NTNU

Kunnskap for en bedre verden

TPK 4450 - Data Driven Prognostics and Predictive Maintenance

# Semester work - Part 3

*Author:*
Krishnan Gopakumar Nair

November, 2020

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`.

For the valve under analysis, we have the total failure rate, $\lambda$ = 17.8 x $10^{-6}$ $h^{-1}$ which results in an MTTF = 5.6 x $10^4$ h

Time based maintenance includes two practices :-

- Clock-based maintenance policy
- Age based maintenance policy

In order to find the optimum period of preventive maintenance based on time, we assume the following : -

- Units after maintenance is considered as good as new.
- Cost of preventive maintenance , C = 50
- Cost of corrective maintenance , k = 500

## Problem 1

### Clock-based maintenance

For the problem we assume that the lifetime of the valve, T follows an exponential distribution with the parameter $\lambda$ ,i.e, T $\sim$ $f_T(t)$, where $f_T(t) = \lambda e^{-\lambda t}$.

Let preventive maintenance be done at time intervals of $t_c$ and if an item fails in this interval it is subjected to corrective maintenance. Cost in an interval is then given by: -

$$C(t_c) = C + (C + k)W(t_c)$$

where W( $t_c$) is the mean number of failure in the interval $t_c$. Asymptotic cost function in the interval $t_c$ is then given by,

$$\overline{C}(t_c) = \frac{C + (C + k)W(t_c)}{t_c} \tag{1}$$

This cost function is used to find the optimum $t_c$ by minimizing the cost. Therefore it is safe to assume that probability of having more than one failure in the optimum $t_c$ is negligible. We have,

$$W(t_c) = \sum_{n=1}^{\infty} nPr(N(t_c) = n))$$

where N($t_c$) is the number of failure in an interval and
Pr(N($t_c$) = n ) = Pr($T_1$ + $T_2$ + ... + $T_n$ ) < $t_c$) using the approximation made earlier,W($t_c$)

becomes,

$$W(t_c) \approx 1\text{x}Pr(N(t_c) = 1)$$
$$W(t_c) = Pr(T \leq t_c) = \int_0^{t_c} f_T(t)\, dt$$

Put in (1),

$$\overline{C}(t_c) = \frac{C + (C + k)\int_0^{t_c} f_T(t)\, dt}{t_c} \tag{2}$$

$$\overline{C}(t_c) = \frac{C + (C + k)(1 - e^{-\lambda t_c})}{t_c}$$

Since for an exponential distribution, $\int_0^{t_c} f_T(t)\, dt = (1 - e^{-\lambda t_c})$.

The code required for calculation is attached as Appendix A. It was observed that the values for the cost kept on reducing with increasing $t_c$ for an exponential distribution and hence there is no optimal period. This makes sense as an exponential distribution has a constant failure probability and is memory less. Hence does not support preventive maintenance.

**Age-based maintenance**

The same applies for Age-based maintenance, so no optimal interval.

## Problem 2

Now assuming that T follows a Weibull distribution, with a shape parameter $\beta = 3$, we have now, T $\sim$ $f_T$(t), and $f_T$(t) $= \beta\lambda^\beta t^{\beta-1} e^{-(\lambda t)^\beta}$, where $\lambda$ is the scale parameter. If we assume that the MTTF given in the question is valid for the Weibull distribution also, the scale parameter is given by: -

$$\lambda = \frac{1}{MTTF}\Gamma\left(\frac{1}{\beta} + 1\right)$$
$$\lambda = \frac{1}{5.6\text{x}10^4}\Gamma\left(\frac{1}{3} + 1\right)$$
$$\lambda = 1.59\text{x}10^{-5}$$

Proceeding in the same way as in Problem 1 and using the $f_T$(t) for a Weibull distribution equation (2) becomes,

$$\overline{C}(t_c) = \frac{C + (C + k)(1 - e^{(-\lambda t_c)^\beta})}{t_c}$$

Solving using python script, we get, Optimal interval $t_c$ = **22906 hours**

The resulting plot is as below: -

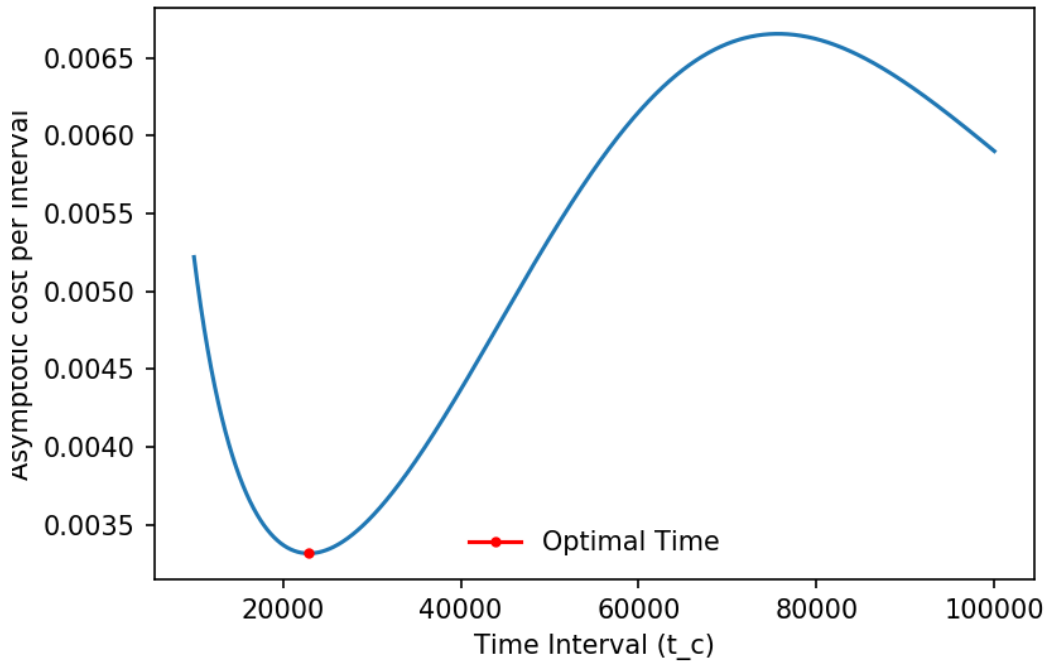


Figure 1: Code for Problem 1

The code is attached as Appendix B.

## Problem 3

The given "cmonitoring.txt" file is loaded into a dataframe in python and the observations are plotted. Figure 2 shows degradation plot for the different test interval and Figure 3 shows the increments in degradation for each unit.

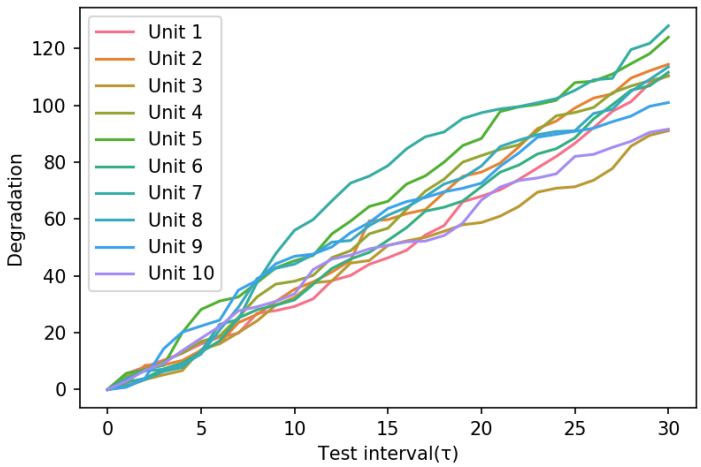


Figure 2: Data plot

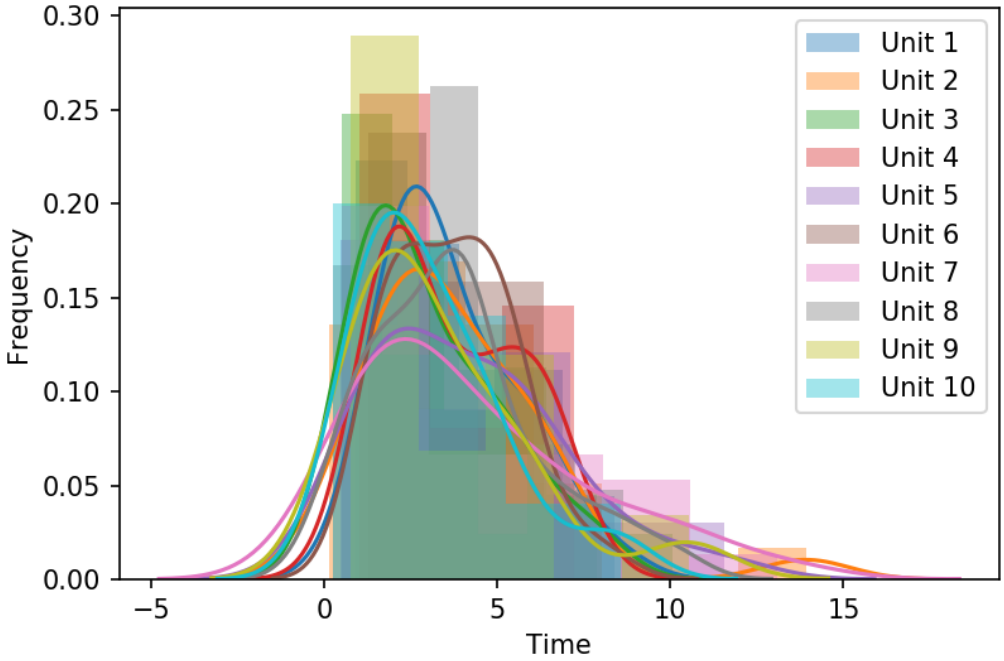


Figure 3: Distribution of increments for the components

From Figure 2 we can see that it is monotonously increasing and hence we can conclude that it is a Gamma process.

The code has been attached as Appendix C.

## Problem 4

The parameters could be estimated using the "scipy.stats.gamma.fit" function in python over the increments. The values obtained were : The shape parameter = 2.378 and the scale parameter = 1.536. The code has been attached as Appendix D.

The parameter could also be estimated using Maximum Likelihood Estimation. It involves the following steps.

1. Start with the PDF of Gamma distribution given by:-

$$f(t) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} t^{\alpha-1} e^{-\beta t}$$

where $\beta$ is called rate parameter (inverse scale parameter) and $\alpha$ is shape parameter. The

Likelihood function is given by: -

$$L(t_i\,;\,\alpha\,,\,\beta) = \prod_{i=1}^{n} f_I(t_i;\alpha\,,\,\beta)$$

2. Take the $ln$ of the Likelihood function, which gives us: -

$$ln\,L(t_i\,;\,\alpha\,,\,\beta) = n(\alpha\,ln\,\beta - ln\,\Gamma(\alpha)) + (\alpha - 1)\sum_{i=1}^{n} ln\,t_i - \beta\,\sum_{i=1}^{n} t_i$$

3. In the next stage take partial derivative of the $ln\,L(t_i\,;\,\alpha\,,\,\beta)$ with respect to $\alpha$ and $\beta$ and equate it to 0.

$$\frac{\partial}{\partial\,\alpha} ln\,L(t_i\,;\,\hat{\alpha}\,,\,\hat{\beta}) = n(ln\,\hat{\beta} - \frac{d}{d\,\alpha} ln\,\Gamma(\hat{\alpha})) + \sum_{i=1}^{n} ln\,t_i = 0$$

$$\frac{\partial}{\partial\,\beta} ln\,L(t_i\,;\,\hat{\alpha}\,,\,\hat{\beta}) = n\frac{\hat{\alpha}}{\hat{\beta}} - \sum_{i=1}^{n} t_i = 0$$

4. Solve the equation to find the $\hat{\alpha}$ and $\hat{\beta}$

## Problem 5

Consider the following assumptions:

- As-good-as-new replacement
- Failures are detected immediately and the unit is replaced.
- The deterioration of the valve is only known at inspection dates (unless a failure occurs).
- Time to replace the valve is neglected. (Instantaneous replacement)
- Cost of preventive maintenance: $C_2 = 50$
- Additional cost due to failures (corrective maintenance): $C_3 = 500$
- Cost of inspection: $C_1 = 10$
- The organization decides to keep the inspection interval as it is $\tau = 12$ weeks.

Let $\sigma_M$ be the first inspection that happens after passing the maintenance threshold value M and $\sigma_L$ be the inspection after passage of the failure threshold value L. Let $T_R i$ be the min( $\sigma_M$, $\sigma_L$). The objective function is then given by: -

$$C_{T_R} = \frac{C_1 k + C_2 + C_3 d(T_R)}{T_R}$$

where,

- $C_1$ is the inspection cost
- $C_2$ is the preventive maintenance cost
- $C_3$ is the corrective maintenance cost
- $k = \frac{T_R}{\tau}$
- $d(T_R)$ is downtime at $(T_R)$

## Problem 6

A total of 1000 trajectories have been simulated and the plot obtained is shown in Figure 4 .

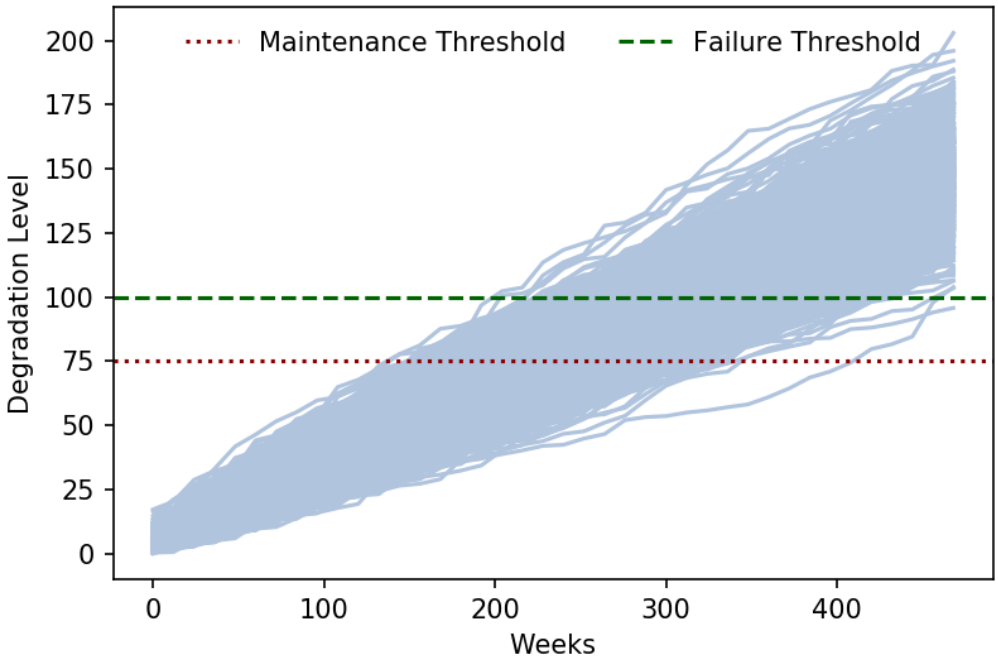


Figure 4: Distribution of increments for the components

The code is attached as Appendix E.

## Problem 7

### Part a

Using interpolation, the histogram obtained for the passage time $(\sigma_L)$ is given in Figure 5.

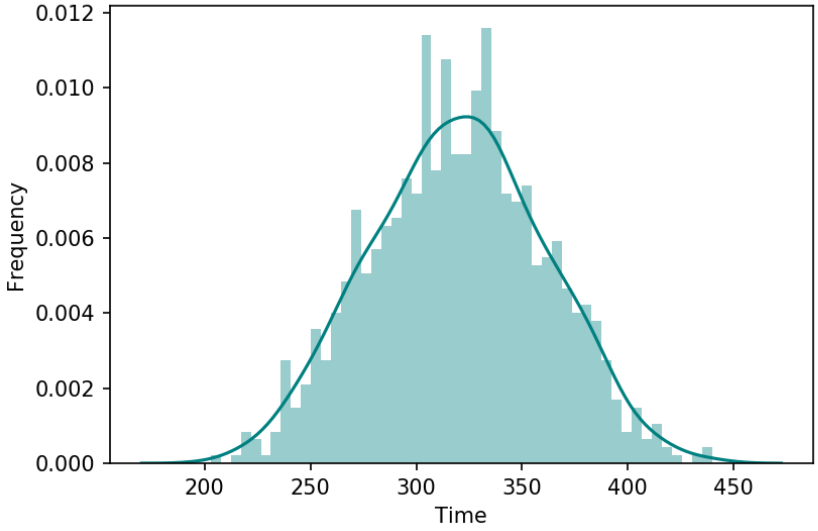


Figure 5: Histogram for passage time ($\sigma_L$)

**Part b**

The histogram obtained for the passage time ($\sigma_M$) is given in Figure 6.

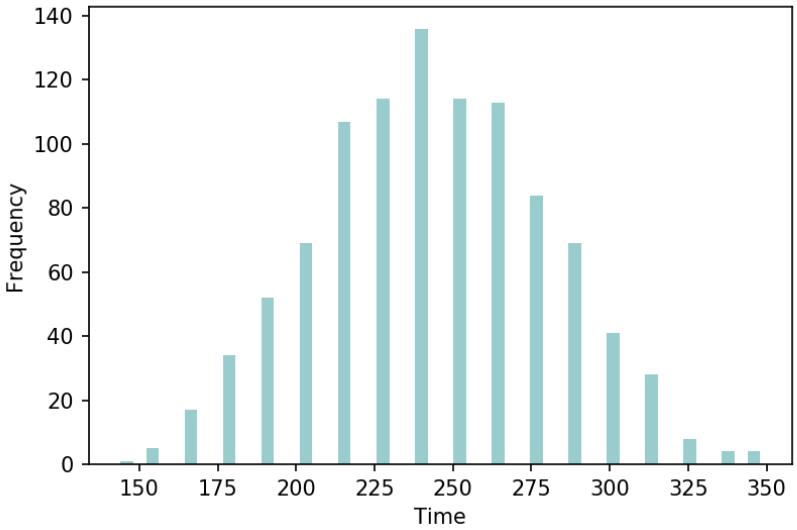


Figure 6: Histogram for passage time ($\sigma_M$)

The code for both the plots are attached as Appendix F.

## Problem 8

The MTTF is found by taking the mean of the list of values found for $\sigma_L$ as part of Problem 7.a. The MTTF was found to be **317.38 weeks** , which is **53320 hours**

The code is attached as Appendix G.

## Problem 9

The MTBR is found by taking the using list of values found for $\sigma_L$ and $\sigma_M$ as part of Problem 7. The MTBR was found to be **241.6 weeks** , which is **40589 hours**

The code is attached as Appendix H.

## Problem 10

The asymtotic cost $\overline{C_{T_n}}$ is given by,

$$\overline{C_{T_n}} = \frac{1}{n}\sum_{i=1}^{m} C_{T_{n\,i}}$$
$$\overline{C_{T_n}} = \frac{C_1 k_i + C_2 + C_3}{T_{Ri}}$$

where, $C_3$ is given by,

$$C_3 = \begin{cases} 0 & \text{for } \sigma_L > \sigma_M \\ 500 & \text{for } \sigma_L \leq \sigma_M \end{cases}$$

Note that all notations denote the same as in Problem 5.

Using python, the value of $\overline{C_{T_n}}$ was found to be **1.0448 cost/week**.

The code is attached as Appendix I.

## Problem 11

The same function used to find the value in problem 10 is run as a loop to obtain the asymptotic cost for different values of maintenance threshold, M. The resulting plot is shown in Figure 7.

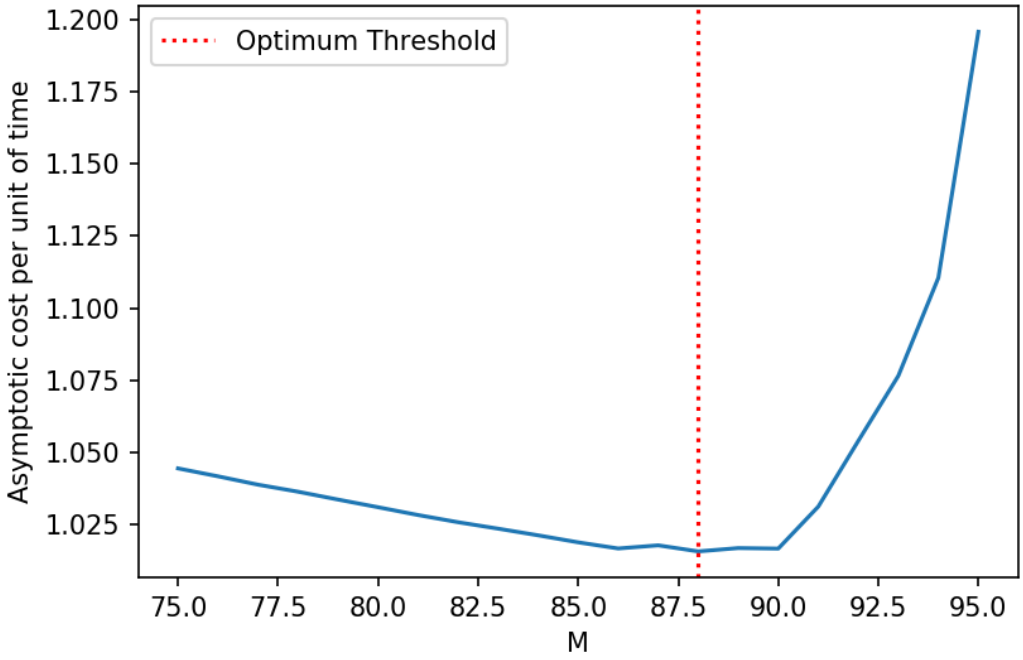


Figure 7: Asymptotic Cost Vs Maintenance Threshold

## Problem 12

The optimal value of Maintenance Threshold from the plot was found to be **88** units.

The code for problem 11 and 12 are attached as Appendix J.

# Appendix

## Appendix A: Code for PM when exponential distribution

The python script is as below

```
# ================================================================================
# Problem 1
# ================================================================================

t_c = np.linspace(730,87600,120)
Cost = []

def MeanCost(t_c):
    MC = (C + (C+k)*(1-math.exp(-Lambda*t_c)))/t_c
    return MC

for element in t_c:
    Cost.append(MeanCost(element))

index = Cost.index(min(Cost))
print (t_c[index])
```

Figure 8: Code for Problem 1

## Appendix B: Code for PM when Weibull distribution

The python script is as below

```
# ================================================================================
# Problem 2
# ================================================================================
t_c = np.linspace(10000,100000,5001)
Cost = []
def MeanCost(t_c):
    F_t = 1-math.exp(math.pow(-Lambda*t_c,3))
    MC = (C + (C+k)*F_t)/t_c
    return MC

for element in t_c:
    Cost.append(MeanCost(element))
index = Cost.index(min(Cost))
print (t_c[index])
plt.plot(t_c, Cost)
plt.plot(t_c[index],Cost[index], marker='o', markersize=3, color="red", label = "Optimal Time")
plt.legend(frameon=False, loc='lower center', ncol=2)
plt.savefig("Problem 2-clock", dpi = 150)
```

Figure 9: Code for Problem 2 for clock based PM

## Appendix C: Plot of data and increment

The python script is as below

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

dataset = pd.read_csv('cmonitoring.txt', sep=",", header = None)
dataset.columns = ["Unit 1", "Unit 2", "Unit 3", "Unit 4", "Unit 5",
                   "Unit 6", "Unit 7", "Unit 8", "Unit 9", "Unit 10"]

sns.lineplot(data=dataset, dashes=False)
plt.xlabel("Test interval(τ)")
plt.ylabel("Degradation")
plt.legend()
plt.savefig("Problem 3 point plot", dpi = 150)
plt.close()

increment = dataset.diff().drop([0])
for column in increment:
    columnVal = increment[column]
    sns.distplot(columnVal, label = column )
plt.xlabel("Time")
plt.ylabel("Frequency")
plt.legend()
plt.savefig("Problem 3 increments", dpi = 150)
```

Figure 10: Code for Problem 3

## Appendix D: Parameter estimation

The python script is as below

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

shape, loc, scale = gamma.fit(increment, floc=0)

print("The shape parameter is given by:", round(shape,3))
print("The location parameter is given by:", round(loc,3))
print("The scale parameter is given by:", round(scale,3))
```

Figure 11: Code for Problem 4

## Appendix E: Simulated trajectories

The python script is as below

```
# ============================================================================
# Problem 6
# ============================================================================
trajectory = {}

for item in range(1, 1001):
    x = 0
    DegradationLevel = []
    for element in range(40):
        x += gamma.rvs(shape, loc, scale)
        DegradationLevel.append(x)
    trajectory[f'T{item}'] = DegradationLevel
trajectories = pd.DataFrame(trajectory)


time = [i for i in range(0, 40*12, 12)]
for i in range(1, 1001):
    plt.plot(time, trajectories[f'T{i}'], color = "lightsteelblue")
plt.axhline(y=75, color = 'darkred',label = "Maintenance Threshold",linestyle='dotted')
plt.axhline(y=100, color = 'darkgreen',label = "Failure Threshold",linestyle='dashed')
plt.xlabel("Weeks")
plt.ylabel("Degradation Level")
plt.legend(frameon=False, loc='upper center', ncol=2)
plt.savefig("Problem 6",dpi = 150)
```

Figure 12: Code for Problem 6

## Appendix F: Problem 7

The python script is as below

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

Lvalues =  []
time = [i for i in range(0, 40*12, 12)]
for i in range(1, 1001):
    if any(trajectories[f'T{i}'].values >= 100):
        Lvalues.append(np.interp(100, trajectories[f'T{i}'], time))
sigmaL = np.array(Lvalues)
sns.distplot(sigmaL, bins=50, color = "Teal")
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.savefig("Problem 7a",dpi = 150)
plt.show()
```

Figure 13: Code for Problem 7a

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

sigmaM =  []
time = [i for i in range(0, 40*12, 12)]
for i in range(1,1001):
    for j in range(len(trajectories['T1'])):
        if trajectories[f'T{i}'][j] >= 75:
            sigmaM.append(time[j])
            break
sns.distplot(sigmaM, kde = False, bins=50, color = "Teal")
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.savefig("Problem 7b",dpi = 150)
plt.show()
```

Figure 14: Code for Problem 7b

## Appendix G: MTTF

The python script is as below

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

MTTF = sigmaL.mean()

print ("MTTF = ",MTTF)
```

Figure 15: Code for Problem 8

## Appendix H: MTBR

The python script is as below

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

t = 0
for i in range(len(sigmaL)):
    t += min(sigmaL[i], sigmaM[i])
MTBR = t/len(sigmaL)

print("MTBR =",MTBR)
```

Figure 16: Code for Problem 9

## Appendix I: Asymptotic cost

The python script is as below

```
# ============================================================================
# Problem 10
# ============================================================================

ci = 10
C = 50
K = 500
asymptoticCost = []
time = [i for i in range(0, 40*12, 12)]
for i in range(1,1001):
    if any(trajectories[f'T{i}'].values >= 100):
        sigmaL = np.interp(100, trajectories[f'T{i}'], time)
        k_L = sigmaL//12
    for j in range(len(trajectories[f'T{i}'])):
        if trajectories[f'T{i}'][j] >= 75:
            sigmaM = time[j]
            k_M = time[j]/12
            break
    if sigmaL > sigmaM:
        cost = (ci*k_M + C)/sigmaM
    else:
        cost = (ci*k_L + C + K)/sigmaL
    asymptoticCost.append(cost)

cost = np.array(asymptoticCost).mean()
print ("Asymptotic Cost per unit time =", cost)
```

Figure 17: Code for Problem 10

## Appendix J: Optimum Asymptotic cost

The python script is as below

```python
# ================================================================================
# Problem 11 & 12
# ================================================================================

def AsymptoticCost(trajectories,M):
    ci = 10
    C = 50
    K = 500
    asymptoticCost = []
    time = [i for i in range(0, 40*12, 12)]
    for i in range(1,1001):
        if any(trajectories[f'T{i}'].values >= 100):
            sigmaL = np.interp(100, trajectories[f'T{i}'], time)
            k_L = sigmaL//12
        for j in range(len(trajectories[f'T{i}'])):
            if trajectories[f'T{i}'][j] >= M:
                sigmaM = time[j]
                k_M = time[j]/12
                break
        if sigmaL > sigmaM:
            cost = (ci*k_M + C)/sigmaM
        else:
            cost = (ci*k_L + C + K)/sigmaL
        asymptoticCost.append(cost)
    return np.array(asymptoticCost).mean()

Cost = []
M = [i for i in range(75,96)]
for i in range(75, 96):
    Cost.append(AsymptoticCost(trajectories, i))

sns.lineplot(M, Cost)
OptimumThreshold = M[Cost.index(min(Cost))]
print("Optimum Threshold:",OptimumThreshold)
plt.axvline(OptimumThreshold, color='red', linestyle='dotted', label = "Optimum Threshold")
plt.xlabel('M')
plt.ylabel('Asymptotic cost per unit of time')
plt.legend()
plt.savefig("Problem 11",dpi = 150)
plt.show()
```

Figure 18: Code for Problem 11 and 12