DEVELOPMENT OF STRAIN-BASED FATIGUE LIFE CALCULATION SOFTWARE FOR VARIABLE AMPLITUDE LOADING DATA

MOHD RADZI BIN ABD RASHID

BACHELOR OF ENGINEERING

UNIVERSITI MALAYSIA PAHANG

2010

UNIVERSITI MALAYSIA PAHANG

BORANG PENGESAHAN STATUS TESIS

JUDUL: **DEVELOPMENT OF STRAIN-BASED FATIGUE LIFE CALCULATION SOFTWARE FOR VARIABLE AMPLITUDE LOADING DATA**

SESI PENGAJIAN: **2010/2011**

Saya **MOHD RADZI BIN ABD RASHID (831114-01-5735)**
(HURUF BESAR)

mengaku membenarkan tesis (Sarjana Muda/~~Sarjana~~ /~~Doktor Falsafah~~)* ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Tesis adalah hakmilik Universiti Malaysia Pahang (UMP).
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan tesis ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. **Sila tandakan (√)

☐	**SULIT**	(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)
☐	**TERHAD**	(Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan)
√	**TIDAK TERHAD**	

Disahkan oleh:

(TANDATANGAN PENULIS)

Alamat Tetap:

4004-3 JLN MENGKUANG,
KG MAK CHILI,
24000 KEMAMAN,
TERENGGANU.

Tarikh: **06 DISEMBER 2010**

(TANDATANGAN PENYELIA)

Nama Penyelia:

CHE KU EDDY NIZWAN BIN CHE KU HUSIN

Tarikh: **06 DISEMBER 2010**

CATATAN:* Potong yang tidak berkenaan.

** Jika tesis ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali tempoh tesis ini perlu dikelaskan sebagai atau TERHAD.
Tesis dimaksudkan sebagai tesis bagi Ijazah doktor Falsafah dan Sarjana secara Penyelidikan, atau disertasi bagi pengajian secara kerja kursus dan penyelidikan, atau Laporan Projek Sarjana Muda (PSM).

DEVELOPMENT OF STRAIN-BASED FATIGUE LIFE CALCULATION SOFTWARE FOR VARIABLE AMPLITUDE LOADING DATA

MOHD RADZI BIN ABD RASHID

Thesis submitted in partial fulfilment of the requirements
for the award of the degree of
Bachelor of Mechanical Engineering

Faculty of Mechanical Engineering
UNIVERSITI MALAYSIA PAHANG

DECEMBER 2010

UNIVERSITI MALAYSIA PAHANG

FACULTY OF MECHANICAL ENGINEERING

I certify that the thesis entitled *"Development of strain-Based Fatigue Life Calculation Software for Variable Amplitude Loading Data"* is written by *Mohd Radzi Bin Abd Rashid*. We have examined the final copy of this thesis and in our opinion; it is fully adequate in terms of scope and quality for the award of the degree of Mechanical Engineering. We herewith recommend that it be accepted in fulfilment of the requirements for the degree of Mechanical Engineering.

Mr Abdul Rahim Bin Ismail Signature
Lecturer Faculty of Mechanical Engineering
Universiti Malaysia Pahang

SUPERVISOR'S DECLARATION

I hereby declare that we have checked this project report and in our opinion this project is satisfactory in terms of scope and quality for the award of the degree of Bachelor of Mechanical Engineering.

Signature:

Name of Supervisor: CHE KU EDDY NIZWAN BIN CHE KU HUSIN

Position: Lecturer Faculty of Mechanical Engineering

Date: 6 DECEMBER 2010

STUDENT'S DECLARATION

I hereby declare that the work in this report is my own except for quotations and summaries which have been duly acknowledged. The report has not been accepted for any degree and is not concurrently submitted for award of other degree.

Signature:

Name: MOHD RADZI BIN ABD RASHID

ID Number: MA08017

Date: 6 DECEMBER 2010

Special Dedication to my family members,
my friends, my fellow colleague
and all faculty members

For all your care, support and believe in me.

ACKNOWLEDGEMENT

First and foremost, I am very grateful to the Almighty ALLAH S.W.T for giving me this opportunity to accomplish my Final Year Project.

Firstly, I wish to express my deep gratitude to my Supervisor, Mr Che Ku Eddy Nizwan Bin Che Ku Husin for all his valuable guidance, assistance and support all through this work.

Secondly, I wish to thank lecturers for their suggestions and support on this project. Their comments on this project are greatly appreciated. My thanks are also to all my friends who have involved and helped me in this project.

Most importantly, I extent my gratitude to my parents who have encouraged me throughout my education and I will always be grateful for their sacrifice, generosity and love.

ABSTRACT

This thesis presents the development of strain-based fatigue life calculation software for variable amplitude loading data. The main objective of this study is to develop calculation software for fatigue life prediction using MATLAB®. The software allows life predictions to quickly provide fatigue crack initiation using SAESUS data and road loading history on car lower suspension arm. In addition, the fatigue life was predicted using strain life approach subjected to variable amplitude loading. Coffin Manson are the method that provides in the software. Rainflow cycle counting method will be use to extract the cycle from time series data. Then the Palmgren-Rules equation was utilized to calculate cumulative damage. As a result, the GUI will display the result from the method using. From the software development, it can contribute to all user for calculate life prediction especially for variable amplitude loading. Thus the software does not need higher cost and also user friendly.

ABSTRAK

Tesis ini membentangkan satu pembangunan perisian pengiraan hayat lesu bagi data bebanan pelbagai amplitud. Objektif utama adalah untuk membangunkan satu perisian pengiraan untuk analisis hayat lesu menggunakan MATLAB®. Perisian ini membenarkan ramalan hayat untuk permukaan retak dengan menggunakan data SAESUS dan data lengan di bawah ampaian kereta semasa perjalanan. Dalam pada itu, hayat lesu diramal dengan menggunakan pendekatan hayat lesu yang dikenakan untuk pembebanan pelbagai amplitud. Model Coffin Manson merupakan kaedah yang disediakan dalam perisian ini. Kaedah pengiraan aliran hujan dapat digunakan dalam mengembangkan setiap kitaran dari data siri masa. Kemudian, jumlah kerosakan kumulatif dapat ditentukan dengan aturan Palmgren Miner. Keputusannya, satu paparan dapat dipaparkan hasil daripada menggunakan kaedah-kaedah yang disediakan. Perisian ini dapat menyumbang dalam memudahkan pengguna khususnya pelajar bagi mengira sesuatu jangka hayat lesu. Dalam pada itu, perisian ini tidak memerlukan kos yang tinggi dan ianya mesra pengguna.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

ε_a	Strain amplitude
ε_f	True fracture ductility
ε'_f	Fatigue ductility coefficient
σ	True stress, local stress
$\Delta\sigma$	Stress range
σ_a	Local stress amplitude
σ_m	Local mean stress
σ_{max}	Local maximum stress
σ'_f	Fatigue strength coefficient
E	Modulus of elasticity
N_f	Fatigue life
b	Fatigue strength exponent
c	Fatigue ductility exponent
$2N_f$	Reversals to failure
$\Delta\varepsilon/2$	Total strain amplitude

LIST OF ABBREVIATIONS

ASTM	American Society for Testing and Materials
GUI	Graphical User Interface
SAE	Society of Automotive Engineers
SWT	Smith Watson Topper

CHAPTER 1

INTRODUCTION

1.1 PROJECT BACKGROUND

This is project about Software Development of Strain-Based Fatigue Life Calculation for Variable Amplitude Loading Data using MATLAB® GUI. This project will use a method that need to be taken into consideration to successfully accomplish this project. The methods that are going to use is Coffin Manson.

Fatigue is the most important failure mode to be considered in a mechanical design. Under the action of oscillatory tensile stresses of sufficient magnitude, a small crack will initiate at a point of the stress concentration. Once the crack is initiated, it will tend to grow in a direction orthogonal to the direction of the oscillatory tensile loads. There are several reasons for the dominance of this failure mode and the problems of designing to avoid it: the fatigue process is inherently unpredictable, as evidenced by the statistical scatter in laboratory data; it is often difficult to translate laboratory data of material behavior into field predictions; it is extremely difficult to accurately model the mechanical environments to which the system is exposed over its entire design lifetime; and environmental effects produce complex stress states at fatigue-sensitive hot spots in the system. It can be thought that fatigue can involve a very complicated interaction of several processes and/or influences (Stephens et al. 2001)

A graphical user interface (GUI) is a pictorial interface to a program. A good GUI can make programs easier to use by providing them with a consistent appearance and with intuitive controls like pushbuttons, list boxes, sliders, menus, and so forth. The GUI should behave in an understandable and predictable manner, so that a user knows what to expect when he or she performs an action (Hunt et al. 2001).

MATLAB® is viewed by many users not only as a high-performance language for technical computing but also as a convenient environment for building graphical user interfaces (GUI). Data visualization and GUI design in MATLAB® are based on the Handle Graphics System in which the objects organized in a Graphics Object Hierarchy can be manipulated by various high and low level commands. If using MATLAB®7 the GUI design more flexible and versatile, they also increase the complexity of the Handle Graphics System and require some effort to adapt to.

1.2 PROBLEM STATEMENT

The current software is able to calculate fatigue life for variable amplitude loading data but it's difficult to get the software because of the higher cost. In this development country, the software has been created to display the value that had been calculated. Same for this project, fatigue life calculation software has been developed and will able to display at the MATLAB® program that is GUIDE. The advantages of this GUIDE is it will not only display the value but it will also able to explain the purpose of this program with interesting button and figure and also can guide the users to use this program.

1.3 OBJECTIVES

i. Design MATLAB® GUI for Fatigue Life Calculation

   To create and design GUI using GUIDE in MATLAB® Software package to make an easier for the user to use. The design in GUI must be user-friendly to make sure the user understand to use it.

ii. To Develop Algorithm for Calculating Fatigue Life

Develop the Fatigue Life Prediction Algorithm using method Coffin Manson by implement iteration method to solve the equation. This method will display a fatigue life in MATLAB® GUI.

1.4 HYPOTHESIS

Hypothesis of this project is when the software successfully develops, the data from SAESUS will be used to calculate fatigue life for variable amplitude loading data. This software calculates fatigue life using Coffin Manson method. This software also able to display time domain data and rainflow histogram based on material properties select and loading data used.

1.5 SCOPE OF RESEARCH

The first element need to be considered for scope of this project is development on Strain Life Fatigue Model. This model only focused on one method which is Coffin Manson.

The second element is software that becomes the main part of this project. The software that use in this project is Graphical User Interface Development Environment (GUIDE) in MATLAB® software package. This software is to design and create the GUI layout to make a user-friendly for user. For this GUIDE software is divide into two, first is GUI layout design with a consistent appearance and with intuitive controls like pushbuttons, list boxes, sliders, menus, and so forth. And second is for the program M-File, must design and use the right coding to make sure the design in GUI layout is work properly like what is needed.

CHAPTER 2

LITERATURE REVIEW

2.1 FATIGUE

Fatigue is the process of progressive localized permanent structural change occurring in a material subjected to conditions that produce fluctuating stresses and strains at some point or points and that may culminate in cracks or complete fracture after a sufficient number of fluctuations. If the maximum stress in the specimen does not exceed the elastic limit of the material, the specimen returns to its initial condition when the load is removed. A given loading may be repeated many times, provided that the stresses remain in the elastic range. Such a conclusion is correct for loadings repeated even a few hundred times. However, it is not correct when loadings are repeated thousands or millions of times. In such cases, rupture will occur at a stress much lower than static breaking strength. This phenomenon is known as fatigue (Stephens et al 2001).

To be effective in averting failure, the designer should have a good working knowledge of analytical and empirical techniques of predicting failure so that during the pre-described design, failure may be prevented. That is why; the failure analysis, prediction, and prevention are of critical importance to the designer to achieve a success (Stephens et al 2001).

Fatigue design is one of the observed modes of mechanical failure in practice. For this reason, fatigue becomes an obvious design consideration for many structures, such as aircraft, bridges, railroad cars, automotive suspensions and vehicle frames. For these structures, cyclic loads are identified that could cause fatigue failure if the design

is not adequate (Stephens et al. 2001). The basic elements of the fatigue design process are illustrated in Figure 2.1.

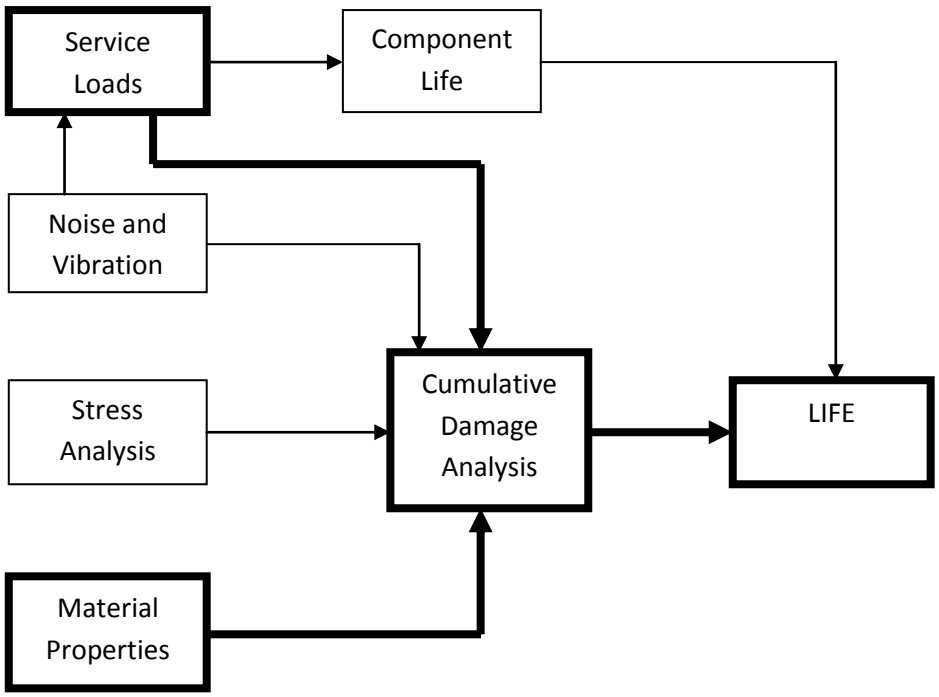


Figure 2.1: The basic elements for the fatigue design process.

Sources: Rise et. al (1988)

Service loads, noise and vibration: Firstly, a description of the service environment is obtained. The goal is to develop an accurate representation of the loads, deflections, strains, noise, vibration etc. that would likely be experienced during the total operating life of the component. Loading sequences are developed from load histories measured and recorded during specific operations. The most useful service load data is recording of the outputs of strain gages which are strategically positioned to directly reflect the input loads experienced by the component or structure. Noise and vibration has also effect on insight in the modes and mechanics of component and structural behaviour. An objective description of the vibration systems can be done in terms of frequency and amplitude information (Rise et. al 1988).

.

Stress analysis: The shape of a component or structure and boundary conditions dictates how it will respond to service loads in terms of stresses, strains and deflections. Analytical and experimental methods are available to quantify this behaviour. Finite element techniques can be employed to identify areas of both high stress, where there may be potential fatigue problems, and low stress where there may be potential for reducing weight. Experimental methods can be used in situations where components or structures actually exist. Strain gages strategically located can be used to quantify strains at such critical areas (Rise et. al 1988).

Material properties: A fundamental requirement for any durability assessment is knowledge of the relationship between stress and strain and fatigue life for a material under consideration. Fatigue is a highly localized phenomenon that depends very heavily on the stresses and strains experienced in critical regions of a component or structure. The relationship between uniaxial stress and strain for a given material is unique, consistent and, in most cases, largely independent of location. Therefore, a small specimen tested under simple axial conditions in the laboratory can often be used to adequately reflect the behaviour of an element of the same material at a critical area in a component or a structure. However, the most critical locations are at notches even when loading is uniaxial (Rise et. al 1988).

Cumulative damage analysis: The fatigue life prediction process or cumulative damage analysis for a critical region in a component or structure consists of several closely interrelated steps as can be seen in Figure 2.2 separately. A combination of the load history (Service Loads), stress concentration factors (Stress Analysis) and cyclic stress-strain properties of the materials (Material Properties) can be used to simulate the local uniaxial stress-strain response in critical areas. Through this process it is possible to develop good estimates of local stress amplitudes, mean stresses and elastic and plastic strain components for each excursion in the load history. Rainflow counting can be used to identify local cyclic events in a manner consistent with the basic material behaviour. The damage contribution of these events is calculated by comparison with material fatigue data generated in laboratory tests on small specimens. The damage fractions are summed linearly to give an estimate of the total damage for a particular load.history (Rise et. al 1988).

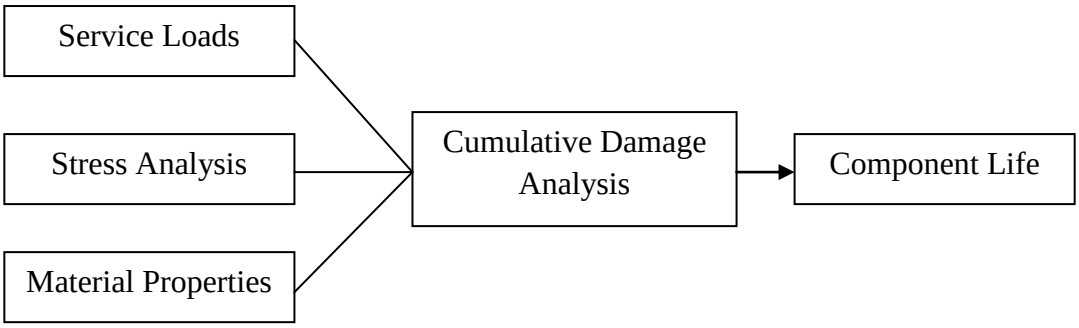


Figure 2.2: The cumulative damage analysis process

Sources: Ariduru (2004)

Component test: It must be carried out at some stage in a development of a product to gain confidence in its ultimate service performance. Component testing is particularly in today's highly competitive industries where the desire to reduce weight and production costs must be balanced with the necessity to avoid expensive service failures (Ariduru 2004).

Fatigue life estimates are often needed in engineering design, specifically in analyzing trial designs to ensure resistance to cracking. A similar need exists in the troubleshooting of cracking problems that appear in prototypes or service models of machines, vehicles, and structures. That is the reason that the predictive techniques are employed for applications ranging from initial sizing through prototype development and product verification. The functional diagram in Figure 2.3 shows the role of life prediction in both preliminary design and in subsequent evaluation-redesign cycles, then in component laboratory tests, and finally in field proving the tests of assemblies or composite vehicles and a conventional stress analysis might lead to a assumption of safety that does not exist (Ariduru 2004).

2.2 STRESS-LIFE BASED APPROACH (S-N METHOD)

For the fatigue design and components, several methods are available. All require similar types of information. These are the identification of candidate locations for fatigue failure, the load spectrum for the structure or component, the stresses or strains at the candidate locations resulting from the loads, the temperature, the corrosive environment, the material behaviour, and a methodology that combines all these effects to give a life prediction. Prediction procedures are provided for estimating life using stress life (Stress vs. Number of cycle's curves), hot-spot stresses, strain life, and fracture mechanics. With the exception of hot-spot stress method, Figure 2.3 shows all these procedures have been used for the design of aluminium structures.

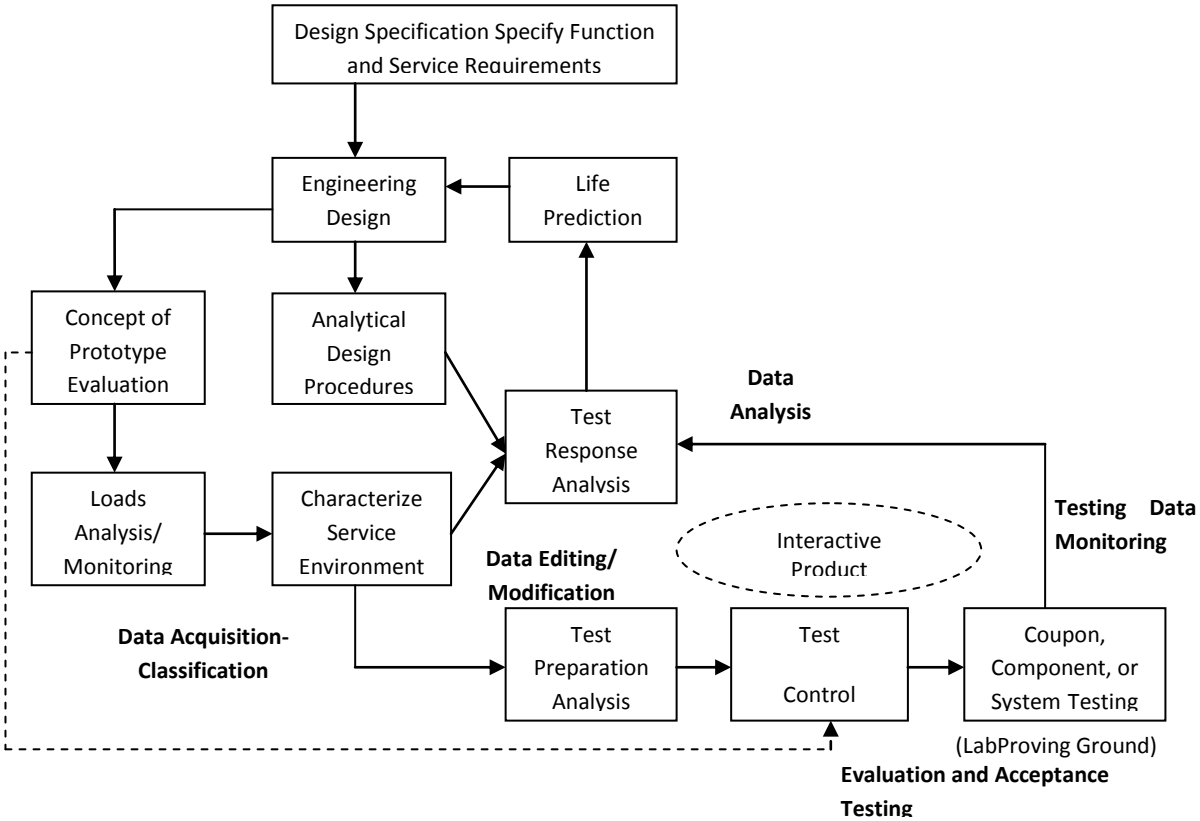


Figure 2.3: Functional diagram of engineering design and analysis

Sources: Rise et al. (1988)

Since the well-known work of Wöhler in Germany starting in the 1850's, engineers have employed curves of stress versus cycles to fatigue failure, which are often called S-N curves (stress-number of cycles) or Wöhler's curve (Lalanne et al.1999). Since the well-known work of Wöhler in Germany starting in the 1850's,engineers have employed curves of stress versus cycles to fatigue failure, which are often called S-N curves (stress-number of cycles) or Wöhler's curve.

The basis of the stress-life method is the Wöhler S-N curve, that is a plot of alternating stress, S, versus cycles to failure, N. The data which results from these tests can be plotted on a curve of stress versus number of cycles to failure. This curve shows the scatter of the data taken for this simplest of fatigue tests. A typical S-N material data can be seen in Figure 2.4. The arrows imply that the specimen had not failed in 10^7 cycles (Lalanne et al. 1999)

.

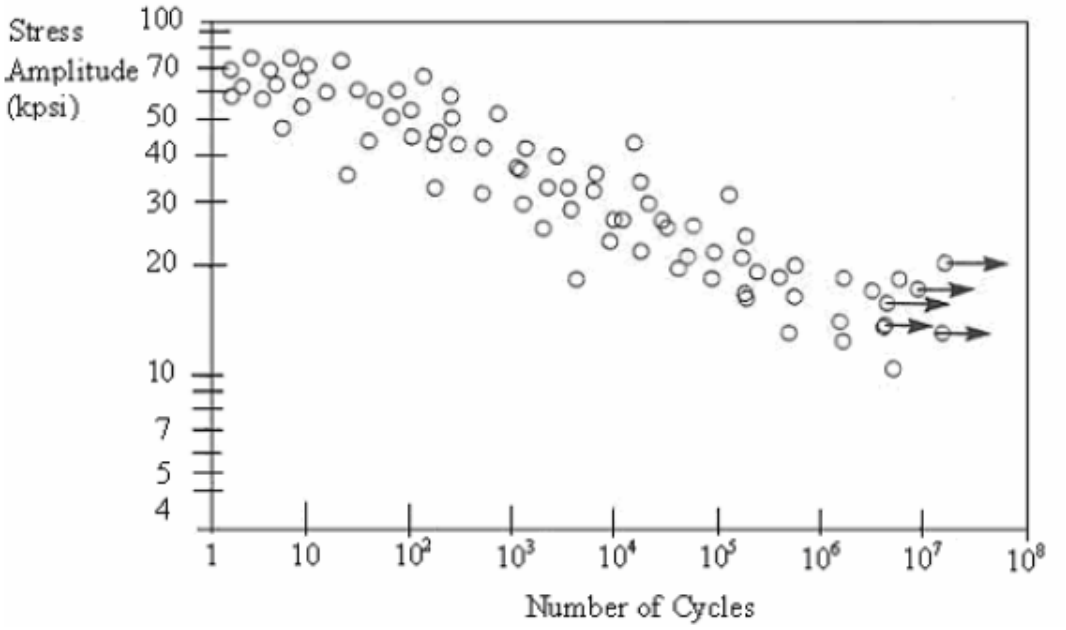


Figure 2.4: A typical S-N material data

Sources: Ariduru (2004)

The approach known as stress-based approach continues to serve as a widespread used tool for the design of the aluminium structures. Comparing the stress-time history at the chosen critical point with the S-N curve allows a life estimate for the component to be made.

Stress-life approach assumes that all stresses in the component, even local ones, stay below the elastic limit at all times. It is suitable when the applied stress is nominally within the elastic range of the material and the number of cycles to failure is large. The nominal stress approach is therefore best suited to problems that fall into the category known as high-cycle fatigue. High cycle fatigue is one of the two regimes of fatigue phenomenon that is generally considered for metals and alloys. It involves nominally linear elastic behaviour and causes failure after more than about 10^4 to 10^5 cycles. This regime associated with lower loads and long lives, or high number of cycles to produce fatigue failure. As the loading amplitude is decreased, the cycles-to-failure increase (Lalanne et al. 1999).

2.3 STRAIN BASED LIFE FATIGUE

Also known as low cycle fatigue which mean is repeated cyclic loadings that cause significant plastic cracking after a relatively small number of cycles-hundreds or thousands. Low cycle fatigue typically occurs as a result of repeated localized yielding near stress raisers, such as holes, fillets and notches, despite the elastic deformation occurring over the bulk of the component. Uniaxial testing is performed on several smooth (unnotched) specimens under different cyclic deformation levels in typical low fatigue test. Each specimen follows a given constant stress amplitude, completely reversed, cyclic strain. That is, the mode of testing is strain control instead of stress control. Stress response is monitored during cyclic loading, and the number of cycles to failure is recorded for these tests. The result from several tests is necessary to determine the cyclic stress to curve and the strain life curve for the material (Yung-Li et al. 2005).

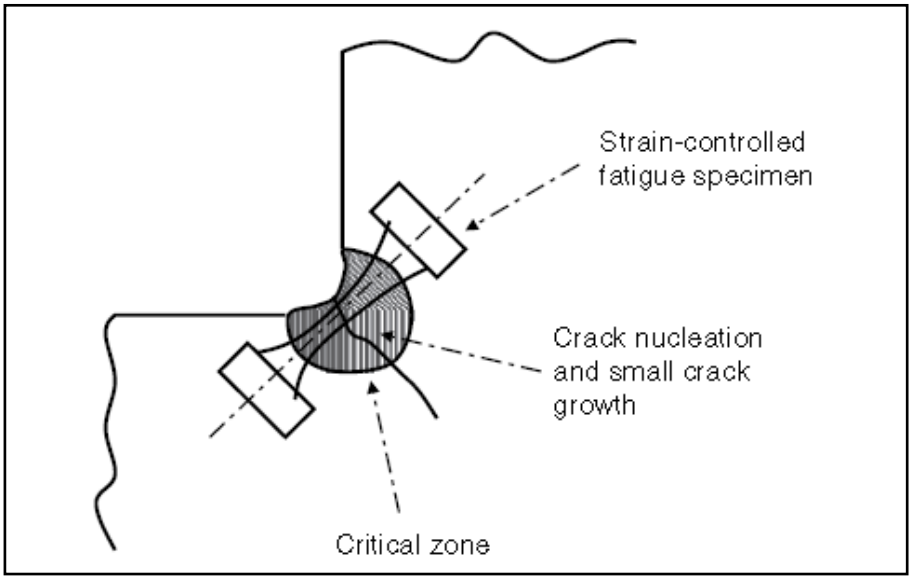


Figure 2.5: Concept of the local strain approach

Sources: Yung-Li et al. (2005)

2.3.1 Coffin-Manson Theory

The Coffin-Manson (Coffin 1954; Manson 1965) relation empirically relates the cycles to final failure, N_f, to the plastic strain amplitude, $\Delta\varepsilon_p / 2$. Expressed in the usual form

$$\frac{\Delta\varepsilon}{2} = \frac{\sigma'_f}{E}(2N_f)^b + \varepsilon'_f(2N_f)^c \qquad (2.1)$$

which E is modulus of elasticity, ε_a is the total strain amplitude, $2N_f$ is reversals to failure, $\sigma`_f$ is fatigue strength coefficient, $b$ is fatigue strength exponent, $\varepsilon`_f$ is fatigue ductility coefficient and c is fatigue ductility exponent. Figure 2.6 has been resolved into elastic and plastic strain component from the steady-state hysteresis loops (Stephens et al 2001).

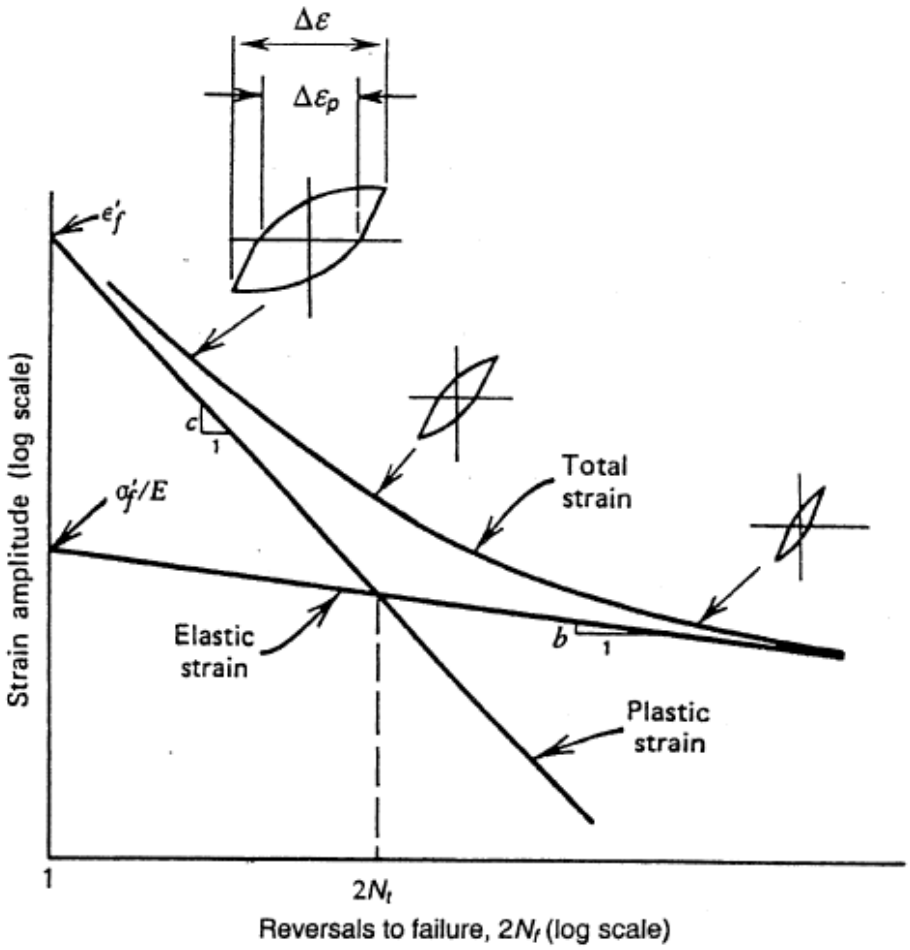


Figure 2.6: Strain-life showing total, elastic, and plastic strain components

Sources: Stephens et al. (2001)

2.3.2 Mean Stress Effects

Strain-controlled deformation and fatigue behavior discussed in the previous sections were for completely reversed straining, $R = \varepsilon_{min} / \varepsilon_{max} = -1$. In many applications, however, a mean strain can be present. Strain-controlled cycling with a mean strain usually results in a mean stress which may relax fully or partially with continued cycling. This relaxation is due to the presence of plastic deformation, and therefore, the rate or amount of result, there is more mean stress relaxation at larger strain amplitude. A model for predicting the amount of mean stress relaxation as a

function of cycles was proposed. Stress relaxation is different from cyclic softening and can occur in a cyclically stable material.

In Morrow`s (1968) method, the elastic term in the strain-life equation is modified by the mean stress. The modification is consistent with observations that the mean stress effects are significant at low values of plastic strain, where elastic strain dominates. Unfortunately, it correctly predicts that the ratio of elastic to plastic strain is dependent on mean stress, which is not true. Figure 2.7 shows Morrow mean stress correction.

An alternative version of Morrow`s mean stress parameter were both the plastic and elastic terms are affected by the mean stress is given by

$$\frac{\Delta\varepsilon}{2}=\varepsilon_a=\frac{\sigma_f\text{`}-\sigma_m}{E}(2N_f)^b+\varepsilon_f\text{`}\left(\frac{\sigma_f\text{`}-\sigma_m}{\sigma_f\text{`}}\right)^{\varepsilon/b}(2N_f)^c \qquad (2.2)$$

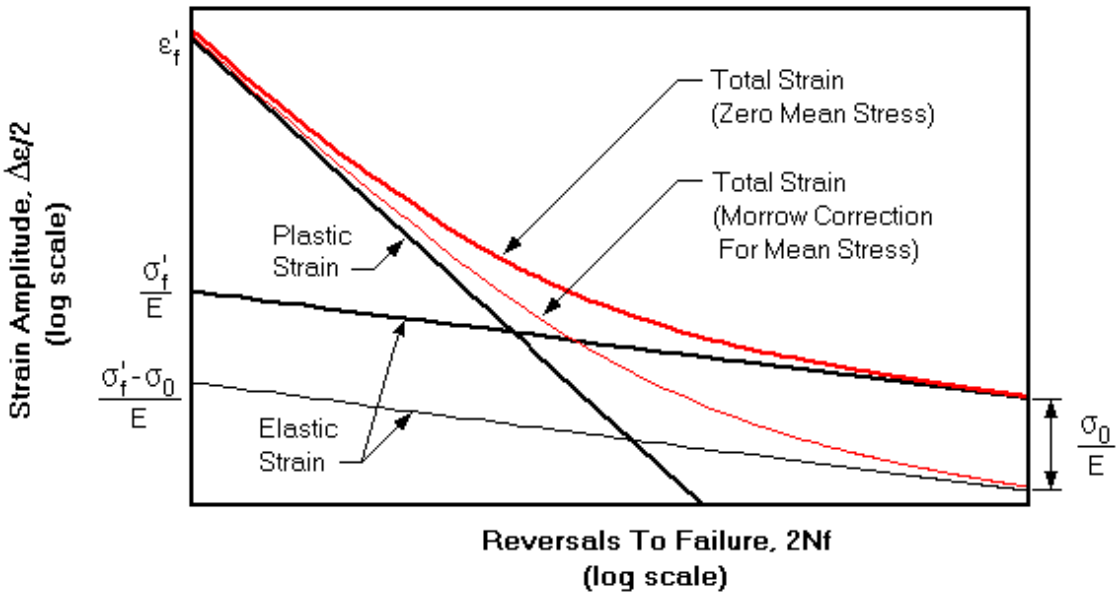


Figure 2.7: Morrow Mean Stress Correction for Strain Life Fatigue Analysis.

Sources: Browell (2006)

Smith, Watson and Topper (SWT) (1970) suggested a different equation to account for the presence of mean stresses. It has the limitation that it is undefined for negative maximum stresses. The physical interpretation of this is that no fatigue damage occurs unless tension is present at some point during the loading.

The equation is

$$\sigma_{max}\varepsilon_a E=(\sigma_f`)^2(2N_f)^{2b}+\sigma_f`\varepsilon_f`E(2N_f)^{b+c} \tag{2.3}$$

where σ_{max} is maximum stress and ε_a is alternating strain. This equation is based on assumption that for different combinations of strain amplitude, ε_a, and mean stress, σ_m, the product $\sigma_{max}\,\varepsilon_a$ remains constant for a given life. Figure 2.8 shows SWT mean stress correction.

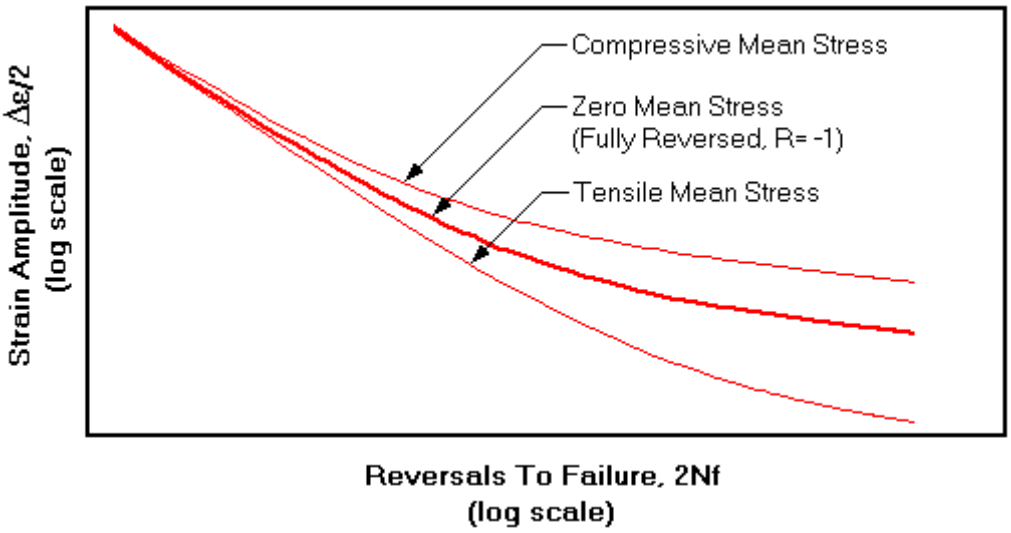


Figure 2.8: SWT Mean Stress Correction for Strain Life Fatigue Analysis.

Sources: Browell (2006)

2.4 CYCLE COUNTING

Cycles can be counted using time histories of the loading parameter of interest, such as force, torque, stress, strain, acceleration, or deflection. In Figure 2.9, one complete stress cycle in a time domain is related to a closed hysteresis loop in the local stress-strain coordinate and consists of two reversals. The reversal can be described as the event of unloading or loading (Yung-Li et al 2005).

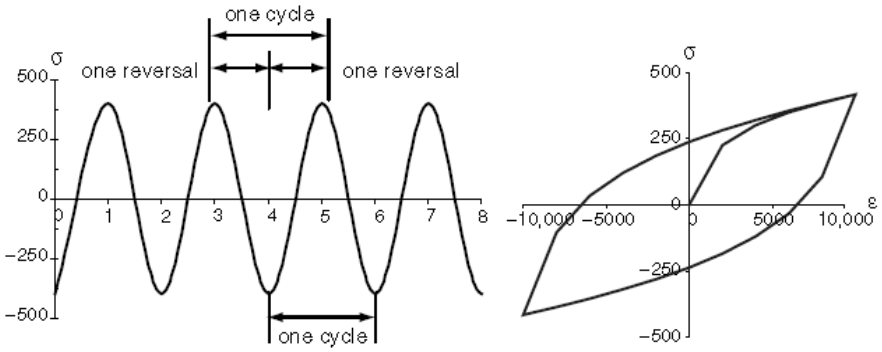


Figure 2.9: Definition of cycles and reversals

Source: Yung-Li et al (2005)

Over the years, one-parameter cycle counting methods such as level crossing, peak-valley, and range counting have been commonly used for extracting the number of cycles in a complex history. These methods are unsatisfactory for the purpose of describing a loading cycle and fail to link the loading cycles to the local stress-strain hysteresis behaviour that is known to have a strong influence on fatigue damage analysis. Thus, these methods are considered inadequate for fatigue damage analysis. However, the following sections present descriptions of the one-parameter cycle counting methods as an illustration of the techniques used in cycle counting and for comparison to the more effective two-parameter cycle counting methods.

For variable fatigue loading amplitude, cycle counting method is using for make analysis easier change to simple form or discrete. This method is widely used in life prediction strain model. The usual method of cycle counting used were as level crossing, peak valley, range counting and 'rainflow' (ASTM E1049,1985).

In this counting method, the magnitude of the loads in the load-time history has to be divided into a number of levels. This process is shown in Figure 2.10. One count at a specific level is defined when a portion of the load-time history with a negative slope passes through this level below a reference load. The reference load level is usually determined by the mean of the complete load-time history. A variation of this method is to count all of the levels crossing with the positive-sloped portion of the load-time history.

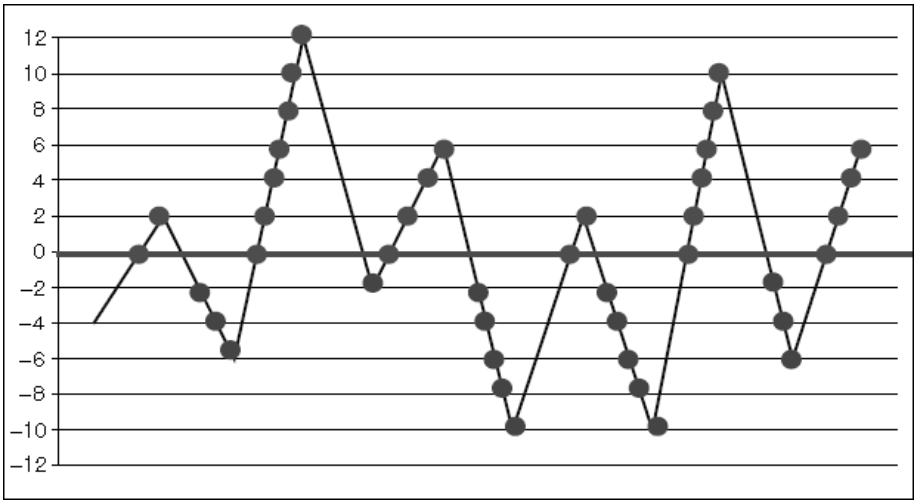


Figure 2.10: Level crossing counting of a service load-time history

Source: Yung-Li et al (2005)

Figure 2.11 show the tabulated and plotted results using the level-crossing count from the load-time history in Figure 2.10. Once all the counts are determined, they are use to form cycles. The cycle extraction rule follows that the most damaging fatigue cycles can be derived by first constructing the largest possible cycle, followed by the second largest possible cycle, and so on. This process repeated until all available counts are used up. Table 2.1 summarizes the cycle counting results.

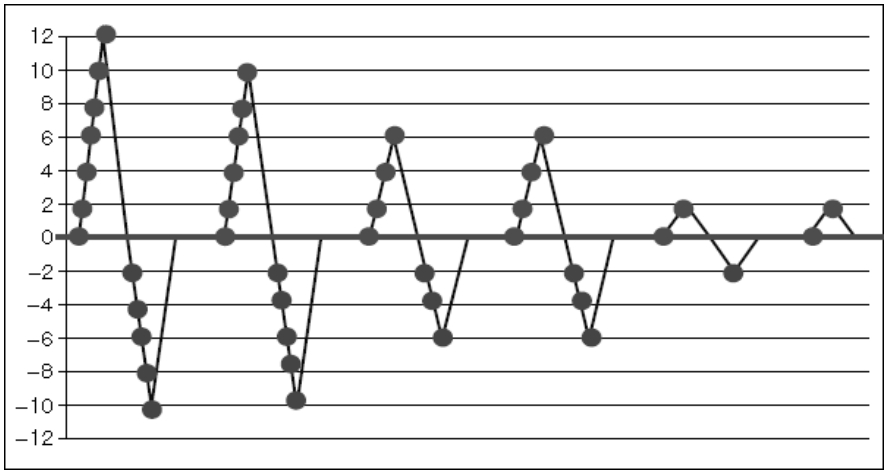


Figure 2.11: A process to generate cycles from level crossing counts.

Source: Yung-Li et al (2005)

Table 2.1: Tabulated cycle extracted from the level crossing counts

Range	Cycles
22	1
20	1
12	2
4	1

Source: Yung-Li et al (2005)

This counting technique first identifies the counts of peaks and valleys in a load-time history and subsequently constructs the possible cycles from the most to the least damaging events according to the extracted peak-valley counts. The peak is the transition point where a positive-sloped segment turns into a negative-sloped segment, and the valley is the point where a negative-sloped segment changes to a positive-slope one. Peaks above and valleys below a reference load level are counted. Table 2.2 shows tabulated results from the peak-valley counting in the load-time history in Figure 2.12(a). The process to generate the cycles from the peak-valley counts is illustrated in Figure 2.12(b). Table 2.3 summarizes the final cycle counting results.

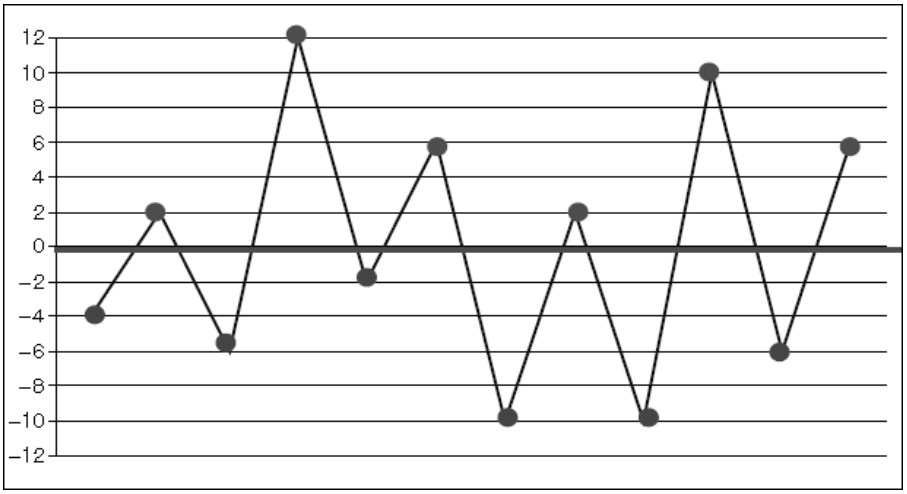


Figure 2.12(a): Peak-valley counting of services load-time history

Source: Yung-Li et al (2005)

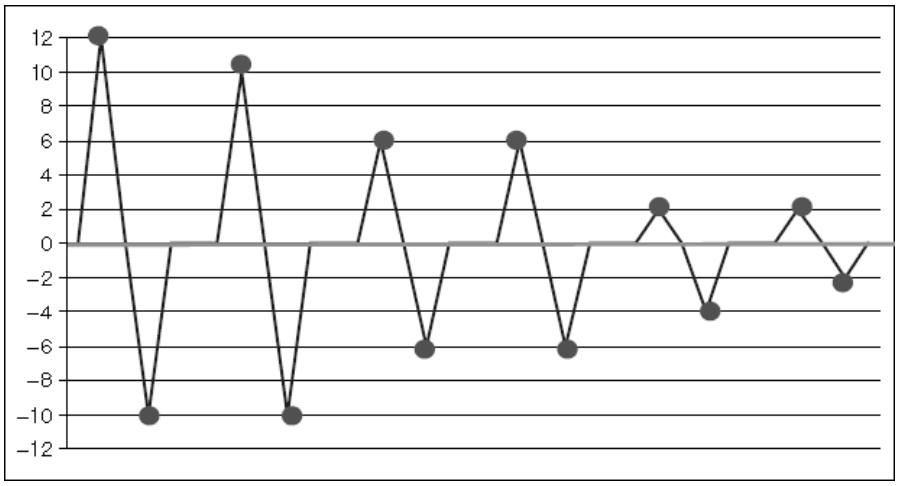


Figure 2.12(b): A process to derive cycles from a peak-valley counting

Source: Yung-Li et al (2005)

Table 2.3: Tabulated cycles extracted from peak-valley counts

Range	Cycles
22	1
20	1
12	2
6	1
4	1

Source: Yung-Li et al (2005)

For range counting method, this counting technique defines one count as a range, the height between a successive peak and valley. Positive ranges and negative ranges are defines on the positively sloped reversals and negatively sloped reversals, respectively. Each range represents on one-half cycle(reversal). Figure 2.13 illustrates the counts of positive and negative ranges. Table 2.4 lists the summary of the summary of the range counts and Table 2.5 shows the final cycles extracted from the range counts.

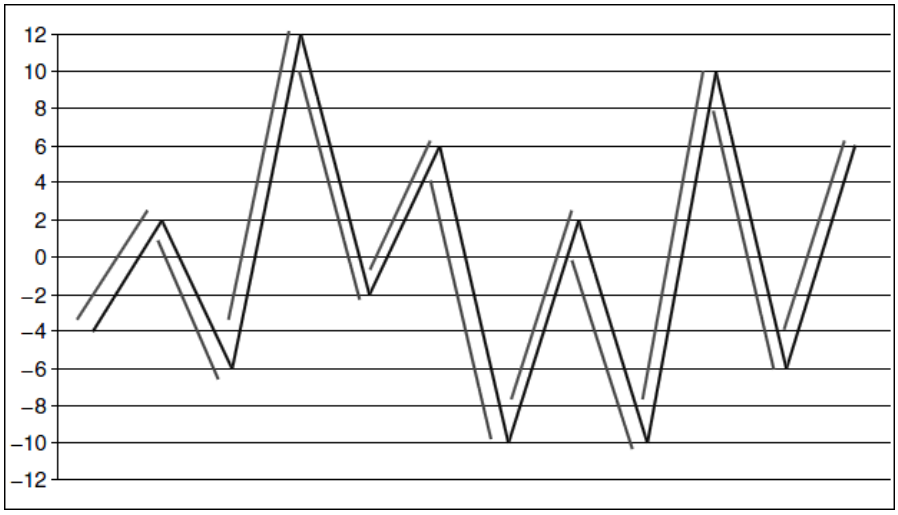


Figure 2.13: Range counting of service load-time history

Source: Yung-Li et al (2005)

Two-parameter cycles counting methods, such as the rainflow cycle counting method, cam faithfully represent variable-amplitude cyclic loading. Dowling (1979) states that the rainflow counting method is generally regarded as the method leading to better predictions of fatigue life. It can identify events in a complex loading sequence that is compatible with constant-amplitude fatigue data. Matsuishi and Endo (1968) originally developed the rainfow cycle counting method based on the analogy of raindrops falling on a pagoda roof and running down the edges of the roof. A number of variations of this original scheme have been published for various applications.

Table 2.4: Tabulated range counts results from the range counting method

Range	Counts
+20	1
+18	1
+12	2
+8	1
+6	1
-8	1
-12	1
-14	1
-16	2

Source: Yung-Li et al (2005)

Table 2.5: Tabulated cycle extracted from the range counts

Range	Counts
20	0.5
18	0.5
16	1
14	0.5
12	1.5
8	1
6	0.5

Source: Yung-Li et al (2005)

Figure 2.14 shows the rules that identify the two possible closed cycles in a time history where stress is the load parameter. The new time history shown in Figure 2.14(a) is generated by cutting all the points prior to and including the highest peak and by appending these data to the end of the original history. An additional highest peak is included n the new time history to close the largest loop for conservatism. The three consecutive stress points (S_1, S_2, S_3) define the two consecutive ranges as $\Delta S_1 = |S_1\text{-}S_2|$ and $\Delta S_2 = |S_2\text{-}S_3|$. If $\Delta S_1 \leq \Delta S_2$ is extracted, and if $\Delta S_1 > \Delta S_2$, no cycle is counted. The first cycle formed by two data points from -2 to 6 is extracted. A new load-time history is generated by connecting the point before -2 and the point after 6 to each other. This is illustrated in Figure 2.14(b). The same process is repeated until all the cycles are identified. This repetition is illustrated in Figure 3.7(c)-(g). The rainflow cycle counting results are tabulated in Table 2.6

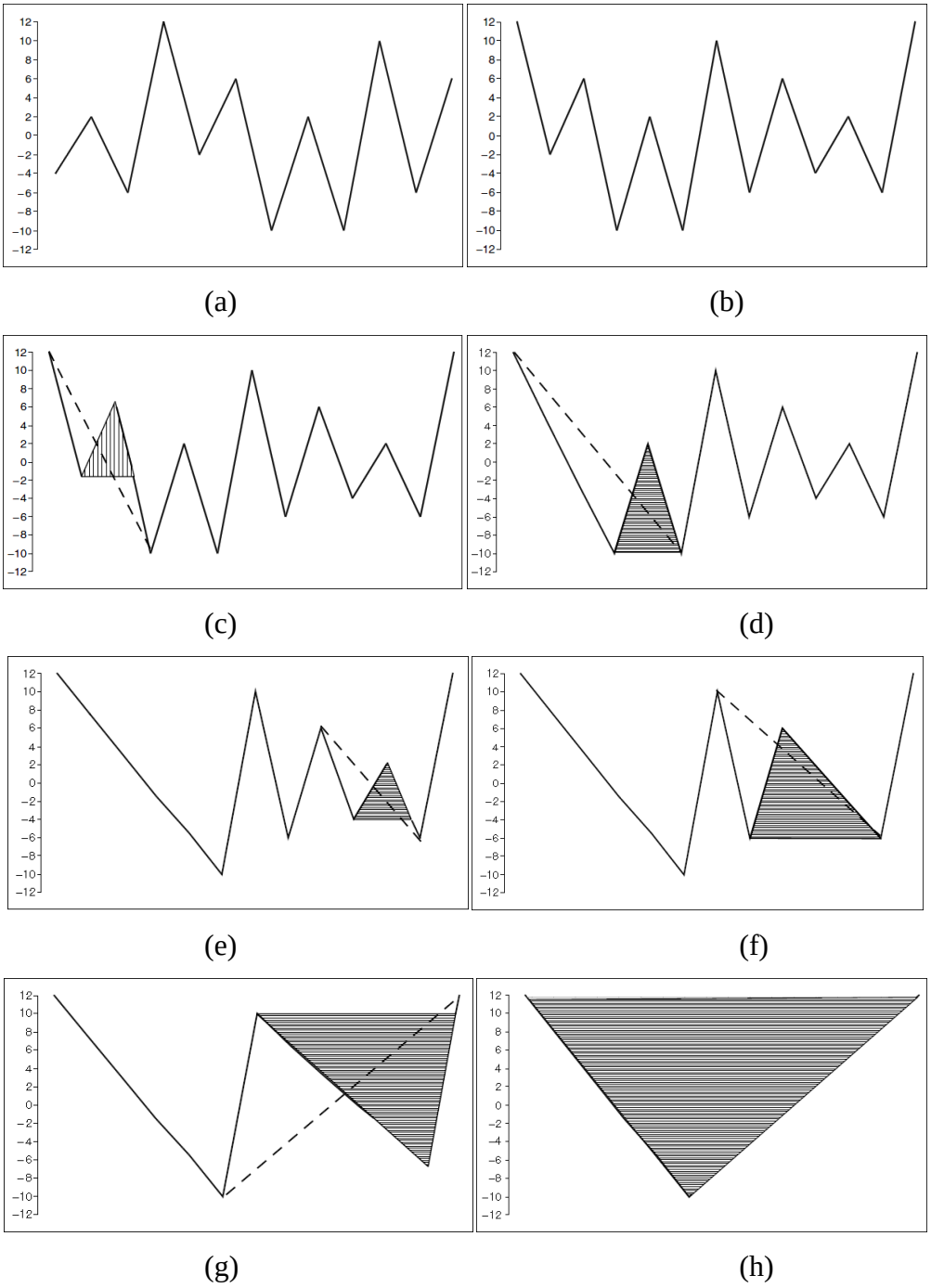


Figure 2.14: Sequence of the fatigue ‘rainflow’ cycle counting method based on standard ASTM E-1049

Source: Yung-Li et al (2005)

Table 2.6: Summary of cycle counting results

No. of Cycles	From	To	Range	Mean
1	-2	6	8	2
1	-10	2	12	-4
1	-4	2	6	-1
1	-6	6	12	0
1	10	-6	16	2
1	12	-10	22	1

Source: Yung-Li et al (2005)

2.5 LINEAR DAMAGE RULE (MINER'S RULE)

The Linear Damage Rule was first proposed by Palmgren (1924) and was further developed by Miner (1945). Today the method is commonly known as *Miner's Rule*. It widely used and still regarded as important tools in determining fatigue life.

This Palmgren-Miner is a linear concept depends on assumptions that changes occur in cycle are not uniformly affect the fatigue life. (Zahavi,1996). Therefore, the damage of one cycle, $D_{i,}$ can be calculate using equation

$$D_i = \frac{1}{N_i} \tag{2.12}$$

Where N_i is number cycles of failure at constant amplitude. To count accumulated damage for various amplitude cyclic loading, a cumulative fatigue damage model can be used known as Palmger-Miner rule. This rules has been developed based on the failure of the various stages of loading, and it can defined as

$$D = \sum \frac{n_i}{N_i} = \sum r_i \tag{2.13}$$

Wheres $n_{i\ \text{is}}$ cycle counting at stress level σ_i and N_i is fatigue life at the same for constant amplitude (Memon et al 2002). Calculation of cumulative at the three levels of

amplitude, n_1, n_2 and n_3 using Palmgren-Miner rule can be defined through eq 2.14 with related show in Figure 2.15.

$$D = \frac{n_1}{N_1} + \frac{n_2}{N_2} + \frac{n_3}{N_3} \tag{2.14}$$

If the component have 100% damages and will result on crack, Palgrem-Miner equation is

$$\sum \frac{n_i}{N_1} = 1 \tag{2.15}$$

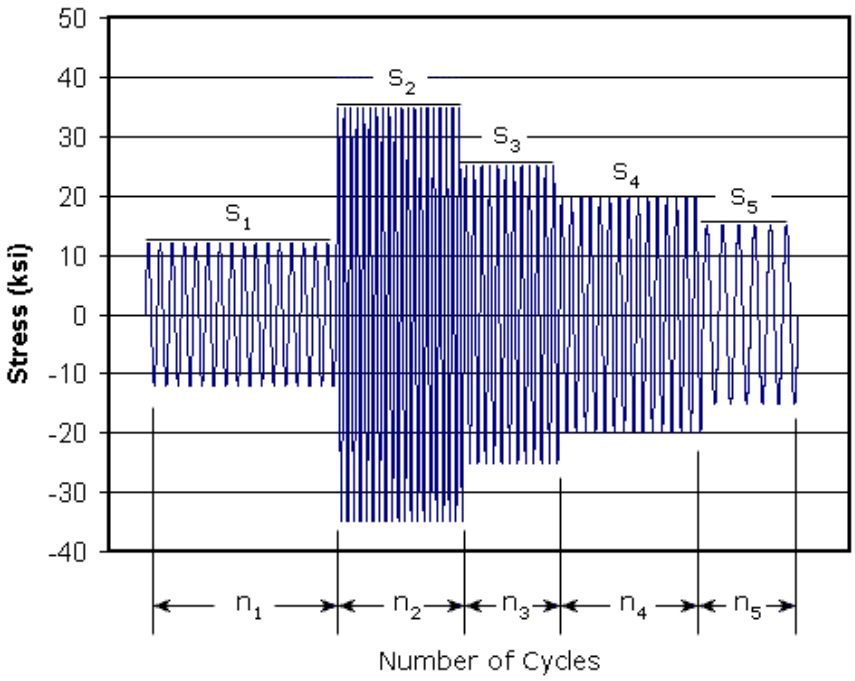


Figure 2.15: Stress spectrum

Source: engrasp (2010)

2.6 STEADY STATE CYCLIC STRESS-STRAIN

Fatigue life can be characterized by the steady-state behavior because for constant strain-amplitude controlled testing, the stress-strain relationship becomes stable after rapid hardening or softening in the initial cycles corresponding to the first several percent of the total fatigue life. The cyclic stable stress-strain response is the hysteresis loop and is identified in Figure 2.16. The hysteresis loop defined by the total strain range (Δε) and the total stress range (Δσ) represents the elastic plus elastic work on a material undergoing loading and unloading. Usually, the stabilized hysteresis loop is taken at half of the fatigue life (Yung-Li et al 2005).

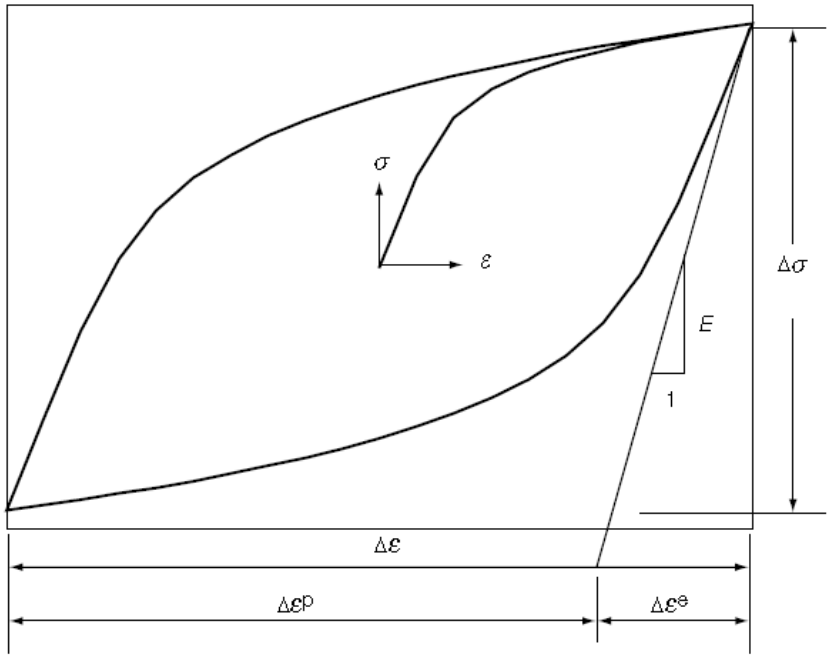


Figure 2.16: Hysteresis loop

Source: Yung-LI et al (2005)

When a family of stabilized hysteresis loops with various strain amplitude levels is plotted on the same axes as shown in Figure 2.17, a cyclic stress-strain curve is defined by the locus of the loop tips (Yung-Li et al 2005).

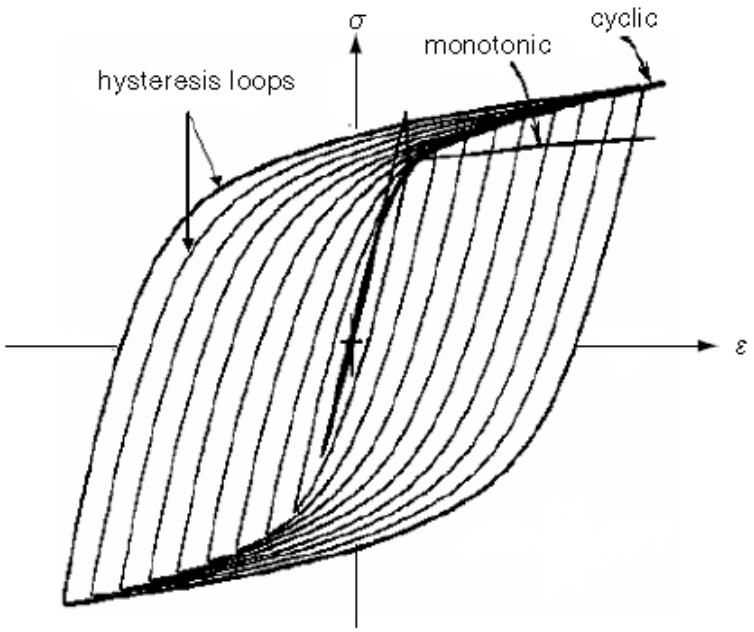


Figure 2.17: Cyclic stress-strain curve

Source: Yung-Li et al (2005)

2.7 MATLAB®

The MATLAB® technical computing language and development environment is used in a variety fields, such as image and signal processing, control systems, financial modeling, and computational biology. MATLAB® offers many specialized routines through domain specific add-ons, called "toolboxes", and a simplified interface to high-performance libraries such as BLAS, FFTW and LAPACK. These features appeal to domain experts who can quickly than with a low-level language such as C (Hunt, 2001).

MATLAB® is an interactive, matrix-based system for scientific and engineering numeric computation and visualization. It can solve complex numerical problems in a fraction of the time with a programming language such as Fortran or C. The name MATLAB® is derived from MATrix LABoratory (Hunt, 2001).

The software has developed from early public domain versions of the late 1970s to a mature product in the 1990s. The numerical routines, the graphical output and the elements for the construction of graphical user interfaces from a unity that can easily be used for teaching and learning (Jia and Schaufelberger, 1995). Some of the early roots of MATLAB are still visible; from a computer science point of view, the product is less developed then from the point of view of algorithms and graphics. The old concept on one file per function is still valid in version 4, and makes the package heavy and demanding on resources (Hunt, 2001).

MATLAB® is common-driven, and the well-structured help facilities is needed for operation because of the many names of the functions that have to be known or found for using the system appropriately. Figure 2.18 contains an example of a newly launched MATLAB® Desktop (Hunt, 2001).

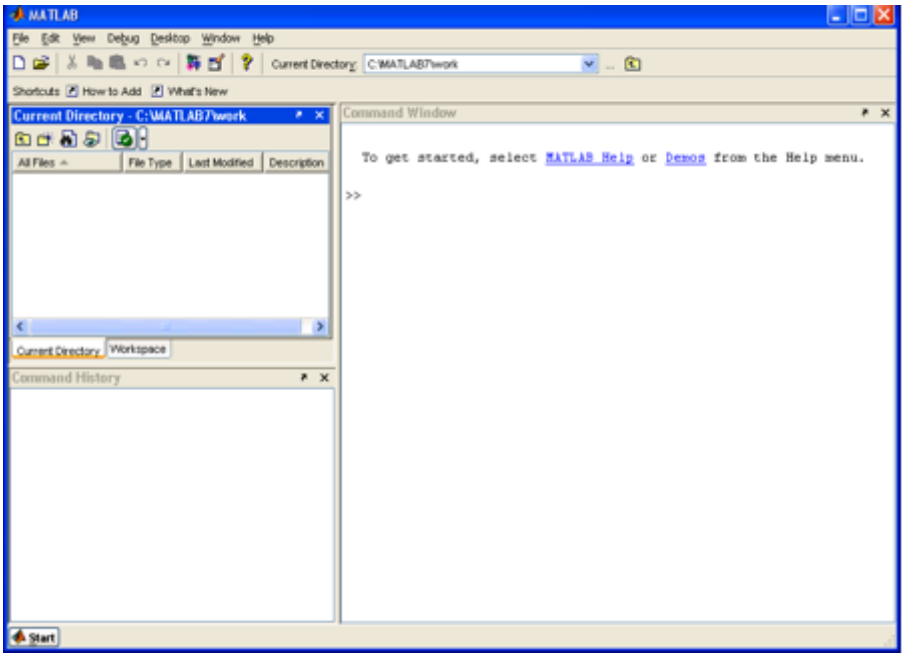


Figure 2.18: A MATLAB Desktop

Source: mathworks (2010)

MATLAB® also provides an interactive tool called GUIDE (this stands for Graphical User Interface Development Environment) that greatly simplifies the task of building a GUI. The Layout Editor looks like Figure 2.19.

A graphical user interface (GUI) is a pictorial interface to a program. A good GUI can make programs easier to use by providing them with a consistent appearance and with intuitive controls like pushbuttons, list boxes, sliders, menus, and so forth (R. Hunt, 2001). The GUI should behave in an understandable and predictable manner, so that a user knows what to expect when he or she performs an action. For example, when a mouse click occurs on pushbutton, the GUI should initiate the action described on the label of the button. This chapter introduces the basic elements of the MATLAB® GUIs. The chapter does not contain a complete description of components or GUI features, but it does provide the basics required to create functional GUIs for your programs (Hunt, 2001).

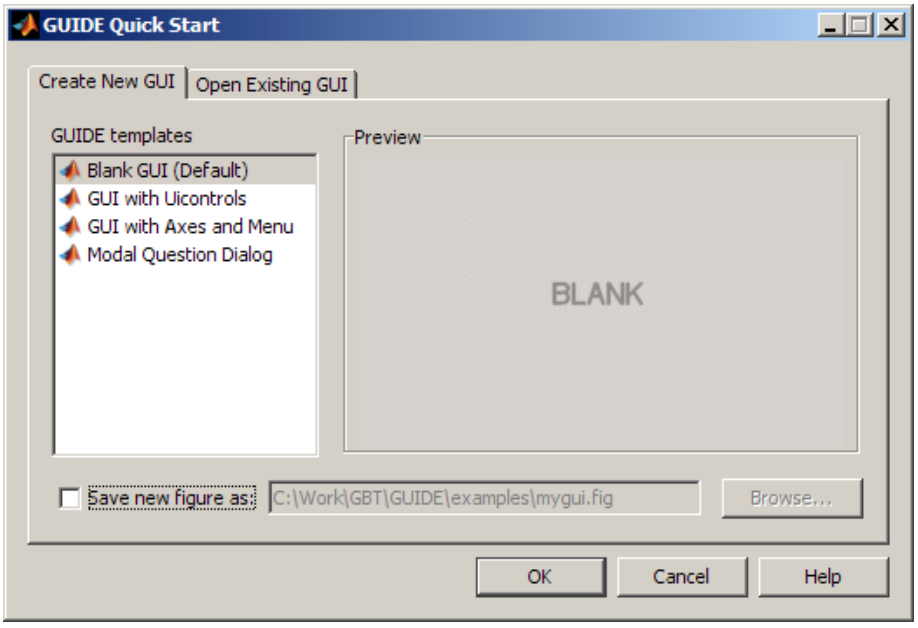


Figure 2.19: The Layout Editor

Source: mathworks (2010)

Applications that provide GUIs are generally easier to learn and use since the person using the application does not need to know what commands are available or how they work. The action that results from a particular user action can be made clear by the design of the interface (Hunt, 2001).

2.8 MATLAB® GRAPHICAL USER INTERFACE (GUI)

A graphical user interface provides the user with a familiar environment in which to work. This environment contains pushbuttons, toggle buttons, lists, menus, text boxes, and so forth. All of which are already familiar to the user, so that he or she can concentrate on using the application rather than on the mechanics involved in doing things. However, GUIs are harder for the programmer because a GUI-based program must be prepared for mouse clicks (or possibly keyboard input) for any GUI element at any time. Such inputs are known as events, and a program that responds to events is said to be *event driven.* The three principal elements required to create a MATLAB® Graphical User Interfaces are (Hunt, 2001):-

i. Components. Each item on a MATLAB® GUI (pushbuttons, labels, edit boxes, etc.) is a graphical component. The types of components include graphical controls (pushbuttons, edit boxes, lists, sliders, ect.), static elements (frames and text strings), menus, and axes. Graphical controls and static elements are created by the function uicontrol, and menus are created by the functions uimenu and uicontextmenu. Axes, which are used to display graphical data, are created by the function axes. Figure 2.20 shows an example of GUIDE tool window

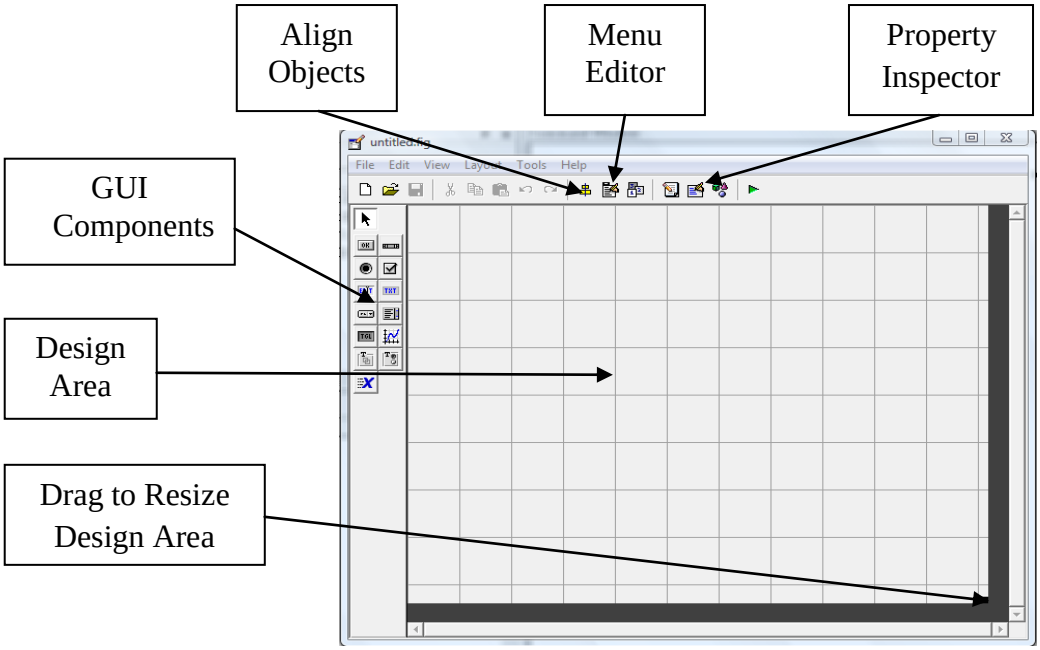


Figure 2.20: The Guide tool window

ii. Figures. The components of a GUI must be arranged within a figure, which is a window on the computer screen. In the past, figures have been created automatically whenever we have plotted data. However, empty figures can be created with a function figure and can be used to hold any combination of components.

iii. Callbacks. Finally, there must be some way to perform an action if a user clicks a mouse on a button or types information on a keyboard. A mouse click or a key press in an event, and the MATLAB® program must be respond to each event if the program is to perform its function. For example, if a user clicks on a button, that event must cause the MATLAB® code that implements the function of the button to be executed. The code executed in response to an event is known as a call back. There must be a call back to implement the function of each graphical component on the GUI. Figure 2.21 shows the pushbutton with Callback.

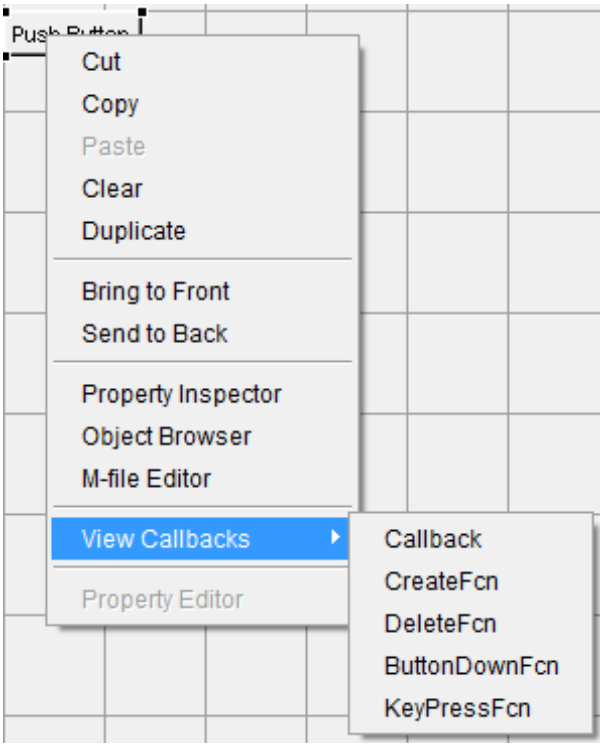


Figure 2.21: The Push Button with Callback

2.9 EXISTING FATIGUE SOFTWARE

2.9.1 Glyphworks

Gylphworks is a powerful data processing system for engineering test data analysis with specific application to durability and fatigue analysis. Designed to handle huge amounts of data, GlyphWorks providers a graphical, process-oriented environment. It can simply create an analysis workflow by ‘drag’ and ‘drooping’ analysis building blocks. The example of GlyphWorks is shown in Figure 2.20.

In addition to general signal processing, GlyphWorks provides leading fatigue analysis capabilities for measured data. Unique capabilities include the ability to help specify accelerated durability tests, saving both time and money in environment qualification and product validation.

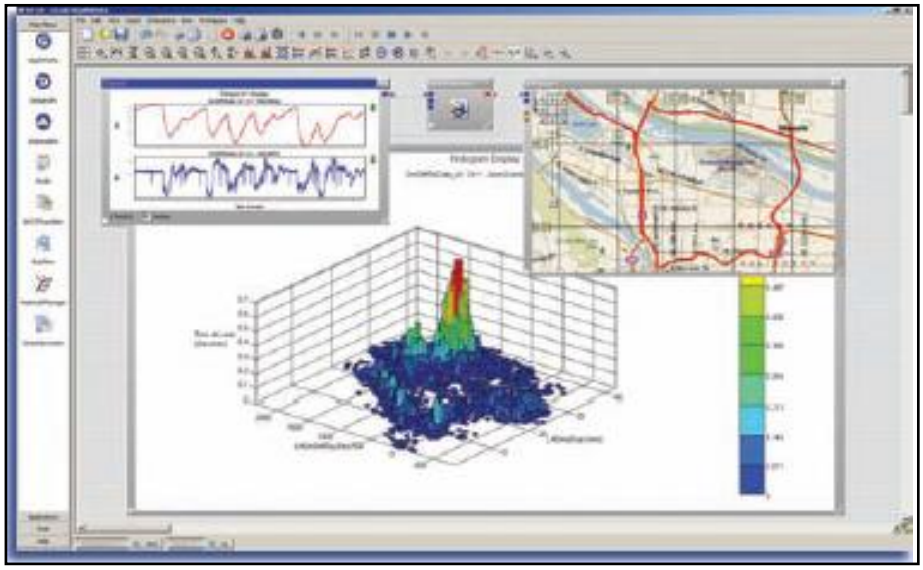


Figure 2.22: GlyphWorks Interface

Source: ncode (2010)

GlyphWorks Fundamental provides visualization and basic manipulation, frequency spectrum analysis and filtering capability. Report layout can be interactively designed and these reports automatically generated by including as part of the analysis process. Fundamentals are a pre-requisite for all other product options (nCode 2001).

The Fundamentals is package includes support for a wide range of data formats and types of displays. The Super glyph capability enables multiple analysis functions to be encapsulated as a single glyph that can be saved re-used.

GlyphWorks provides the industry-leading fatigue analysis technology that needs to calculate fatigue life from measured data. It can correct for mean stress and surface finish effects, even back calculate from each data channel to determine a scale or fatigue concentration factor required to achieve a target life. Then it can review damage histograms to determine which load cycles were most damaging, and even output damage time histories to show exactly when the damage occurred. A database with commonly used fatigue data is also provided (nCode 2001).

i. **Stress-Life** method uses a nominal stress approach for high-cycle condition or non-metallic applications. A wide range of methods are provided for defining the SN curves including the ability to interpolate multiple material data curves for mean stress effects. For ultimate flexibility, Python scripting enables the definition of custom fatigue methods and material models

ii. **Strain-Life** method is more appropriate for more severe loading conditions (low-cycle fatigue) – where local elastic-plastic strain controls the fatigue life. Supported methods include the Coffin-Manson-Basquin formula with additional mean stress corrections such as Morrow and Smith-Watson-Topper.

iii. **Crack Growth** provides linear elastic fracture mechanics in the GlyphWorks environment. It provides a complete fracture mechanics capability using industry standard methodologies, an open environment for users to embed their own algorithm – and the advanced reporting and quality assurance capabilities of GlyphWorks. Built-in growth laws includes NASGRO3, Form an, Paris, Walker and more. Predicting how a crack will propagate after initiation is now easy.

2.9.2 MSC Fatigue Software

MSC.Fatigue uses three life prediction methods. These are *total life, crack initiation,* and *crack propagation.* Total life is apply named in that only the total life of the component is of concern and not when a crack will initiate or how quickly it will grow.

The three methods are related to each other by the fact that the total number of cycles to failure, N_f, equals the number of cycles to initiate a crack, N_i, plus the number of cycles to propagate that crack, $N_{p.}$ The three methods have grown out of different needs over the decades using different techniques and having different degrees of accuracy as shown in Figure 2.23. So in theory this equation is true, but in practice when applying the three methods to the same problem, rarely, if ever does it ad up.

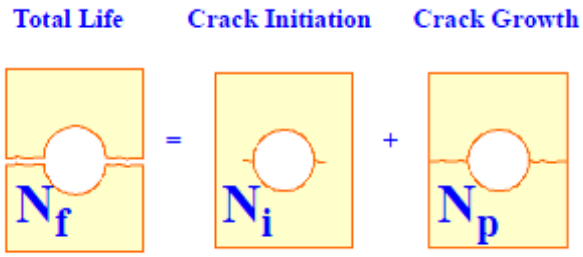


Figure 2.23: Life Prediction Methods

Source: MSC.Fatigue QuickStart Guide

In reality however, rarely are all three methods used on the same problem, mainly because different industries adopt different analysis methods depending on the driving design philosophy.

CHAPTER 3

METHODOLOGY

3.1 INTRODUCTION

In this chapter, the methodology can be divide into two; which is develop algorithm for analysis fatigue data using Coffin Manson, Morrow and Smith Watson Topper. Another method is develop interface using MATLAB® GUI for display some parameter related to this project.

3.2 FLOW CHART OF PROJECT

From the Figure 3.1, after get the topic of the project its goes to case study to find more related information and to deep knowledge about the project. Find the information whether on internet, journal, books or anything else that related to the topic.

For this project the software that has to use is MATLAB® GUI. First, study about the algorithm development using rainflow and also using fatigue calculation for variable amplitude loading method. And second study about the software programming and understand how to use it. For this software has divide by two parts, first is GUI layout design with a consistent appearance and with intuitive controls like pushbuttons, list boxes, sliders, menus, and so forth. Secondly is for the program M-File, must design and use the right coding to make sure the design in GUI layout is work properly like what is needed.

When the two parts have done, test it to make sure the software that has been design is work properly. If not, identify the problem and overcome it again. After software part have work properly, simulate and testing it whether is okay or not. And troubleshoot this part if not okay until get the satisfied result. If the testing is work properly and correctly, finally these projects have done and can be submit the about this project.

3.3 DEVELOPMENT ALGORITHM FOR FATIGUE LIFE CALCULATION

The calculation of fatigue life data for variable amplitude, the first thing to do is to find effective cycles in the fatigue data. In this research an algorithm computer has been developed for the cycle by using three point cycle “rainflow” counting (ASTM 1049, 1985). A flow chart for develop algorithm cycle “rainflow” counting shown in Figure 3.2.

3.4 DEVELOPMENT ALGORITHM OF FATIGUE LIFE CALCULATION FOR VARIABLE AMPLITUDE LOADING

To determine fatigue damage in a signal fatigue, a computer algorithm was developed. The developed algorithm is based on strain life as Coffin Manson (Coffin 1954; Manson 1965). Theory information for the model can be referred to the section 2.3. Flow chart for the development of fatigue life calculation algorithm is shown in Figure 3.3. Algorithm development process of fatigue damage calculation is as follows:

i. Various amplitude fatigue data should be input to the algorithm that has been developed.

ii. Range, maximum strain, minimum strain and average strain for each fatigue cycle been determine using cycle ‘rainflow’ counting method.

iii. The largest range of cycles was determined from available cycle to obtain the maximum strain(ε_{max})and minimum strain(ε_{min}).

iv. Choose the model Coffin Manson (Coffin 1954; Manson 1965). Because of this model has two unknown of reversal to failure ($2N_f$) at both side, the iteration method can be used to obtain the value of reversal to failure ($2N_f$). *M* and *M1* represents the value of reversal to failure on the strain, *e*. The initial value of M1 is set as 10 and will be repeated by taking the value of *M* to obtain value *M=M1*. Constant value of *E* is 204GPa, *b* = -0.092 and *c* = -0.445.

v. Then input the value of reversal to failure, $2N_f$ for each cycle 'rainflow'. Fatigue damage for each cycle is then calculated by using the rules of Palmgren-Miner. The total amount of fatigue damage was determined by taking the cumulative value of fatigue damage for each cycle.

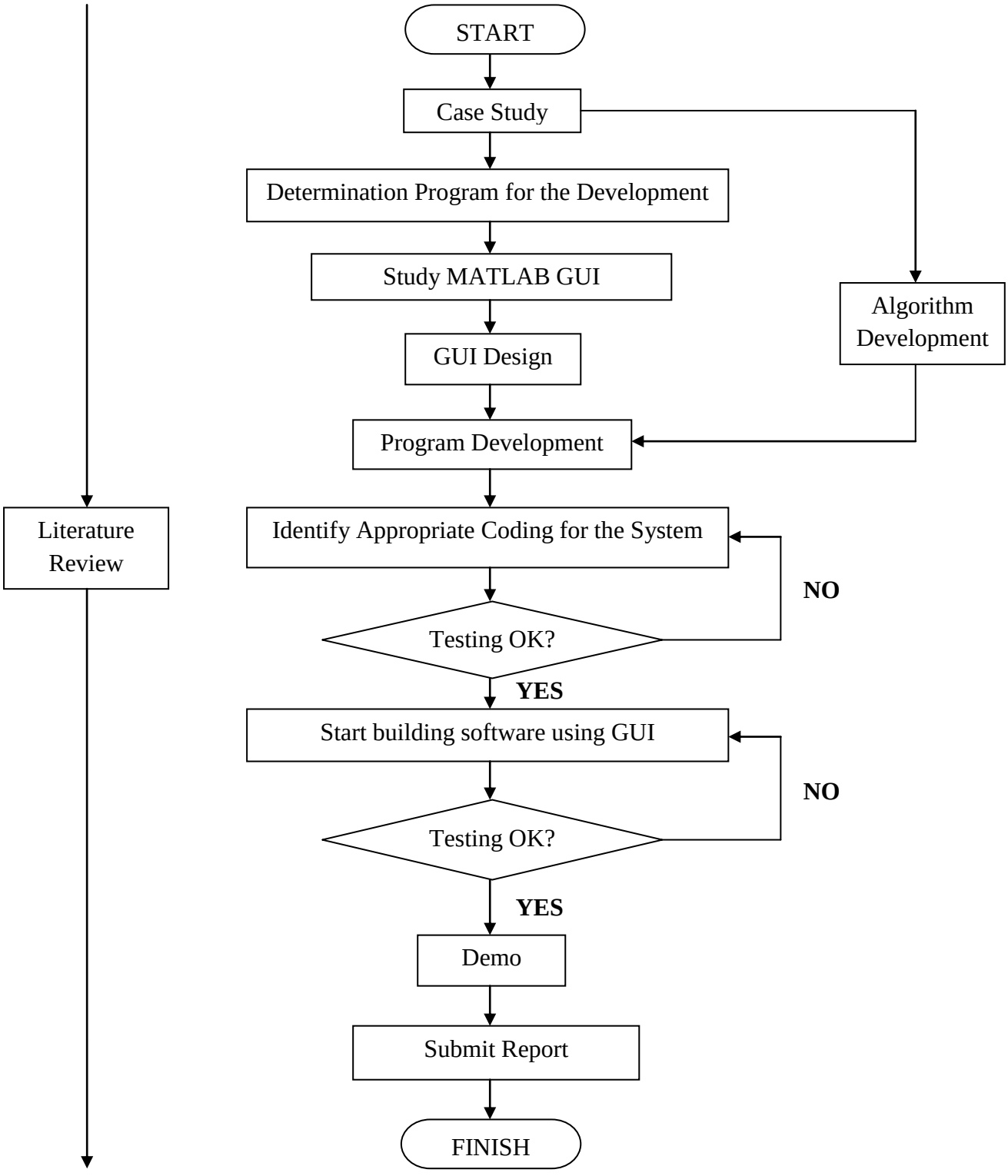


Figure 3.1: Flow Chart of Project

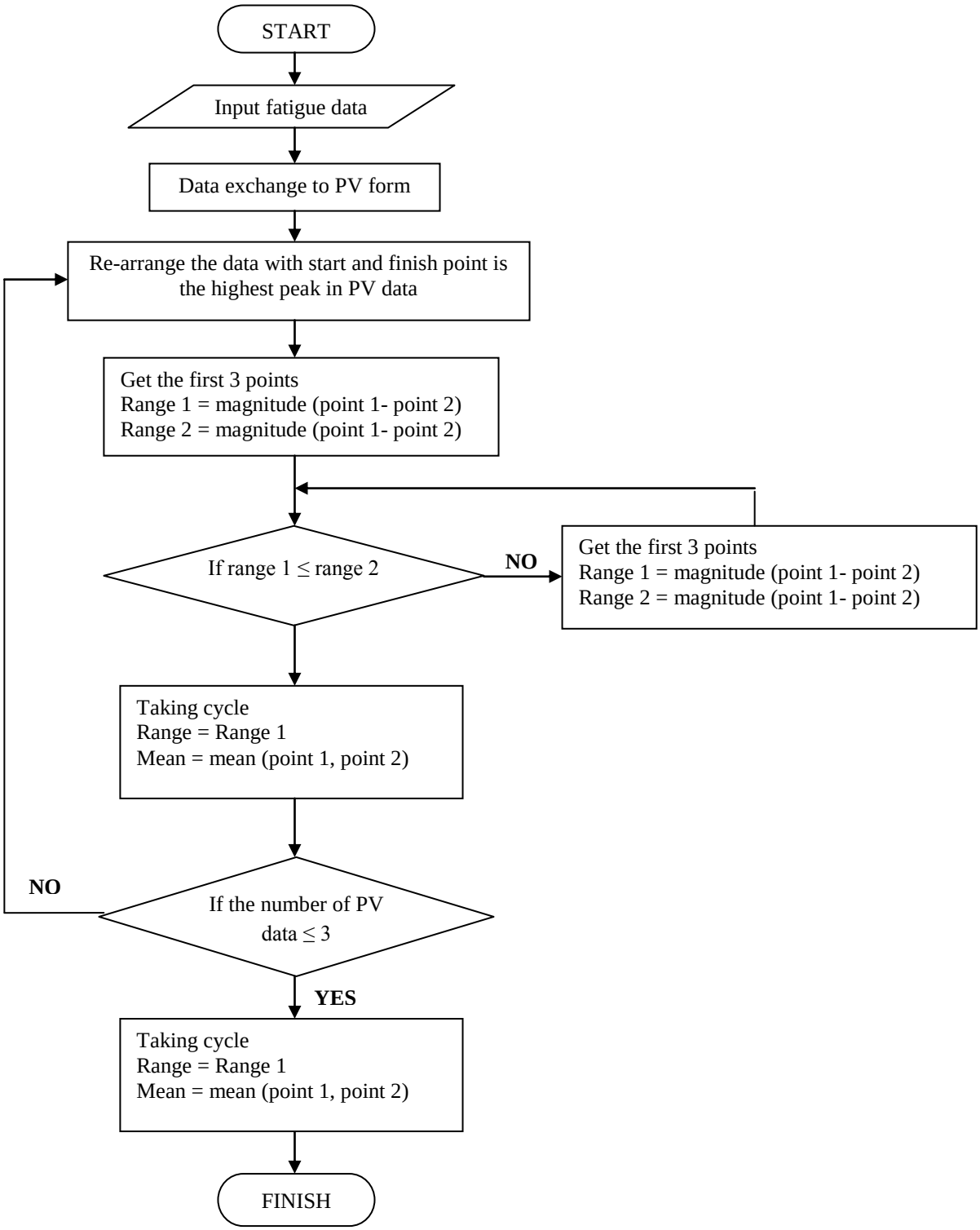


Figure 3.2: Flow chart for algorithm development using rainflow cycle counting

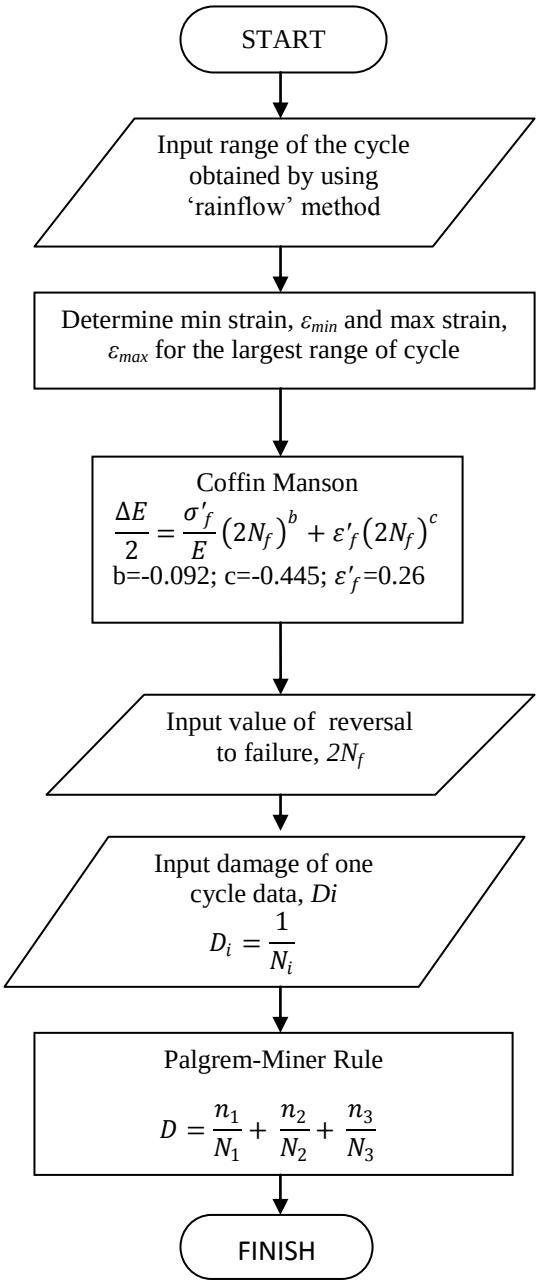


Figure 3.3: Flow chart for development algorithm fatigue life calculation for variable amplitude loading

3.5 DEVELOP INTERFACE USING GRAPHICAL USER INTERFACE

This is the main part of this project. GUIDE, the MATLAB® graphical user interfaces (GUIs). This tool allows a programmer a layout the GUI, selecting and aligning the GUI components to be placed in it. Once the components are in place, the programmer can edit their properties: name, color, size, font, text to display, and so forth.

When guide saves the GUI, it creates working program including skeleton functions that the programmer can modify to implement the behavior of the GUI. Figure 3.4 show the layout GUI after done with the designation with a few basic components that had been used like push button.

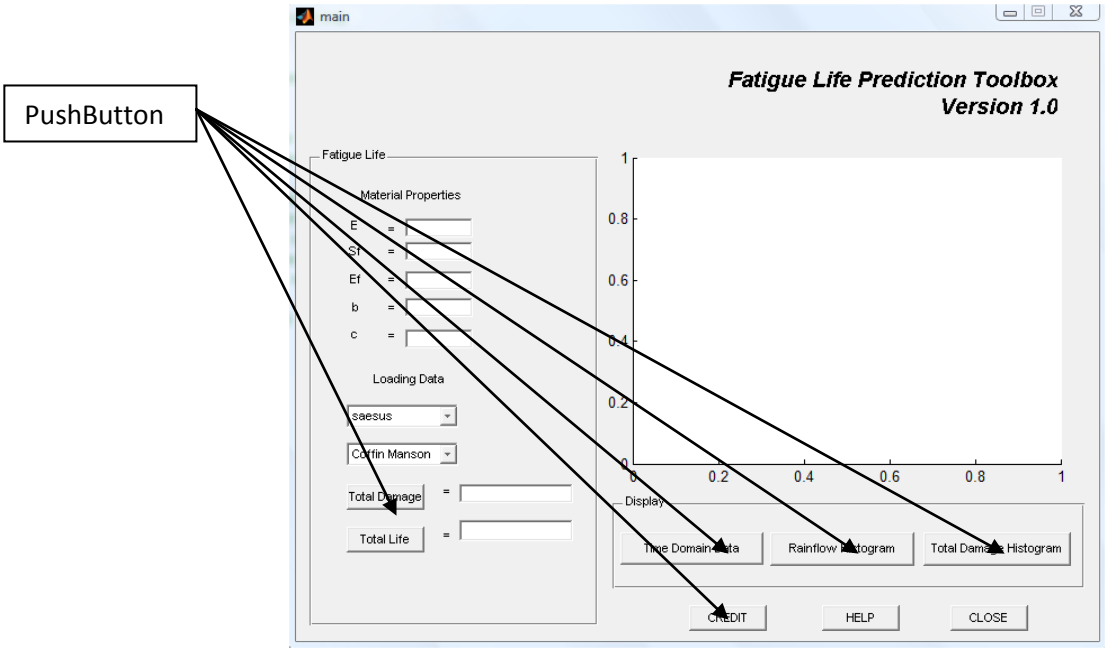


Figure 3.4: Layout GUI (Main Window)

All the material properties need to be filling before calculate the fatigue life. The user can choose a data set to prompt a loading history by clicking time domain data pushbutton. This software only calculates fatigue life using Coffin-Manson.

For example, if the user wants to display rainflow histogram, first click pushbutton “rainflow histogram” at the interface. This is the way to callback function rainflow before one dataset of cycle range display. The 3D histogram will visual at axes as shown in Figure 3.5.

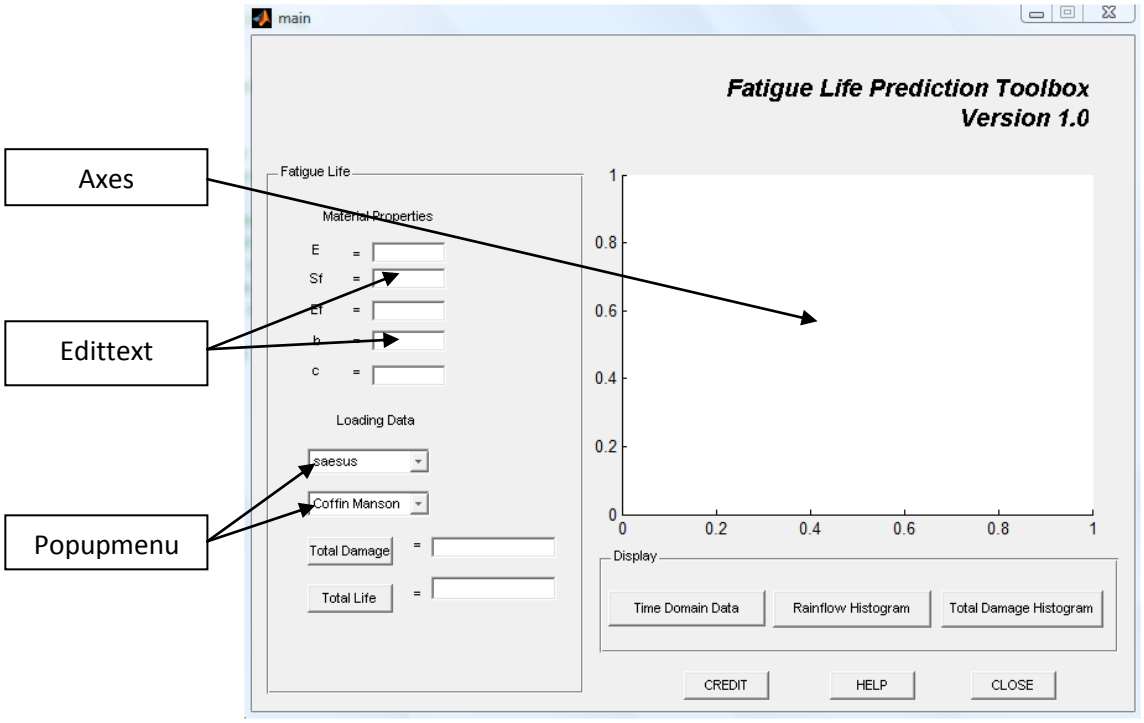


Figure 3.5: Layout GUI (Display Axes)

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 INTRODUCTION

This chapter consists of the discussions on the result from the MATLAB® GUI layout that has been developed using Graphical User Interface (GUI). The GUI in this project has performed the task from the data loading. This chapter also will explain the main menu of GUI.

4.2 MAIN MENU OF THE GUI

The menu of GUI in this project contains a few buttons that has been named as shown in Figure 4.1. The fatigue life prediction panel contain two pushbutton will explain in the next sub chapter. The user needs to fill all the material properties before calculating the fatigue life. This software also provide the user to display time loading history, rainflow histogram and total damage histogram. All this will display in the axes box. The other pushbutton is credit which is containing the detail about GUI developer and also the supervisor as shown in Figure 4.2. The user can use the help to get help when using this software.

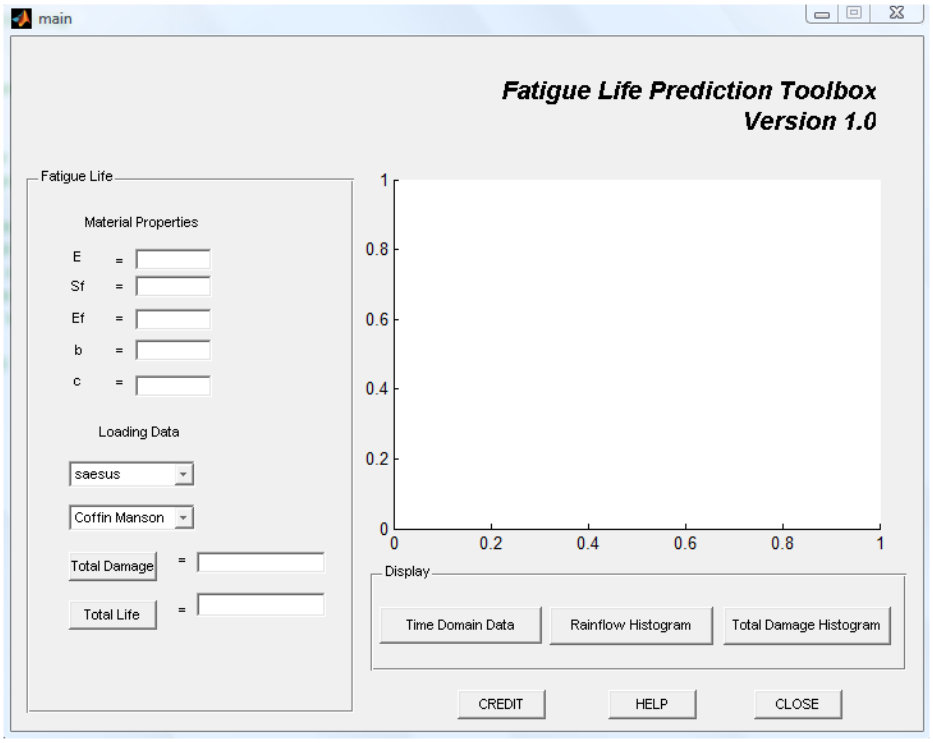


Figure 4.1: Main Interface

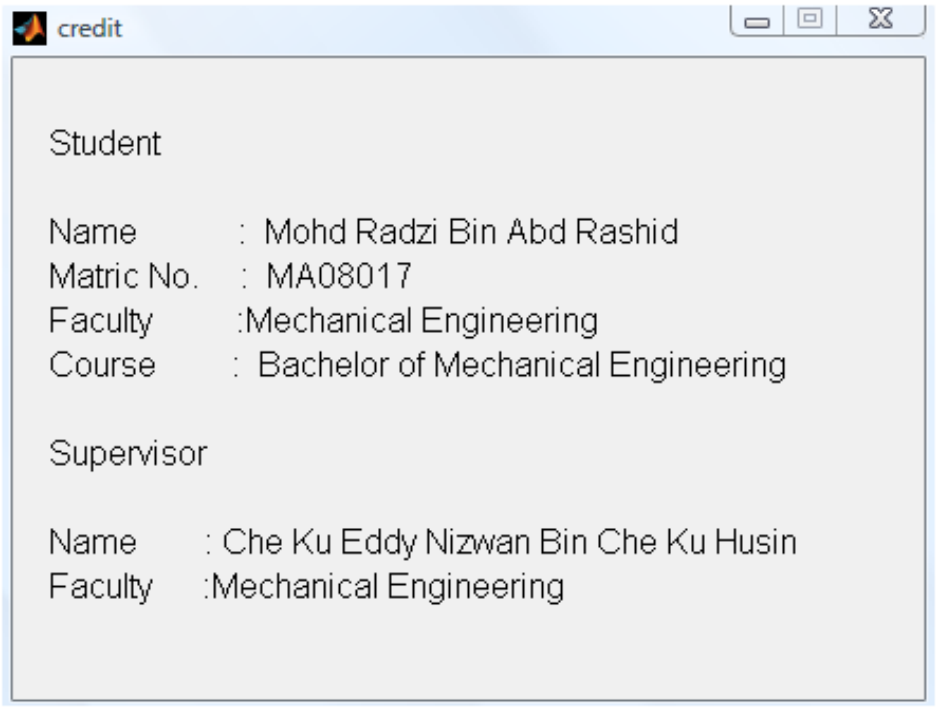


Figure 4.2: Credit Interface

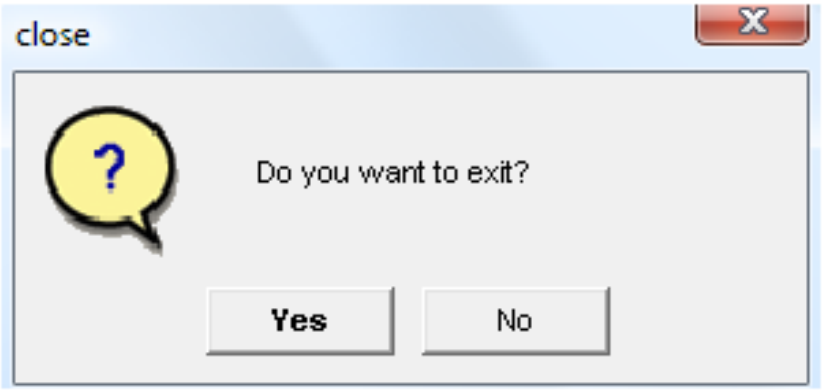


Figure 4.3: Close Interface

When the user click Close button on the GUI, the close appears as shown in Figure 4.3 which is by clicking the YES button closes both the close programme and the GUI calls it. But when the user clicking the NO button closes just the programme.

4.3 INTERFACE MATLAB® GUI SOFTWARE

4.3.1 Display the loading history

The material properties need to be filling under the material properties panel. The users need to choose the loading data and the method for calculating the fatigue life. By clicking the time domain data pushbutton, the loading history will appear at the axes as shown in Figure 4.4.

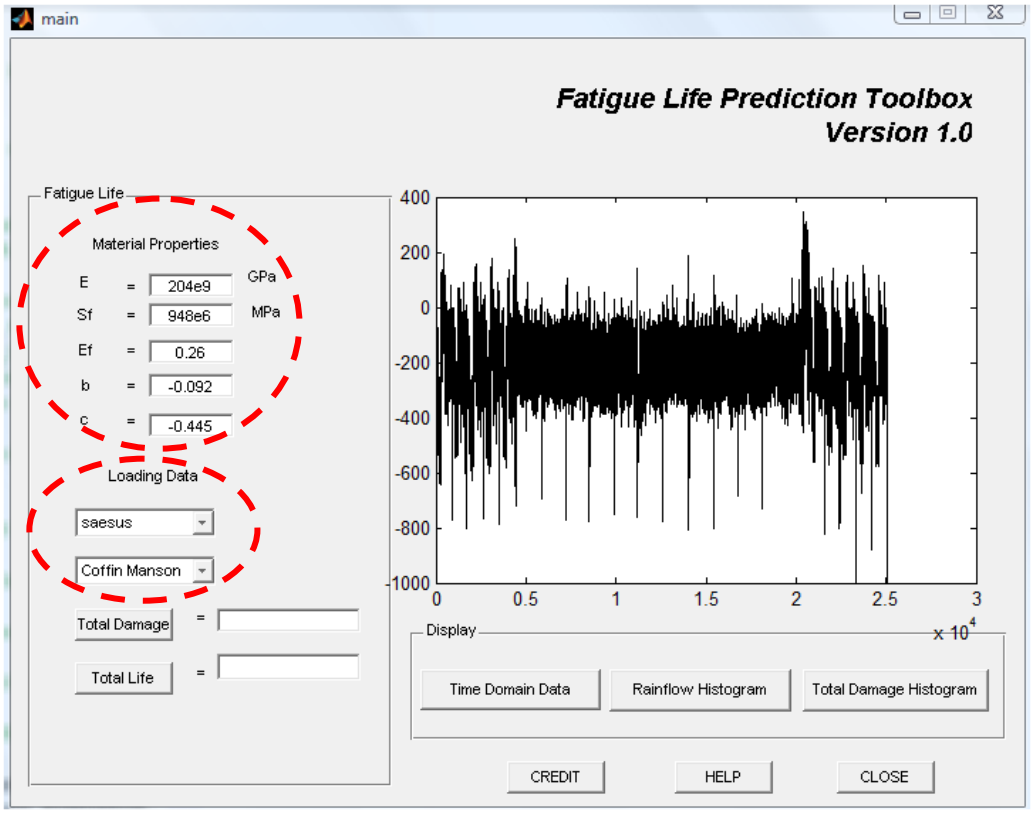


Figure 4.4: Time loading history

Loading is another major input to the fatigue analysis. Loading information can be obtained using a number of different methods. Several types of variable amplitude loading history from SAE standard. For this software, the load history has a predominantly compressive (negative) mean that is referred as suspension history. The variable amplitude load-time histories are shown in Figure 4.5. The term SAESUS represent the load-time history for the suspension respectively. The considered load-time histories are based on the SAE's profile. The abscissa is the time, in seconds.

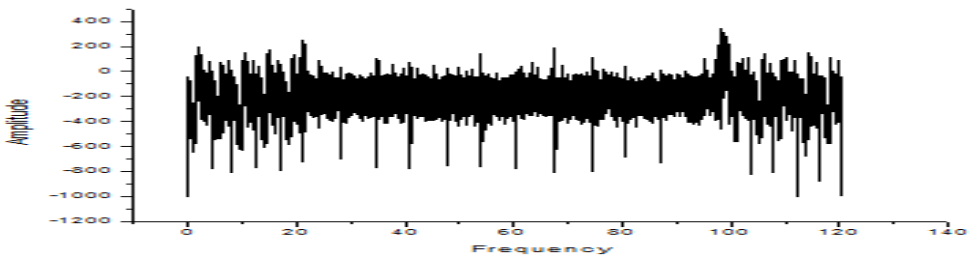


Figure 4.5: SAE standard suspension (SAESUS) loading

4.3.2 Display the rainflow histogram

The rainflow histogram can be display by clicking the rainflow histogram pushbutton. The cycle from the time loading data extracted by peak to peak using rainflow cycle counting method. This cycle counting method can be referred in literature review. The M-files of rainflow cycle shown in Appendix. The Figure 4.6 has shown the interface after clicking the pushbutton.

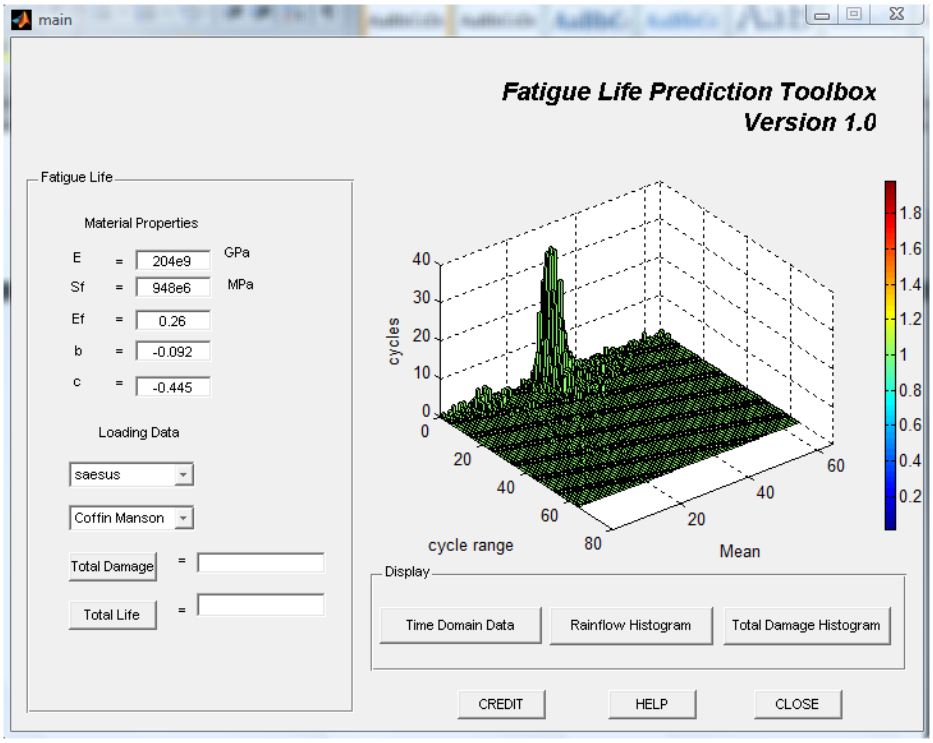


Figure 4.6: Rainflow histogram

4.3.3 Display the total damage histogram

The total damage can be display by clicking pushbutton total damage histogram. Figure 4.7 shown the interface. The detail of total damage histogram will be discussed in next sub chapter. Total life and total damage value can be calculated by clicking the pushbutton. The interface as shown in Figure 4.8 and Figure 4.9.

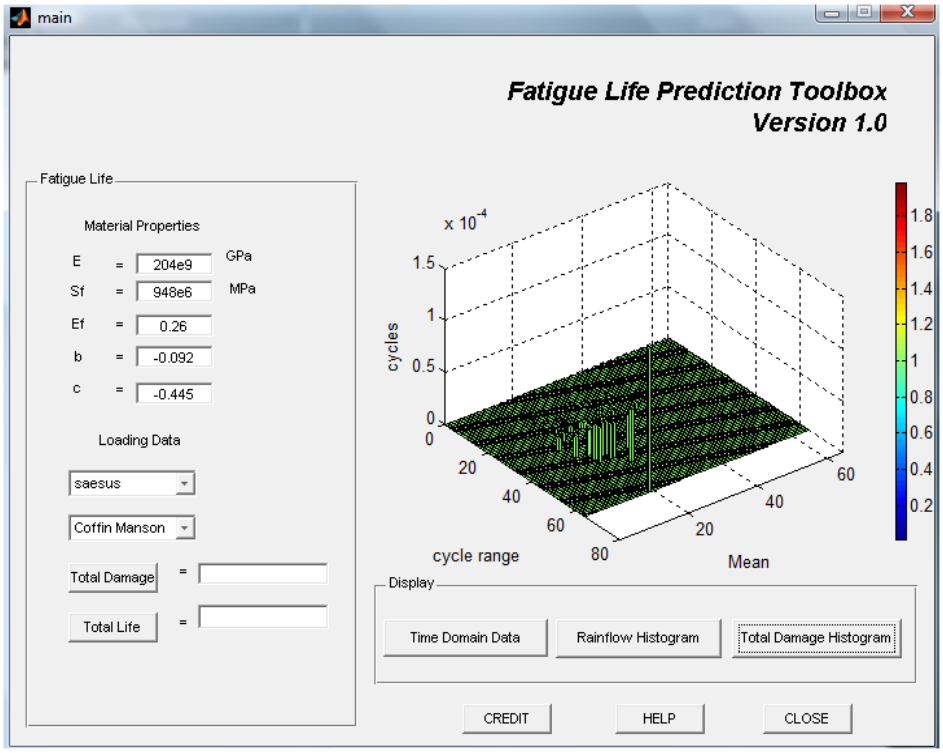


Figure 4.7: Total damage histogram

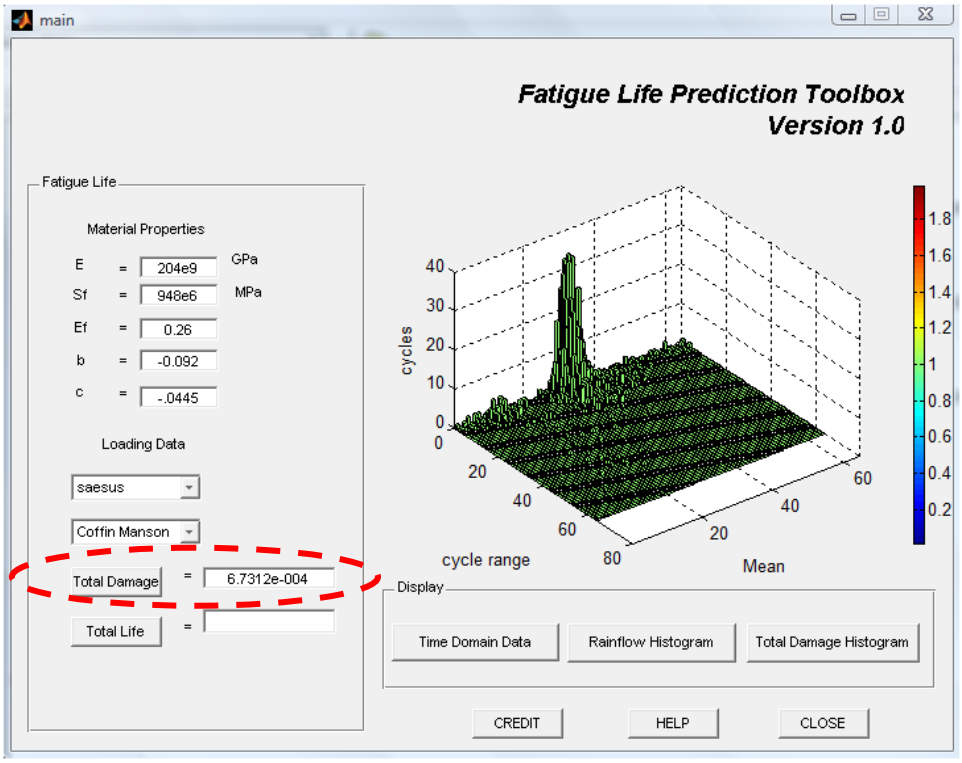


Figure 4.8: Total damage value

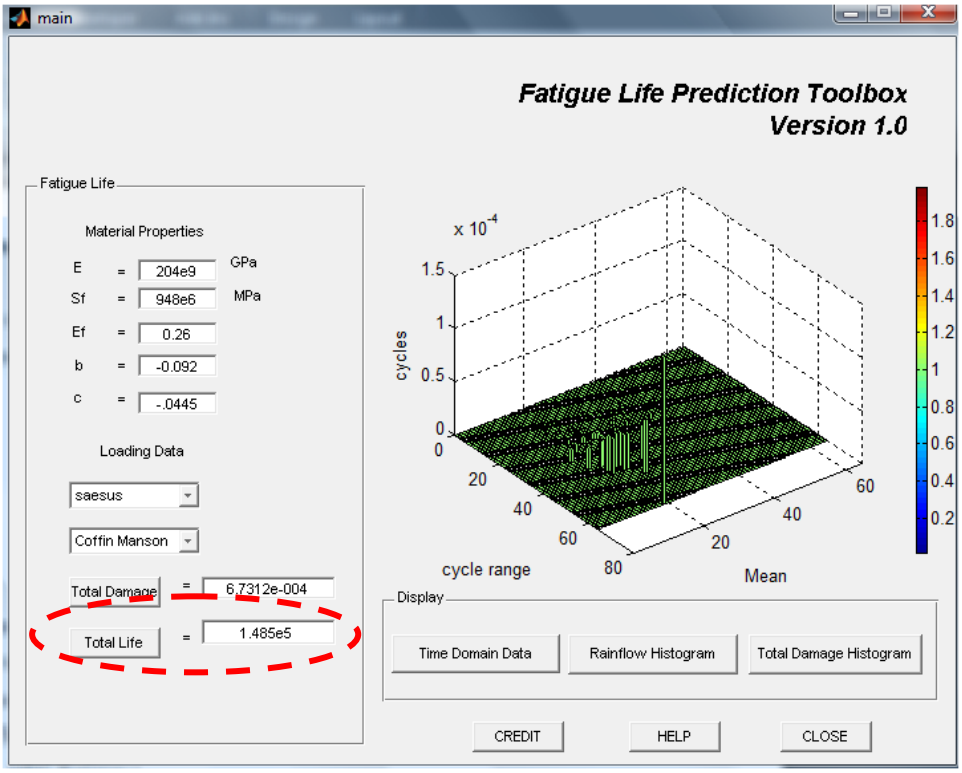


Figure 4.9: Total life value

4.4 DISCUSSION

For fatigue life prediction, strain-life approach is used using Coffin-Manson model. A steel material used for the purpose of stimulation is SAE1045 steel. This material selected because it is a material that commonly used in the automotive industry. In addition, these materials are also chosen because it is widely used in the investigation of the fatigue life parameter determination *'Society of Automotive Engineers Fatigue Design and Evaluation', SAEFDE* (Pals & Stephen 2004). Mechanical properties of SAE1045 are shown in the Table 4.1.

For the simulation using software that has been developed, the fatigue data had been input will converted to the cycle of fatigue by using the *rainflow*. The data that be used is SAESUS data. Each cycle fatigue is displayed in a three-dimensional surface shown in Figure 4.10. In Figure 4.10, x-axis represents the range of strains, the y-axis represents the strain mean and z-axis represents the number of fatigue cycles in the

same range and mean. The results obtained in the majority of the signal cycle fatigue have a range of small amplitude.

Table 4.1: Mechanical properties of SAE1045

Mechanical Properties	SAE1045 Steel
Modulus of Elasticity, *E* (Gpa)	204
Fatigue Strength Coefficient, $\sigma_{f`}$ (Mpa)	948
Fatigue Strength Exponent, *b*	-0.092
Fatigue Ductility Exponent, *c*	-0.445
Fatigue Ductility Coefficient, $\varepsilon_{f`}$	0.26

Sources: nSoft 2001

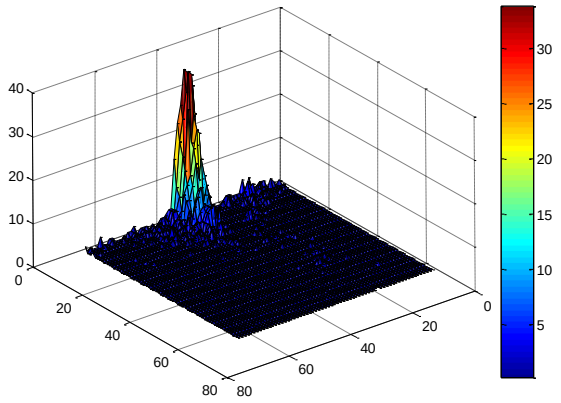


Figure 4.10: SAESUS cycle distribution in three-dimensional surface

Fatigue damage for each cycle is calculated using model Coffin-Manson (Eq. 2.1). Figure 4.11 shows the three-dimensional surface plot of fatigue damage in the fatigue data that been analyse. In Figure 4.11, the x-axis represents the range of strain for the cycle that obtained by using the 'rainflow', y-axis represents the strain mean and z-axis represents the fatigue damage to the cycle that has the same range and mean. The value of fatigue damage is displayed on the surface get from the fatigue damage in a cycle time's total of the cycle in a fatigue signal. The number of cycles is shown in the surface distribution of the cycle. From the surface of fatigue damage, the high fatigue damage was found in the cycle there is a high cycle range. While the cycle in the range

of small strain gives the fatigue damage low even has a lot of cycles. Palmgren-Miner rules are used to determine the overall of fatigue life for variable loading. Overall damage value is the total of cumulative fatigue damage for each cycle fatigue. Then, the fatigue life obtained by calculating the reciprocal value of the overall fatigue damage.

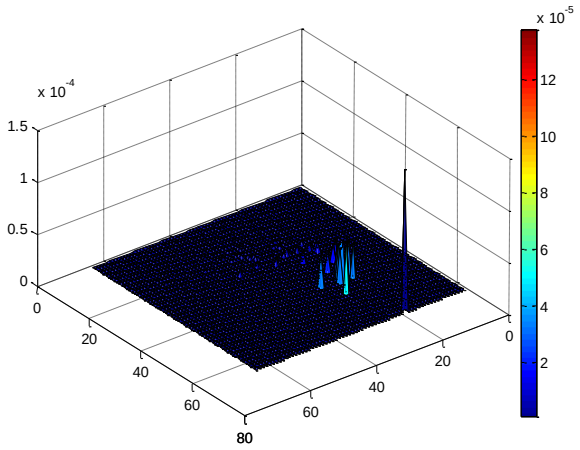


Figure 4.11: SAESUS fatigue damage distribution three-dimensional surface for fatigue damage

4.4.1 Comparison result with MSC Fatigue Software

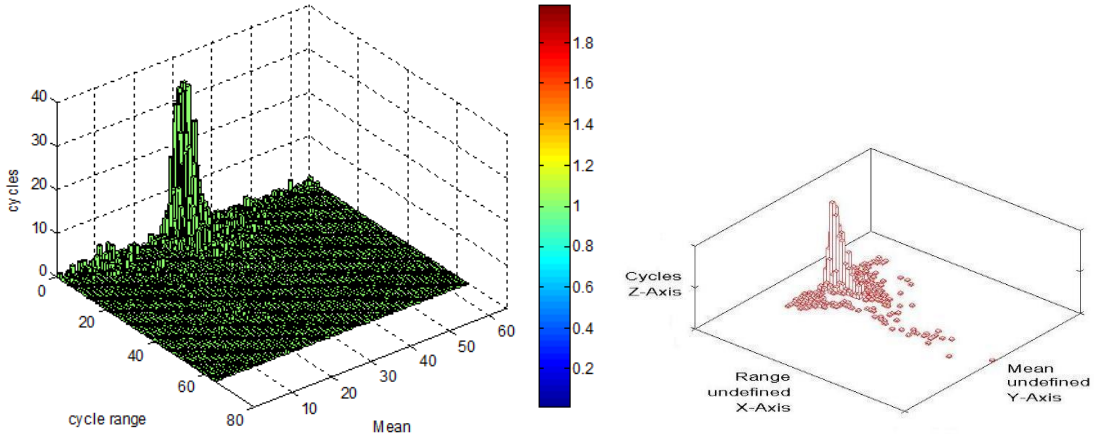
The comparison between the software that has been developed and the MSC.Fatigue software show the result of total life in Table 4.2.

Table 4.2: Total life result

GUI Software	MSC Fatigue
1.485e5 (in sec)	1.644e5 (in sec)

From the result above, it show two different value of the total life between using GUI software and MSC.Fatigue. The different between two value because of MSC.Fatigue use the complex and more accurate algorithm compare to the GUI software. The iteration method that used in GUI software give less decimal places compare to MSC.Fatigue software.

The distribution of rainflow cycle that display either from GUI software or MSC.Fatigue is most the same. For the GUI software, the raiflow histogram plot with cycle range and mean is 64 bin. The Figure 4.12(a) and 4.12(b) shows the rainflow histogram for each software.



(a) (b)

Figure 4.12: Cycle distribution in three dimension histogram; (a) GUI Software (b) MSC.Fatigue

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 CONCLUSION

This project is successfully developed. Graphical User Interface (GUI) for calculating fatigue life using variable amplitude loading data has been presented. There are many function in using MATLAB®, with the function in MATLAB®, it can create GUIDE and design the layout of the GUI. From the GUI, it can show many thing based on its application. For this project, the GUI is creating to display the total life and total damage based on variable amplitude loading data which is only focus on the Low Cycle Fatigue by using Coffin-Manson method. The knowledge about this has been studied from the literature review.

The first objective for this thesis is to create and design GUI using GUIDE in MATLAB® Software package to make an easier for the user to use. The design in GUI must be user-friendly to make sure the use understand to use it. The next objective is Develop the Fatigue Life Prediction Algorithm using methods Coffin Manson by implement iteration method to solve the equation. The data from SAESUS will be used to calculate fatigue life for variable amplitude loading data. This method will display a fatigue life in MATLAB® GUI. The result can achieve after this software running smoothly using the model fatigue.

The objective of this project is to interface the MATLAB® GUI that is achieved. The main contribution of this project is to interfacing the GUI.

5.2 RECOMMENDATION

The recommendations for future development of this software are:

i. This software only uses Coffin-Manson method for calculating fatigue life. So for the future this software can be developed further by adding another method such as Morrow, Smith, Watson and Topper (SWT) and DuQuesnay.

ii. The other algorithm development methods can be used for enhancement this software. There is many other iterations method where it is faster and more converge to the fixed point.

REFERENCES

Ariduru S. 2004. *Fatigue Life Calculation by Rainflow Cycle Counting Method*. M. Sc. Thesis. Middle East Technical University.

ASTM Standard E1049. 1985. *Standard Practice for Cycle Counting in Fatigue Analysis.* Philadelphia: ASTM

Christian Lalanne, 1999. *Mechanical Vibration & Shock, Fatigue Damage, Volume IV,* Taylor and Francis Books, Inc.

D. Kujawski, F. Ellyin. 1984. A Cumulative Damage Theory for Fatigue Crack Initiation and Propagation. *International Journal of Fatigue,* 6:119-137.

G. Bhuyan, O. Vosikovsky. 1989. Prediction of Fatigue Crack Initiation Lives for Welded Plate T-joints Based on the Local Stress-Strain Approach. *International Journal of Fatigue,* 11:153-159.

Hanselman D., Field B. L. 2005. *Mastering MATLAB7*. Pearson & Prentice Hall, Inc.

J. Szusta, A. Seweryn. 2010. Low-Cycle Fatigue Model of Damage Accumulation – The Strain Approach. *Engineering Fracture Mechanics,* 77:1604-1616.

L. Molent, M. McDonald, S. Barter, and R. Jones. 2008. Evaluation of Spectrum Fatigue Crack Growth Using Variable Amplitude Data. *International Journal of Fatigue,* 30:119-137.

Manson S.S 1965. Fatigue: a complex subject – some simple approximation. *Experimental Mechanics 5*: 193-226

Masahiro Jono. 2005. Fatigue Damage and Crack Growth Under Variable Amplitude Loading with Reference to the Counting Methods of Stress–Strain Ranges. *International Journal of Fatigue,* 27:1006-1015.

Matsuishi, M. & Endo, T. 1968. *Fatigue of materials subjected to varying stress.* Fukuoka: Japan Society of Mechanical Engineers.

Memon, I.R., Zhang, X. & Cui, D. 2002. Fatigue Life Prediction of 3-D Problems by Damage Mechanics with Two-Block Loading. *International Journal of Fatigue* 24: 29-37.

Miner, M.A. 1945. Cumulative Damage In Fatigue. *Journal of Applied Mechanics* 67: 159-164

Morrow, J.D., 1968. Fatigue Design Handbook. In: *Advances in engineering*. Warrendale: Society of Automotive Engineers. 4:21-29.

Morrow, D. & Vold, H. 1997. *Compression of Time Histories Used for Component Fatigue Evaluation SAE930403 in PT-67 in Recent Developments in Fatigue Technology*. USA: Society of Automotive Engineers (SAE).

Nizwan C. K. E, 2009. *Pembangunan Algoritma Penyuntingan Data Lesu Menggunakan Kaedah Penjelmaan Masa-Frekuensi*. S. Sn. Tesis. Universiti Kebangsaan Malaysia.

nSoft® User Manual. 2001. *nSoft V5.3 Online Documentation*. Sheffield: nCode International Ltd.

Palmgren, A. 1924. Die Lebensdauer von Kugellagern. *Verfahrenstechinik*. Berlin, 68: 339-341.

Ramberg, W., & Osgood, W. R. 1943. Description of Stress-Strain Curves by Three Parameters. Technical Note No. 902. Washington DC: National Advisory Committee For Aeronautics.

Richard C.Rise, Brian N.Leis, Drew V.Nelson, Henry D.Berns, Dan Lingenfelser, M.R. Mitchell, 1988. *Fatigue Design Handbook, Society of Automotive Engineers,* Inc.

Richard, C. Rise, Leis, B. N. & Drew Nelson. 1997. *Fatigue Design Handbook*. 3rd Edition. Warrendele: Society of Automotive Engineers.

Smith, K. N., Watson, P. & Topper, T. H. 1970. A stress-strain function for the fatigue of metals. *Journal of Materials*, JMLSA 5 (4): 767-778.

Sonsino, C. M. 2007. Fatigue testing under variable amplitude loading, *International Journal of Fatigue* 29 (6): 1080-1089.

Stephens, R. I., Ali Fatemi, Stephens, R. R. & Henry, O. F. 2001. *Metal Fatigue in Engineering*. New York: John Wiley & Sons, Inc.

Stephens, R. I., 2001. *Metal Fatigue in Engineering*. 2nd Edition. New York: John Wiley & Sons, Inc.

Stephens, R. I., Dindinger, P. M. & Gunger, J. E. 1997. Fatigue Damage Editing for Accelerated Durability Testing Using Strain Range and SWT Parameter Criteria. *International Journal of Fatigue* 19: 599-606.

J. Palm III W., *Introduction to MATLAB7 for Engineers*, 2005, McGraw Hill, Americas, New York, NY.

Yung-Li Lee, Jwo Pan, Hathaway, R. & Barkey, M. 2005. *Fatigue Testing and Analysis (Theory and Practice)*. United Kingdom: Elsevier Inc.

APPENDICES A1

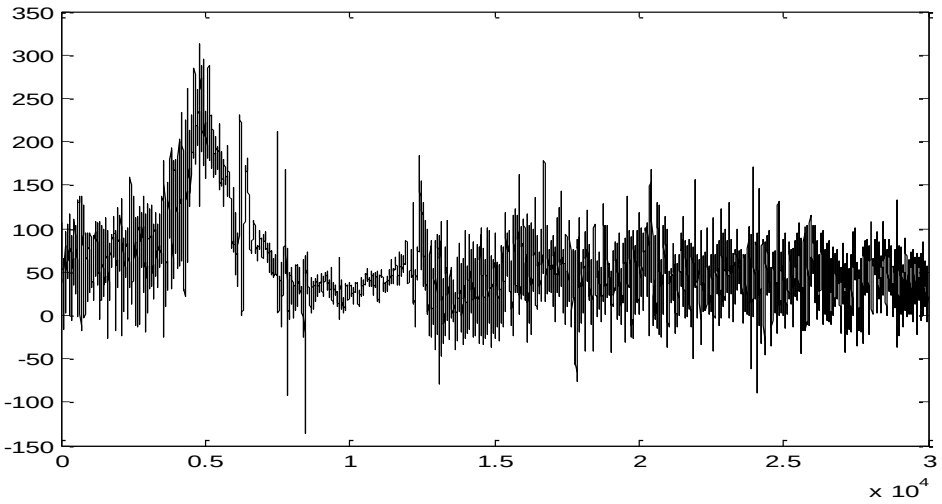


Figure 6.1: Time loading history for D1

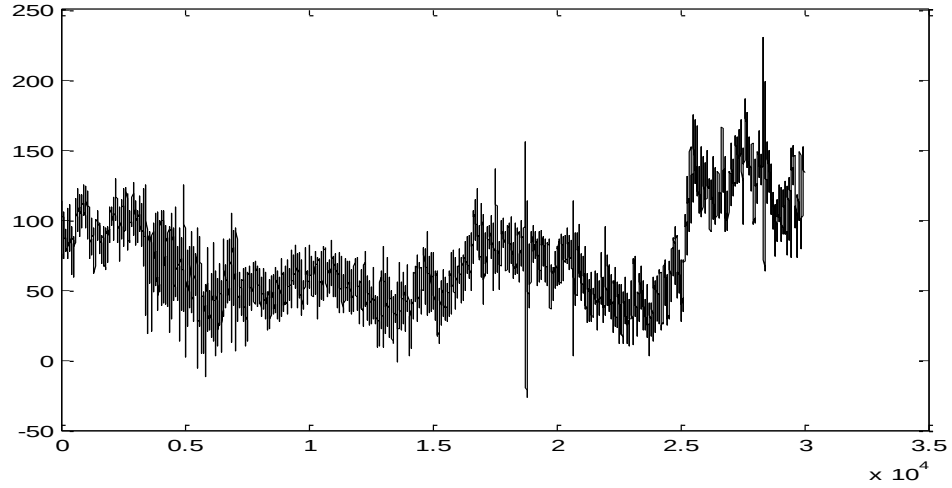


Figure 6.2: Time loading history for D2

APPENDICES A2

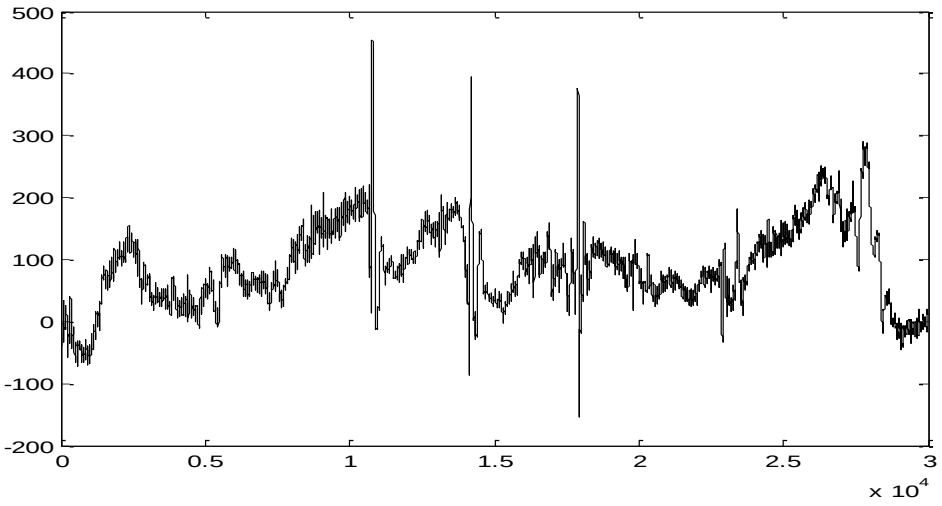


Figure 6.3: Time loading history for D3

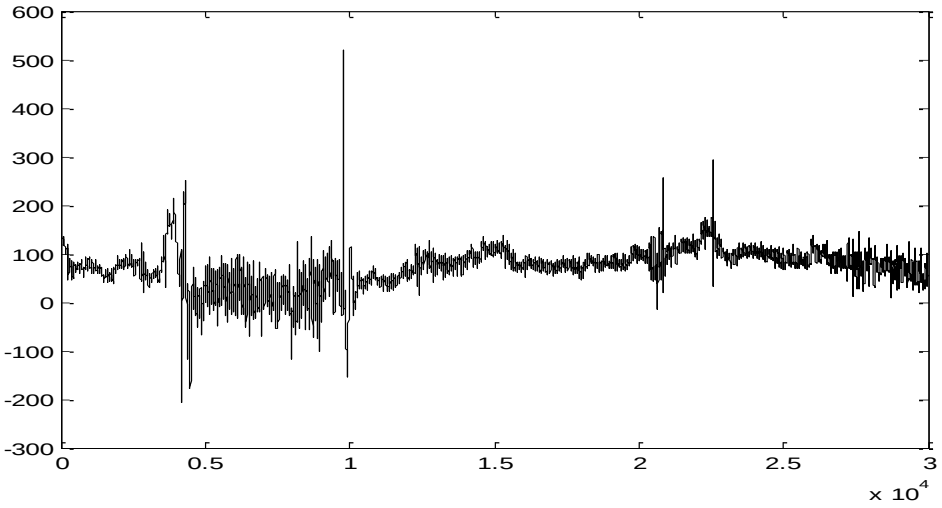


Figure 6.4: Time loading history for D4

APPENDICES B1

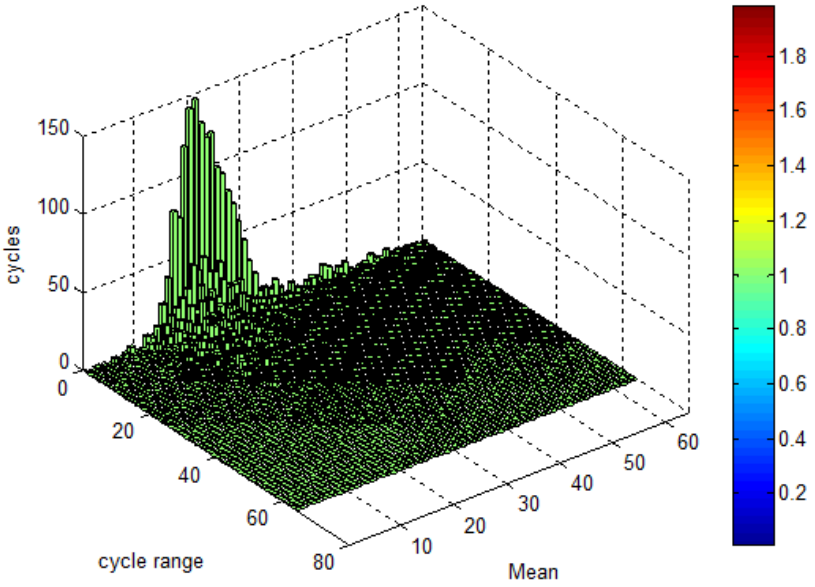


Figure 6.5: Rainflow histogram for D1

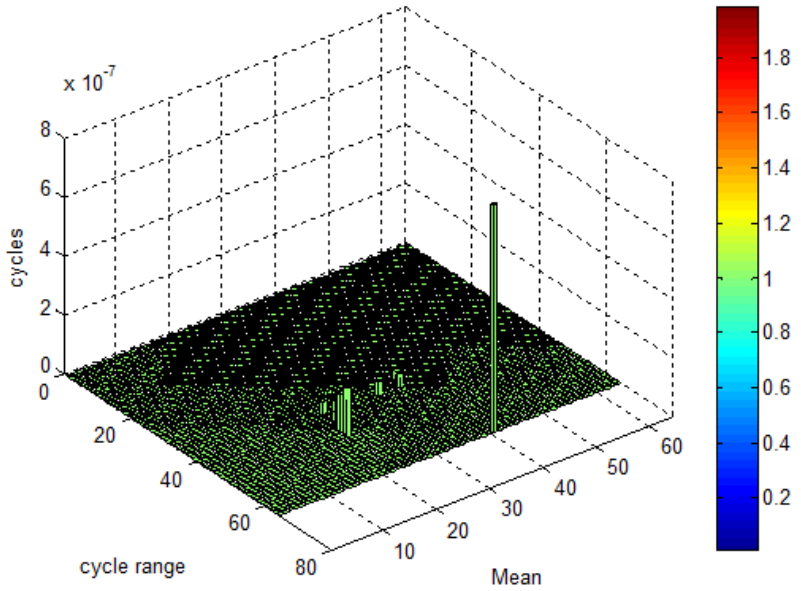


Figure 6.6: Total damage histogram for D1

APPENDICES B2

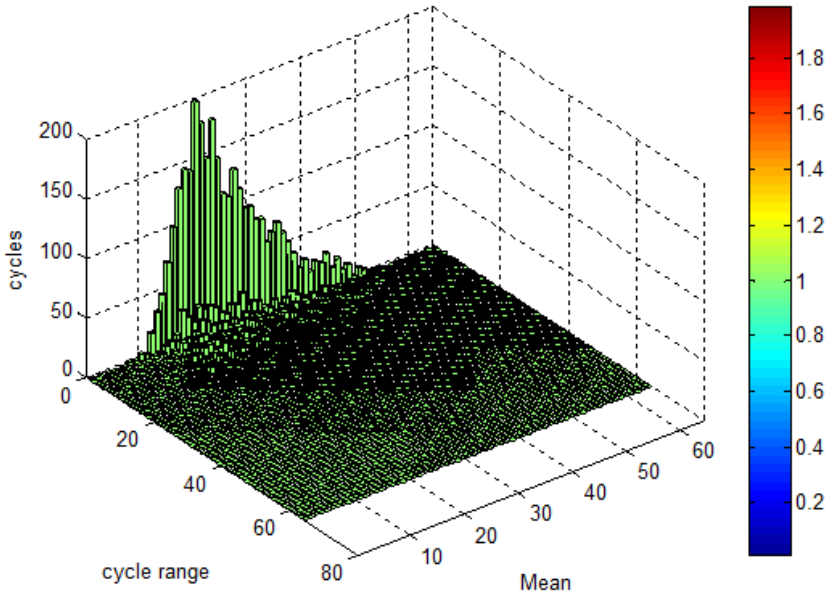


Figure 6.7: Rainflow histogram for D2

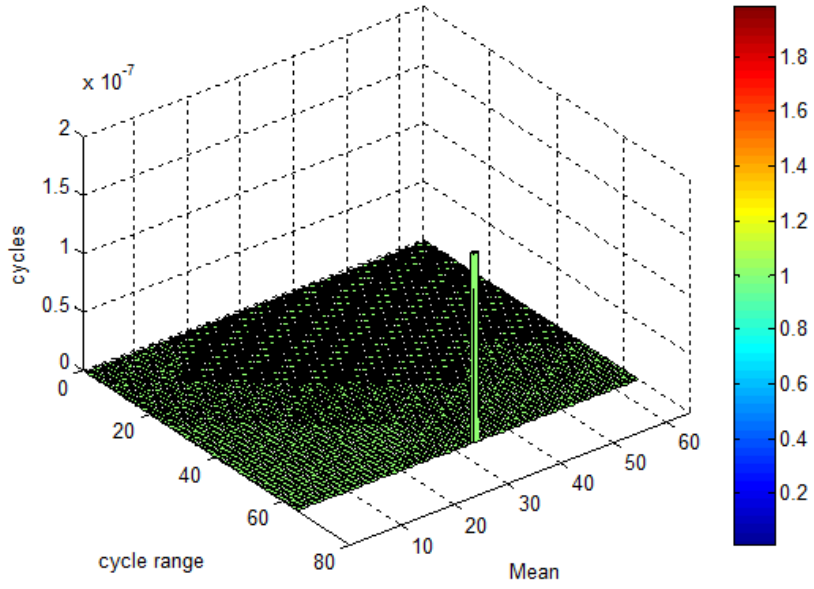


Figure 6.8: Total damage histogram for D2

APPENDICES B3

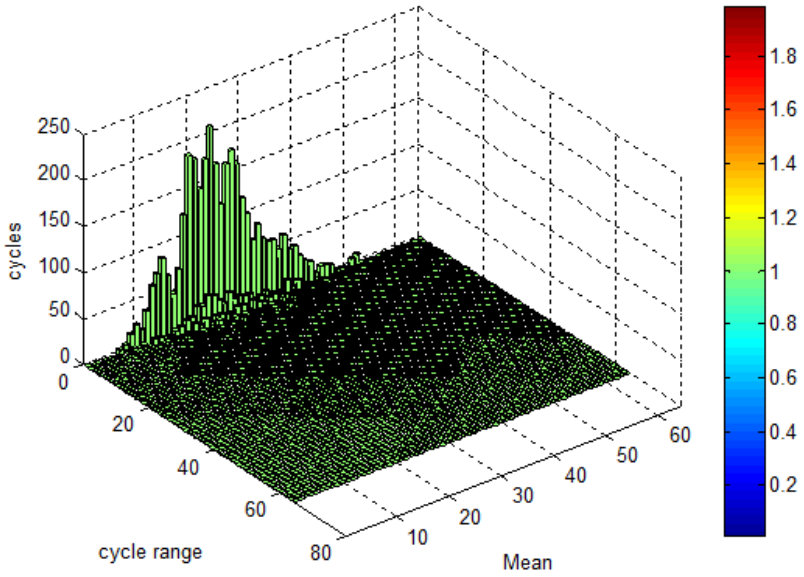


Figure 6.9: Rainflow histogram for D3

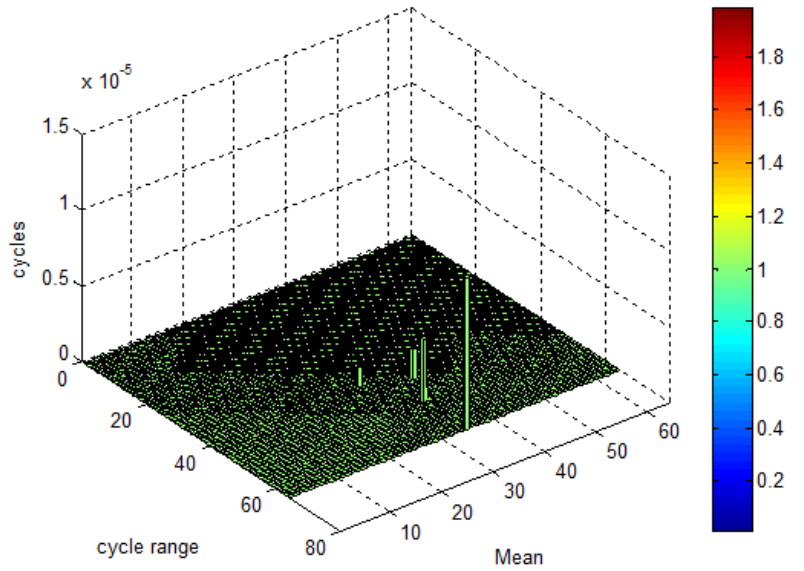


Figure 6.10: Total damage histogram for D3

APPENDICES B4

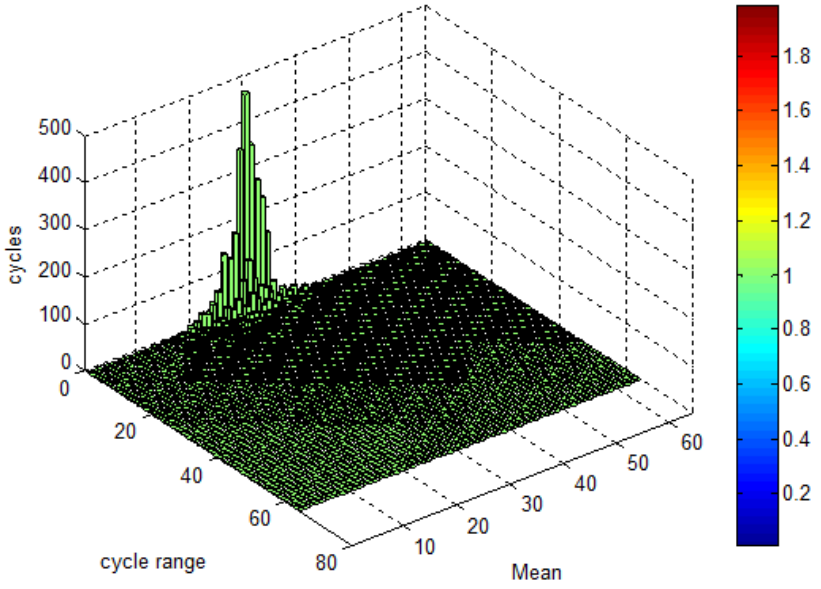


Figure 6.11: Rainflow histogram for D4

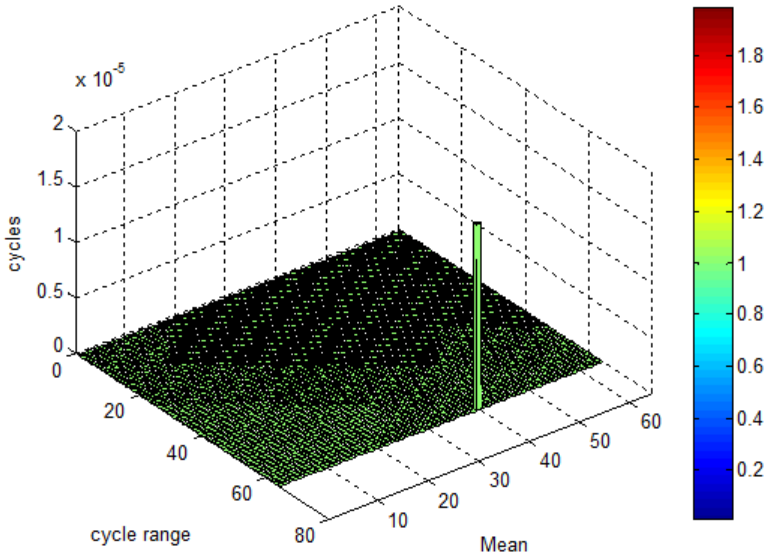


Figure 6.12: Total damage histogram for D4

APPENDICES C1

```
function varargout = main(varargin)
% MAIN M-file for main.fig
%      MAIN, by itself, creates a new MAIN or raises the existing
%      singleton*.
%
%      H = MAIN returns the handle to a new MAIN or the handle to
%      the existing singleton*.
%
%      MAIN('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MAIN.M with the given input arguments.
%
%      MAIN('Property','Value',...) creates a new MAIN or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before main_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to main_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help main

% Last Modified by GUIDE v2.5 24-Oct-2010 01:24:07

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @main_OpeningFcn, ...
                   'gui_OutputFcn',  @main_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before main is made visible.
function main_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to main (see VARARGIN)

% Choose default command line output for main
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes main wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = main_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
```

```
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of edit8 as a double


% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit9 as text
%        str2double(get(hObject,'String')) returns contents of edit9 as a double

% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10 as a double

% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11 as a double

% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

val = get(hObject,'Value');
str = get(hObject, 'String');
switch str{val};
case 'saesus' % User selects saesus
handles.current_data = handles.saesus;
case 'D1' % User selects D1
handles.current_data = handles.D1;
case 'D2' % User selects D2
handles.current_data = handles.D2;
case 'D3' % User selects D3
handles.current_data = handles.D3;
case 'D4' % User selects D4
handles.current_data = handles.D4;
end
guidata(hObject,handles)

% Hints: contents = get(hObject,'String') returns popupmenu2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu2

% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
```

```
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

load saesus.txt;
assignin('base','saesus',saesus);
z=saesus.*10e-6;
assignin('base','z',z);
[cyclerange, Emin, Emax, meancycle,maxcycle]=RFCFunct(z);
assignin('base','cyclerange',cyclerange);
assignin('base','Emin',Emin);
assignin('base','Emax',Emax);
assignin('base','meancycle',meancycle);
assignin('base','maxcycle',maxcycle);
cyclerange=cyclerange.*0.5;
assignin('base','cyclerange',cyclerange);
Emin=Emin.*0.5;
assignin('base','Emin',Emin);
Emax=Emax.*0.5;
assignin('base','Emax',Emax);
meancycle=meancycle.*0.5;
assignin('base','meancycle',meancycle);
maxcycle=maxcycle.*0.5;
assignin('base','maxcycle',maxcycle);
range=maxcycle/64:maxcycle/64:maxcycle;
assignin('base','range',range);
[damage]=DamageFunct(range)
assignin('base','damage',damage);

X=[cyclerange',meancycle'];
assignin('base','X',X);

      Z=hist3(X,[64 64]);
      assignin('base','Z',Z);
for a=1:64
   assignin('base','a',a);
   D(a,:)=Z(a,:).*damage(a);
   assignin('base','D',D);
end
totaldamage0=sum(D)
```

```
assignin('base','totaldamage0',totaldamage0);
totaldamage=sum(totaldamage0)
assignin('base','totaldamage',totaldamage);
set(handles.edit12,'string',totaldamage);

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

load saesus.txt;
assignin('base','saesus',saesus);
z=saesus.*10e-6;
assignin('base','z',z);
[cyclerange, Emin, Emax, meancycle,maxcycle]=RFCFunct(z);
assignin('base','cyclerange',cyclerange);
assignin('base','Emin',Emin);
assignin('base','Emax',Emax);
assignin('base','meancycle',meancycle);
assignin('base','maxcycle',maxcycle);
cyclerange=cyclerange.*0.5;
assignin('base','cyclerange',cyclerange);
Emin=Emin.*0.5;
assignin('base','Emin',Emin);
Emax=Emax.*0.5;
assignin('base','Emax',Emax);
meancycle=meancycle.*0.5;
assignin('base','meancycle',meancycle);
maxcycle=maxcycle.*0.5;
assignin('base','maxcycle',maxcycle);
range=maxcycle/64:maxcycle/64:maxcycle;
assignin('base','range',range);
[damage]=DamageFunct(range)
assignin('base','damage',damage);

X=[cyclerange',meancycle'];
assignin('base','X',X);

      Z=hist3(X,[64 64]);
      assignin('base','Z',Z);
for a=1:64
   assignin('base','a',a);
   D(a,:)=Z(a,:).*damage(a);
   assignin('base','D',D);
end
totaldamage0=sum(D)
assignin('base','totaldamage0',totaldamage0);
totaldamage=sum(totaldamage0)
```

```
assignin('base','totaldamage',totaldamage);
NF=1/totaldamage
assignin('base','NF',NF);
set(handles.edit13,'string',NF);

function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12 as text
%        str2double(get(hObject,'String')) returns contents of edit12 as a double

% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit13 as text
%        str2double(get(hObject,'String')) returns contents of edit13 as a double

% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on button press in pushbutton3.
```

```
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);
cla;

popup_sel_index=get(handles.popupmenu2,'value');
switch popup_sel_index
   case 1
      load saesus.txt;
      assignin('base','saesus',saesus);
      data=saesus;
      assignin('base','data',data);
      T=1;
      assignin('base','T',T);
      t=[T:T:T*length(data)];
      assignin('base','t',t);
      plot(t,data,'color','k');
   case 2
      load D1.txt;
      assignin('base','D1',D1);
      data=D1;
      assignin('base','data',data);
      T=1;
      assignin('base','T',T);
      t=[T:T:T*length(data)];
      assignin('base','t',t);
      plot(t,data,'color','k');
   case 3
      load D2.txt;
      assignin('base','D2',D2);
      data=D2;
      assignin('base','data',data);
      T=1;
      assignin('base','T',T);
      t=[T:T:T*length(data)];
      assignin('base','t',t);
      plot(t,data,'color','k');
   case 4
      load D3.txt;
      assignin('base','D3',D3);
      data=D3;
      assignin('base','data',data);
      T=1;
      assignin('base','T',T);
      t=[T:T:T*length(data)];
      assignin('base','t',t);
      plot(t,data,'color','k');
```

```
    case 5
        load D4.txt;
        assignin('base','D4',D4);
        data=D4;
        assignin('base','data',data);
        T=1;
        assignin('base','T',T);
        t=[T:T:T*length(data)];
        assignin('base','t',t);
        plot(t,data,'color','k');
end

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);
cla;

popup_sel_index=get(handles.popupmenu2,'value');
switch popup_sel_index
    case 1
        load saesus.txt;
        assignin('base','saesus',saesus);
        z=saesus;
        assignin('base','z',z);
        [cyclerange, Emin, Emax, meancycle]=RFCFunct(z);
        assignin('base','cyclerange',cyclerange);
        assignin('base','Emin',Emin);
        assignin('base','Emax',Emax);
        assignin('base','meancycle',meancycle);

                X=[cyclerange',meancycle'];
                assignin('base','X',X);
                Z=hist3(X,[64 64]);
                assignin('base','Z',z);
                h = bar3(Z);
                assignin('base','h',h);
                for i=1:length(h)
                    zdata = ones(6*length(h),4);
                    assignin('base','zdata',zdata);
                    k=1;
                    assignin('base','k',k);
                    for j=0:6:(6*length(h)-6);
                        assignin('base','j',j);
                        k=k+1;
                        assignin('base','k',k);
                    end
```

```
            set(h(i),'Cdata',zdata);
        end
        colormap jet
        colorbar
        xlabel('Mean');ylabel('cycle range');zlabel('cycles');

case 2
    load D1.txt;
    assignin('base','D1',D1);
    z=D1;
    assignin('base','z',z);
    [cyclerange, Emin, Emax, meancycle]=RFCFunct(z);
    assignin('base','cyclerange',cyclerange);
    assignin('base','Emin',Emin);
    assignin('base','Emax',Emax);
    assignin('base','meancycle',meancycle);

        X=[cyclerange',meancycle'];
        assignin('base','X',X);
        Z=hist3(X,[64 64]);
        assignin('base','Z',z);
        h = bar3(Z);
        assignin('base','h',h);
        for i=1:length(h)
            zdata = ones(6*length(h),4);
            assignin('base','zdata',zdata);
            k=1;
            assignin('base','k',k);
            for j=0:6:(6*length(h)-6);
                assignin('base','j',j);
                k=k+1;
                assignin('base','k',k);
            end
            set(h(i),'Cdata',zdata);
        end
        colormap jet
        colorbar
        xlabel('Mean');ylabel('cycle range');zlabel('cycles');

case 3
    load D2.txt;
    assignin('base','D2',D2);
    z=D2;
    assignin('base','z',z);
    [cyclerange, Emin, Emax, meancycle]=RFCFunct(z);
    assignin('base','cyclerange',cyclerange);
    assignin('base','Emin',Emin);
    assignin('base','Emax',Emax);
    assignin('base','meancycle',meancycle);
```

```
        X=[cyclerange',meancycle'];
        assignin('base','X',X);
        Z=hist3(X,[64 64]);
        assignin('base','Z',z);
        h = bar3(Z);
        assignin('base','h',h);
        for i=1:length(h)
           zdata = ones(6*length(h),4);
           assignin('base','zdata',zdata);
           k=1;
           assignin('base','k',k);
           for j=0:6:(6*length(h)-6);
              assignin('base','j',j);
              k=k+1;
              assignin('base','k',k);
           end
           set(h(i),'Cdata',zdata);
        end
        colormap jet
        colorbar
        xlabel('Mean');ylabel('cycle range');zlabel('cycles');

case 4
   load D3.txt;
   assignin('base','D3',D3);
   z=D3;
   assignin('base','z',z);
   [cyclerange, Emin, Emax, meancycle]=RFCFunct(z);
   assignin('base','cyclerange',cyclerange);
   assignin('base','Emin',Emin);
   assignin('base','Emax',Emax);
   assignin('base','meancycle',meancycle);

        X=[cyclerange',meancycle'];
        assignin('base','X',X);
        Z=hist3(X,[64 64]);
        assignin('base','Z',z);
        h = bar3(Z);
        assignin('base','h',h);
        for i=1:length(h)
           zdata = ones(6*length(h),4);
           assignin('base','zdata',zdata);
           k=1;
           assignin('base','k',k);
           for j=0:6:(6*length(h)-6);
              assignin('base','j',j);
              k=k+1;
              assignin('base','k',k);
           end
```

```
            set(h(i),'Cdata',zdata);
        end
        colormap jet
        colorbar
        xlabel('Mean');ylabel('cycle range');zlabel('cycles');

    case 5
        load D4.txt;
        assignin('base','D4',D4);
        z=D4;
        assignin('base','z',z);
        [cyclerange, Emin, Emax, meancycle]=RFCFunct(z);
        assignin('base','cyclerange',cyclerange);
        assignin('base','Emin',Emin);
        assignin('base','Emax',Emax);
        assignin('base','meancycle',meancycle);

        X=[cyclerange',meancycle'];
        assignin('base','X',X);
        Z=hist3(X,[64 64]);
        assignin('base','Z',z);
        h = bar3(Z);
        assignin('base','h',h);
        for i=1:length(h)
            zdata = ones(6*length(h),4);
            assignin('base','zdata',zdata);
            k=1;
            assignin('base','k',k);
            for j=0:6:(6*length(h)-6);
                assignin('base','j',j);
                k=k+1;
                assignin('base','k',k);
            end
            set(h(i),'Cdata',zdata);
        end
        colormap jet
        colorbar
        xlabel('Mean');ylabel('cycle range');zlabel('cycles');
end

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


load saesus.txt;
assignin('base','saesus',saesus);
```

```
z=saesus.*10e-6;
assignin('base','z',z);
[cyclerange, Emin, Emax, meancycle,maxcycle]=RFCFunct(z);
cyclerange=cyclerange.*0.5;
assignin('base','cyclerange',cyclerange);
Emin=Emin.*0.5;
assignin('base','Emin',Emin);
Emax=Emax.*0.5;
assignin('base','Emax',Emax);
meancycle=meancycle.*0.5;
assignin('base','meancycle',meancycle);
maxcycle=maxcycle.*0.5;
assignin('base','maxcycle',maxcycle);
range=maxcycle/64:maxcycle/64:maxcycle;
assignin('base','range',range);
[damage]=DamageFunct(range)

X=[cyclerange',meancycle'];
assignin('base','X',X);

      Z=hist3(X,[64 64]);
      assignin('base','Z',Z);

for a=1:64
   D(a,:)=Z(a,:).*damage(a);
   assignin('base','D',D);
end
totaldamage0=sum(D)
totaldamage=sum(totaldamage0)
NF=1/totaldamage

      h = bar3(D);
      for i=1:length(h)
         zdata = ones(6*length(h),4);
         assignin('base','zdata',zdata);
         k=1;
         assignin('base','k',k);
         for j=0:6:(6*length(h)-6);
            assignin('base','j',j);
            k=k+1;
            assignin('base','k',k);
         end
         set(h(i),'Cdata',zdata);
      end
      colormap jet
      colorbar
      xlabel('Mean');ylabel('cycle range');zlabel('cycles'
% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

delete(handles.figure1)
```

APPENDICES C2

```
function [cyclerange, Emin, Emax, meancycle,maxcycle]=RFCFunct(z)
N=length(z);
fs=200;
T=1/fs;
ts=N/fs;
t=[ts/N:ts/N:ts];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%define peak or valley for the 1st point
if (z(1)>z(2)),
   peak(1)=z(1);
   i=2;
   j=1;
   k=1;
   m=2;
else
   valley(1)=z(1);
   i=1;
   j=2;
   k=2;
   m=1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

b=1;
for a=1:(length(z)-1)
   if z(a)==z(a+1),
   else
      c(b)=z(a);
      b=b+1;
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%pick peak n valley
for n=1:(length(c)-2),
   if (c(n+1)>max(c(n),c(n+2))),
      peak(i)=c(n+1);
      tpeak(i)=(n+1)*T;
      i=i+1;
   elseif (c(n+1)<min(c(n),c(n+2))),
      valley(j)=c(n+1);
```

```
        tvalley(j)=(n+1)*T;
        j=j+1;
    else
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%create new data(peak n valley data)
for l=1:length(peak),
    newData(k)=peak(l);
    tnewData(k)=tpeak(l);
    k=k+2;
end

for p=1:length(valley),
    newData(m)=valley(p);
    tnewData(m)=tvalley(p);
    m=m+2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%peak to peak

x=max(newData)%max peak

for q=1:length(newData)
    if (newData(q)==x),
    r=q;%point max peak;
    break
    else
    end
end

if newData(length(newData))<newData(length(newData)-1),
    if newData(1)<newData(2),
        if newData(length(newData))>newData(1),
            s=1:r;
            u=r:length(newData)-1;
        else
            s=1:r;
            u=r:length(newData);
        end
    elseif newData(1)>newData(2),
        if newData(length(newData))>newData(1)
            s=2:r;
            u=r:length(newData)-1;
```

```
        else
            s=1:r;
            u=r:length(newData)-1;
        end
    else
    end
else
    if newData(1)<newData(2),
        if newData(length(newData))>newData(1),
            s=1:r;
            u=r:length(newData);
        else
            s=2:r;
            u=r:length(newData)-1;
        end
    elseif newData(1)>newData(2),
        if newData(length(newData))>newData(1),
            s=2:r;
            u=r:length(newData);
        else
            s=1:r;
            u=r:length(newData)-1;
        end
    else
    end
end

PeakToPeak=[newData(u) newData(s)] ;
% subplot(2,2,2)
% plot(PeakToPeak);
% title('Time Histories(peak & valley-peak to peak)');
% ylabel('Amplitude [microstrain]')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%rainflow cycle counting
PTP=PeakToPeak;
Npoint=1;
Ncycle=1;
L=length(PTP);

while L>3
    range1=abs(PTP(Npoint)-PTP(Npoint+1));
    mean1=min([PTP(Npoint) PTP(Npoint+1)])+range1/2;
    range2=abs(PTP(Npoint+1)-PTP(Npoint+2));

     if range1<=range2,
        cyclerange(Ncycle)=range1;
```

```
        meancycle(Ncycle)=mean1;
        Emin(Ncycle)=min([PTP(Npoint) PTP(Npoint+1)]);
        Emax(Ncycle)=max([PTP(Npoint) PTP(Npoint+1)]);

        Ncycle=Ncycle+1;
        if Npoint==2
           PTP=[PTP(1) PTP(Npoint+2:length(PTP))];
        else
           PTP=[PTP(1:Npoint-1) PTP(Npoint+2:length(PTP))];
        end

        L=length(PTP);
        Npoint=1;
     else
        Npoint=Npoint+1;
     end
end

cyclerange(Ncycle)=abs(PTP(Npoint)-PTP(Npoint+1));
Emin(Ncycle)=min([PTP(Npoint) PTP(Npoint+1)]);
Emax(Ncycle)=max([PTP(Npoint) PTP(Npoint+1)]);
meancycle(Ncycle)=min([PTP(Npoint) PTP(Npoint+1)])+(abs(PTP(Npoint)-
PTP(Npoint+1))/2);
maxcycle=max(cyclerange);
```

APPENDICES C3

```
function [damage]=DamageFunct(cyclerange)
% cyclerange=[50000 60000 70000].*1e-6;

b=-0.092;
c=-0.445;
E=204e9;
Ef=0.26;
Sf=948e6;
dataNf=[1:1:99 100:10:990 1000:100:9900 10000:10000:2e8];
for j=1:length(dataNf)
   dataEa(j)=Sf/E*((dataNf(j))^b)+Ef*((dataNf(j))^c);
end
for i=1:length(cyclerange)
   if cyclerange(i)<dataEa(length(dataEa))
      damage(i)=0;
   else
      for k=1:length(dataNf)
         error=abs(cyclerange(i)-dataEa(k))/cyclerange(i)*100;
         if error<5
            NF=dataNf(k);
            damage(i)=1/NF;
            break
         else
         end
      end
   end
end
```

UNDERGRADUATE PROJECT RESEARCH (GANTT CHART)