek.append(tauousch[len(tauousch) - 1])
    # Make predictions on the bootstrapped test data
    y_pred_samples[i] = model.predict(x_test)
    intercept_samples[i] = model.intercept_

tau_pred = model.predict(x_test)
g = np.mean(gas)
gstd = np.std(gas)
tou = np.mean(touschek)
toustd = np.std(touschek)
print("gas lifetime:", g, "  ", 2*gstd)
print("touschek lifetime:", tou, "  ", 2*toustd)

# Calculating uncertainty on regression coefficients
coef_std = np.std(coef_samples, axis=0)
intercept_std = np.std(intercept_samples)
# Calculating error bars on predicted values
y_pred_mean = np.mean(y_pred_samples, axis=0)  # Mean of
    sampled predictions
y_pred_std = np.std(y_pred_samples, axis=0)  # Standard
    deviation of sampled predictions

# Calculating Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred_mean)
print("MSE:", mse)

# Calculating Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred_mean)
print("MAE:", mae)

# Displaying regression coefficients with their uncertainty
for i, coef in enumerate(model.coef_):
    print("Coeff", i, ": ", coef, "  ", coef_std[i])
print("intercept :", model.intercept_, "  ", intercept_std)

print('R-squared:', model.score(x_test, y_test))

a = model.intercept_
b = model.coef_

Z = []
Delta = []

for i in range(len(tauinv)):
    k = a + b * yinv[i]
    Delta.append(taupct[i] - k)
    Z.append(k)


tautousch = []
```

```
for i in range(len(tauinv)):
    k = tauinv[i] - a
    tautousch.append(1 / k)

# Removing data points with tautousch > 700
for i in range(len(tautousch)-1,-1,-1):
    if tautousch[i] >= 700:
        LA.pop(i)
        LB.pop(i)
        LC.pop(i)
        LD.pop(i)
        tautousch.pop(i)
        CMOS1L1.pop(i)
        CMOS1L2.pop(i)
        CMOS1L3.pop(i)
        CMOS1L4.pop(i)
        CMOS1L5.pop(i)
        CMOS1L6.pop(i)
        CMOS1L7.pop(i)
        CMOS1L8.pop(i)
        CMOS1L9.pop(i)
        CMOS1L10.pop(i)
        CMOS1L11.pop(i)
        CMOS2L1.pop(i)
        CMOS2L2.pop(i)
        CMOS2L3.pop(i)
        CMOS2L4.pop(i)
        CMOS2L5.pop(i)
        CMOS2L6.pop(i)
        CMOS2L7.pop(i)
        CMOS2L8.pop(i)
        CMOS3L1.pop(i)
        CMOS3L2.pop(i)
        CMOS3L3.pop(i)
        CMOS3L4.pop(i)
        CMOS3L5.pop(i)
        CMOS3L6.pop(i)
        CMOS3L7.pop(i)
        CMOS4L1.pop(i)
        CMOS4L2.pop(i)
        CMOS4L3.pop(i)
        CMOS4L4.pop(i)
        CMOS4L5.pop(i)
        CMOS4L6.pop(i)
        CMOS4L7.pop(i)
        CMOS4L8.pop(i)
        CMOS4L9.pop(i)
        CMOS4L10.pop(i)
        CMOS4L11.pop(i)
        Delta.pop(i)
        y.pop(i)
        tauinv.pop(i)
        Z.pop(i)
        yinv.pop(i)
        fwd_pow.pop(i)




# Create lists for values where y > 100 and y < 100
y_sup_60 = [y_val for y_val in y if y_val > 100]
tauinv_sup_60 = [tauinv[i] for i, y_val in enumerate(y) if
    y_val > 100]

y_inf_40 = [y_val for y_val in y if y_val < 100]
tauinv_inf_40 = [tauinv[i] for i, y_val in enumerate(y) if
    y_val < 100]

# Generate the equation and coefficient of determination
    label
equation = f'1/tau = ({b.item():.2e}  {coef_std[0].item():.2
    e})/Y + {a.item():.2e}  {intercept_std.item():.2e}'
m = model.score(x_test, y_test)
label = f'{equation}\nR  = {m:.3f}'

# Plot the graph using different colors for values where y >
     60 and y < 40
fig, ax = plt.subplots()
plt.scatter(y_sup_60, tauinv_sup_60, color='blue', label='y
    > 100')
plt.scatter(y_inf_40, tauinv_inf_40, color='green', label='y
     < 100')
plt.plot(y, Z, 'r-', label='invert of tau predicted versus
    beamsize')
plt.text(0.99, 0.7, label, ha='right', va='top', fontsize=8,
     transform=ax.transAxes, bbox=dict(facecolor='white',
    edgecolor='black', boxstyle='round'))
plt.xlabel("Beamsize in  m ")
plt.ylabel("Inverse lifetime in   h   ")
plt.legend()
plt.title('Inverse of lifetime vs beamsize ')

# Generate the equation using the variables
plt.figure(20)
plt.scatter(y, Delta, label='residuals')
plt.xlabel("Beamsize in  m ")
plt.ylabel('residuals for Inverse lifetime in hour')
plt.legend()
plt.title('residuals vs beamsize ')
```

```
# Extract 'y' values as features and 'tautousch' as the
    target
features = y
dataset = Bunch(data=features)
feature_names = ['y']
df = pd.DataFrame(dataset.data, columns=feature_names)
dataset.target = tautousch
df['tau'] = dataset.target
df.head()

# Calculate the correlation matrix
corr = df.corr()
print(corr)
print(df.corr().abs().nlargest(6, 'tau').index)

# Define columns and create dataframes 'x' and 'Y' for train
    -test split
columns = ['y']
data = {'y': df['y']}
x = pd.DataFrame(data, columns=columns)
Y = df['tau']

# Perform train-test split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, Y,
    test_size=0.3, random_state=5)

n_iterations = 10000  # Number of bootstrap iterations
x['intercept'] = 1
coef_samples = np.zeros((n_iterations, x.shape[1]))  # x
    represents your independent variables
y_pred_samples = np.zeros((n_iterations, len(y_test)))
intercept_samples = np.zeros(n_iterations)  # y_test
    represents your test dependent values

touschek = []
for i in range(n_iterations):
    # Bootstrap sampling of the data
    x_boot, y_boot = resample(x_train, y_train)
    # Create a linear regression model and train it on the
        bootstrapped data
    model = LinearRegression()

    model.fit(x_boot, y_boot)
    # Save the sampled regression coefficients
    coef_samples[i] = model.coef_
    # Make predictions on the bootstrapped test data
    y_pred_samples[i] = model.predict(x_test)
    intercept_samples[i] = model.intercept_
```

```
# Calculate uncertainty on regression coefficients
coef_std = 2 * np.std(coef_samples, axis=0)
intercept_std = 2 * np.std(intercept_samples)
# Calculate error bars on predicted values
y_pred_mean = np.mean(y_pred_samples, axis=0)  # Mean of
    bootstrapped predictions
y_pred_std = np.std(y_pred_samples, axis=0)  # Standard
    deviation of bootstrapped predictions

# Calculate Mean Squared Error (MSE) and Mean Absolute Error
     (MAE)
mse = mean_squared_error(y_test, y_pred_mean)
print("MSE:", mse)
mae = mean_absolute_error(y_test, y_pred_mean)
print("MAE:", mae)

# Display regression coefficients with their uncertainty
for i, coef in enumerate(model.coef_):
    print("Coeff", i, ":", coef, "  ", coef_std[i])
print("intercept:", model.intercept_, "  ", intercept_std)

print("R-squared:", model.score(x_test, y_test))

a = model.intercept_
b = model.coef_

Z2 = []
Delta2 = []

for i in range(len(tautousch)):
    k = a + b * y[i]
    Delta2.append(tautousch[i] - k)
    Z2.append(k)

# Generate the equation and coefficient of determination
    label for touschek lifetime
equation = f'tousch = ({model.coef_.item():.2e}  {coef_std
    [0].item():.2e})*Y {model.intercept_.item():.2e}  {
    intercept_std.item():.2e}'
m = model.score(x_test, y_test)
label2 = f'{equation}\nR  = {m:.3f}'

# Plot touschek lifetime versus beamsize
fig, ax1 = plt.subplots()
plt.figure(8)
plt.scatter(y, tautousch, label='touschek lifetime versus
    beamsize')
plt.plot(y, Z2, color="red", label='touschek lifetime fitted
    ')
plt.text(0.65, 0.8, label2, ha='right', va='top', fontsize
    =8, transform=ax1.transAxes, bbox=dict(facecolor='white',
     edgecolor='black', boxstyle='round'))
plt.xlabel("Beamsize in  m ")
plt.ylabel('tau in hour')
plt.legend()
plt.title('touschek lifetime versus beamsize ')
plt.show()

# Plot residuals for touschek lifetime versus beamsize
plt.figure(21)
plt.scatter(y, Delta2, label='residuals')
plt.xlabel("Beamsize in  m ")
plt.ylabel('residuals in hour')
plt.legend()
plt.title('residuals for touschek lifetime vs beamsize ')
plt.show()
```

Listing 4: Touschek lifetime measurement script