NTNU
Kunnskap for en bedre verden

TPK 4450 - Data Driven Prognostics and Predictive Maintenance

Semester work - Part 2

Author:
Krishnan Gopakumar Nair

October, 2020

Table of Contents

All the python script has been attached as screenshots in appendix. The GitHub link to the code is : `https://github.com/krishngo/TPK-4450---Data-Driven-Prognostic-Maintenance.git`.

Problem 1

From the candidates Gamma process could be removed since to be a Gamma proces, it has to have non-negative independent increments and based on the observations that is not the case.

1 Part A

Problem 2

Method of least squares can be used to estimate the optimal parameter value and it is done by minimizing the sum of squared residuals, i.e., errors.

For a linear regression model, we have,

$$Y(t) = X(t) + \varepsilon(t)$$

Y(t) is the observed condition at time t
X(t) is the actual deterioration
ε(t) is the random error

For a linear regression model, $X(t) = c + at$. From the question, it is same to consider that c = 0 since it is given that $X(t) = at$.

Taking partial derivatives and setting them to 0, we get,

$$a = \frac{\sum_{i=1}^{n} t_i Y_i}{\sum_{i=1}^{n} t_i^2}$$

The value has been found using python and is obtained as **a = 0.101**. The plot of X(t) is as below.

Comment: Inorder to use the LSE method accurately, the task was also completed without considering c=0. It is attached as **Appendix A**. The python code screenshot for both the solutions is also attached.

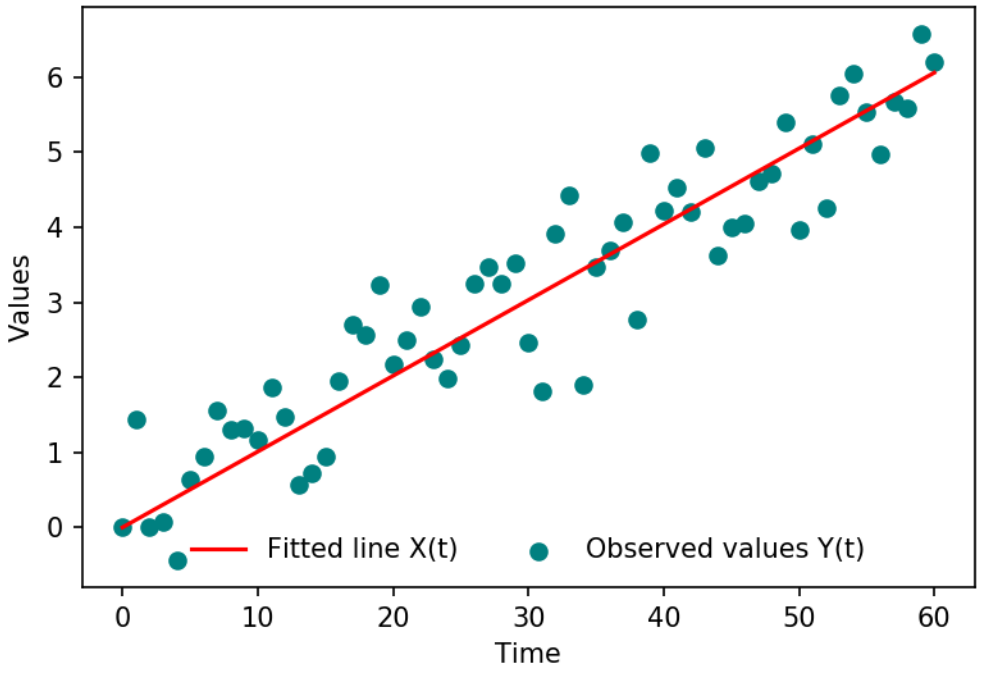


Figure 1: Plot of X(t) alongside Y(t)

Problem 3

We have the error, $\varepsilon(t) = Y(t) - X(t)$, and that it follows normal distribution. In-order to determine the parameters of the error distribution we use the methods of likelihood estimation. Since it follows a normal distribution, we have to find μ and σ. The probability density function of normal distribution is given by,

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}(\frac{x-\mu}{\sigma})^2} \tag{1}$$

The likelihood function is given by.

$$L(\mu_i, \sigma | Y_i, ..., Y_N) = f(Y_1|\mu_1, \sigma) \times .... \times f(Y_N|\mu_N, \sigma)$$

For the error distribution, $\mu_N = \hat{Y_N}$. Therefore,

$$L(\mu_i, \sigma | Y_1, ..., Y_N) = f(Y_1|\hat{Y_1}, \sigma) \times .... \times f(Y_N|\hat{Y_N}, \sigma)$$

From (1) we get,

$$L(\mu_i,\sigma|Y_1,...,Y_N) = \frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-1}{2}(\frac{Y_1-\hat{Y_1}}{\sigma})^2} \times .... \times \frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-1}{2}(\frac{Y_N-\hat{Y_N}}{\sigma})^2} \times$$
$$L(\mu_i,\sigma|Y_1,...,Y_N) = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N e^{\frac{-1}{2}\sum\limits_{i=1}^{N}(\frac{Y_i-\hat{Y_i}}{\sigma})^2}$$

Taking ln values, we get

$$ln(L) = N \times ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2}\sum_{i=1}^{N}(\frac{Y_i-\hat{Y}_i}{\sigma})^2$$

Taking partial derivative with respect to $\hat{Y_i}$ and equating it to 0, we obtain

$$\frac{\partial ln(L)}{\partial \hat{Y_i}} = \frac{1}{2\sigma^2} \times 2 \times \sum_{i=1}^{N}(Y_i - \hat{Y_i}) = 0 \tag{2}$$

Since all errors has the same distribution with the same μ and σ, we have $\sum\limits_{i=1}^{N}\hat{Y_i} = \mathrm{N}\hat{Y_i}$ Therefore (2) becomes,

$$\hat{Y_i} = \frac{\sum\limits_{i=1}^{n}(Y_i)}{N} \tag{3}$$

This is the estimate for μ_L of the error distribution.

Now, taking partial derivative with respect to σ and solving we get

$$\sigma = \frac{\sum\limits_{i=1}^{N}(Y_i - \hat{Y_i})^2}{N} \tag{4}$$

which is the estimate for σ_L for the error distribution.

In (3) and (4) $Y_i = \varepsilon_i(t) = Y_i(t) - X_i(t)$

Finding the value of μ_L and σ_L using python script, we get, μ_L = **0.075** and σ_L = **0.647**. The code is attached as **Appendix B**

Problem 4

A total of 10^4 trajectories has been simulated using python with the parameters ($\hat{a}$=0.101,$\hat{\mu_L}$ = 0.075, $\hat{\sigma_L}$ = 0.647). The plot of the trajectories, though a bit messy, has been attached below.

The code is attached as **Appendix C**

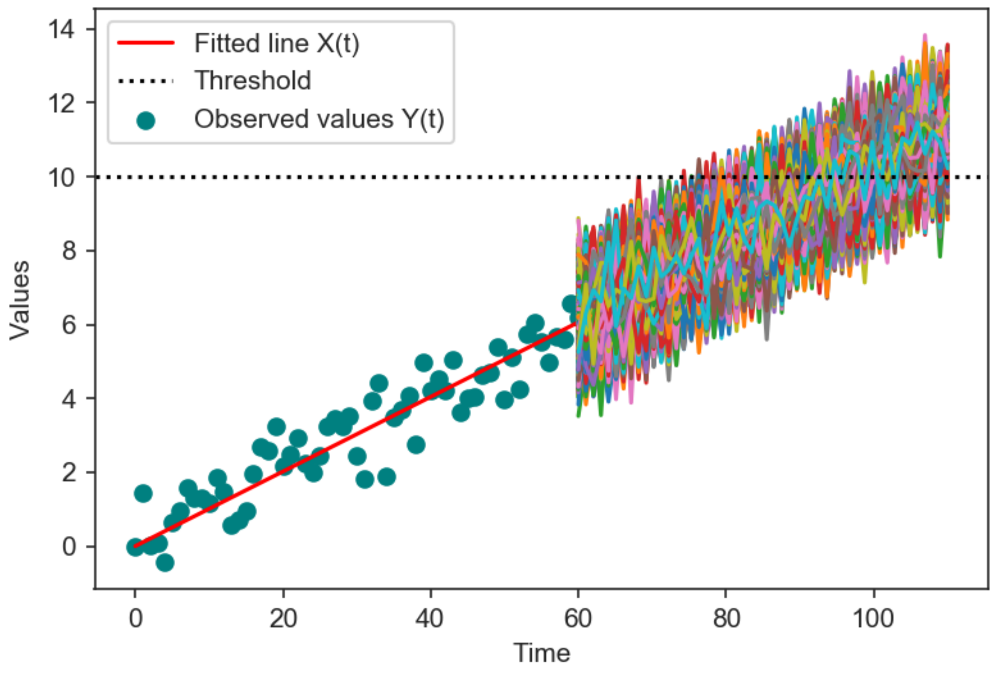


Figure 2: The simulated trajectories

Problem 5

Threshold was set for $y(t_j) > 10$ and then the remaining useful life was estimated. The histogram plot obtained is as below.

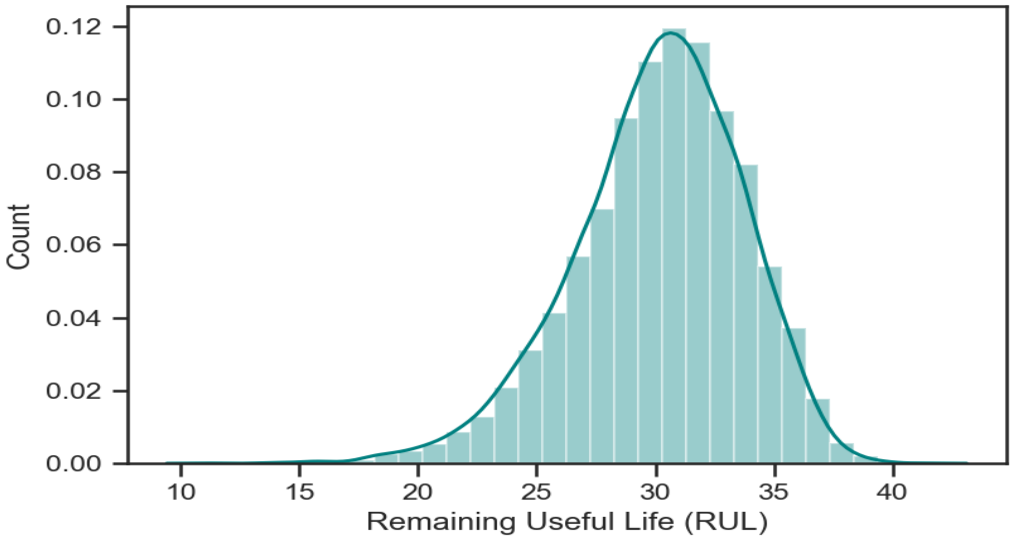


Figure 3: The histogram for RUL

The code is attached as **Appendix D**.

2 Part B

Problem 6

Now, we consider a Wiener process with a linear drift. Increment I = Y(t+Δt) - Y(t) and for the given set of observations, Δt = 1. The distribution of the increments of a Wiener process with drift follows a **normal distribution**. The dataset is imported into python environment and the parameters ($\hat{\mu_W}$ and $\hat{\sigma_W}$) was found. The value obtained are: $\hat{\mu_W}$ **= 0.103** and $\hat{\sigma_W}$ **= 0.887**. The code is attached as **Appendix E**

Problem 7

Similiar to problem 4, 10^4 trajectories have been simulated over a time horizon of (60,500) with Δt = 1. The plot of the trajectories is as below.

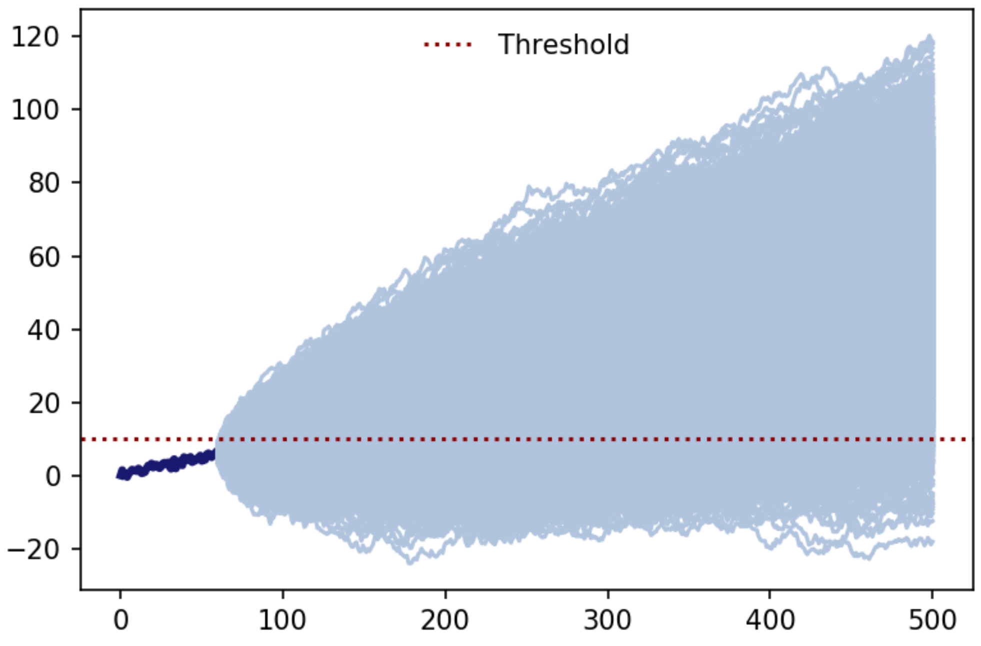


Figure 4: The simulated trajectories when considering increments

The code has been attached as **Appendix F**

Problem 8

Repeating the same logic for Problem 5, the histogram obtained is as below.

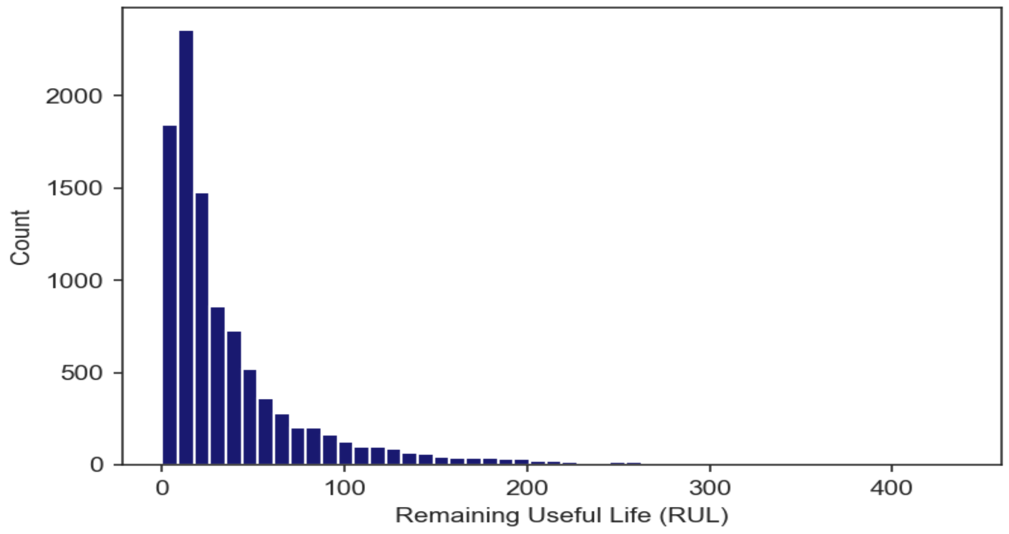


Figure 5: The histogram for RUL with increments

The code has been attached as **Appendix G**

Problem 9

Based on the hints given, the pdf of a Wiener process with linear drift is Inverse Gaussian distributed is,

$$f_T(t;\mu,\lambda) = \sqrt{\frac{\lambda}{2\pi t^3}} e^{\left(\frac{\lambda}{2\mu^2}\right)\left[\frac{(t-\mu)^2}{t}\right]}, \text{for t>0, } \mu > 0,\ \lambda > 0 \tag{5}$$

Then the PDF of the RUL(t_j) becomes:

$$f_{RUL}(t_j) = f(h;\mu_1,\lambda_1)$$

where the estimates of μ_1 and λ_1 is given by,

$$\hat{\mu_1} = \frac{L - y(t_j)}{\hat{\mu_W}} \quad and \quad \hat{\lambda_1} = \left(\frac{L - y(t_j)}{\hat{\sigma_W}}\right)^2 \quad , for\ \Delta t = 1$$

The value for $\hat{\mu_1}$ and $\hat{\lambda_1}$ was found using python and is used as input in (5) to find the points of the PDF of the RUL(t_j). The histogram obtained in problem 8 is then converted into a density histogram and the PDF is plotted on top of it. The resulting plot is as below. The python script is attached as **Appendix H**.

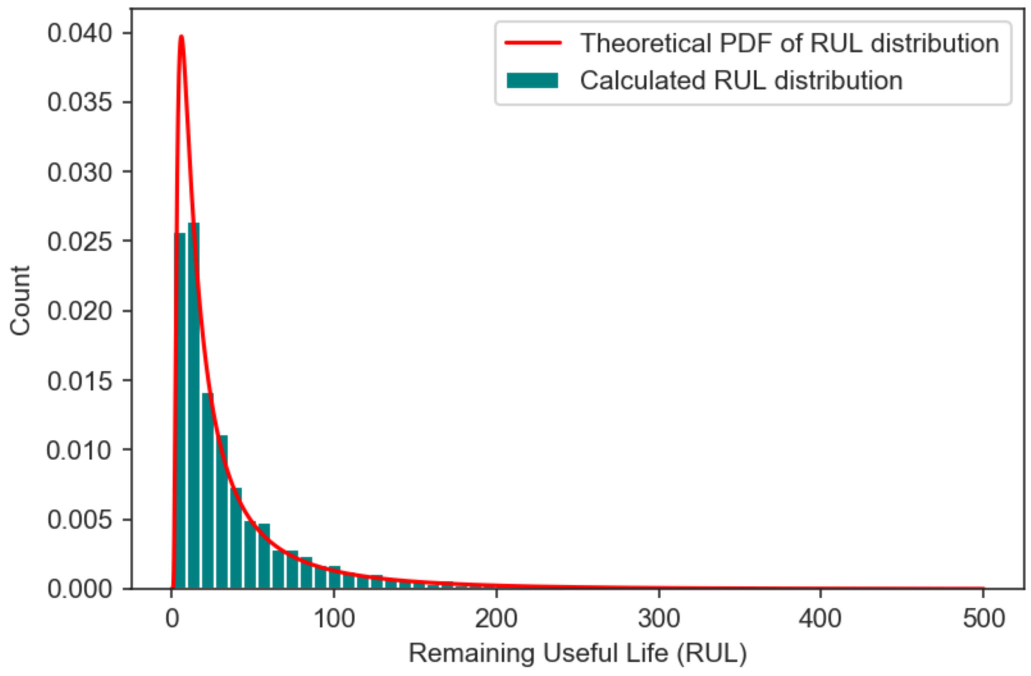


Figure 6: The histogram for RUL with increments with the PDF

Problem 10

(a) In-order to find the $\Pr(\text{RUL}(t_j) \leq 34)$, the logic used is to use python script to count the number of items less than 34 from problem 5 and divide it by the total number of items. The value obtained is **0.658**

(b) The same process is repeated, this time on the values obtained as part of problem 8. The probability obtained is **0.638**

(c) For the inverse gaussian distribution the pdf of the function has to be integrated over the interval (0.0001,34)[starting value grater than 0] to find the probability. For performing integration trapezoidal method is implemented in python and the probability obtained is **0.69**

The codes for all the problems has been attached as **Appendix I**

Comment: When the probability was assessed for 10.a) it was noted that the probability depended a lot on the time step. For instance, when the time horizon was taken to be between 60 to 110 and divided into 50 the above value was obtained, but when divided into 100 the probability becomes 0.88 and is found to be varying when that division is changed again. This could be due to the fact that there are more points that are below 94 than that is above when divided into 100. It would be great if a feedback on why this happens could be given.

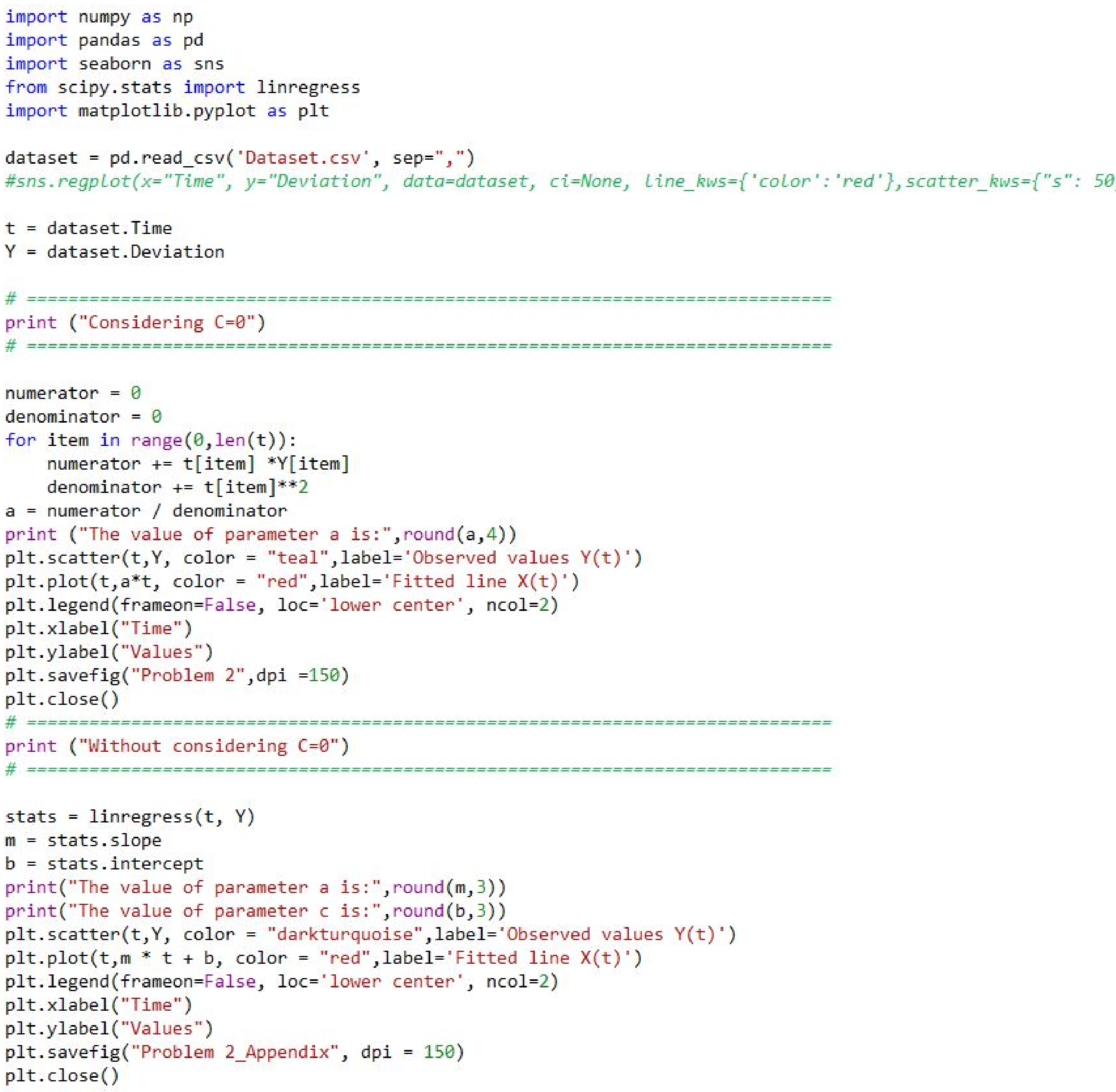
Problem 11

If we have data till t = 80, we can say that we no have more information regarding the degradation process. As a result we can say that the uncertainties associated with the process is a little more removed. This would lead to a better estimation of parameters and less spread histograms.

Appendix

Appendix A: Estimation without considering C=0

The value of parameters are , $\mathbf{a} = \mathbf{0.094}$ and $\mathbf{c} = \mathbf{0.293}$.The python code is as below:



```python
import numpy as np
import pandas as pd
import seaborn as sns
from scipy.stats import linregress
import matplotlib.pyplot as plt

dataset = pd.read_csv('Dataset.csv', sep=",")
#sns.regplot(x="Time", y="Deviation", data=dataset, ci=None, line_kws={'color':'red'},scatter_kws={"s": 50

t = dataset.Time
Y = dataset.Deviation

# =============================================================================
print ("Considering C=0")
# =============================================================================

numerator = 0
denominator = 0
for item in range(0,len(t)):
    numerator += t[item] *Y[item]
    denominator += t[item]**2
a = numerator / denominator
print ("The value of parameter a is:",round(a,4))
plt.scatter(t,Y, color = "teal",label='Observed values Y(t)')
plt.plot(t,a*t, color = "red",label='Fitted line X(t)')
plt.legend(frameon=False, loc='lower center', ncol=2)
plt.xlabel("Time")
plt.ylabel("Values")
plt.savefig("Problem 2",dpi =150)
plt.close()
# =============================================================================
print ("Without considering C=0")
# =============================================================================

stats = linregress(t, Y)
m = stats.slope
b = stats.intercept
print("The value of parameter a is:",round(m,3))
print("The value of parameter c is:",round(b,3))
plt.scatter(t,Y, color = "darkturquoise",label='Observed values Y(t)')
plt.plot(t,m * t + b, color = "red",label='Fitted line X(t)')
plt.legend(frameon=False, loc='lower center', ncol=2)
plt.xlabel("Time")
plt.ylabel("Values")
plt.savefig("Problem 2_Appendix", dpi = 150)
plt.close()
```

Figure 7: Code for Problem 2

The corresponding plot received is:

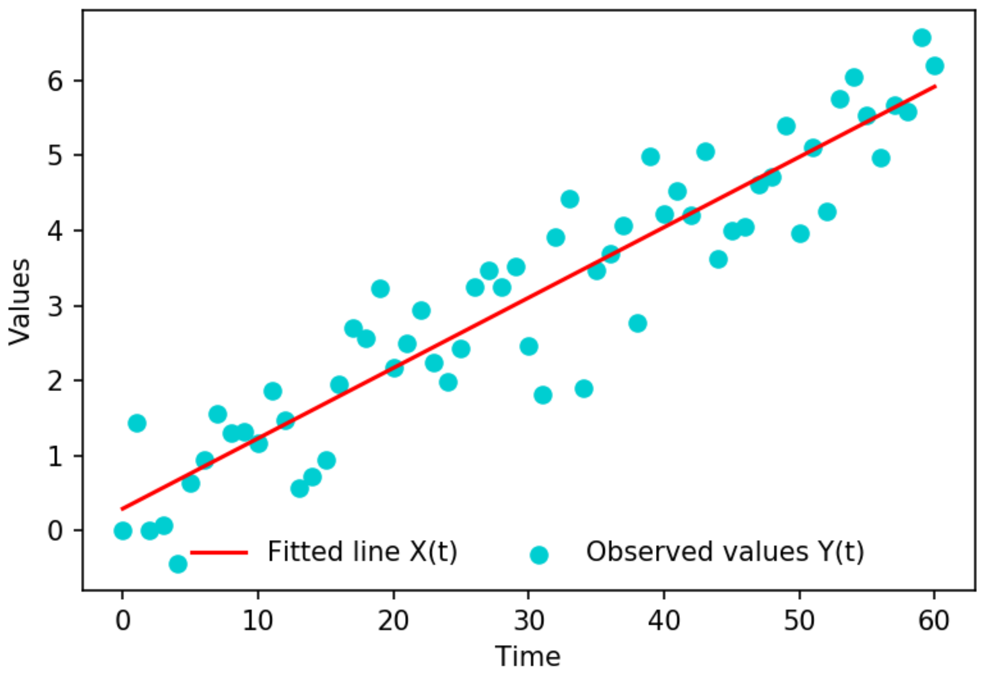
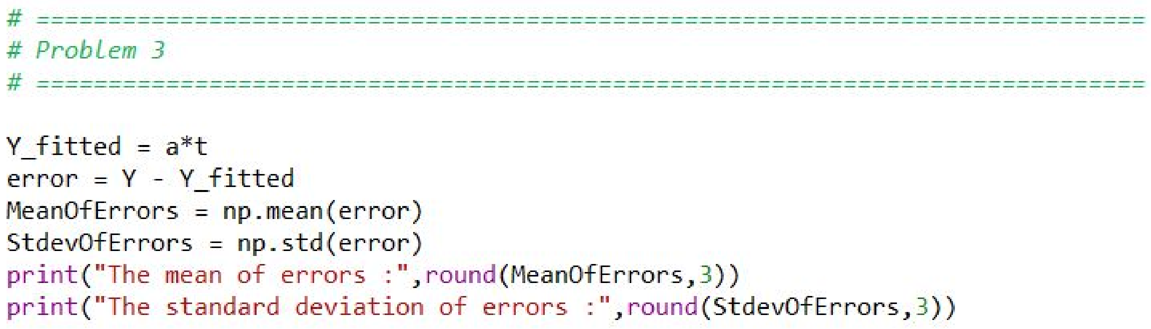


Figure 8: Code for Problem 2

Appendix B: Mean and SD for error distribution

The python script is as below

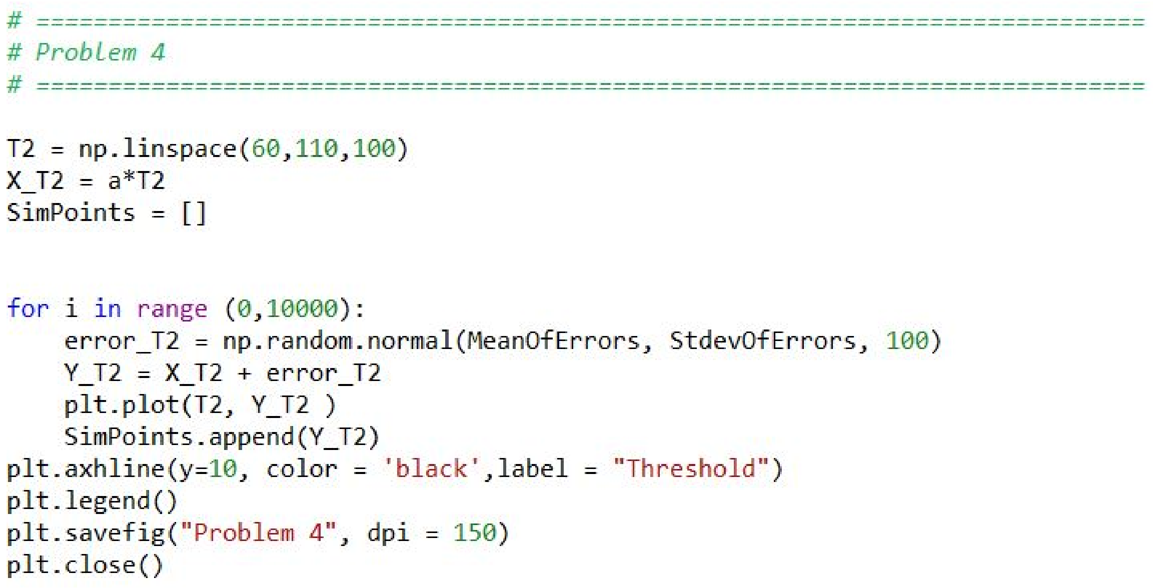


```
# ============================================================================
# Problem 3
# ============================================================================

Y_fitted = a*t
error = Y - Y_fitted
MeanOfErrors = np.mean(error)
StdevOfErrors = np.std(error)
print("The mean of errors :",round(MeanOfErrors,3))
print("The standard deviation of errors :",round(StdevOfErrors,3))
```

Figure 9: Code for Problem 3

Appendix C: Simulated trajectories

The python script is as below

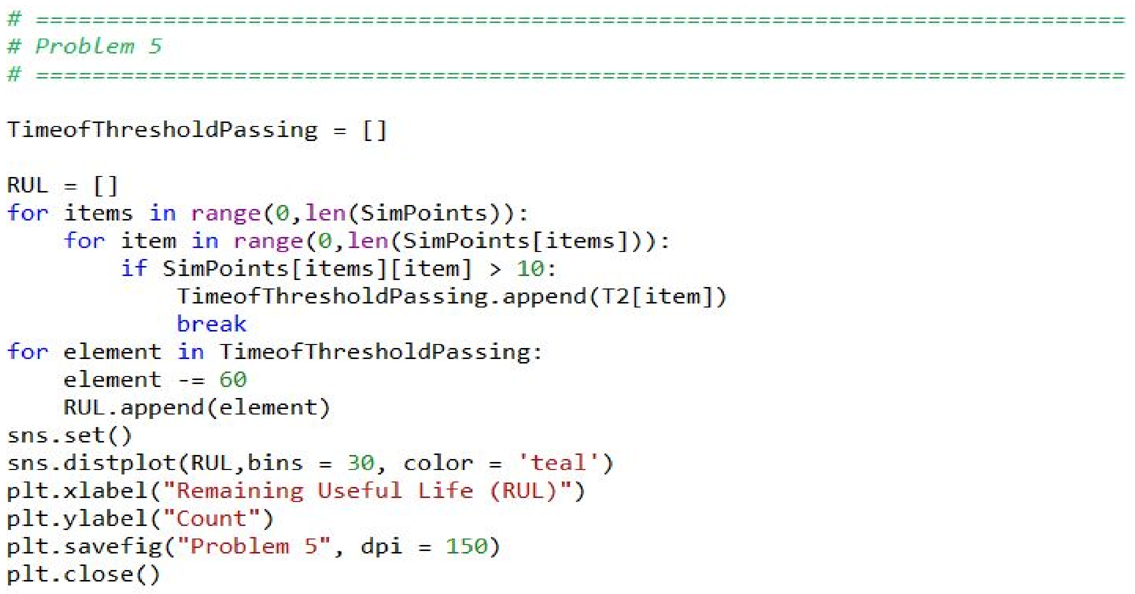


```
# ==============================================================================
# Problem 4
# ==============================================================================

T2 = np.linspace(60,110,100)
X_T2 = a*T2
SimPoints = []


for i in range (0,10000):
    error_T2 = np.random.normal(MeanOfErrors, StdevOfErrors, 100)
    Y_T2 = X_T2 + error_T2
    plt.plot(T2, Y_T2 )
    SimPoints.append(Y_T2)
plt.axhline(y=10, color = 'black',label = "Threshold")
plt.legend()
plt.savefig("Problem 4", dpi = 150)
plt.close()
```

Figure 10: Code for Problem 4

Appendix D: Remaining Useful life estimation

The python script is as below

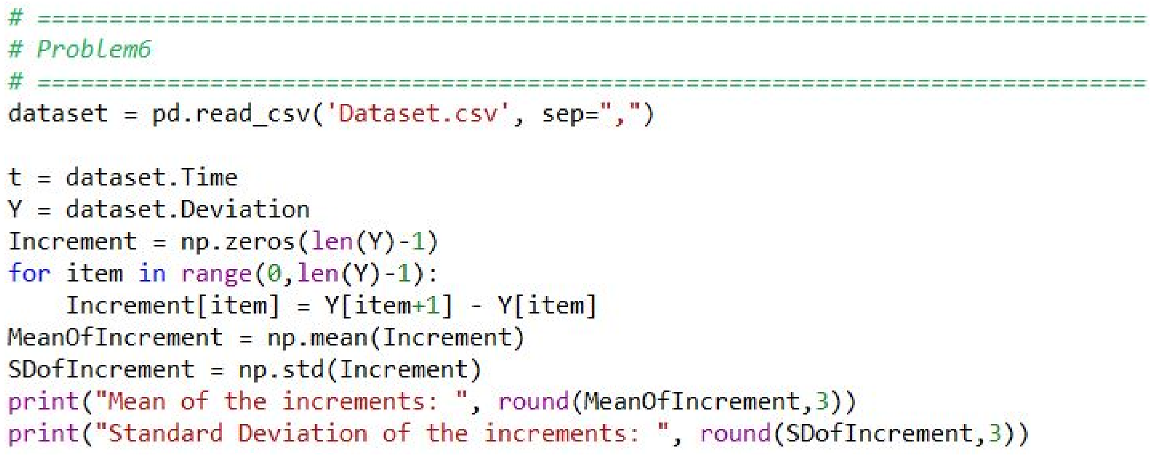


```
# ==============================================================================
# Problem 5
# ==============================================================================

TimeofThresholdPassing = []

RUL = []
for items in range(0,len(SimPoints)):
    for item in range(0,len(SimPoints[items])):
        if SimPoints[items][item] > 10:
            TimeofThresholdPassing.append(T2[item])
            break
for element in TimeofThresholdPassing:
    element -= 60
    RUL.append(element)
sns.set()
sns.distplot(RUL,bins = 30, color = 'teal')
plt.xlabel("Remaining Useful Life (RUL)")
plt.ylabel("Count")
plt.savefig("Problem 5", dpi = 150)
plt.close()
```

Figure 11: Code for Problem 5

Appendix E: Calculations of parameters of increments

The python script is as below

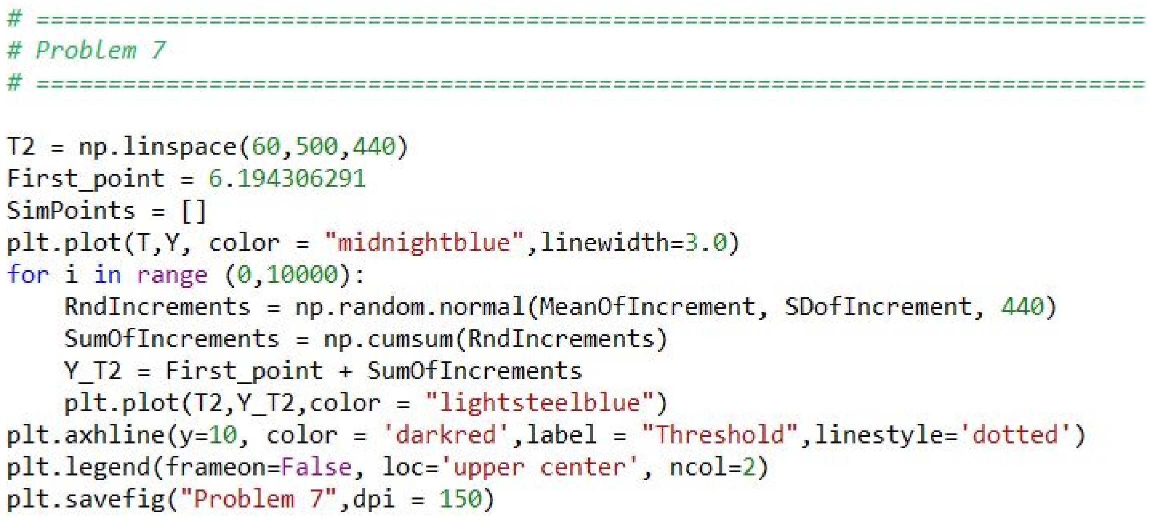


```
# ============================================================================
# Problem6
# ============================================================================
dataset = pd.read_csv('Dataset.csv', sep=",")

t = dataset.Time
Y = dataset.Deviation
Increment = np.zeros(len(Y)-1)
for item in range(0,len(Y)-1):
    Increment[item] = Y[item+1] - Y[item]
MeanOfIncrement = np.mean(Increment)
SDofIncrement = np.std(Increment)
print("Mean of the increments: ", round(MeanOfIncrement,3))
print("Standard Deviation of the increments: ", round(SDofIncrement,3))
```

Figure 12: Code for Problem 6

Appendix F: Simulated trajectories for increments

The python script is as below

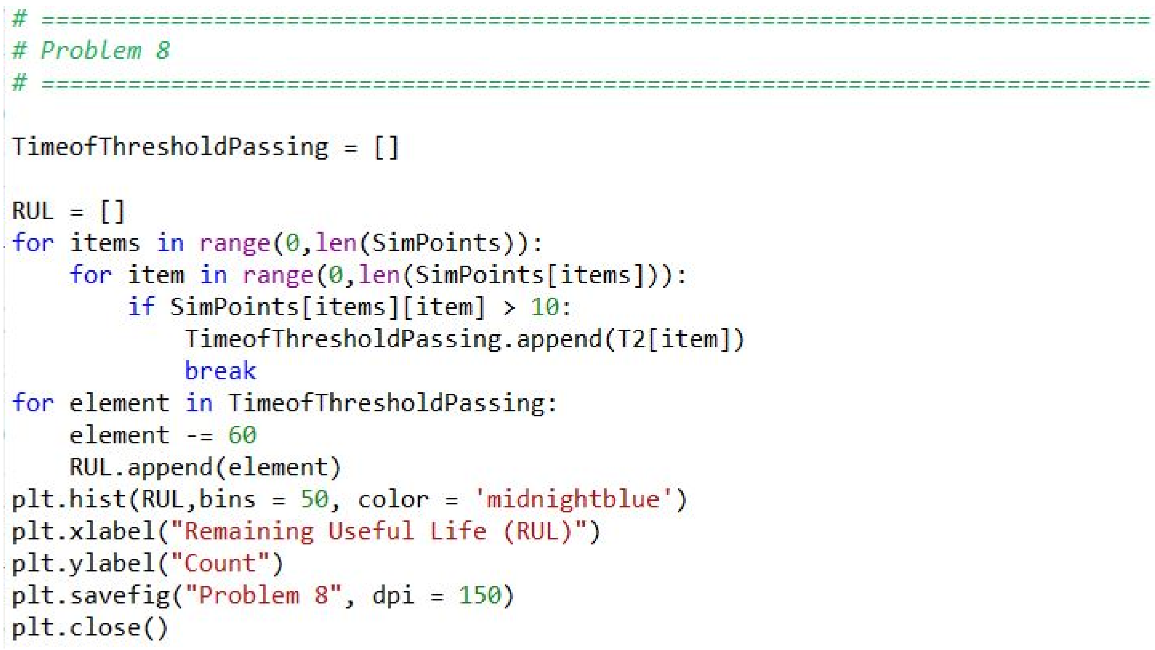


```
# ============================================================================
# Problem 7
# ============================================================================

T2 = np.linspace(60,500,440)
First_point = 6.194306291
SimPoints = []
plt.plot(T,Y, color = "midnightblue",linewidth=3.0)
for i in range (0,10000):
    RndIncrements = np.random.normal(MeanOfIncrement, SDofIncrement, 440)
    SumOfIncrements = np.cumsum(RndIncrements)
    Y_T2 = First_point + SumOfIncrements
    plt.plot(T2,Y_T2,color = "lightsteelblue")
plt.axhline(y=10, color = 'darkred',label = "Threshold",linestyle='dotted')
plt.legend(frameon=False, loc='upper center', ncol=2)
plt.savefig("Problem 7",dpi = 150)
```

Figure 13: Code for Problem 7

Appendix G: RUL with increments

The python script is as below

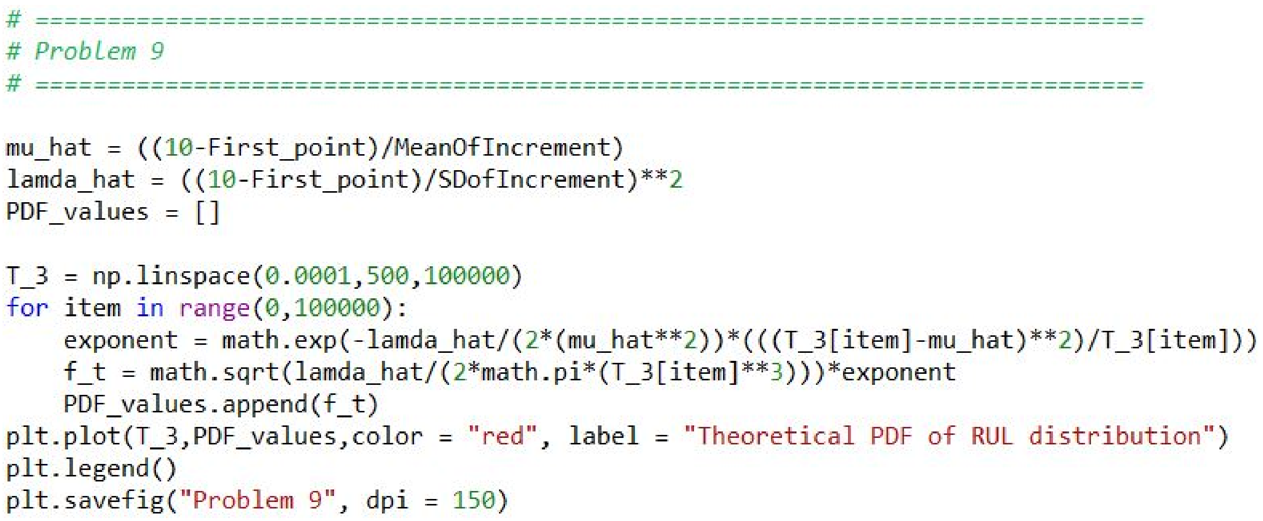


```
# ===========================================================================
# Problem 8
# ===========================================================================

TimeofThresholdPassing = []

RUL = []
for items in range(0,len(SimPoints)):
    for item in range(0,len(SimPoints[items])):
        if SimPoints[items][item] > 10:
            TimeofThresholdPassing.append(T2[item])
            break
for element in TimeofThresholdPassing:
    element -= 60
    RUL.append(element)
plt.hist(RUL,bins = 50, color = 'midnightblue')
plt.xlabel("Remaining Useful Life (RUL)")
plt.ylabel("Count")
plt.savefig("Problem 8", dpi = 150)
plt.close()
```

Figure 14: Code for Problem 8

Appendix H: RUL with increments and PDF

The python script is as below

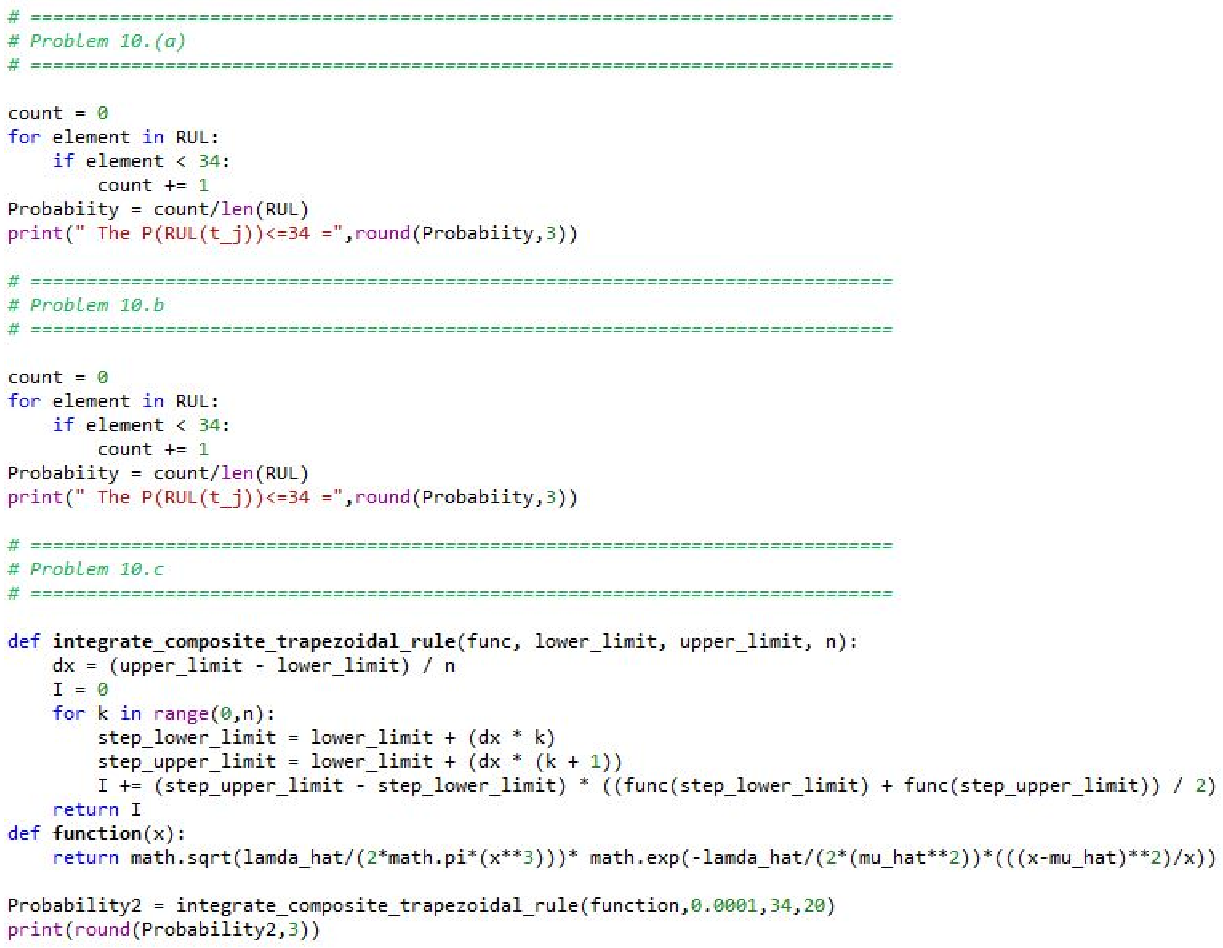


```
# ===========================================================================
# Problem 9
# ===========================================================================

mu_hat = ((10-First_point)/MeanOfIncrement)
lamda_hat = ((10-First_point)/SDofIncrement)**2
PDF_values = []

T_3 = np.linspace(0.0001,500,100000)
for item in range(0,100000):
    exponent = math.exp(-lamda_hat/(2*(mu_hat**2))*(((T_3[item]-mu_hat)**2)/T_3[item]))
    f_t = math.sqrt(lamda_hat/(2*math.pi*(T_3[item]**3)))*exponent
    PDF_values.append(f_t)
plt.plot(T_3,PDF_values,color = "red", label = "Theoretical PDF of RUL distribution")
plt.legend()
plt.savefig("Problem 9", dpi = 150)
```

Figure 15: Code for Problem 9

Appendix I: Probability Assessment

The python script is as below



```
# ==============================================================================
# Problem 10.(a)
# ==============================================================================

count = 0
for element in RUL:
    if element < 34:
        count += 1
Probabiity = count/len(RUL)
print(" The P(RUL(t_j))<=34 =",round(Probabiity,3))

# ==============================================================================
# Problem 10.b
# ==============================================================================

count = 0
for element in RUL:
    if element < 34:
        count += 1
Probabiity = count/len(RUL)
print(" The P(RUL(t_j))<=34 =",round(Probabiity,3))

# ==============================================================================
# Problem 10.c
# ==============================================================================

def integrate_composite_trapezoidal_rule(func, lower_limit, upper_limit, n):
    dx = (upper_limit - lower_limit) / n
    I = 0
    for k in range(0,n):
        step_lower_limit = lower_limit + (dx * k)
        step_upper_limit = lower_limit + (dx * (k + 1))
        I += (step_upper_limit - step_lower_limit) * ((func(step_lower_limit) + func(step_upper_limit)) / 2)
    return I
def function(x):
    return math.sqrt(lamda_hat/(2*math.pi*(x**3)))* math.exp(-lamda_hat/(2*(mu_hat**2))*(((x-mu_hat)**2)/x))

Probability2 = integrate_composite_trapezoidal_rule(function,0.0001,34,20)
print(round(Probability2,3))
```

Figure 16: Code for Problem 10