B. TECH. PROJECT REPORT

On

Development of a Failure Simulation Library Based on Concepts of Reliability Engineering

BY

Chinmay Naik

DISCIPLINE OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

DEC 2019

Development of a Failure Simulation Library Based on Concepts of Reliability Engineering

A PROJECT REPORT

Submitted in partial fulfillment of the
requirements for the award of the degrees

of
BACHELOR OF TECHNOLOGY
in

MECHANICAL ENGINEERING

Submitted by:
Chinmay Naik

Guided by:
Dr.Makarand S.Kulkarni
Dr.Bhupesh K. Lad

INDIAN INSTITUTE OF TECHNOLOGY INDORE
DEC 2019

CANDIDATE'S DECLARATION

I hereby declare that the project entitled **"Development of a Failure Simulation Library Based on Concepts of Reliability Engineering"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Mechanical Engineering' completed under the supervision of **Dr. Makarand S. Kulkarni, IIT Bombay** and **Dr. Bhupesh K. Lad, IIT Indore** is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Chinmay Naik (160003031)

CERTIFICATE by BTP Guide(s)

It is certified that the above statement made by the students is correct to the best of my/our knowledge.

Signature
Dr. Bhupesh K. Lad
Associate Professor,
Mechanical Engineering,
IIT Indore

Preface

This report on "Development of a Failure Simulation Library Based on Concepts of Reliability Engineering" is prepared under the guidance of Dr. Makarand S. Kulkarni and Dr. Bhupesh K. Lad.

I have developed a failure simulation library named 'Fsim' based on programming language python 3. I have tried to explain the design and development of different modules and functions of this library through this project report. I have also included a small optimization problem at the end for the demonstration of use of functions in Fsim.

Chinmay Naik (160003031)
B.Tech. IV Year
Discipline of Mechanical Engineering
IIT Indore

Acknowledgements

First, I would like to thank my guide and supervisor Dr. Makarand S. Kulkarni for giving me an opportunity to work on this project and guiding me throughout it. I would also like to thank Dr. Bhupesh K. Lad for giving me his time and valuable guidance as an internal supervisor, along with helping me to be on track and complete this project on time. I would also like to thank Mr. Ram Mohril and all the other fellow members of Industrial Engineering and Smart Manufacturing lab for their constant support in development and presentation of this project.

In my personal view, this project has given me a great work experience and along with the field of work, I have learned a lot of new things in my life for self-improvement as well. It wouldn't have been possible for me to finish this work without the support of all the people mentioned above and hence I would like to offer my sincere thanks to all those who knowingly or unknowingly helped me completing this project.

Chinmay Naik (160003031)
B.Tech. IV Year
Discipline of Mechanical Engineering
IIT Indore

Abstract

This report presents the process of design and development of a failure simulation library named ‘Fsim’ based on the programming language Python 3. The functions of this library are based on concepts of Reliability. This library helps us to simulate the given system of components for a given amount of time and given number of simulations and get the estimates of possible results along with their distributions. Fsim involves mainly two modules named ‘Text’ and ‘Solution’. The Text module gives the results of simulation in form of a text file. The Solution module gives the results of the simulation in form of user defined data structure named ‘Solution’ inside the code itself so that the values can be used for further calculations and solving complicated problems. Each of these modules includes functions for different configurations of components (e.g. series, parallel, k out of m, standby components, competitive failure modes etc.) and different maintenance methods (e.g. Run to failure, Time based maintenance, Age based maintenance etc.). Finally, an optimization problem for maintenance scheduling of a given system of components is included for the demonstration of working of this library.

Table of Contents

List of Figures

List of Tables

Chapter-1

Introduction

1.1 General Background

'Nothing is immortal in this world. Everything has a beginning, and everything has an end.'

The world of mechanical engineering is no exception to this philosophy. Every machine, every mechanism, every assembly and every component follow this rule and thus has a limited life. But no-one knows this life and no-one can accurately predict when a component can fail. However, we can experimentally observe and determine the probability of failure of a component in the field of Reliability Engineering. Reliability, i.e. the probability of survival of component till given time, can be helpful in predicting the nature of its failure.

Using the Reliability calculations on a given system of components, we can calculate different parameters. But sometimes it might get complicated to visualize the behavior of a system in given conditional environment merely based on the Reliability calculations. To remove this complexity and get comparatively more direct results needed, we can use the concepts of Reliability to simulate the given system multiple times for required runtime and get an estimation of different parameters needed for further analysis. The failure simulation library 'Fsim' does the same job and gives the results in visual as well as in encapsulated user defined variable format.

1.2 Reliability Engineering [2]

Reliability engineering is a sub-discipline of systems engineering that emphasizes dependability in the lifecycle management of a product. Reliability, describes the ability of a system or component to function under stated conditions for a specified period of time. Reliability is closely related to availability, which is typically described as the ability of a component or system to function at a specified moment or interval of time.

In Reliability engineering, we deal with the failure probability of a given component or system of components for given amount of time. The Reliability function for a component is represented as:

R(t) = The probability of survival of component till time t.
F(t) = The probability of Failure of component before time t = 1 - R(t).

Every Reliability function can be represented as the integration of the probability distribution function over time till infinity as follows:

$R(t) = P(T > t) = 1-F(t) = 1- \int_0^\infty f(t).\,dt$

Where,

t = time for calculation
T = time of failure
f(t) = probability density function

1.3 The simulation approach

The Reliability calculations done on a system gives all the required data in probabilistic format. Although this data can be really accurate, it might get difficult for an ordinary user to interpret theses results and convert them to the parameters of his own interest. To eliminate this issue, we can use the concepts in Reliability theory to simulate the given system for given time and given number of times to get an estimate of possible results in form of those parameters, which are directly beneficial for an ordinary user e.g. owner of a factory.

Simulation can be done by doing a couple of things for every happening event:

i) Pseudo Random Number generation.[3]
ii) Inverting the Failure function.[1]

These two when combined can simulate a component for given time by generating failure time as well as repair time using a random number generated based on its failure distribution.

1.4 Use of existing modules in Python

The failure simulation library 'Fsim' is built in python and it involves 2 models:
1. Text module
2. Solution module

However, there are some pre-existing modules in python that are used in development of this library. Some of them are listed below:

1. Random:

   Random is an inbuilt module in Python that uses PNRG algorithm (Pseudo-random number generator) to generate random numbers based on uniform distribution in given range. These results will undergo an inverse transformation function of the Reliability function and convert it to a randomly generated number in corresponding distribution.

2. Csv:

   Csv is an inbuilt module in Python that helps us to make and edit comma separated variable files (.csv files) which can be otherwise viewed or edited using a spreadsheet software like MS Excel.

3. Math:

   Math is again an inbuilt module in Python that is used in performing complex mathematical operations like (e.g. exponential and logarithmic calculations)

4. Numpy[4]:

   Numpy is one of the most popular python libraries, and is used for complicated mathematical calculations.

5. Matplotlib[5]:

   When it comes to plotting the graph, Matplotlib is probably the best Python library that exists. The Pyplot module in Matplotlib is very helpful in plotting graphs like bar graph, spline curve pie-charts etc.

Chapter-2

Methodology

2.1 Basics of Reliability engineering:

2.1.1. Introduction

All the functions in the library Fsim are based the concepts of Reliability. As discussed earlier, Reliability is nothing but the probability of survival of a component till a given time t. This can be represented mathematically as:

$R(t) = 1 – F(t) = P(T > t) = 1 - \int_0^\infty f(t).dt$

Where,

R(t) = Reliability of component till time t.
F(t) = Failure probability of component till time t.
t = time for calculation
T = time of failure
f(t) = probability density function of corresponding failure distribution.

There are various failure distributions a component may undergo. Although we are considering only the most popular failure distribution observed in most of the components i.e. the Weibull distribution.

2.1.2 Conditional Reliability

Condition Reliability is the Reliability of a component to survive for time t, provided it has already survived for time T_0. This simply follows the rules of Conditional Probability.

$$R(t|T_0) = \frac{R(T_0+t)}{R(T_0)}$$

2.1.2 The Weibull Distribution

Weibull Distribution is one of the most common failure distributions seen in components. Following are the details of Weibull Distribution:

Probability Distribution function: $f(t) = \frac{\beta}{\eta}\left(\frac{t-\gamma}{\eta}\right)^{\beta-1}.e^{-\left(\frac{T-\gamma}{\eta}\right)^{\beta}}$

Cumulative Distribution function: $F(t) = 1 - e^{-\left(\frac{t-\gamma}{\eta}\right)^{\beta}}$

Where $f(t) \geq 0$, $t \geq 0$ and $\beta, \eta \geq 0$

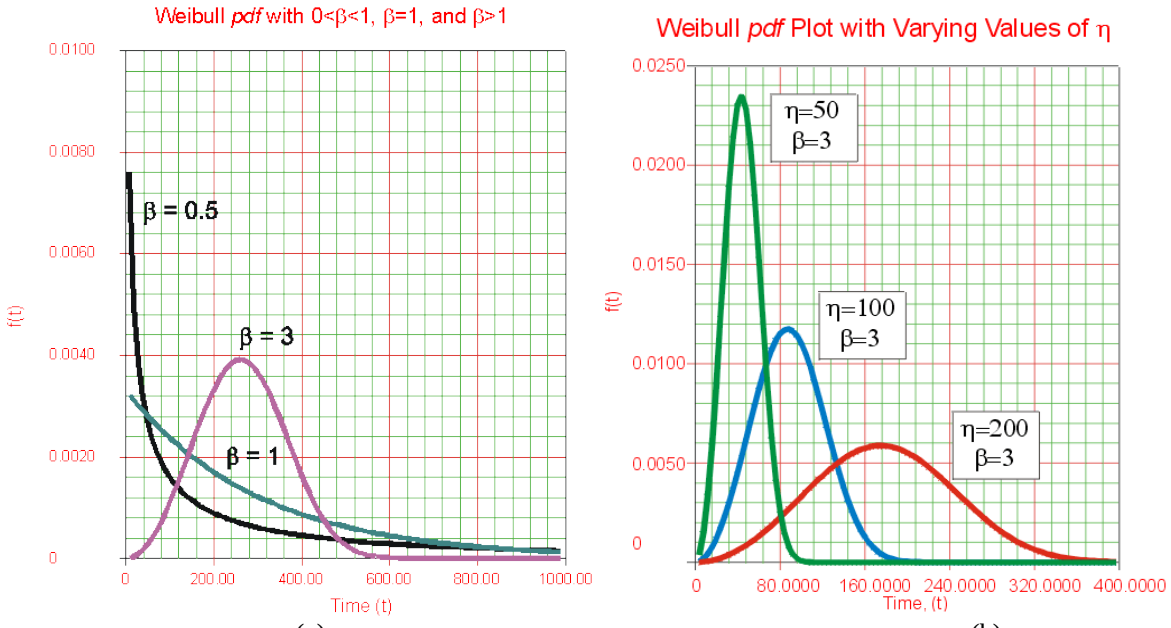
β: Shape parameter

η: Scale parameter

γ: Location parameter

While considering the usual cases, we can ignore the location parameter to narrow down our formula for Reliability to: $R(t) = e^{-\left(\frac{t}{\eta}\right)^{\beta}}$

2.1.3 Variation of probability density function based on variation of shape and scale parameters



(a) (b)

Fig.1: Variation of the probability density function with (a) Variation in β (Shape parameter) while keeping η (Scale parameter) constant (b) Variation in β (Shape parameter) while keeping η (Scale parameter) constant

In these figures we can clearly see how the probability density function depends on the shape and scale parameters. When the Scale parameter is closed to 1, the failure rate component is much less dependent on the time. As its value increases, it becomes more probable that the component might fail due to wear. Similarly, as the Shape parameter increases, the MTTF (Mean time to failure) of component increases.

2.2 Method of Simulation: The Inverse Transform Sampling

Inverse transformation function or inverse transform sampling is a basic method of pseudo-random number generation from any probability distribution, given its cumulative density function.

Since the 'Random' module in python, like any other language or programming platform, can generate uniform pseudo-random numbers in given range, we can use following way to generate random number based on given distribution:

1. Generate a random number 'u' from the standard uniform distribution in the interval [0,1].
2. Find the inverse transform function of $F^{-1}(x)$ of the Cumulative Density Function of given distribution.
3. Compute $X = F^{-1}(u)$. The computed random variable has the required distribution.

In case of Weibull distribution,

$$F(t) = 1 - e^{-\left(\frac{t}{\eta}\right)^{\beta}}$$

By solving for inverse transform function for Weibull distribution, we get that:

$$t = F^{-1}(u) = \eta.[-\ln(1-u)]^{\frac{1}{\beta}}$$

By substituting the randomly generated number u in range [0,1] we can generate a random value of time 't' that follows Weibull distribution. A similar equation can be found for the case of conditional reliability:

$$t = F^{-1}(u) = \eta.\left[\left(\frac{T_0}{\eta}\right)^{\beta} - \ln(1-u)\right]^{\frac{1}{\beta}} - T_0$$

This inverse transformation function of Weibull distribution is used throughout the library to generate random failure times in simulations.

2.3 The Gaussian (normal) Distribution and random number generation

Every component undergoes repair/replacement after failure. The MTTR (Mean time to repair) of every component is assumed to be Normally distributed. The Normal distribution, also known as the Gaussian distribution involves 2 parameters in its probability density function:

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

Where,

μ: The mean value of time needed for repair

σ: The standard deviation of time needed for repair

It's not that easy to solve for the inverse transform function for normal distribution. Hence a different approach is used here to generate a random number in Gaussian distribution. This is an approximate, but sufficiently good way of generating a Normally distributed random number:

1. Generate n random numbers $u_1, u_2, \ldots, u_n$ which are uniformly distributed in range [0,1], such that n is (preferably) a multiple of 12.
2. Use following approximation to generate a random number 't' in Gaussian distribution:

$$t = \mu + \left[\frac{(\sum_{i=1}^{n} u_i) - \left(\frac{n}{2}\right)}{\sqrt{\frac{n}{12}}}\right]\sigma$$

Provided the mean value and standard deviation is given, the above formula is used to generate random values in Gaussian distribution. In the simulation programs, at most of the places, n = 12 is used to further simplify the formula to:

$$t = \mu + \left[\sum_{i=1}^{12} u_i) - 6\right]\sigma$$

Chapter-3

Maintenance Strategies and System Configurations

3.1 The concept of maintenance [7]

3.1.1 Introduction

In every system, every component is bound to fail at some time. Depending on the configuration, the state of system depends on the state of individual components. When the components fail, eventually the system fails at one point. It is therefore necessary to keep maintaining the components of system from time to time to ensure its smooth functioning. The process of maintenance has a major role in efficient running of system. The technical meaning of maintenance involves functional checks, servicing, repairing or replacing of necessary devices, equipment, machinery, building infrastructure, and supporting utilities in industrial, business, governmental, and residential installations. Over time, this has come to include multiple wordings that describe various cost-effective practices to keep equipment operational; these activities take place either before or after the failure.

3.1.2 Types of maintenance:

The maintenance can be done on a component or a group of components before or after their failure. There are two important types of maintenance that we are considering in this project.

i. Corrective maintenance

Whenever a component undergoes a sudden failure, it stops working and affects the working of system as well. So, at a certain point the component either needs to be repaired or needs to be replaced by a new component to start working of the system again. This process in technical terms is called as 'corrective maintenance' of component.

- Corrective maintenance is done after the sudden failure of component.
- Corrective maintenance cannot be planned.
- Corrective maintenance is subject to Production losses as well as repair/replacement costs.

In this project, we are considering the Gaussian failure distribution for the time needed for corrective maintenance.

ii. Preventive maintenance

As the name suggests, preventive maintenance is done on a component or a group of components to prevent sudden failures and reduce the system downtime. In preventive maintenance, the component may get repaired or replaced after a certain duration of time to ensure smooth functioning of system.

- Preventive maintenance is done before sudden failure of component to prevent it from happening.
- Preventive maintenance can be, and in fact needs to be planned.
- Preventive maintenance is subject to much less production loss and repair costs as it is planned.

Similar to the corrective maintenance, we are assuming in this project that the time needed for preventive maintenance also follows Gaussian distribution.

3.1.3 Types of Preventive Maintenance

Preventive maintenance can be categorized in two major types:

i. Time-based preventive maintenance:

Time based preventive maintenance is done periodically on a component or a group of components after a certain duration of system uptime.

ii. Age based preventive maintenance:

Age based preventive maintenance is done when the component surpasses a certain age limit and has increased chances of sudden failure.

3.2 Maintenance strategies

A lot of different maintenance strategies exist that work for different system configuration. Following are some of the basic strategies considered in development of this project:

a) Run to failure

In this maintenance strategy, the system of components is allowed to run till failure. Every component undergoes corrective maintenance after sudden failure. Preventive maintenance stands no role in this type of model. This is the best strategy when the failure rate of a component does not depend much on its age and hence preventive maintenance is useless.

b) Time-based preventive maintenance

In this maintenance strategy, each component undergoes preventive maintenance after a certain duration of time assigned to it. We can even make clusters of components that undergo preventive maintenance together. A system undergoing such type of maintenance needs a preventive maintenance schedule.

c) Age-based preventive maintenance

In this maintenance strategy, each component undergoes preventive maintenance after reaching a certain age limit. This type of strategy is really helpful in case of those component which are prone to fail due to aging.

d) Time or age based preventive maintenance

This is mixed maintenance strategy where a component can undergo time-based or age-based preventive maintenance as per user specifications.

3.3 System configurations [7]

The components in a system are connected to each other through a certain configuration which decides the state system as a function of states of individual components.

Some of the common system configurations considered in this project are:

1. **Series configuration**

The series configuration of components is schematically represented as follows:

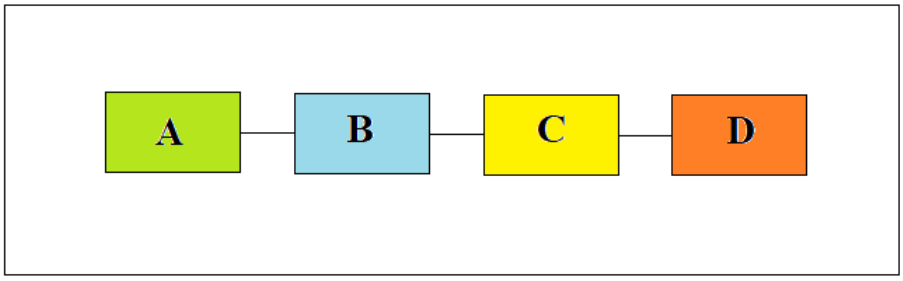


Fig. 2: Schematic of system of four components (A, B, C and D) in series configuration

When any of the components fails, the system will fail. The Reliability of system is the product of the reliabilities of individual components.

R(system) = R(A)R(B)R(C)R(D)

2. Parallel configuration

The parallel configuration of components is schematically represented as follows:

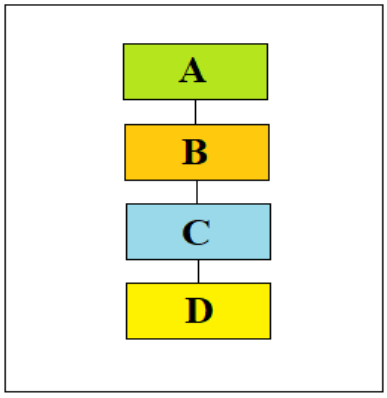


Fig. 3: Schematic of system of four components (A, B, C and D) in parallel configuration

In parallel configuration, the system is working when at least one of the components is active. The Failure probability of system is the product of the failure probabilities of the individual components.

F(system) = F(A) F(B) F(C) F(D)

i.e. R(system) = 1 – (1-R(A)) (1-R(B)) (1-R(C)) (1-R(D))

3. K out of M configuration

This is a generalized configuration of system of M components. When at least K components are active, the system is active. Once the number of active components goes below M, the system fails.

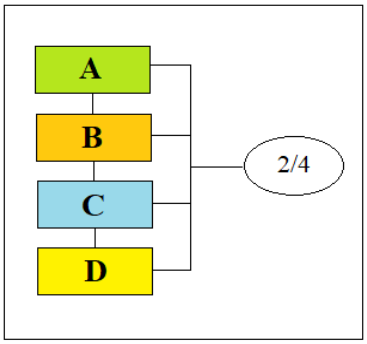


Fig. 4: Schematic of system of four components (A, B, C and D) in k out of m configuration (k=2, M=4)

Series and Parallel configurations are special cases of this configuration. When k = 1, system is in parallel and when k = m, system is in series.

4. Standby redundancy

In this configuration, a component or more are kept as standby for a primary component. The redundant components don't share any load and are in cold standby mode. When the primary component fails, the component in standby mode gets activated through a switch.

A schematic representation of standby container is as follows :

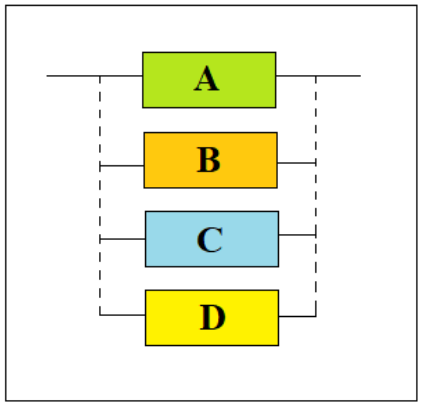


Fig. 5: Schematic of standby container of four components, where A is the primary component and rest 3 are in standby

Chapter-4

Development of individual programs for different maintenance strategies and system configurations

4.1 Abbreviations used in program file names:

- Comp = Simulation of a single component
- Block = Simulation on a block(assembly) of component
- RTF = Run to failure analysis
- TBM = Time-based (preventive) Maintenance
- ABM = Age-based (preventive) Maintenance
- TABM = Time or Age based (preventive) Maintenance
- Series = Components inside the block are in series configuration
- Parallel = Component inside the block are in parallel configuration
- Kom = Component inside the block ae in K out of M configuration
- MS = Multiple Standby mode
- PS = Perfect switch
- IPS = Imperfect switch
- CFM = Competing Failure Modes
- Allowed = Corrective as well as preventive maintenance is allowed during system uptime
- Notallowed = Corrective as well as preventive maintenance is not allowed during system uptime

4.2 Python classes used in programs

- Class Component
- Class Block
- Class failure_mode

(Refer to Appendix A for detailed description of each of these)

4.3 List of individual programs

The detail description of each of the following codes including its meaning, assumptions, input parameters, input file format, output parameters and the execution method is given in the documents that come along with all the individual codes in a separate folder.

4.3.1 Single component programs

1. Comp_RTF
2. Comp_TABM
3. Comp_MS_PS_RTF
4. Comp_MS_IPS_RTF

4.3.2 Component Block (Assembly) programs

1. Block_series_RTF
2. Block_series_TBM
3. Block_series_ABM
4. Block_series_TABM
5. Block_parallel_RTF_Allowed
6. Block_parallel_RTF_Notallowed
7. Block_parallel_TBM
8. Block_parallel_ABM
9. Block_parallel_TABM
10. Block_kom_RTF_Allowed
11. Block_kom_RTF_Notallowed
12. Block_kom_TBM
13. Block_kom_ABM
14. Block_kom_TABM

4.3.3 Multiple standby component blocks programs

1. Block_MS_PS_RTF_Allowed
2. Block_MS_PS_RTF_Notallowed
3. Block_MS_PS_TBM
4. Block_MS_PS_ABM
5. Block_MS_PS_TABM

6. Block_MS_IPS_RTF_Allowed
7. Block_MS_IPS_RTF_Notallowed
8. Block_MS_IPS_TBM
9. Block_MS_IPS_ABM
10. Block_MS_IPS_TABM

4.3.4 Competing failure mode programs

1. CFM_Block_series_RTF
2. CFM_Block_series_TBM
3. CFM_Block_series_ABM
4. CFM_Block_series_TABM
5. CFM_Block_parallel_RTF_Allowed
6. CFM_Block_parallel_RTF_Notallowed
7. CFM_Block_parallel_TBM
8. CFM_Block_parallel_ABM
9. CFM_Block_parallel_TABM
10. CFM_Block_kom_RTF_Allowed
11. CFM_Block_kom_RTF_Notallowed
12. CFM_Block_kom_TBM
13. CFM_Block_kom_ABM
14. CFM_Block_kom_TABM

4.4 Input file format

Almost every program that involves the failure data of multiple components, needs the input file in comma separated variable format (.csv). This file can be generated as well as edited using any spreadsheet software like MS Excel.

Each line in a file should contain the failure data of a component. The data to be filled in each column is mentioned below:

4.4.1 Component-block programs

Every program demands the .csv file in following format

Column 1: Sr. No.

Column 2: Name of component

Column 3: Eta value in Weibull distribution for failure (failures/hr.)

Column 4: Beta value in Weibull distribution for failure

Column 5: Mean value of time needed for corrective maintenance (hrs.)

Column 6: Standard deviation for corrective maintenance time (hrs.)

Column 7: Duration before each preventive maintenance (hrs.)

Column 8: Mean time for preventive maintenance (hrs.)

Column 9: Standard deviation for preventive maintenance time(hrs.)

Column 10: Type of preventive maintenance (0:Time based, 1: Age based)

Column 11: Initial age of components (hrs.)

Column 12: Restoration factor for corrective maintenance

Column 13: Restoration factor for preventive maintenance

A snap of a sample data file is shown below:

1	Component_1	1000	1.4	7	1	2000	5	0.4	0	0	0.1	0.05
2	Component_2	800	1.5	9	1.2	2000	7	1.1	1	0	0.2	0.1
3	Component_3	1300	1.1	8	1.1	2500	7	1	0	0	0.2	0.15
4	Component_4	1200	1.8	5	0.5	2500	4	0.5	1	0	0	0
5	Component_5	2500	1.5	19	3.5	10000	15	2	0	0	0.3	0.2

Table 1: Sample .csv file containing component failure data

4.4.2 Single component Standby programs:

Each program demands a .csv file as input that should contain following failure data for each component:

Column 1: Sr. No.

Column 2: Name of component

Column 3: Eta value in Weibull distribution for failure (failures/hr.)

Column 4: Beta value in Weibull distribution for failure

Column 5: Mean value of time needed for corrective maintenance (hrs.)

Column 6: Standard deviation for corrective maintenance time (hrs.)

Column 7: Duration before each preventive maintenance (hrs.)

Column 8: Mean time for preventive maintenance (hrs.)

Column 9: Standard deviation for preventive maintenance time(hrs.)

Column 10: Type of preventive maintenance (0:Time based, 1: Age based)

Column 11: Initial age of components (hrs.)

Column 12: Restoration factor for corrective maintenance

Column 13: Restoration factor for preventive maintenance

Column 14: Probability of successful switching to the next standby component

A snap of a sample data file is shown below:

1	Primary	1000	1.1	70	10	2000	50	4	0	0	0.1	0.05	0.9
2	standby-1	1000	1.5	90	12	2000	70	11	1	300	0.2	0.1	0.9
3	standby-2	900	1.4	80	11	2500	70	10	0	400	0.2	0.15	0.95
4	standby-3	800	1.5	50	5	2500	40	5	1	100	0.3	0.2	0.8
5	standby-4	800	1.8	60	15	10000	50	12	0	200	0.3	0.2	0

Table 2: Sample .csv file containing component failure data in standby mode

4.4.3 Component-blocks Standby programs:

Each program demands two .csv files:

i. Block data

ii. Component data

Format for Block data file:

1	component_data	3	3	0.8
2	component_data_sb1	3	8	0.9
3	component_data_sb2	3	8	0.7
4	component_data_sb3	3	8	0.8

Table 3: Sample .csv file containing data of blocks in standby mode

Column 1: Serial No.

Column 2: Name of file that contains failure data in all components in that block

Column 3: System configuration of block (1: series, 2: parallel, 3: k out of m)

Column 4: Value of k if any

Column 5: Probability of successful switch to the next block

Format for Component data file:

Column 1: Sr. No.

Column 2: Name of component

Column 3: Eta value in Weibull distribution for failure (failures/hr.)

Column 4: Beta value in Weibull distribution for failure

Column 5: Mean value of time needed for corrective maintenance (hrs.)

Column 6: Standard deviation for corrective maintenance time (hrs.)

Column 7: Duration before each preventive maintenance (hrs.)

Column 8: Mean time for preventive maintenance (hrs.)

Column 9: Standard deviation for preventive maintenance time(hrs.)

Column 10: Type of preventive maintenance (0:Time based, 1: Age based)

Column 11: Initial age of components (hrs.)

Column 12: Restoration factor for corrective maintenance

Column 13: Restoration factor for preventive maintenance

A snap of a sample file containing the failure data of components in the block is shown below:

1	Component_1	900	1.4	70	1	2000	50	0.4	0	300	0.1	0.05
2	Component_2	700	1.5	90	1.2	2000	70	1.1	1	700	0.2	0.1
3	Component_3	1200	1.1	80	1.1	2500	70	1	0	400	0.2	0.15
4	Component_4	1100	1.8	50	0.5	2500	40	0.5	1	200	0	0
5	Component_5	2400	1.5	190	3.5	10000	150	2	0	5000	0.3	0.2
6	Component_6	300	1.2	30	0.1	1000	25	0.1	1	100	0	0
7	Component_7	1100	1.6	60	0.4	3000	50	0.2	0	1000	0	0
8	Component_8	1000	2.1	80	2	2500	60	1.5	1	200	0.15	0.1
9	Component_9	1000	1.3	70	1	2500	60	0.5	0	750	0.15	0.15
10	Component_10	1500	1.7	120	1.5	5000	100	1	0	2000	0.25	0.2

Table 4: Sample .csv file containing failure data of components in the block

4.4.4 Competing failure modes programs:

Each program demands two .csv files:

i. Component data

ii. Failure mode data

Format for component data file

Column 1: Sr. No.

Column 2: Name of component

Column 3: Eta value in Weibull distribution for failure (failures/hr.)

Column 4: Beta value in Weibull distribution for failure

Column 5: Mean value of time needed for corrective maintenance (hrs.)

Column 6: Standard deviation for corrective maintenance time (hrs.)

Column 7: Type of preventive maintenance (0: time based, 1: age based)

Column 8: Restoration factor for corrective maintenance

A snap of a sample data file is shown below:

1	Component_1	cfm_comp_fm_data1	2000	5	0.4	0	0.05
2	Component_2	cfm_comp_fm_data2	2000	7	1.1	1	0.1
3	Component_3	cfm_comp_fm_data3	2500	7	1	0	0.15
4	Component_4	cfm_comp_fm_data4	2500	4	0.5	1	0

Table 5: Sample .csv file containing failure data of components in the block

Format for failure mode data file

Column 1: Sr. No.

Column 2: Name of failure mode

Column 3: Name of .csv file that contains the failure data of component

Column 4: Duration before each preventive maintenance (hrs.)

Column 5: Mean time for preventive maintenance (hrs.)

Column 6: Standard deviation for preventive maintenance time(hrs.)

Column 7: Type of preventive maintenance (0:Time based, 1: Age based)

Column 8: Initial age of components (hrs.)

Column 9: Restoration factor for preventive maintenance

A snap of a sample data file is shown below:

1	FM_1	1000	1.4	7	1	300	0.1
2	FM_2	800	1.5	9	1.2	700	0.2
3	FM_3	1300	1.1	8	1.1	400	0.2
4	FM_4	1200	1.8	5	0.5	50	0
5	FM_5	2500	1.5	19	3.5	2000	0.3

Table 6: Sample .csv file containing data of each failure mode of the component

4.5 Validation of codes

All of the above-mentioned programs are validated using a commercial reliability simulation software 'Blocksim 7.0' by Reliasoft. In all the cases, the values of system and component parameters calculated by both, the software and the python codes give pretty close results (within range of ±1%). The reason behind the small error is that Blocksim uses same seed value for random number generation at beginning of all simulations, whereas these codes don't, and hence give different simulated results in every run. Each code is corrected and modified using the data of a lot of simulation runs in different conditions and the final documentation of validation is done using 27 final test cases that cover every configuration mentioned above.

Chapter-5

Development of Failure Simulation Library 'Fsim'

5.1 Introduction

'Fsim' is the Failure simulation library developed in Python 3 which contains programs for all the basic maintenance strategies and system configurations mentioned in Chapter 3, in form of functions. This library helps us to get simulation results of given system configuration in given conditions by writing a single line of code.

Download link: https://drive.google.com/open?id=1qBzl3WFcV_MFIgLySK8CMuwHd8VM41pl

5.2 Modules in library

'Fsim' contains two modules named 'Text.py' and 'Solution.py'. Each of them is designed for a different kind of job. The Text module can be really helpful for an ordinary user who just needs the overall simulation analysis of given problem, whereas the Solution module is made for developers and programmers who want to use the values of results inside their code for solving complex problems.

5.2.1 Text.py module:

The text module will run the simulation and save it's results in a text file (.txt) file with user-given name.

The input parameters are:

- Input file:

  This is a 'comma separated variable' file (.csv) which contains the failure distribution data of components in the system. It can be easily generated and edited using a spreadsheet software like Microsoft excel.

  (The format of csv file is given in a separate word file for each function in the module)
- Time to run the simulation: (in hrs.)
  - Number of simulations:

    User can run any number of simulations to get the distribution of results. More the number of simulations, more is the time and memory needed to execute, but better is the accuracy of results.
- Output file name:

When all the simulations are complete and final results are calculated, they will be stored in a new text file with user given name.

(User should prefer to give a different name for each new file. Else, it will erase the previous data and store the new results)

Output parameters vary from function to function. But these are some of the common output parameters that are present in many of the functions:

System parameters:

- Total downtime of system
- Number of downtimes of system
- Total number of corrective maintenances in system
- Total number of preventive maintenances in system
- Availability of system

Component parameters:

- Number of corrective maintenances of individual component
- Number of preventive maintenances of individual component.

A snap of a sample output file generated by the text module is shown below:

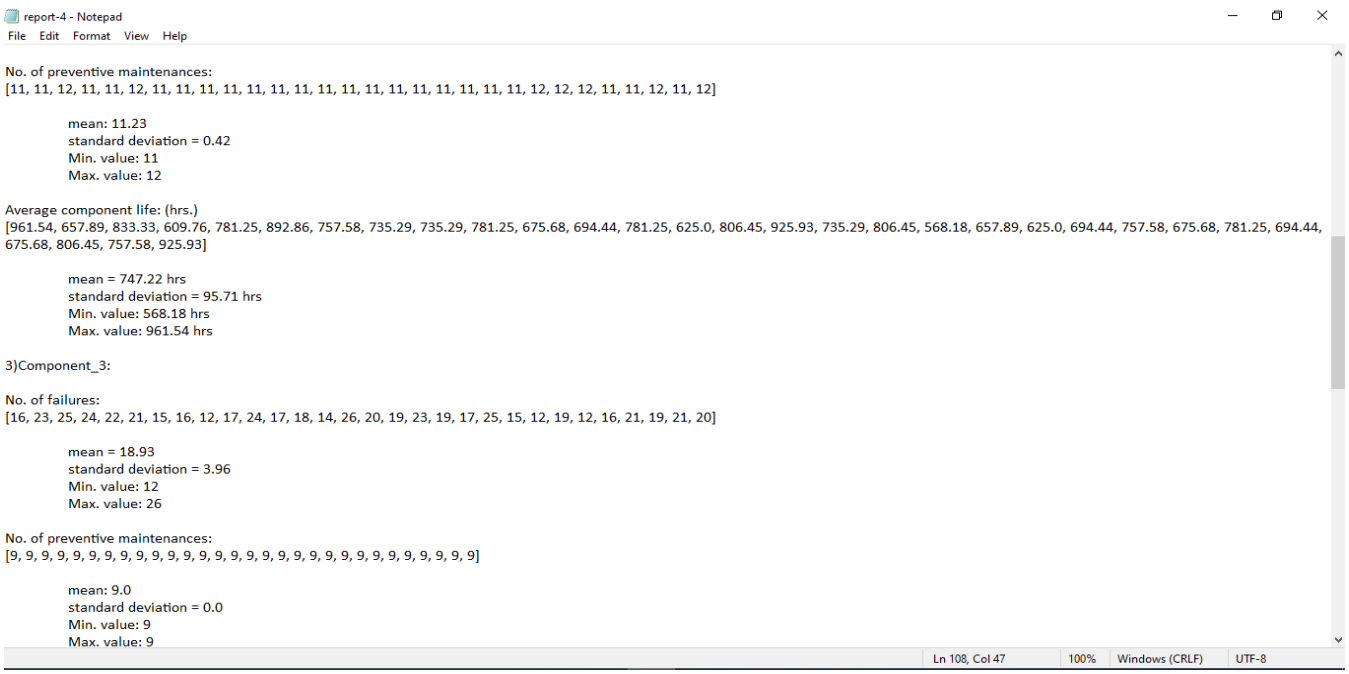


Fig.6: Snap of sample output file generated by Text module containing array, mean, standard deviation, minimum and maximum value of each parameter

5.2.2 Solution.py Module:

The solution module will run the simulation and save it's results in a user defined data format in python named 'Solution'.

The input parameters are:

- Input file:

This is a 'comma separated variable' file (.csv) which contains the failure distribution data of components in the system. It can be easily generated and edited using a spreadsheet software like Microsoft excel.

(The format of csv file is given in a separate word file for each function in the module)

- Time to run the simulation: (in hrs.)
- Number of simulations:

User can run any number of simulations to get the distribution of results. More the number of simulations, more is the time and memory needed to execute, but better is the accuracy of results.

The output:

Every function in this module will return a defined data structure named 'solution' which will contain the results of simulation stored inside it.
Every 'solution' will contain some specific parameters in it. These parameters may vary from function to function. But here are some of the common parameters listed below:

Sys_av : The availability of system
Sys_DT: Total downtime of system
Sys_UT: Total uptime of the system
Sys_NODT: Total number of system downtimes
Sys_NOCM: Total number of corrective maintenances in system
Sys_NOPM: Total number of preventive maintenances in system etc.
Comp_NOCM: Number of corrective maintenances done on individual component
Comp_NOPM: Number of preventive maintenances done on individual component

Every parameter is again a user-defined data type named ‘Para’ which contains following 5 variables:

Array: This contains the list of individual simulation results in an array(list) format

Mean: This will contain the mean value of results. (Float value)

Std: This will contain the standard deviation of results. (Float value)

Min: This will contain the minimum value among all. (Float value)

Max: This will contain the maximum value among all. (Float value)

(E.g. ‘Sol3.Sys_av.mean’ will give the mean value of system availability in the Solution named ‘Sol3’)

5.3 Classes used in library

- Class solution
- Class para
- Class component
- Class Block
- Class failure_mode

(The detailed description of each of these classes is given in Appendix-A)

Chapter-6

Demonstration of use of 'Solution.py' through a maintenance scheduling problem

6.1 Problem statement

A list of systems is given with their respective details. A system contains some n number of components in series configuration. Following data corresponding to each component is given in a csv file:

1. Component Name
2. Eta value (Weibull failure distribution)
3. Beta value (Weibull failure distribution)
4. Mean value for corrective maintenance time
5. Standard deviation for corrective maintenance time
6. Time needed for preventive maintenance of component
7. Restoration factor for corrective maintenance
8. Restoration factor for preventive maintenance
9. Initial age of the component

Following are the constraints that need to be followed:

1. The system works continuously 24hrs a day for 6 days a week.
2. Every Sunday is a holiday and system stops for that 24 hours.
3. Whenever a component fails, the system stops working till the component gets repaired/replaced.
4. Preventive maintenance of components is always preferred on Sundays as there is no production loss.
5. The production loss on weekdays is assumed to be directly proportional to the system downtime.
6. The cost for maintenance of a component is also assumed to be directly proportional to the maintenance time needed. However, the cost multiplier (cost/hr.) value at weekend will be smaller compared to cost multiplier on weekdays. (E.g. Rs. 5000/hr. on weekdays and Rs. 1000/hr. on weekend)
7. The complete system undergoes Overhaul after a certain duration (e.g. 2yrs)
8. Components undergo preventive maintenance in series. (At a time, a single component will be treated)

We need to find an optimal weekly preventive maintenance schedule for this system for given overhaul duration.

A snap of input files is given below:

Case No.	Input Filename	Weekday cost (Rs.)	Weekend cost (Rs.)	Overhaul period (hrs)
1	case1.csv	10000	2000	15000
2	case2.csv	2000	450	15000
3	case3.csv	7500	2000	20000
4	case4.csv	5000	750	7500
5	case5.csv	10000	2500	12500
6	case6.csv	9000	1000	8500
7	case7.csv	2500	400	16000
8	case8.csv	1000	150	14000
9	case9.csv	4000	1000	7500
10	case10.csv	10000	1500	14500
11	case11.csv	7000	1800	8000
12	case12.csv	11000	2400	16000
13	case13.csv	25000	4000	40000
14	case14.csv	2000	200	15000
15	case15.csv	4500	500	10000

Table 7:.csv file containing data of all the cases

1	component-1	4100	3.1	15	9.1	0	2.6	0	0.25	0.18
2	component-2	9900	2.7	15	9.1	0	4.1	0	0.32	0.14
3	component-3	4800	2.3	19	8	0	4.4	0	0.37	0.17
4	component-4	9200	1.1	18	6.7	0	2.9	0	0.09	0.08
5	component-5	3500	2.3	19	9.7	0	4.5	0	0.34	0.14
6	component-6	4200	2.1	16	9	0	2.7	0	0.12	0.15
7	component-7	4600	3.5	19	8.7	0	4.8	0	0.16	0.17
8	component-8	1500	2.9	18	7.8	0	2.6	0	0.24	0.04
9	component-9	7900	3.4	14	7.9	0	2.9	0	0.23	0.01
10	component-10	5400	2.4	14	5.7	0	2.2	0	0.04	0.08
11	component-11	9600	3.2	17	7.8	0	4.2	0	0.21	0.13
12	component-12	10000	1.3	16	6.4	0	3.2	0	0.03	0.12
13	component-13	2300	1.4	14	5.7	0	3	0	0.08	0.07
14	component-14	8400	1.1	12	9.5	0	2.6	0	0.39	0.1
15	component-15	6400	1.4	12	8.5	0	4	0	0.05	0.01
16	component-16	6200	1.9	15	9.9	0	4.9	0	0.16	0.07
17	component-17	8700	1.3	17	7.8	0	3.4	0	0.15	0.16
18	component-18	8000	3.4	11	9.3	0	4.2	0	0.34	0.15
19	component-19	7500	3.2	15	6	0	3.3	0	0.33	0.06
20	component-20	6000	1.7	18	8.1	0	2.1	0	0.12	0.03

Table 8: Sample .csv file containing failure data of components in one of the cases

6.2 Solution

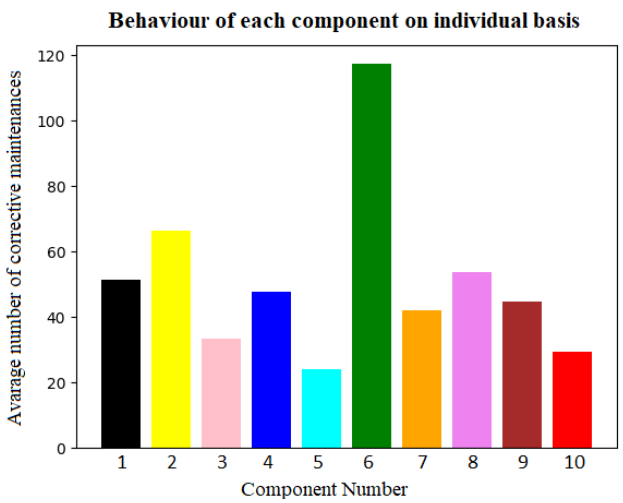
6.2.1 Assumptions to proceed

To solve this problem schedule, we can proceed by making following assumptions:

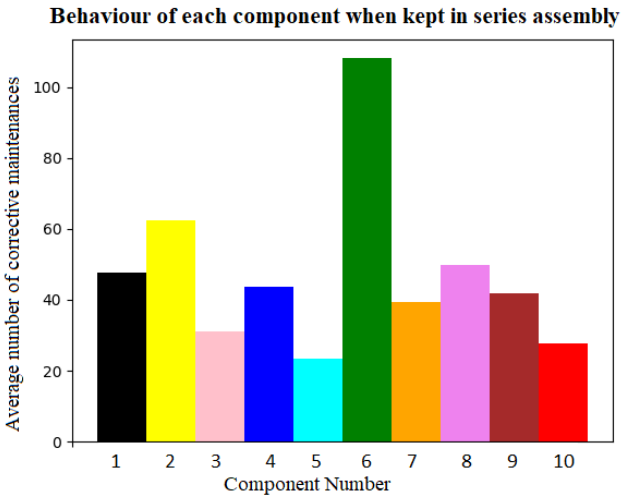
1) Every component behaves the same way he does in an assembly as it does individually for given runtime.
2) Every component follows a curve for total maintenance cost based on downtime associated with corrective as well as preventive maintenance having a global minimum value.

6.2.2 Verification of assumptions

Assumption 1:



(a)



(b)

Fig.7a) Bar graph that show the mean value of number of failures of a component run on individual basis for a given runtime.
Fig.7b) Bar graph that show the mean value of number of failures of all those components in series configuration

These bar graphs clearly show us the similarity in nature of number of failures of all the components on individual basis as well as when kept in a block in series configuration and hence validate our first assumption

Assumption 2:

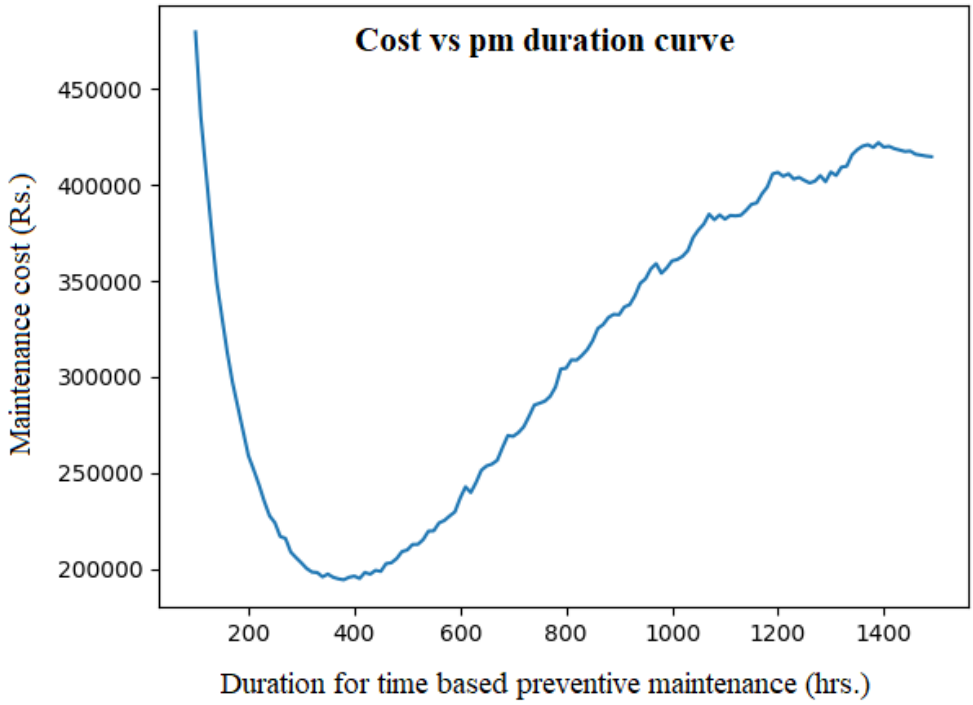


Fig. 8a: The cost vs preventive maintenance duration curve (not considering different cost multiplier on weekends)

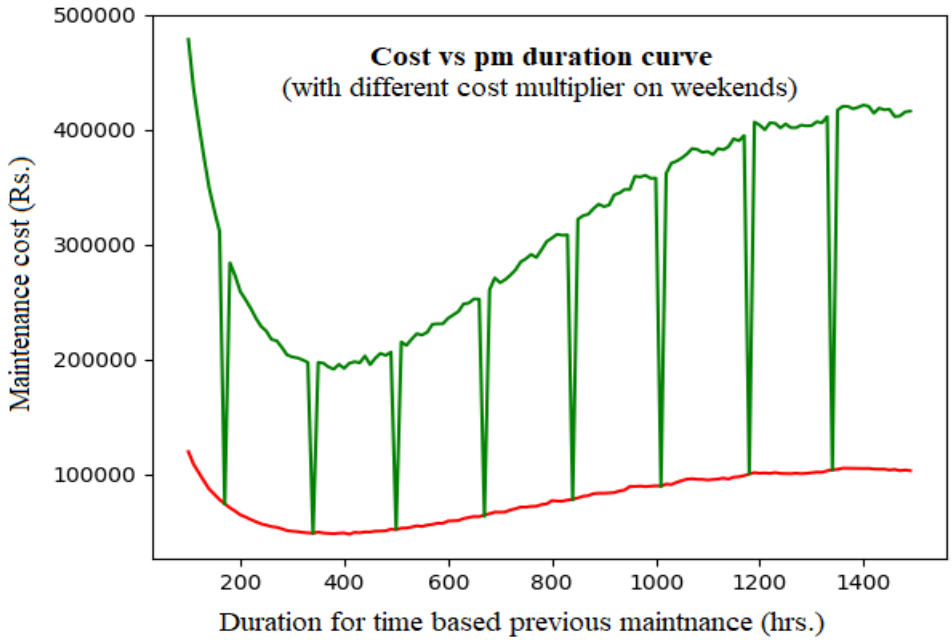


Fig. 8b: The cost vs preventive maintenance duration curve considering different cost multiplier on weekends.

These two graphs verify that there exists a value of preventive maintenance duration for every component for which the maintenance cost needed is minimum.

Using these two assumptions, we can proceed to solve our problem.

6.2.3 The process flow

The process of solving for the optimal preventive maintenance schedule for the given system goes through 4 major steps:

1. Calculating the optimal preventive maintenance duration for a component in terms of weeks on individual basis.
2. Generating a raw maintenance schedule for each week in the duration before the system overhaul.
3. Spotting out the weeks in which the total maintenance time needed exceeds the daily limit (e.g. 21 hrs. with some margin)
4. Adjusting the components in overloaded weeks in the adjecent (comparatively) free weeks to get the final maintenance schedule.

6.2.4 Snap of one of the result files

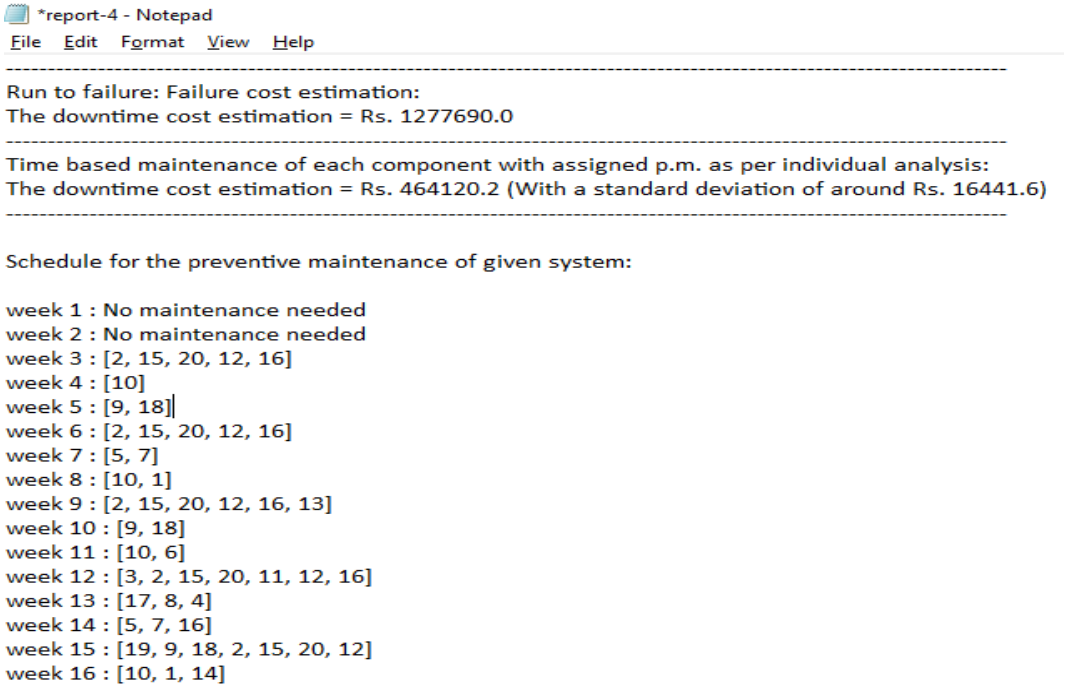


Fig. 9: Snap of a sample result file that contains the weekly schedule for preventive maintenance of all components in given system for the overhaul duration.

6.3 Results of Simulation:

Being a large problem, it wasn't possible to include the detailed result of the complete problem in this space. Hence a simplified version of this problem is given in Appendix-B with the complete program along with the results of the simulation.

Chapter-7

Conclusion and Future Scope

7.1 Conclusion

The failure simulation library 'Fsim' helps us to solve failure simulation problems using two of its modules, Text.py and Solution.py. We can get the required solution by calling the corresponding functions through a single line of code. The Text module helps an ordinary user to get the results of simulations in form of a text file, whereas the Solution module is very useful for a developer or programmer to directly use the calculated values in simulations in his own program.

There are various commercial softwares available in market which can give more detailed and sophisticated results of simulations compared to this library. But this library has its own benefits over them.

The Text module has an only advantage that it's a free module and can be used by any user who may not afford or access the commercial Reliability softwares. But the Solution module has his own strengths. If the results of failure simulations are to be used in a complex algorithm which may involve hundreds and thousands of iterations, it might not be practically possible to do it in time by manually entering the data for each case in the softwares. But using the Solution module, it is possible to use the simulation results directly in the codes and solve complicated problems. In this way, this library can be used by common users who know basic python, as well as developers who want to implement the results in their own algorithms.

7.2 Future scope for growth and improvement

This library, although capable of performing all kind of basic failure simulations for most of the basic maintenance strategies and system configurations, is still in its incipient form and can grow in every possible direction.

Some of the possible future developments in this library:

1. Addition of more modules in library which might be capable of giving the simulation results in more aesthetic and graphical form.
2. Addition of functions that can simulate more complicated component networks.
3. Converting the library into a Graphical User Interface (GUI) which will make it much easier to use
4. Adding more and more variables to enable this library to solve more complicated real-world problems.

References

1. *Inverse transform method, 2010 by Karl Sigman , http://www.columbia.edu/~ks20/4404-Sigman/4404-Notes-ITM.pdf*
2. *Institute of Electrical and Electronics Engineers (1990) IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY ISBN 1-55937-079-3*
3. Barker, Elaine; Barker, William; Burr, William; Polk, William; Smid, Miles (July 2012). *"Recommendation for Key Management" (PDF). NIST* Special Publication 800-57. *NIST. Retrieved 19 August 2013.*
4. *Releases – numpy/numpy". Retrieved 11 November 2019 – via GitHub.*
5. *Releases – matplotlib"*
6. *Charles Eblings, An introduction to Reliability and Maintainability engineering*

Appendices

Appendix A

List of all the classes used in library

- **Class Component**
  This class contains all the information and functions associated with a component.

```
class component():

    def __init__(self):
        self.a = 0
        self.b = 0
        self.mean = 0
        self.s = 0
        self.component_time = 0
        self.component_name = 'none'
        self.number_of_repairs = 0
        self.number_of_maintenances = 0
        self.maintenance_duration = 0
        self.maintenance_mean = 0
        self.maintenance_s = 0
        self.maintenance_type = 0
        self.RF = 0
        self.RFC = 0
        self.RFP = 0
        self.initial_age = 0
        self.fm_filename = ''
        self.fm_list = []
        self.nofm = 1

        self.uptimes = []
        self.downtimes = []
        self.maintenances = []
        self.downtime_track = []

        self.total_uptimes = []
        self.total_downtimes =  []
        self.total_failures = []
        self.total_maintenances = []
        self.total_number_of_repairs = []
        self.total_number_of_maintenances = []
        self. availability = []
        self.average_life = []
        self.total_switchfails = []
```

```
    def generate_failure_time(self):
        u = random.uniform(0,1)
        life = self.a * (math.log((math.e**((self.component_time/self.a)**self.b)/u))) ** (1 / self.b) -
self.component_time
        return(life)

    def generate_repair_time(self):
        temp_total = 0

        for i in range(0, 12):
            u = random.uniform(0, 1)
            temp_total += u

        N = self.mean + self.s * (temp_total - (12 / 2)) / (12 / 12) ** 0.5
        return (N)

    def generate_maintenance_time(self):
        temp_total = 0
        for i in range(0, 12):
            u = random.uniform(0, 1)
            temp_total += u

        N = self.maintenance_mean + self.maintenance_s * (temp_total - (12 / 2)) / (12 / 12) ** 0.5
        return (N)
```

- **Class Block**

  Contains the details of a Block of components.

```
    class block():

        def __init__(self):

            self.filename = ''
            self.type = 1      #1:series 2:Parallel 3:component
            self.state = 1
            self.number_of_components = 0
            self.component_state = []
            self.component_list = []
            self.switchprob = 1
            self.number_of_failures = 0
            self.k = 1

            self.total_failures = []
            self.total_uptimes = []
            self.total_switchfails = []
            self.availability = []
            self.average_life = []
```

- **Class failure_mode**

  Contains the details of different failure modes.

```
class failure_mode():

    def __init__(self):
        self.a = 0
        self.b = 0
        self.mean = 0
        self.s = 0
        self.component_time = 0
        self.initial_age = 0
        self.fm_name = 'none'
        self.downtime = []
        self.average_life = []
        self.number_of_repairs = 0
        self.RFC = 0

        self.total_number_of_repairs = []

    def generate_failure_time(self):
        u = random.uniform(0, 1)
        life = self.a * (math.log((math.e ** ((self.component_time / self.a) ** self.b) / u))) ** (
                1 / self.b) - self.component_time
        return (life)

    def generate_repair_time(self):
        temp_total = 0

        for i in range(0, 12):
            u = random.uniform(0, 1)
            temp_total += u

        N = self.mean + self.s * (temp_total - (12 / 2)) / (12 / 12) ** 0.5
        return (N)
```

- **Class Solution**

  Contains the details of solution returned by the Solution.py module

```
class solution():

    def __init__(self):
        self.sys_av = para()
        self.sys_DT = para()
```

```
        self.sys_UT = para()
        self.sys_NODT = para()
        self.sys_AL = para()
        self.sys_NOCM = para()
        self.sys_NOPM = para()
        self.comp_NOCM = []
        self.comp_NOPM = []
        self.comp_AL = []
        self.comp_UT = []
        self.comp_SF = []
        self.block_SF = []
        self.fm_NOCM =[]
```

- **Class para**

  Contains the details like mean, standard deviation, minimum and maximum value of each parameter.

```
class para():
    def __init__(self):
        self.array = []
        self.mean = 0
        self.std =  0
        self.min = 0
        self.max= 0

    def set(self):
        if len(self.array) is 0:
            self.mean = 0
            self.std = 0
            self.min = 0
            self.max = 0

        else:
            self.mean = round(np.mean(self.array), 2)
            self.std = round(np.std(self.array), 2)
            self.min = min(self.array)
            self.max = max(self.array)
```

Appendix B

Simplified version of maintenance scheduling problem

In this problem, n components along with their failure data are generated randomly.

Code:

```
from Fsim import Solution as S
import random
import math
import csv
import numpy as np

class component():

    def __init__(self):
        self.a = 0
        self.b = 0
        self.mean = 0
        self.s = 0
        self.component_time = 0
        self.component_name = 'none'
        self.number_of_repairs = 0
        self.number_of_maintenances = 0
        self.maintenance_duration = 0
        self.maintenance_mean = 0
        self.maintenance_s = 0
        self.maintenance_type = 0
        self.RF = 0
        self.RFC = 0
        self.RFP = 0
        self.initial_age = 0
        self.fm_filename = ''
        self.fm_list = []
        self.nofm = 1

        self.uptimes = []
        self.downtimes = []
        self.maintenances = []
        self.downtime_track = []

        self.total_uptimes = []
        self.total_downtimes =  []
        self.total_failures = []
        self.total_maintenances = []
        self.total_number_of_repairs = []
        self.total_number_of_maintenances = []
```

```
        self. availability = []
        self.average_life = []
        self.total_switchfails = []



    def generate_failure_time(self):
        u = random.uniform(0,1)
        life = self.a * (math.log((math.e**((self.component_time/self.a)**self.b)/u))) ** (1 / self.b) -
self.component_time
        return(life)

    def generate_repair_time(self):
        temp_total = 0

        for i in range(0, 12):
            u = random.uniform(0, 1)
            temp_total += u

        N = self.mean + self.s * (temp_total - (12 / 2)) / (12 / 12) ** 0.5
        return (N)

    def generate_maintenance_time(self):
        temp_total = 0

        for i in range(0, 12):
            u = random.uniform(0, 1)
            temp_total += u

        N = self.maintenance_mean + self.maintenance_s * (temp_total - (12 / 2)) / (12 / 12) ** 0.5
        return (N)


def opt_pmtime_singlecomp(comp,j):

    x = time_to_run//144
    min_c = time_to_run*DTC_weekday
    min_x = 1
    count = 0
    for i in range(1,x+1):
        Sol =
S.Comp_TABM(comp.component_name,comp.a,comp.b,comp.mean,comp.s,0,i*144,0,0,comp.initial_age,c
omp.RFC,comp.RFP,time_to_run,1000)
        cost = Sol.sys_DT.mean*DTC_weekday + Sol.sys_NOPM.mean*pm_time[j]*DTC_weekend

        if cost < min_c:
            min_c= cost
            min_x = i
        else:
            count+=1
```

```
        if count>5:
            break

    print(comp.component_name, " : ",min_x,"weeks")

    return min_x*144

no_of_comp = int(input("Enter number of random components to generate: "))
time_to_run = int(input("Enter the overhaul time for system: "))
DTC_weekday = int(input("Enter the Downtime cost due to production loss on weekdays: "))
DTC_weekend= int(input("Enter the Downtime cost for maintenance at weekends: "))
weeks = time_to_run//144

component_list = []
pm_time = []

with open('problem_gen_data.csv','w+',newline='') as data:
    data.truncate()
    writer = csv.writer(data)

    for i in range(0, no_of_comp):
        name = 'component-' + str(i + 1)
        random_eta = round(random.uniform(1500, 10000), -2)
        random_beta = round(random.uniform(1.1, 3.5), 1)
        random_CM_mean = round(random.uniform(10, 20), 0)
        random_CM_std = round(random.uniform(5, 10), 1)
        random_PM_mean = round(random.uniform(2, 5), 1)
        random_PM_std = round(random.uniform(0, 1), 1)
        random_RFC = round(random.uniform(0, 0.4), 2)
        random_RFP = round(random.uniform(0, 0.2), 2)

        trial = component()
        trial.component_name = name
        trial.a = random_eta
        trial.b = random_beta
        trial.mean = random_CM_mean
        trial.s = random_CM_std
        trial.maintenance_mean = 0
        trial.maintenance_s = 0
        trial.maintenance_type = 0
        trial.initial_age = 0
        trial.RFC = random_RFC
        trial.RFP = random_RFP

        pm_time.append(random_PM_mean)


writer.writerow([i+1,trial.component_name,trial.a,trial.b,trial.mean,trial.s,0,trial.maintenance_mean,trial.mai
ntenance_s,trial.maintenance_type,trial.initial_age,trial.RFC,trial.RFP])
```

```
        component_list.append(trial)

print("pmtime:",pm_time)

print("\nRun to failure analysis for individual component: ")
print("-----------------------------------------------------------------------------------------------------------------------------------------")
c=0

for i in range(0,no_of_comp):

    Sol =
S.Comp_RTF(component_list[i].component_name,component_list[i].a,component_list[i].b,component_list[i
].mean,component_list[i].s,component_list[i].initial_age,component_list[i].RFC,time_to_run,1000)
    print("\nNo. of failures of ",component_list[i].component_name," : ", Sol.sys_NOCM.mean)
    print("Total component downtime = ",Sol.sys_DT.mean,"hrs.")
    print("Failure cost = Rs.",round(Sol.sys_DT.mean*DTC_weekday,2))
    c += Sol.sys_DT.mean*DTC_weekday

print("\nSum of individual costs = Rs.",round(c,2))
print("-----------------------------------------------------------------------------------------------------------------------------------------")

print("\nOptimal preventive maintenance time for individual component (Planned on weekends): ")
print("-----------------------------------------------------------------------------------------------------------------------------------------")
c = 0

ind_pmtime = []
ind_week =[]

for i in range(0,no_of_comp):
    time = opt_pmtime_singlecomp(component_list[i],i)
    ind_pmtime.append(time)
    Sol =
S.Comp_TABM(component_list[i].component_name,component_list[i].a,component_list[i].b,component_lis
t[i].mean,component_list[i].s,pm_time[i],time,component_list[i].maintenance_mean,component_list[i].maint
enance_s,component_list[i].initial_age,component_list[i].RFC,component_list[i].RFP,time_to_run,1000)
    #print("Corrective maintenances: ",Sol.sys_NOCM.mean)
    #print("Preventive maintenances: ",Sol.sys_NOPM.mean)
    #print("Optimal preventive maintenance duration for  ",component_list[i].component_name," : ", time)
    #print("Failure cost =
Rs.",Sol.sys_DT.mean*DTC_weekday+Sol.sys_NOPM.mean*pm_time[i]*DTC_weekend,"\n")
    c += Sol.sys_DT.mean*DTC_weekday+Sol.sys_NOPM.mean*pm_time[i]*DTC_weekend

print("\nSum of individual costs = Rs.",round(c,2))
print("-----------------------------------------------------------------------------------------------------------------------------------------")

print("\nTIME FOR GROUP ANALYSIS:\n")

with open('problem_gen_data1.csv','w+',newline='') as data:
    data.truncate()
```

```
    f = open('problem_gen_data.csv', 'r')
    writer = csv.writer(data)
    with f:
        reader = csv.reader(f)
        for row in reader:
            a = row
            a[6] = ind_pmtime[int(a[0])-1]
            writer.writerow(a)

print("-----------------------------------------------------------------------------------------------------------------------")
print("\nRun to failure: Failure cost estimation: \n")

Sol = S.Block_series_RTF('problem_gen_data', time_to_run, 1000)

print("The downtime cost = Rs.", DTC_weekday*Sol.sys_DT.mean)
print("-----------------------------------------------------------------------------------------------------------------------")

print("\nTime based maintenance of each component with assigned p.m. as per individual analysis: \n")

Sol1 = S.Block_series_TBM('problem_gen_data1', time_to_run, 1000)
cost = 0
for i in range(0,no_of_comp):
    cost += Sol1.comp_NOPM[i].mean*DTC_weekend*pm_time[i]
print("The downtime cost = Rs.",DTC_weekday*Sol1.sys_DT.mean+cost)
print("-----------------------------------------------------------------------------------------------------------------------")

for i in ind_pmtime:
    ind_week.append(int(round(i/144,0)))

print("Weeks: ",ind_week)

print("\nLet's see what happens if we use these values: \n")

mtime = []
comps = []

ind_week2 =[]
for i in ind_week:
    if i not in ind_week2:
        ind_week2.append(i)

ind_week2.sort()
freq = []
for i in ind_week2:
    count = 0
    for j in ind_week:
        if j is i:
            count += 1
    freq.append(count)
print("\nind_week2: ",ind_week2)
```

```
print("\nfreq: ",freq)

comps2 = []

for i in ind_week2:
    comps2.append([])

for i in range(0,len(ind_week)):
    for j in range(0,len(ind_week2)):
        if ind_week[i] is ind_week2[j]:
            comps2[j].append(i+1)

print("\ncomps2: ",comps2)

mlist2 = []
for i in range (0,weeks):
    mlist2.append([])

for i in range(0,len(comps2)):
    j = len(comps2[i])
    for k in range(0,j):
        for w in range(0,weeks):
            if (w+1)%ind_week[i] is 0:
                mlist2[w].append(comps2[i][k])

print("\nmlist2: ",mlist2)

mtime = []

for i in range(0, weeks):
    sum = 0
    for j in mlist2[i]:
        sum += pm_time[j - 1]
    mtime.append(round(sum, 1))

print("\n", mtime)

faulty_weeks = []
for i in range(0,weeks):
    if mtime[i]>21:
        faulty_weeks.append(i+1)

print("\nFaulty weeks: ",faulty_weeks)

#Faulty weeks adjustment:
Issues = 0
for i in faulty_weeks:
    if mtime[i-2] is 0:
        if mtime[i-2] + mtime[i-1]<38:
            Issues += 0
```

```
        else:
            Issues += 0.1
    elif mtime[i] is 0:
        if mtime[i-1]+mtime[i] <38:
            Issues += 0
        else:
            Issues += 0.1
    else:
        if mtime[i-1]+mtime[i]+mtime[i-2] <62:
            Issues += 0
        else:
            Issues += 1
print("\nIssues: ",Issues)


print("\nNow let's deal with these faulty weeks: ")

itcount = 0

while len(faulty_weeks) > 0:

    i = faulty_weeks[0]

    a = mlist2[i-2]
    b = mlist2[i-1]
    c = mlist2[i]

    p = 0
    q = 0
    r = 0

    for num in a:
        p += pm_time[num-1]
    for num in b:
        q += pm_time[num-1]
    for num in c:
        r += pm_time[num-1]

    while(q>21):

        tc = b[-1]
        if p<r:
            a.append(tc)
        else:
            c.append(tc)

        del b[-1]

        p = 0
        q = 0
```

```
        r = 0

        for num in a:
            p += pm_time[num - 1]
        for num in b:
            q += pm_time[num - 1]
        for num in c:
            r += pm_time[num - 1]

    mlist2[i - 2] = a
    mlist2[i - 1] = b
    mlist2[i] = c

    itcount += 1

    print("\nAfter iteration",itcount,": ")

    print("\n",mlist2)

    new_faulty_weeks = []
    mtime = []

    for i in range(0,weeks):
        sum = 0
        for j in mlist2[i]:
            sum += pm_time[j-1]
        mtime.append(round(sum,1))

    print("\n", mtime)

    for i in range(0, weeks):
        if mtime[i] > 21:
            new_faulty_weeks.append(i + 1)

    print("\nFaulty weeks: ", new_faulty_weeks)

    faulty_weeks = new_faulty_weeks

    if itcount>weeks/2:
        break


#Printing the results:

print("\n\nSchedule for the preventive maintenance of given system: ")

for i in range(0,weeks):
    if len(mlist2[i]) is 0:
        print('week', i + 1, ": No maintenance needed ")
    else:
```

```
    print('week', i + 1, ":", mlist2[i])
```

Input

Enter number of random components to generate: 25

Enter the overhaul time for system: 15000 hrs. (2 years i.e. 104 weeks)

Enter the Downtime cost due to production loss on weekdays: 5000 Rs.

Enter the Downtime cost for maintenance at weekends: 1000 Rs.

Output Results

Run to failure: Failure cost estimation:

The downtime cost = Rs. 8149000.0

Time based maintenance of each component with assigned p.m. as per individual analysis:

The downtime cost = Rs. 2572478.0

Schedule:

Schedule for the preventive maintenance of given system:

week 1: No maintenance needed
week 2: No maintenance needed
week 3: [17, 25, 16, 18]
week 4: [24, 19, 4]
week 5: [3, 5, 8, 12, 21, 2]
week 6: [17, 25, 16, 18, 6]
week 7: [7, 13]
week 8: [24]
week 9: [17, 25, 16, 18, 4]
week 10: [3, 5, 8, 12, 21, 2]
week 11: [11, 20, 23, 19]
week 12: [24, 17, 25, 16, 18]
week 13: [13, 6]
week 14: [7, 1, 19, 16, 17]
week 15: [3, 5, 8, 12, 21, 2]
week 16: [24, 15, 18, 25, 4, 14]
week 17: [6, 18]

week 18: [9, 10, 22, 17, 25]
week 19: [13, 16, 19, 4, 24]
week 20: [3, 5, 8, 12, 21, 2]
week 21: [7, 17, 25, 16, 18]
week 22: [11, 20, 23]
week 23: [13, 6]
week 24: [24, 17, 25, 16, 18]
week 25: [3, 5, 8, 12, 21, 2]
week 26: [19, 4]
week 27: [17, 25, 16, 18]
week 28: [7, 24, 14]
week 29: [1, 19, 6, 16, 17]
week 30: [3, 5, 8, 12, 21, 2]
week 31: [15, 13, 18, 25, 4]
week 32: [24, 16]
week 33: [11, 20, 23, 17, 25]
week 34: [18, 19, 4, 2]
week 35: [7, 3, 5, 8, 12, 21]
week 36: [9, 10, 22, 24, 17]
week 37: [13, 6, 18, 16, 25]
week 38: No maintenance needed
week 39: [17, 25, 16, 18]
week 40: [3, 5, 8, 12, 21, 2]
week 41: [19, 4, 24]
week 42: [7, 17, 25, 16, 18]
week 43: [13, 6, 14, 17]
week 44: [11, 20, 23, 24, 16]
week 45: [3, 5, 8, 12, 21, 2]
week 46: [15, 1, 19, 18, 25]
week 47: [4, 13]
week 48: [24, 17, 25, 16, 18]
week 49: [7, 6, 19, 4]
week 50: [3, 5, 8, 12, 21, 2]
week 51: [17, 25, 16, 18]
week 52: [24]
week 53: [13, 6, 18, 16, 23]
week 54: [9, 10, 22, 17, 25]
week 55: [3, 5, 8, 12, 21]
week 56: [7, 24, 19, 4, 2, 20]
week 57: [17, 25, 16, 18, 11]
week 58: [24]
week 59: [1, 19, 6, 16, 17]
week 60: [3, 5, 8, 12, 21, 2]
week 61: [15, 13, 18, 25, 4]
week 62: [14]
week 63: [7, 17, 25, 16, 18]
week 64: [24, 19, 4]
week 65: [3, 5, 8, 12, 21, 2]
week 66: [11, 20, 23, 17, 25]
week 67: [13, 6, 18, 16]

week 68: [24]
week 69: [17, 25, 16, 18]
week 70: [7, 3, 5, 8, 12, 21]
week 71: [19, 4, 2, 25]
week 72: [9, 10, 22, 24, 17]
week 73: [13, 6, 18, 16]
week 74: [15, 19, 16, 17]
week 75: [3, 5, 8, 12, 21, 2]
week 76: [24, 1, 18, 25, 4, 14]
week 77: [7, 11, 20, 23]
week 78: [17, 25, 16, 18, 6]
week 79: [13, 19, 4, 24]
week 80: [3, 5, 8, 12, 21, 2]
week 81: [17, 25, 16, 18]
week 82: No maintenance needed
week 83: [13, 6, 18]
week 84: [7, 24, 17, 25, 16]
week 85: [3, 5, 8, 12, 21, 2]
week 86: [19, 4]
week 87: [17, 25, 16, 18, 14]
week 88: [11, 20, 23, 24, 8]
week 89: [15, 19, 6, 16, 17]
week 90: [9, 10, 22, 3, 5, 21]
week 91: [7, 1, 13, 18, 25]
week 92: [24, 12, 2, 4]
week 93: [17, 25, 16, 18]
week 94: [19, 4]
week 95: [3, 5, 8, 12, 21, 2]
week 96: [24, 17, 25, 16, 18]
week 97: [13, 6]
week 98: [7, 18, 16]
week 99: [11, 20, 23, 17, 25]
week 100: [3, 5, 8, 12, 21, 2]
week 101: [19, 4, 24]
week 102: [17, 25, 16, 18, 6]
week 103: [13]
week 104: [24]
