Probabilistic damage tolerance analysis using inspection data from integrated sensors

Maria Isabel Vallejo Ciro
Manuel José Carvajal Loaiza

Degree work report as requirement to obtain the Title of Mechanical Engineer

External Advisor
Ph.D. Juan David Ocampo de los Ríos
Saint Mary's University, San Antonio, TX, EEUU

Internal Advisor
Ph.D.(c) Liliana Marcela Bustamante Góez
Universidad de Antioquia

Universidad de Antioquia
Engineering Faculty
Mechanical Engineering
Medellín, Antioquia, Colombia
2021

Cite	Vallejo Ciro & Carvajal Loaiza [1]
Reference IEEE (2020)	[1] M. I. Vallejo Ciro & M. J. Carvajal Loaiza, "Probabilistic damage tolerance analysis using inspection data from integrated sensors", Bachelor's degree project, Mechanical Engineering, Universidad de Antioquia, Medellín, Antioquia, Colombia, 2021.

Mechanical Design Research Group.

Internship University: Saint Mary's University

Centro de Documentación Ingeniería (CENDOI)

Universidad de Antioquia - www.udea.edu.co

Chancellor: John Jairo Arboleda Céspedes.

Dean: Jesús Francisco Vargas Bonilla.

Chair: Pedro León Simanca.

Dedication

For my mom Ana Cecilia, my sister Claudia,
my grandmother Ana de Jesús,
and family.
—M.I.V.C.

For my parents Nelson and Beatriz, my sister Daniela
my grandfather Ernesto,
my cousin Laura,
and family.
—M.J.C.L.

Acknowledgments

The authors are grateful to Dr. Juan Ocampo, Liliana Bustamante, and Dr. Junes Villarraga, who were our guides through all this process and taught us more than just academic. To Mechanical Design Group from Universidad de Antioquia. To St. Mary's University, Fundación Universidad de Antioquia and Programa de Semilleros de Investigación. for the grants that made possible this work. Likewise, to the Federal Aviation Administration for the grant that allowed the development of the methodology in SMART-DT. And to our alma mater Universidad de Antioquia.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Fatigue failures are common failures within the aeronautical field. microcracks appear after many repetitions of cyclic stresses, then, these microcracks grow until a point of no return is reached and the growth becomes unstable and imminent. It is hard to do reliable estimations of a system subject to fatigue due to multiple random factors that affect the material, geometry, and stresses, among others. In this work, to estimate this type of failure, a probabilistic analysis is performed, where each parameter from the model is represented as a probability density function. This work presents a software application to perform fatigue failure analysis using MATLAB and SMART|DT[1]. This last program follows the standards issued by the Federal Aviation Administration of United States (FAA). The application implements a Bayesian inference process to update the model's crack size distribution when inspections are performed, to add more accuracy to risk predictions. This application is expected to support decisions about when to perform inspections based on an allowable desired risk for the fleet.

***Keywords* — Bayesian updating, Damage Tolerance, Probability of Failure, Residual Strength, Fracture Toughness, Probabilistic methods.**

RESUMEN

Las fallas a fatiga son fallas comunes dentro del campo aeronáutico. Luego de muchas repeticiones de esfuerzos cíclicos, microgrietas aparecen, estas crecen hasta alcanzar un punto de no retorno en el cuál terminan de crecer de manera inestable e inminente. Es difícil hacer estimaciones confiables de un sistema sujeto a fatiga debido a múltiples factores aleatorios que afectan al material, la geometría y los esfuerzos, entre otros. En este trabajo, para estimar este tipo de falla, se utiliza un análisis probabilístico, donde cada parámetro del modelo es representado como una función de densidad de probabilidad. Este trabajo presenta una aplicación para realizar análisis de falla por fatiga usando los programas MATLAB y SMART|DT[1], este último sigue las normas emitidas por la administración federal de aviación de Los Estados Unidos (FAA), e implementa un proceso de inferencia bayesiana para actualizar la distribución de tamaño de grietas del modelo cada que se realiza una inspección para proporcionar más precisión a las predicciones de riesgo. Se espera que esta aplicación sea de soporte a la hora de tomar decisiones sobre cuando realizar inspecciones en la flota basadas en el riesgo que se quiera tomar.

***Palabras clave* — Actualización Bayesiana, Tolerancia al Daño, Probabilidad de Falla, Esfuerzo Residual, Resistencia a la Fractura, Métodos probabilísticos.**

I. INTRODUCTION

Fatigue failures are common failures within the aeronautical field. They are present when a material is subject to cyclic and variable loading and the magnitude of these stresses is always less than the material yield point. If there are many stresses repetitions (cycles), microcracks begin to appear. These microcracks grow with the following cycles until a point of no return is reached and the crack grows unstably and imminently giving as result a failure and the separation of the part [2]. For this type of cracks, it is hard to do reliable estimations of a system subject to fatigue due to multiple random factors that affect the material, geometry, and stresses[3], [4]. However, in recent years, the capacity of prognosis for crack behavior has been improved. There are better devices and methods to detect cracks of smaller sizes, such as penetrant liquids, Eddy current, ultrasound, or radiography. All of these has allowed, especially the aeronautical industry, to develop better designs based on the denominated damage tolerance, which applies fracture mechanics principles, and it has led to an increase in airplane safeness to fatigue. This means the airplane is designed to bear cracks of a specific length without presenting a failure.

The damage tolerance analysis approach using fatigue crack growth has been the leading tool for aircraft design and continuing airworthiness evaluation. Damage tolerance is used to evaluate the fatigue life and the residual strength of aircraft components to establish the durability and inspection requirements. To better assess the durability and inspection requirements, it is required to assess variations in loading, material, and geometry. Therefore, a comprehensive probabilistic damage tolerance analysis is essential.

A probabilistic analysis consists in representing each parameter from the model as probability density functions instead of doing so through punctual estimations, just as it is done in a deterministic model[5]. To better estimate the Probability of Failure within the probabilistic damage tolerance analysis framework, it is required to develop distributions for loading, material, and geometry to consider real-world airplane to airplane variations. When inspection data becomes available, either as a finding or no finding, it can be used to update the probabilistic damage tolerance analysis distribution modeling assumptions.

This work presents a software application created in MATLAB as support to perform fatigue failure analysis through a probabilistic methodology which includes Bayesian inference to update the crack size distribution at a given inspection and its subsequent Probability of Failure using inspection information. Two examples are presented to demonstrate the methodology. This application was developed using fracture mechanics together with Monte Carlo probabilistic method, in addition to the program SMART|DT[6], and implements a Bayesian Inference process to update crack size distributions every time an inspection is performed in order to increase the accuracy for risk predictions [7], [8]. AFGROW is a software initially developed by the United States Air Force (USAF) to model crack growth for different geometries and loads. SMART|DT is based on design standards issued by the Federal Aviation Administration of United States (FAA). With this application, it will be possible to include inspections data from integrated sensors in the mathematical model, and this way, the application can work as support for scheduling future inspections based on an updated curve of Probability of Failure that will be more accurate respect to reality. Also, it is expected that this methodology will be added in a module in SMART|DT. This work was developed during a research internship at Saint Mary's University in San Antonio, TX, USA.

II. OBJECTIVES

A. General Objective

Create a software application based on damage tolerance analysis, probabilistic methods, and Bayesian inference through SMART|DT that allows to compute the probability of failure in airplanes, add inspections and update the probability of failure based on data found in each inspection using integrated sensors.

B. Specific Objectives

- Understand fracture mechanics physical phenomena for metallic materials and implementing damage tolerance using stresses spectrums and components geometries to estimate the crack growth curve.

- Create a MATLAB code to carry out, within the crack growth analysis, a field inspection and/or repair and to update the failure probability distribution executing SMART|DT.

- Implement Bayesian Inference to update the initial crack size distribution for each inspection.

III. THEORETICAL FRAMEWORK

A. Fracture Mechanics

Materials, for example, steel or aluminum, often have imperfections due, in many cases, to the manufacturing process. These imperfections may be due to dislocations, porosities, crystal imperfections, welding, among others. The detection of these cracks depends on the inspection method. To estimate the propagation of these cracks Fracture Mechanics principles are used. Hence, it is necessary to know the initial crack size or be assumed, in addition to mechanical properties such as yield strength, fracture toughness, and its geometry.

There are three different modes of loading for crack surface displacement: Mode I is referent to opening, this mode is the most representative for damage and it is the most researched. Mode II corresponds to planar shearing or sliding, this occurs when the crack faces slide in the direction parallel to the principal crack direction, and Mode III is tearing mode, this occurs when the crack faces slide in the direction perpendicular to the principal crack direction. The last mode does not occur very often. The three modes are shown in Fig. 1.

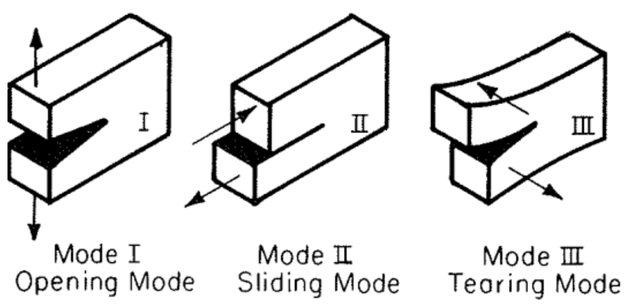


Fig. 1. Loading modes[2]

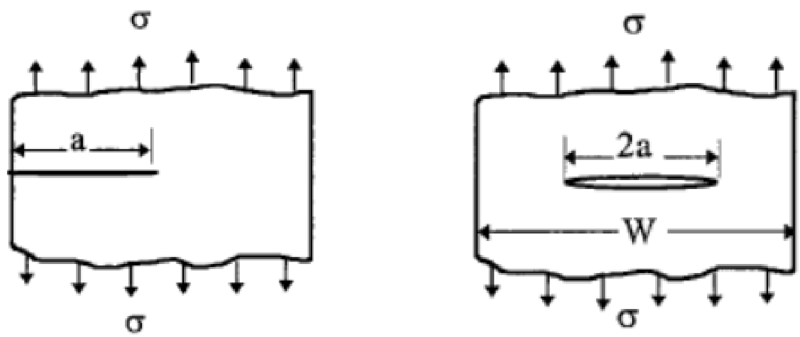
1. Stress Intensity Factor

The stress intensity factor, K, is the energy or intensity of the stresses around the crack tip. When this stress intensity factor reaches and exceeds a threshold (K_{TH}), the cracks start to grow. K depends on the geometrical configuration, type of crack and the stresses involved.[2]

For loading mode I, which is the loading mode used for the analysis within this work, the equation for K is the following

$$K = \beta \cdot \sigma \cdot \sqrt{\pi a} \qquad [PSI\sqrt{in}\,]\ [MPa\sqrt{m}] \tag{1}$$

Where σ is the remote stress applied to the component, a is crack length, and β is a correction factor that depends on the specimen and cracks geometry, for example, for the configuration shown in Fig. 2.a, $\beta = 1.12$, and for the configuration shown in Fig. 2.b, $\beta = \sqrt{\sec\left(\frac{\pi a}{W}\right)}$.



a) b)

Fig. 2. a) Edge crack in a semi-infinite body.
b) Centre crack in a strip of finite width.[9]

2. Fracture Toughness

Fracture toughness of the material is the value of the critical stress intensity factor, K_C. When *Kc* is reached, the crack grows rapidly and unstably. A similitude between Fracture toughness and yield stress can be made saying that fracture toughness is the limiting value for stress intensity factor and yield stress is the limiting value for applied stresses. *Kc* depends on the specimen thickness, it varies until plane strain conditions are reached and it becomes constant, this fracture toughness is represented by K_{IC}.[2] Fig. 3 shows a schematic curve displaying the relationship between thickness and the critical fracture toughness.

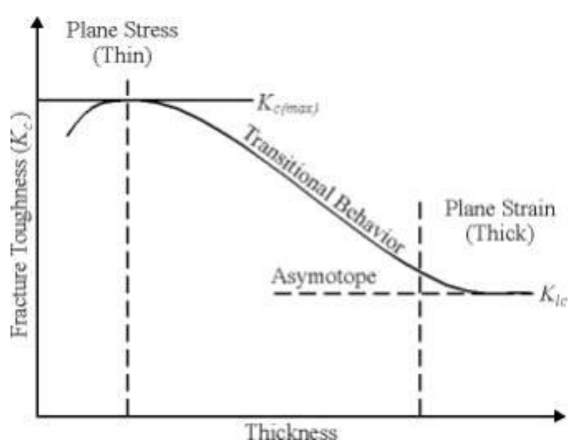


Fig. 3. Fracture Toughness AFGROW[10]

3. Crack Growth Rate Curve

da/dN represents the crack growth rate, where crack length, a, is differentiated in terms of cycles, N, and it can be plotted vs ΔK using the following the equation.

$$\Delta K = K_{max} - K_{min} = \beta \Delta \sigma \sqrt{\pi a} \tag{2}$$

Where $\Delta\sigma$ is the remote stress applied to the component, Fig. 4. shows $\Delta\sigma$ schematically.

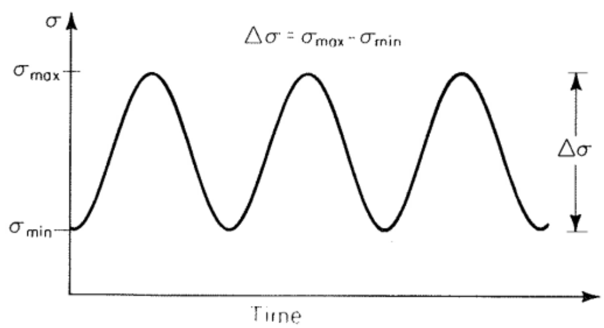


Fig. 4. Δσ through time[2].

Fig. 5. shows a schematic plot for *da/dN* versus Δ*K* in logarithmic scale. This curve exposes three different regions, the first one represents crack initiation, and it is associated with threshold K_{TH}, the crack will only grow if ΔK exceeds K_{TH}. This section is not commonly used for designing, only parts for specific applications are designed within this region, for example power trains that operate at very high speeds. The second region is the crack propagation, essentially linear, most applications are designed within this region, and there are several research studies about this region and some equations have been created to represent this phenomenon, for example, the Paris Equation or the NASGRO equation. The third region represents a fracture with high ΔK, this section is reached when K_C is exceeded, it occurs in a very short time, the crack grows rapidly and unstably.

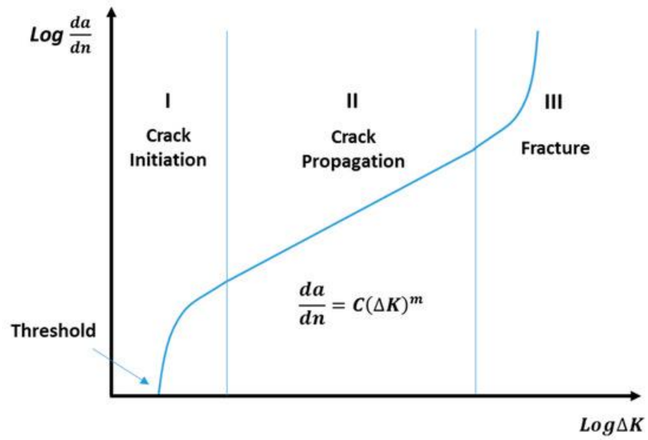


Fig. 5. Three regions of crack growth rate curve [11].

This study is focused on region II and the Paris equation, which is the most accepted for describing the curve in this region, equation 3 is the Paris equation.

$$\frac{da}{dN} = C(\Delta K)^m \quad (3)$$

Where *C* and *m* are material constants and *ΔK* is the stress intensity range. C and m can be found in the literature.

B. Probabilistic Damage Tolerance Analysis

Damage tolerance refers to the ability of structures to sustain cracks for a time before repairing it. Its analysis is based on the physics of fracture mechanics, and it is associated with the second region in Fig. 5. in crack propagation. This concept was introduced by the FAA and the USAF and they use it as "safety by inspection". This approach assumes that components always have cracks, and they propagate with usage. It relies on inspections to repair the cracks and extend the service time of the component. A probabilistic analysis refers to the use of probabilistic distributions or random variables for the different properties used within the methodology of damage tolerance analysis and fracture mechanics.[1]

1. Failure Criteria

a) Probability of Failure (POF):

The probability of failure is defined as the probability that the maximum stress per flight exceeds the Residual Strength of the part. [7]

$$POF = P(\sigma_{maxflight} > RS) \quad (4)$$

b) Residual Strength (RS):

Residual Strength is the structural strength remaining in the presence of a crack, also the residual strength determines the critical crack size.[2]

- **RS by Fracture Toughness**

To know the quantity of the residual strength at any time on the structure based on the material property fracture toughness (K_c), the following equation is used:

$$RS = \frac{K_c}{\beta\sqrt{\pi a(t)}} \quad (5)$$

This type of failure exists when ΔK reaches K_C and imminent grow for the crack occurs.

- **RS by Net Section Yielding**

The amount of residual strength by net section yielding can be calculated using the following equation:

$$S_{NSY} = S_y\left(1 - \frac{(D+a_i+r_{yz})}{W}\right) \quad (6)$$

Where S_y is the yield stress, D is the hole diameter, a_i is the crack length, r_{yz} is the radius of the plastic zone near the crack tip and W is the width of the part.

This failure occurs when the part has been subjected to plastic deformation from the crack tip to the opposite face of the part.

C. Monte Carlo Sampling

Monte Carlo methods are different computational techniques for the solution of mathematical problems, which focuses on random samples. These techniques are used to generate random samples based on a probability density function.[12]

As example, this methodology can be used to estimate the value of PI. It is assumed a circle of radius 0.5 circumscribed by a square of width 1, as shown in Fig. 6. The area of the square is 1 and the area of the circle is $\frac{\pi}{4}$, then, the area of the circle over the area of the square is $\frac{\pi}{4}$.

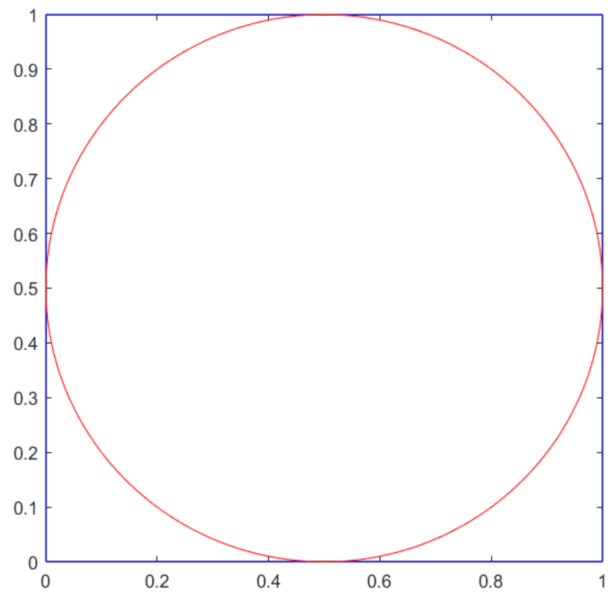


Fig. 6. Circle circumscribed by a square.

Assuming that uniformly distributed random points are generated inside a square from (0,0) and (1,1), the points inside the circle are colored red and outside blue as shown in Fig. 7. Then, the number of points inside the circle is divided by the total number of points, the result will be an approximate value to the division of the areas as shown before, this is, $\frac{\pi}{4}$. If this approximate result is the multiplied by 4, then, the result will be an estimation of π.

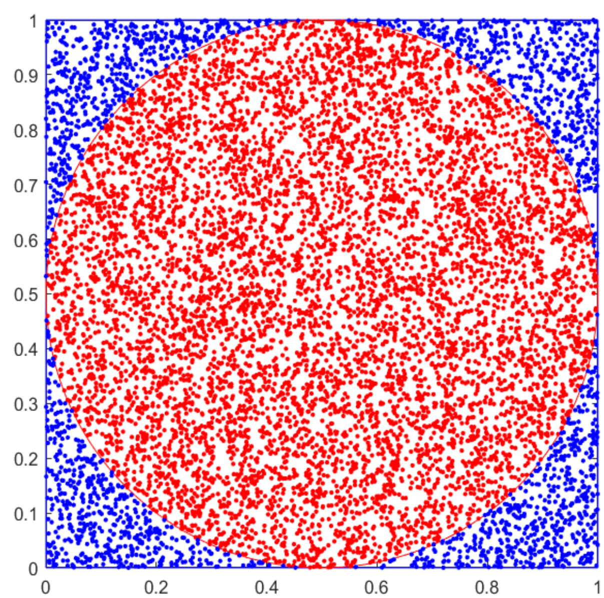


Fig. 7. Monte Carlo sampling

D. Bayesian Updating

Bayes' theorem relates the conditional probability between two events. It is used to calculate the probability of an outcome based on prior knowledge or its association with another event [13]. Bayes' theorem formula is:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{8}$$

Where *P(A|B)* is the probability of event *A* occurring, given event *B* has occurred, *P(B|A)* is the probability of event *B* occurring, given event *A* has occurred, *P(A)* is the probability of event *A* and *P(B)* is the probability of event *B*.

These two events *A* and *B* are independent, this is, the result of event *A* does not affect the probability of event *B*.

Bayesian Inference, involves Bayes' Theorem, is used to update a distribution of an event based on new information. It uses probability distributions instead of deterministic estimations. Bayesian inference is commonly used in machine learning, but it is also applied in fields such as medical and pharmaceutical.

In this case, the Bayesian formula is used to update the probability distributions of the parameters of crack size detected [14].

$$P^{+}(\theta|\boldsymbol{D}) = \frac{L(\boldsymbol{D}|\theta) \cdot P^{-}(\theta)}{NF} \tag{9}$$

Where:

- θ represents the parameters mean(μ) independent variable, and standard deviation(σ) assumed, it will be fixed,
- $\boldsymbol{D}$ represents the vector of the measurements (or inspections),
- $P^{-}(\theta)$ represents the prior distribution of crack size at the time.
- $L(\boldsymbol{D}|\theta)$ represents the likelihood function of the parameters.
- NF Normalization Factor used to get a probability density function.
- $P^{+}(\theta|\boldsymbol{D})$ represents the posterior distribution given the detected crack sizes.

1. Prior Distribution

The prior distribution is known, based on the damage tolerance model at time *t*. the prior distribution is equal to the crack size distribution predicted at time *t*. Assumed that it follows a Lognormal distribution with mean and standard deviation known [8]. Fig. 8 shows the prior distribution methodology used within this work.

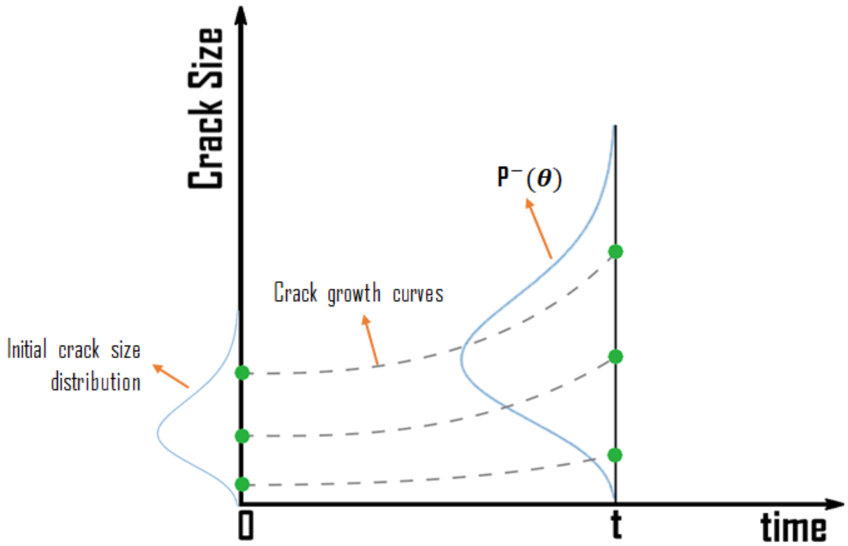


Fig. 8. Prior Distribution.

2. Likelihood Distribution

The likelihood function reflects the degree of agreement between the obtained measurements, *D*, and the output obtained from the mathematical model (Log-normal distribution) used to physically describe the system [8].

It will be dependent on each inspection, and whether a crack is found or not and will have the following equation:

$$L(\boldsymbol{D}|\theta) = L_D(\theta) \cdot L_{ND}(\theta) \tag{10}$$

Where $L_D(\theta)$ is the likelihood function when there is a crack detected and $L_{ND}(\theta)$ when there were no cracks found. Fig. 9 shows Likelihood distribution schematically.

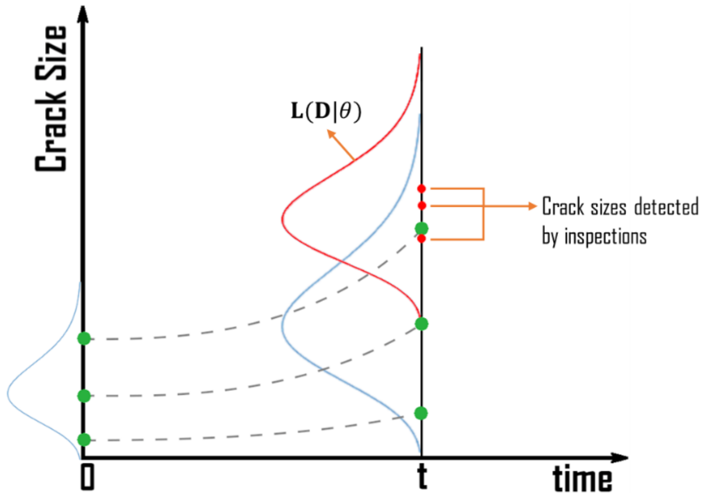


Fig. 9. Likelihood distribution.

- **Likelihood of Detection $L_D(\boldsymbol{\theta})$:**

The likelihood function for detections will have the following equation and will be dependent on every crack detected:

$$\mathrm{L}_D(\theta) = \prod_{i=1}^{N_D} POD_i(D_i(t_i)) \cdot f(D_i(t_i)|\theta) \qquad (11)$$

- <u>Probability of Detection (*POD*)</u>, which depends on the detection method, e.g., Eddy current testing, corresponds to the probability of detection curve. Fig. 10 shows an example of this curve for an inspection method following a log-normal distribution with a mean of 0.06 and a standard deviation of 0.07.

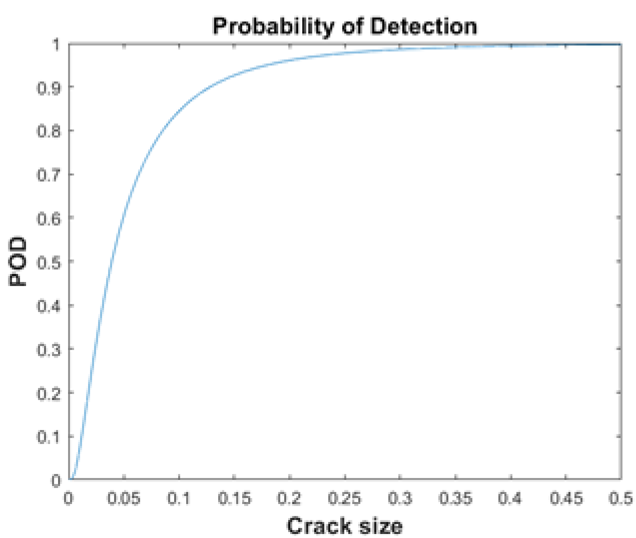


Fig. 10. Probability of Detection example.

- Function $f(D_i(t_i)|\theta)$ For this work, it represents the distribution of means for the crack found D_i, and it is defined as:

$$\text{LogNormal}(D_i|\varphi, h) = \frac{1}{D_i \cdot h \cdot \sqrt{2\pi}} \cdot exp\left\{\frac{-(\log(D_i)-\varphi)^2}{2 \cdot h^2}\right\}, \text{for } D > 0 \tag{12}$$

$$\varphi = Log\left(\frac{\mu^2}{\sqrt{\sigma^2+\mu^2}}\right) \tag{13}$$

$$h = \sqrt{Log\left(\frac{\sigma^2}{\mu^2}+1\right)} \tag{14}$$

Where μ is an independent variable representing the possible mean of cracks and σ follows this statement: "There are two possible ways to decide on the value of σ_k. The first would be through estimation via the mean squared error of $(D_k - M(\boldsymbol{\theta}))$. **The second would be to set it as a fixed parameter based on prior calculations or knowledge.**"[15]: In this case, it is used the same as the standard deviation for the prior distribution.

For the probability density function shown in Eq. 12, the crack size is known, which is D_i and the mean is a random variable. Fig. 11 shows an example of function *f(D).*

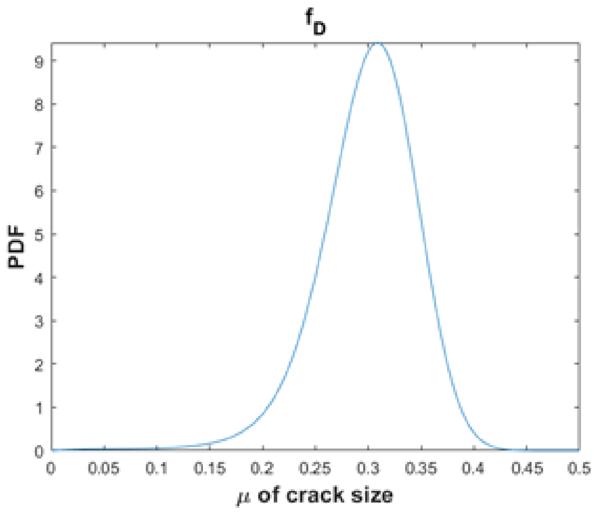


Fig. 11. Function f(D)

- **Likelihood of NO Detection $L_{ND}(\boldsymbol{\theta})$:**

When no cracks are detected, the Likelihood function is defined as:

$$\text{L}_{ND}(\theta) = \prod_{i=1}^{N_{ND}} \int_0^\infty PND_i(D(t_i)) \cdot f(D(t_i)|\theta) dD \tag{15}$$

- Probability of No Detection (PND) corresponds to the probability that the inspection method will not detect a crack and it is defined as:

$$PND = 1 - POD \quad (16)$$

Fig. 12 shows an example of the probability of no detection for an inspection method following a log-normal distribution with a mean of 0.06 and a standard deviation of 0.07.

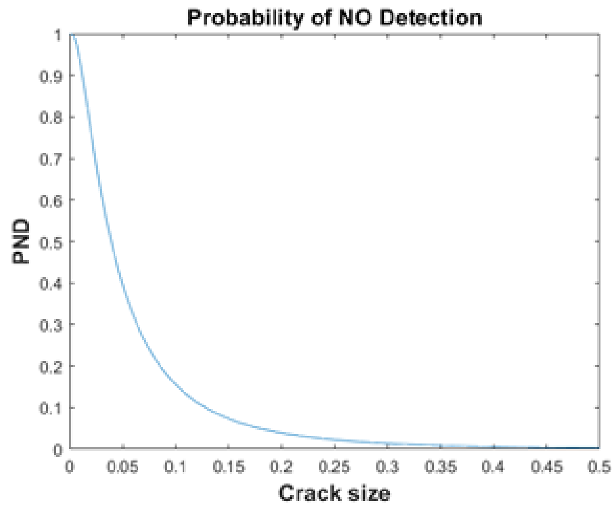


Fig. 12. Probability of No Detection.

- Function $f(D(t_i)|\theta)$ represents is a probability density function (PDF), it follows the same methodology as for detected cracks, but now it has *D* as a random variable too. Fig. 13 shows graphically a representation for distributions of means and crack sizes.
  The integral considers all the values *D* can take.

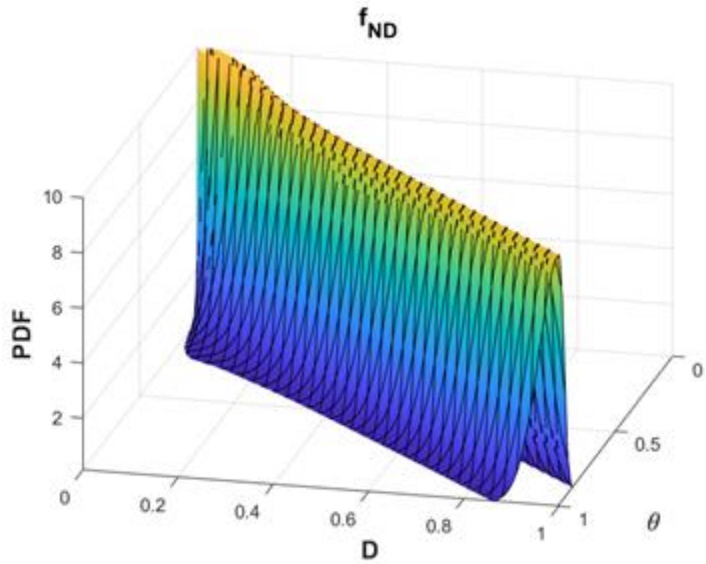


Fig. 13. Function f(ND)

Fig. 14 shows an example for Likelihood function curve for no detection.

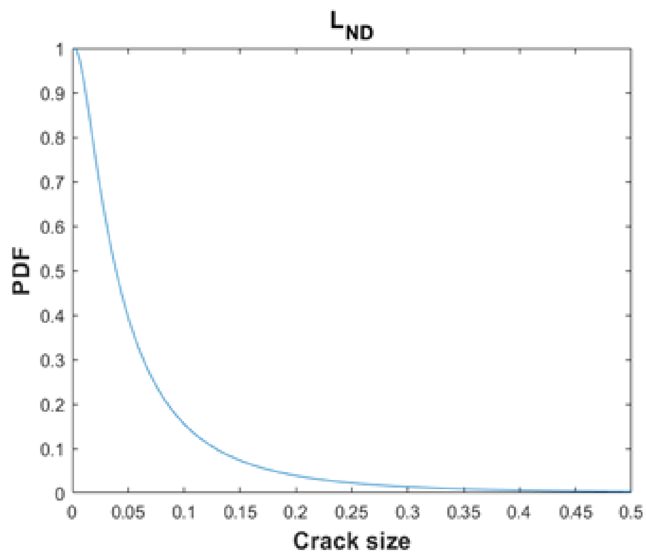


Fig. 14. Likelihood of No Detection.

3. Normalization Factor

It's a normalization factor, so when integrating the posterior distribution, the cumulative density function is equal to 1. Fig. 15 shows a scheme for normalization factor.

$$NF=\int_0^\infty \mathrm{L}(\mathbf{D}|\theta)\cdot \mathrm{P}^-(\theta)\cdot d\theta \quad (17)$$

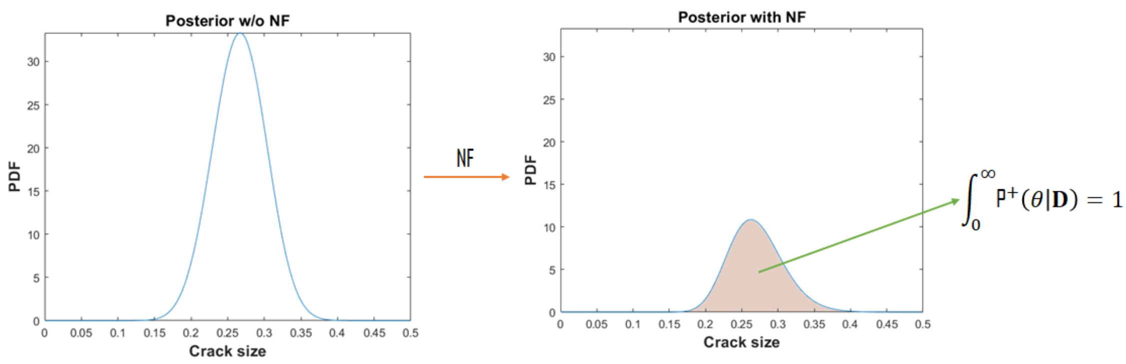


Fig. 15. Normalization Factor.

4. Posterior Distribution

After computing the posterior distribution, it becomes a laborious function, so it is fit to a log-normal distribution and the new parameters are obtained. Fig. 16 shows a prior, likelihood and posterior distribution.

$$P^+(\theta|\boldsymbol{D}) = \frac{L(\boldsymbol{D}|\theta)\cdot P^-(\theta)}{NF} \tag{18}$$

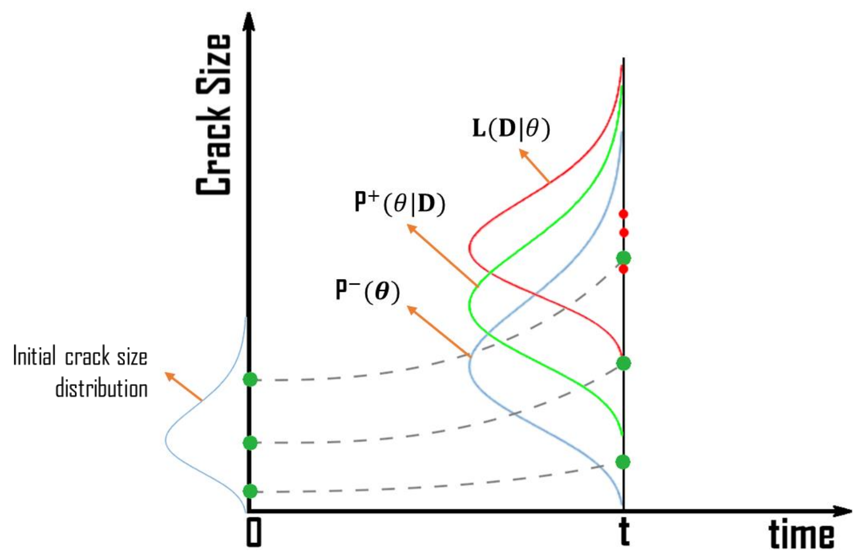


Fig. 16. Prior, Likelihood and Posterior Distributions.

IV. METHODOLOGY

This study was performed with SMART|DT, which does a probabilistic crack growth depending on different material, airplane, and flight properties, and there was a necessity to update the model after each inspection based on what was found within each inspection. To update the model, Bayesian updating was used, and a graphical user interface was created in MATLAB in order to have the possibility to do many updates in a short time. A flowchart showing the methodology followed for the script is exposed in Fig. 17.

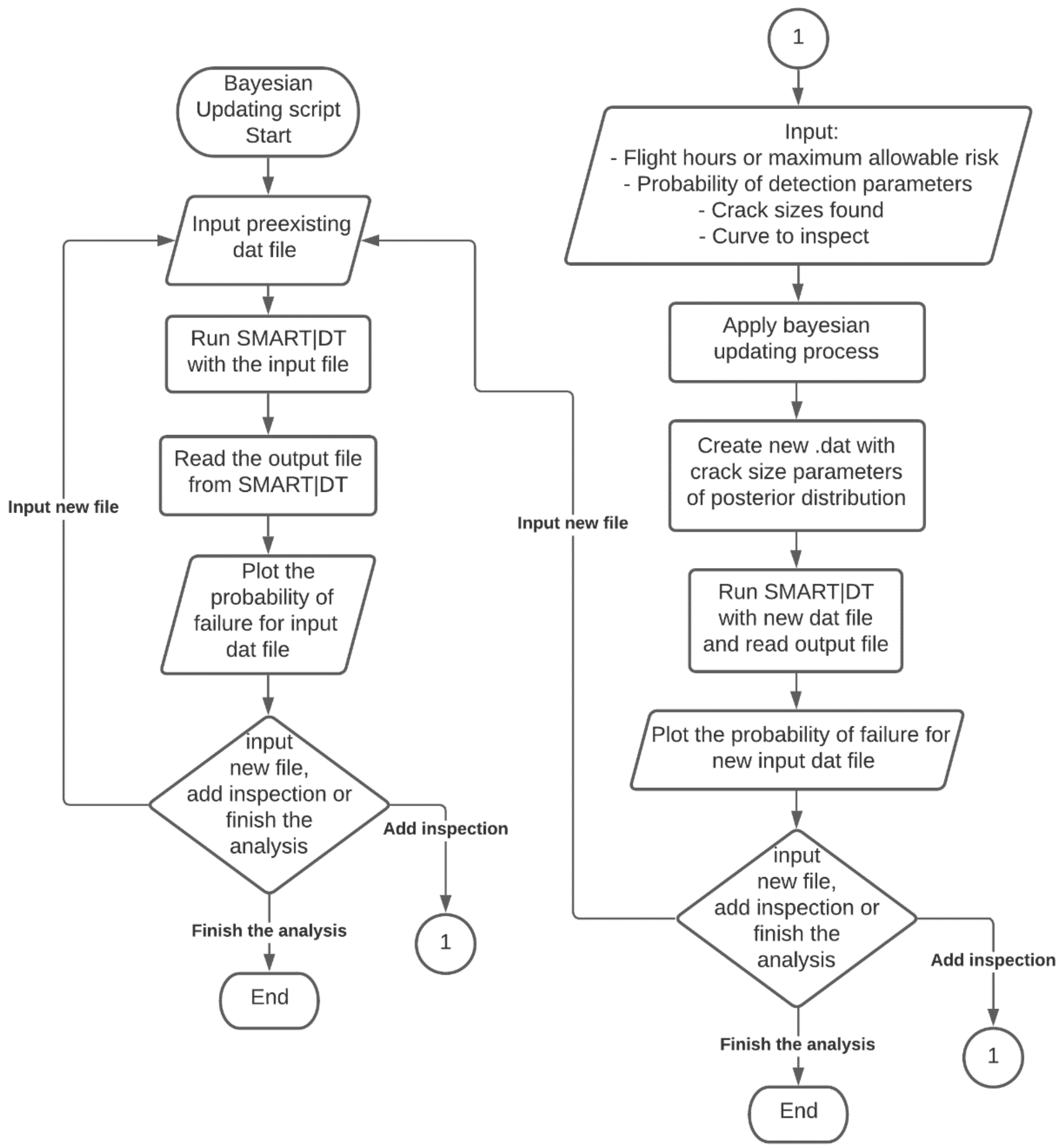


Fig. 17. Flowchart for Bayesian Updating Script.

To do the Bayesian updating, the first step was to have an initial crack size distribution, this distribution represents existing knowledge at the beginning of the analysis based on the manufacturing process. The initial crack size distribution was grew using the damage tolerance models in SMART|DT. This software gives an output file which contains the cumulative crack size distribution, then a fitting was performed for this cumulative crack size following a log-normal distribution and its parameters (mean and standard deviation) were obtained and used in MATLAB using a symbolic variable within an equation.

Subsequently, it is known that the likelihood distribution is a representation of the degree of agreement between the mathematical model and the obtained measurements, to accomplish this, a new distribution was created. This distribution is assumed to follow a log-normal distribution equation, but it has some changes, first, the independent variable is now the crack size mean, and second, instead of "x" a measurement D_i is set, this equation is shown in Eq.12, lastly, based on the prior model, it is assumed that this equation will have the same standard deviation as the prior distribution. Furthermore, there is also another important equation here, this other equation represents how reliable is the measurement obtained, and it is the probability of detection distribution. An example of the probability of detection for Eddy currents is shown in Fig. 10. Assuming that an inspection in an airplane is performed and that inspection found a crack, the likelihood distribution can be expressed as the multiplication of two equations as shown in Eq.11. When cracks are not found during an inspection, a modification to Eq.12 needs to be done due to the inexistence of measurement D_i. To include every possible size that the crack could have, an integration is done from 0 to the infinite in terms of D, and the resulting equation is then multiplied by the probability of detection equation of the method used during the inspection; this process is shown in Eq.15. When there are multiple inspections, a likelihood distribution is created independently for each inspection, then, all these likelihood distributions are multiplied together in order to obtain just one likelihood distribution. Using MATLAB, these equations are created using symbolic equations and the likelihood function is a variable containing the multiplication of them.

After that, the updated distribution needs to follow the rules of a probability density function, and this is, the area under the curve must be 1, to achieve this, the prior and likelihood distributions are multiplied, then, they are integrated in terms of "x" from 0 to the infinite, the result of that integration is called the normalization factor. In MATLAB, this was done by using a loop with increments of 0.001 from 0 to 2 and replacing that value in the equation resulting from the multiplication between likelihood and prior distributions and performing the sum of each resulting value.

Finally, the posterior distribution is going to be the result of the multiplication of the prior and likelihood distributions over the normalization factor, Eq.17. This is then substituted by values in order to have pairs of numbers to apply a fitting to the resulting equation. A log-normal distribution is used to perform the fitting and the results are the mean and standard deviation of the posterior distribution.

These two new parameters now represent the initial crack size distribution and are used to restart the damage tolerance analysis from that time and update the probability of failure for the analyzed part. A flowchart showing the followed process is shown in Fig. 18.

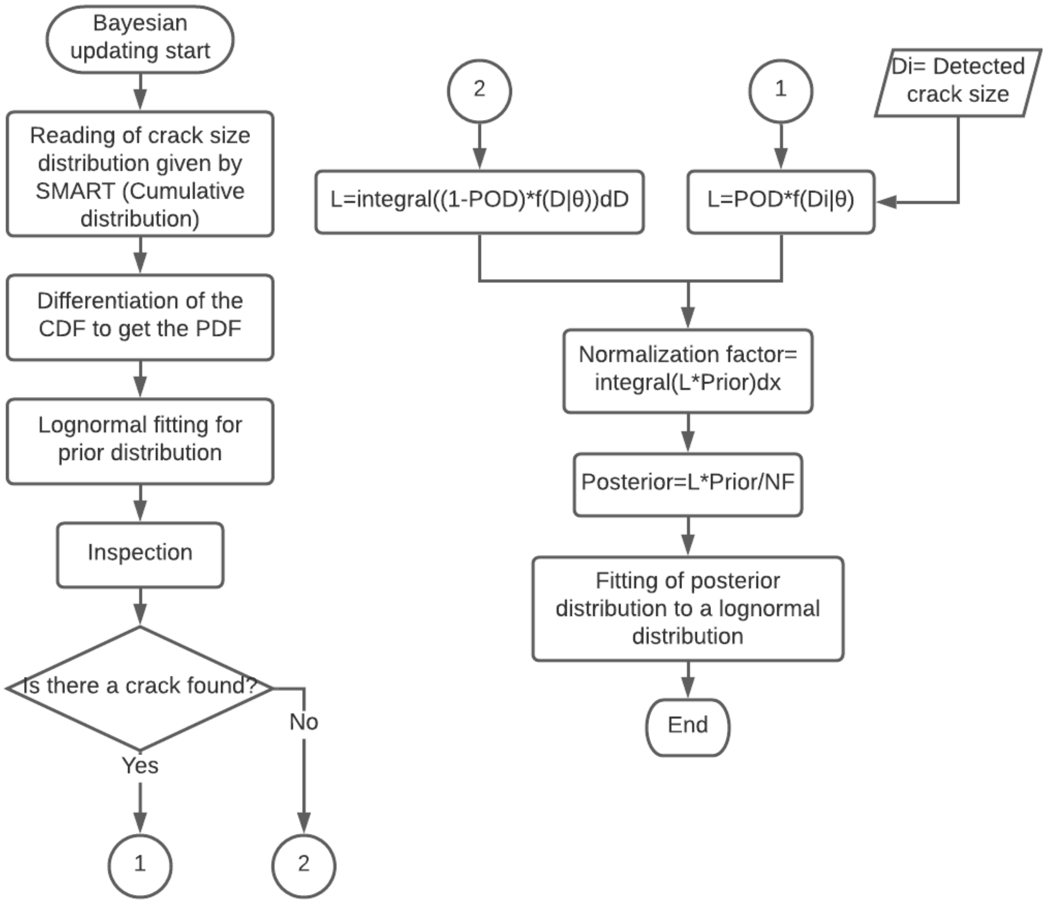


Fig. 18. Flowchart Bayesian Updating Script.

When all the coding for Bayesian updating was ready, the next step was to do an implementation of SMART|DT within MATLAB to run simulations from the first input file and to simulate every update the user would like to do. Fig. 19 shows the resulting graphic user interface created.

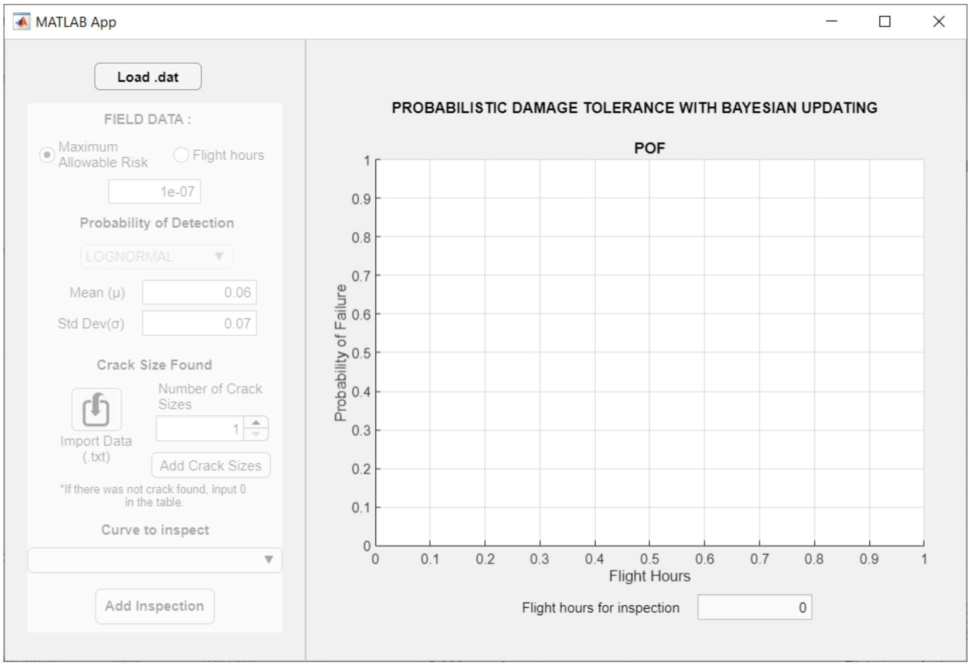


Fig. 19. Program graphical user interface

This graphical user interface (GUI) includes a button at the top, this button is used to import a preexisting .dat file used to do a simulation on SMART|DT. Those files contain aircraft information, simulation method, material properties, inspection information, loading parameters, and a description of that .dat file. When that button is clicked, a new window appears, shown in Fig. 20. It is used for searching and importing a preexisting .dat file, an example of this type of files is shown in Fig. 21. Then, SMART|DT runs the simulation with the selected file. After the simulation is done, the GUI adds to the plot the resulting curve for the probability of failure from the output file of the simulation, an example of this is shown in Fig. 22.

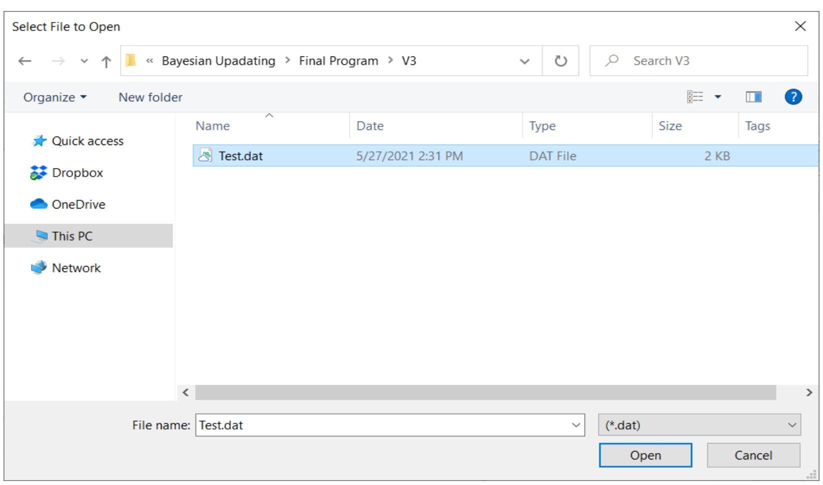


Fig. 20. Search window.

```
! --------------------------------------------------
!            AIRCRAFT INFORMATION
! --------------------------------------------------
TITLE = Wing_Spar
AC_MAKE = Acme
AC_MODEL = Sky Runner
AC_SERIAL_NUM = SR100
AC_TCDS = TCSR100
! --------------------------------------------------
!            METHOD
! --------------------------------------------------
INTEGRATION_METHOD = MC 1000000 2394
POF_MAX_INC = 40000 400
ANALYSIS_TIME_UNITS = flights
! --------------------------------------------------
!            FRACTURE MECHANICS
! --------------------------------------------------
CRACK_GROWTH_CODE = MASTERC_USER MasterCurve_e1000.avsn
INITIAL_CRACK_SIZE = LOGNORMAL 0.005 0.003
FRACTURE_TOUGHNESS = NORMAL 34.9 3.4
YIELD_STRENGTH = DETERMINISTIC 120.0
! --------------------------------------------------
!            INSPECTIONS
! --------------------------------------------------
INSPECTIONS = 0
! --------------------------------------------------
!            LOADING AND EVD PARAMETERS
! --------------------------------------------------
EVD_TYPE = USER 1.465D1 8.0D-1 0.0D0
NUMBER_OF_USAGES = 0
! --------------------------------------------------
!            DESCRIPTION
! --------------------------------------------------
!  RUAG training June 29-30 - 2020
```

Fig. 21 . dat file example.

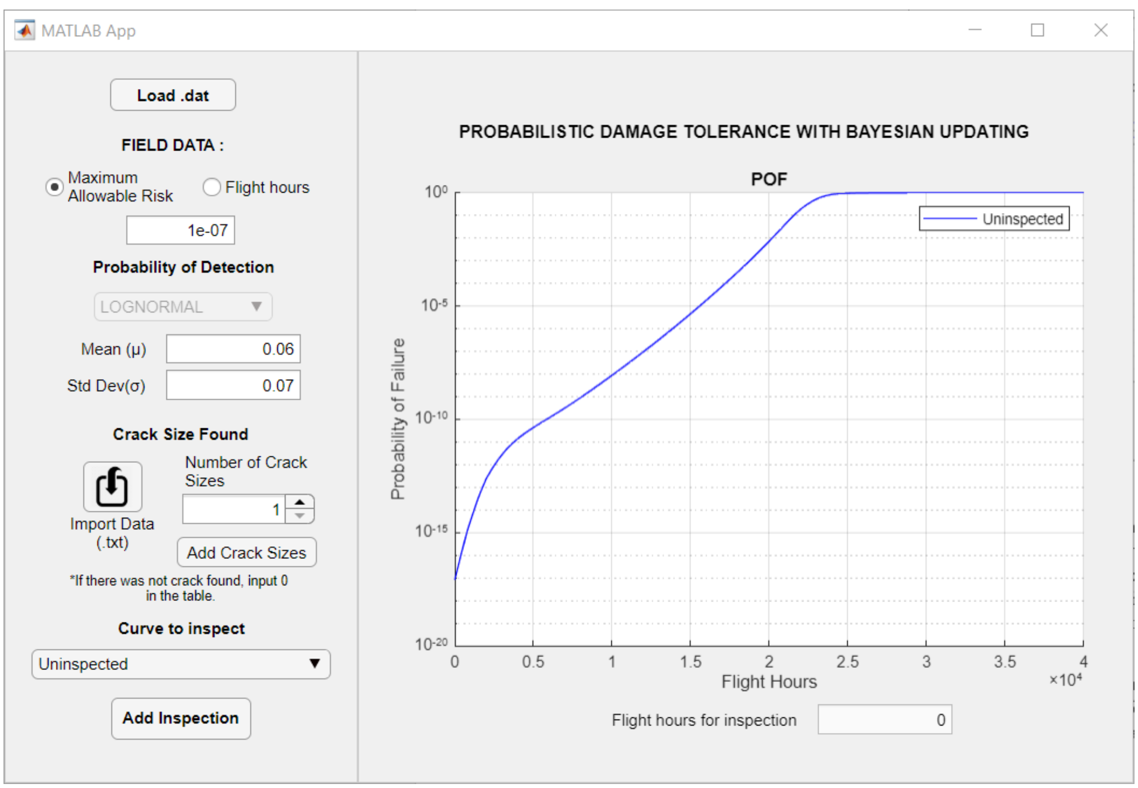


Fig. 22. Curve for the probability of failure.

After there is an existing curve in the plot, all the left panel is activated. This section is used to input the field data. First, there are two options, to select maximum allowable risk or flight hours, and a text box to input the respective risk or flight hours. When “Maximum allowable risk” is selected, the text box is set to have a scientific number format and limits from 0 to 1. When it is selected flight hours, the text box is set to have integer number format and limits from 0 to the infinite.

The next section corresponds to the inspection method and its inherent probability of detection curve. The GUI is programmed to use a log-normal distribution for the probability of detection, and it has two textboxes to input the mean and standard deviation for that log-normal distribution.

Additionally, there are two options to add the crack sizes found. First, it can be done by importing a txt file. That txt file must have each crack size in a different line and when the inspection didn’t detect a crack, that inspection must be represented by a 0. The second option is to manually add each crack size found. For this option, the spinner must be set to the number of crack sizes to add, and after the “Add Crack Sizes” button must be clicked. This will create a new window that has a number of cells equal to the number set in the spinner. These cells must be filled with integers equal or bigger than 0, where 0 also represents that there were no cracks found, then, “Load” button must be clicked to add the input values. An example of this window is shown in Fig. 23.

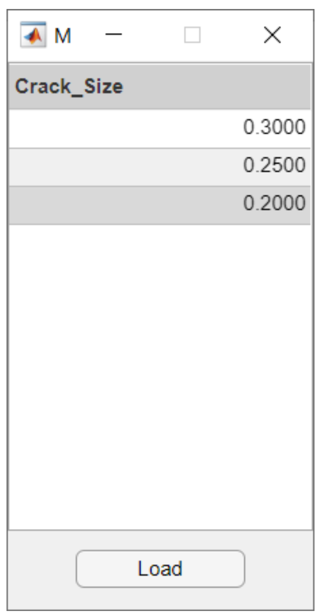


Fig. 23. Crack size found.

Before adding the inspection, there is one last step within the GUI before adding the inspection, this is, to select a curve in which the Bayesian updating inspection will be performed from the dropdown list. Every time there is a new inspection, that curve is added to the dropdown list.

Finally, "Add inspection" button can be clicked. When this button is clicked, it first identifies which curve is selected to perform Bayesian updating and saves its indicative in a variable and imports the values for POF output file from SMART|DT. it searches for the flight hours or risk that was input before and creates a new .dat file to write crack size distribution at the corresponding time. It runs SMART|DT with the .dat file. Then, it imports the crack size distribution from that last file and follows the Bayesian updating process mentioned before using this distribution as the prior distribution. With the resulting parameters from the posterior distribution, it creates two new .dat files using these parameters as the initial crack distribution. The first file only updates the distribution, and the second file, besides updating the distribution, also simulates a repair in the part. These two files are then run in SMART|DT, and the two resulting POF curves are plotted within the GUI, an example of this is shown in Fig. 24. It also adds the two new curves to the dropdown list and sets it to the updated curve without repair. two new windows are also created. The first new window displays 3 curves, they are the prior, likelihood and posterior distributions, and the second window displays only the likelihood distribution, an example of these two windows are shown in Fig. 25 and Fig. 26.

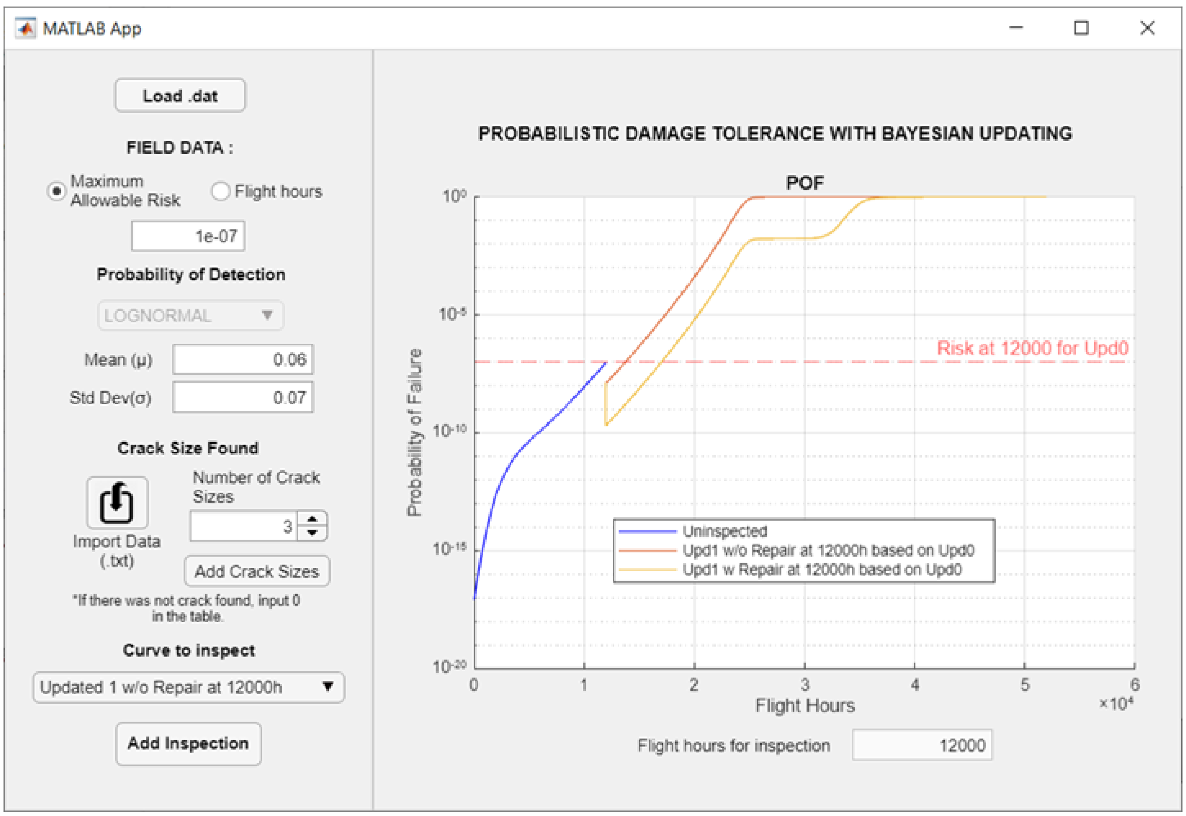


Fig. 24. GUI after added inspection, example.

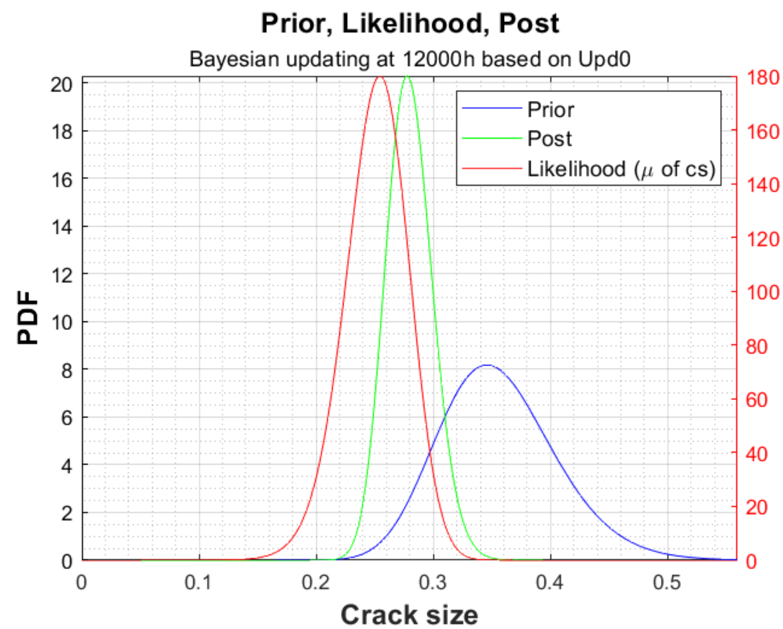


Fig. 25. Prior, Likelihood and posterior distributions, example.

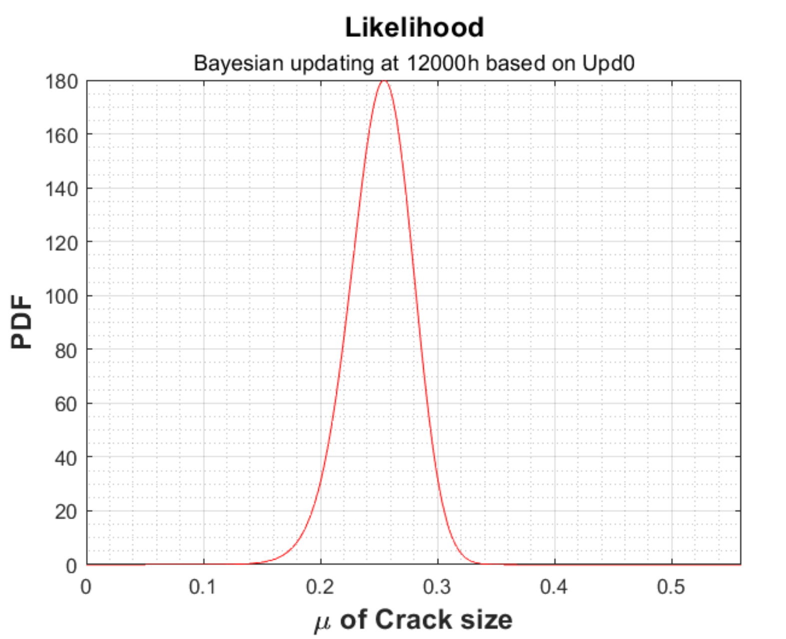


Fig. 26. Likelihood function.

V. EXAMPLE PROBLEMS

A. Example with no detection

This example begins with the probability of failure curve shown in Fig. 22, then, it is assumed to do an inspection at the time when the risk is less than 10^{-7}. The inspection method is assumed to have a probability of detection that follows a log-normal distribution with mean 0.06 in and standard deviation 0.07. Finally, it is assumed that within one inspection there were not cracks found. Fig. 27 shows these aspects mentioned before already selected and included within the GUI.

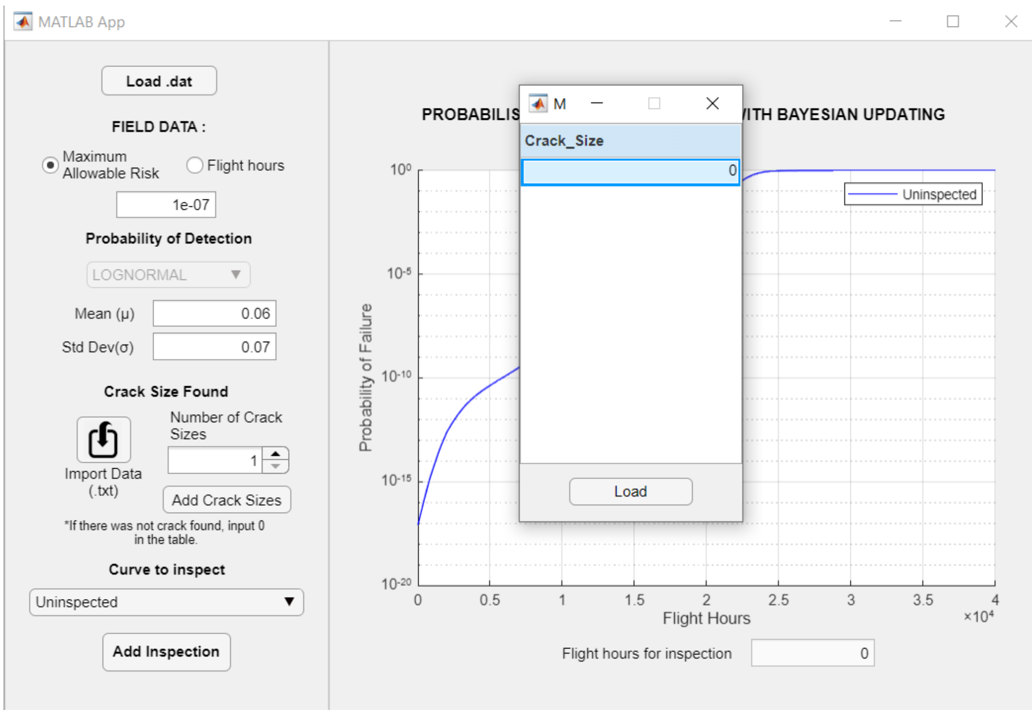


Fig. 27. No detection example setup.

The results for this setup are presented in Fig. 28, including the new probability of failure and the distributions used to perform Bayesian updating process. These last distributions are shown in Fig. 29 and Fig. 30.

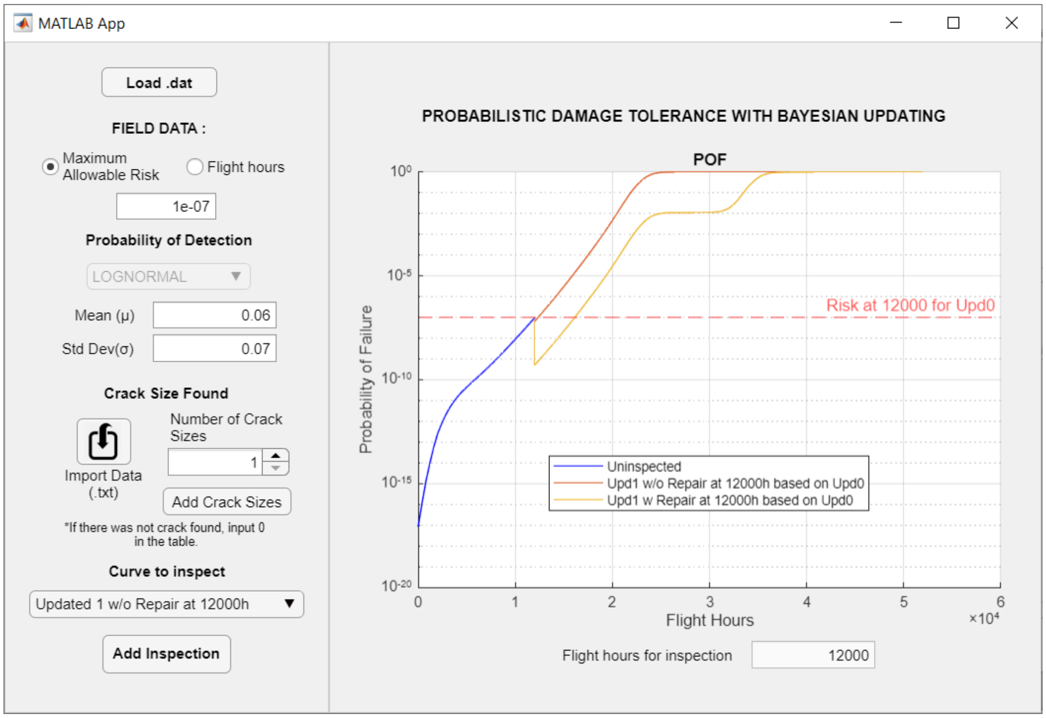


Fig. 28. Updated probability of failure for no detection example.

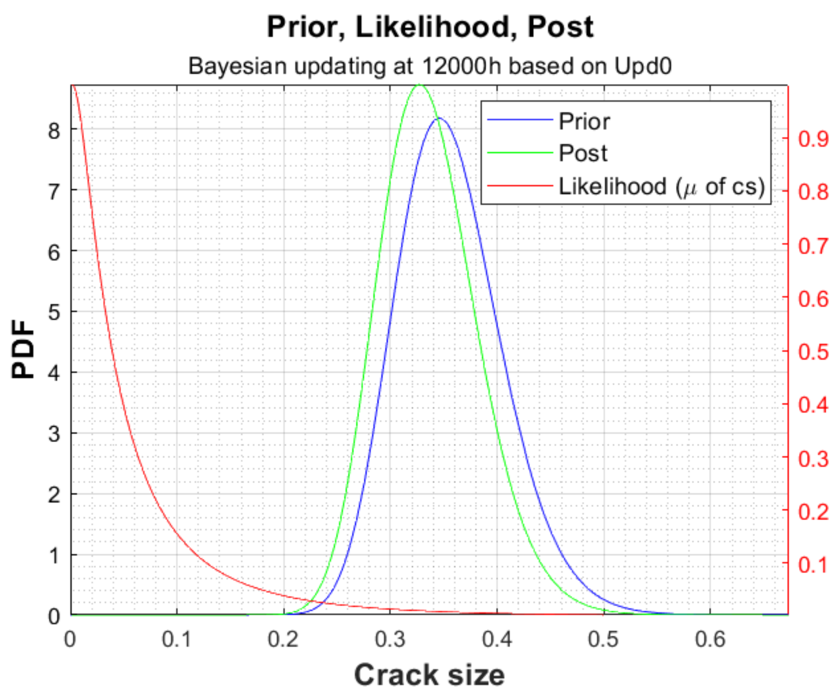


Fig. 29. Prior, likelihood and posterior distributions for no detection example.

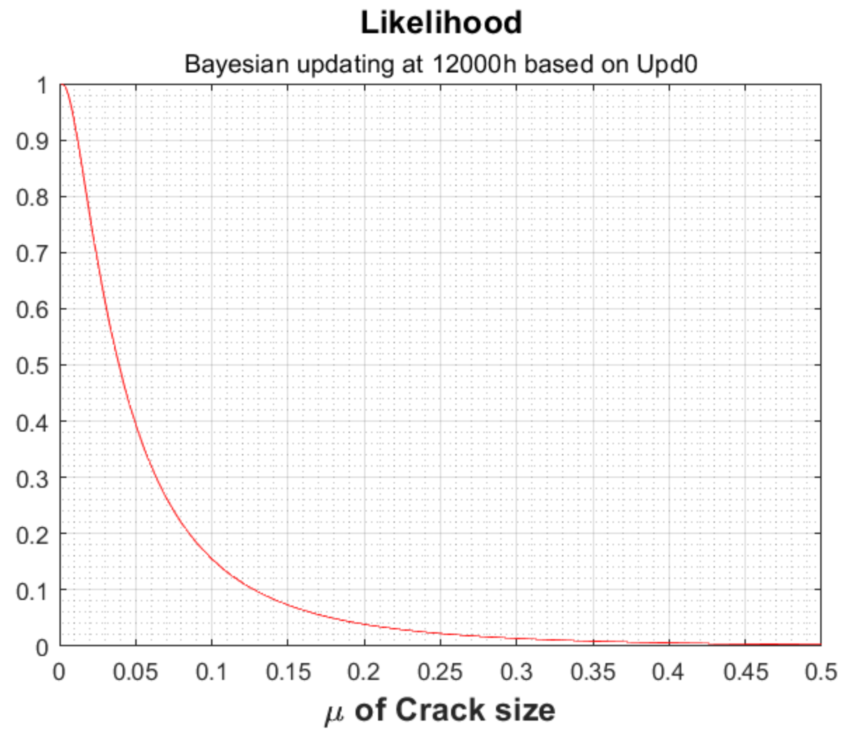


Fig. 30. Likelihood distribution for no detection example.

From Fig. 28, it can be seen that the inspection should be done at 12,000 flight hours to prevent reaching the risk of 10^{-7}. From Fig. 29 it can be seen that due to no cracks found, the mean for crack size distribution became less than the predicted model and as consequence, it can be seen in Fig. 28 that the updated probability of failure is less risky than the initial one.

B. Example with detection

As continuation for the example before, it is assumed that there was no reparation because there were no cracks found and a new inspection was planned at 17000 flight hours. Within this inspection it is assumed that a crack of 0.1in of length was found using the same inspection method than before, so it will follow the same probability of detection distribution. Fig. 31 shows the setup for this example having what was found in the past example already in the GUI.

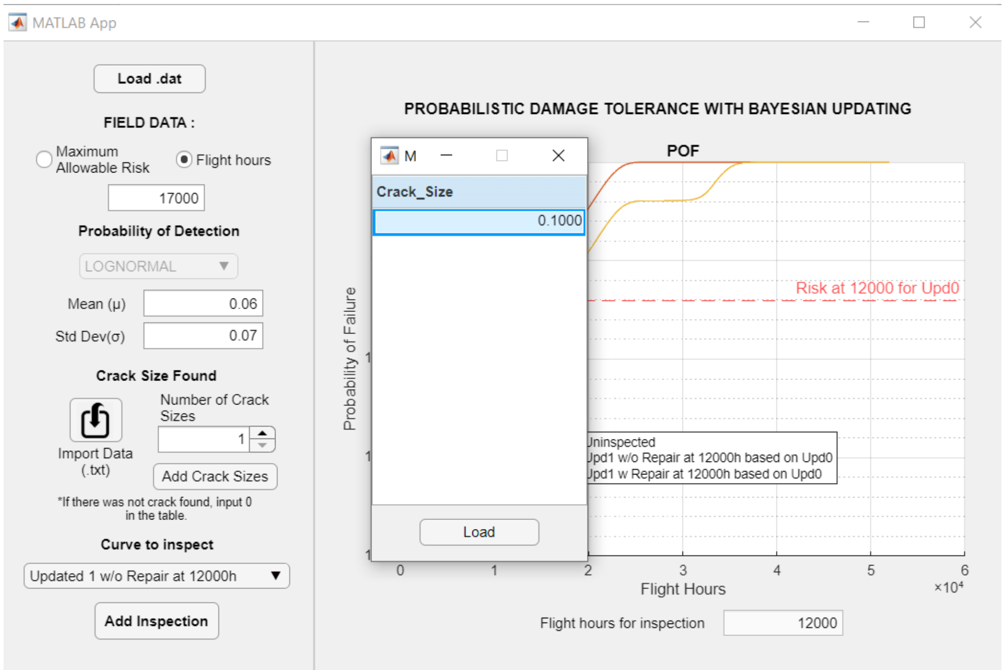


Fig. 31. Example for one detection

The results for this setup are presented in Fig. 32, including the new probability of failure and the distributions used to perform Bayesian updating process. These last distributions are shown in Fig. 33 and Fig. 34.

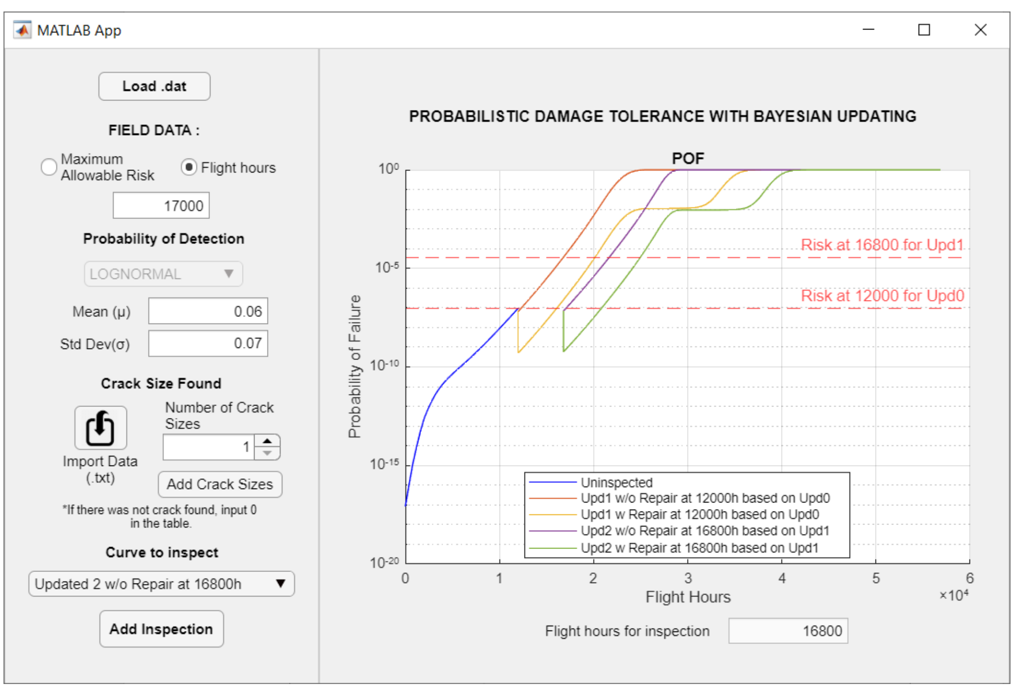


Fig. 32. Updated probability of failure for detection example

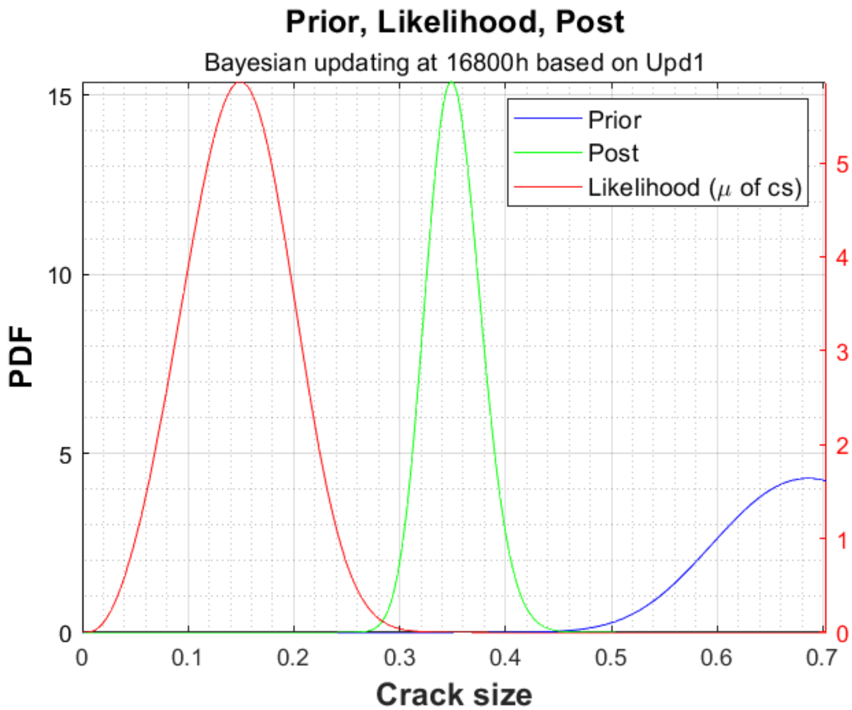


Fig. 33. Prior, likelihood and posterior distributions for detection example.

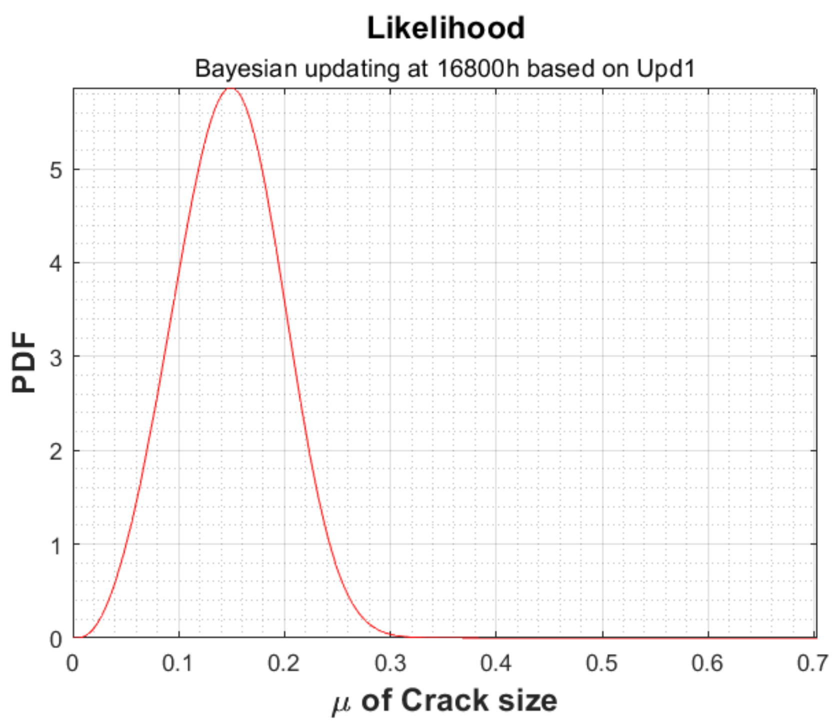


Fig. 34. Likelihood distribution for detection example.

From Fig. 32. It can be seen that the inspection should be done at 16,800 flight hours to prevent reaching the risk (although it was input 17,000 hours, the program takes the hours that the probability of failure has, which is 16,800). From Fig. 33 it can be seen that due to the crack found the mean for crack size distribution also became less than the predicted model and as consequence, it can be seen in Fig. 34 that the updated probability of failure is less risky than the initial one.

VI. CONCLUSION

As conclusion, the objective of this work, which was to create an application capable of updating the crack size distribution for a fleet with information given by inspections, was fulfilled through a MATLAB software application. This software application was made to first run a simulation on SMART|DT with a preexisting setup for cracks in an airplane, and with the results of that simulation, read a prior crack size distribution at a given time. Then, with information from integrated sensors during inspections, the application creates a likelihood crack size distribution. Finally, it computes a new crack size distribution called posterior distribution, using the prior and likelihood distributions obtained before. Additionally, the application runs again a simulation with the new setup to update the fleet probability of failure. The application can do this process as many times as desired, but always based on a probability of failure without repair.

REFERENCES

[1] H. Millwater, J. D. Ocampo, and T. Castaldo, “Probabilistic damage tolerance analysis for general aviation,” in *Advanced Materials Research*, 2014, vol. 891–892, doi: 10.4028/www.scientific.net/AMR.891-892.1191.

[2] J. A. Bannantine, J. J. Comer, and J. L. Handrock, *FUNDAMENTALS OF METAL FATIGUE ANALYSIS*, First. Englewood Cliffs, NJ: Prentice Hall, 1989.

[3] P. H. Wirsching, “Statistical Summaries of Fatigue Data for Design Purposes.,” *NASA Contract. Reports*, no. July, 1983.

[4] F. G. Pascual and W. Q. Meeker, “Estimating Fatigue Curves With the Random Fatigue-Limit Mode Estimating Fatigue Curves With the Random Fatigue-Limit Mode,” 1997, doi: 10.1080/00401706.1999.10485925.This.

[5] M. I. Vallejo, M. J. Carvajal, and J. Ocampo, “Probabilistic Structural Fatigue and Risk Analysis on the PIPER -PA-28 Fleet, A Case Study.”

[6] J. Ocampo *et al.*, “Development of a probabilistic linear damage methodology for small aircraft,” in *Journal of Aircraft*, 2011, vol. 48, no. 6, pp. 2090–2106, doi: 10.2514/1.C031463.

[7] D. T. Rusk, K. Y. Lin, D. D. Swartz, and G. K. Ridgeway, “Bayesian updating of damage size probabilities for aircraft structural Life-Cycle management,” *19th AIAA Appl. Aerodyn. Conf.*, vol. 39, no. 4, 2001, doi: 10.2514/6.2001-1646.

[8] Y. T. Wu, M. Shiao, and H. R. Millwater, “A Bayesian-updating computational method for probabilistic damage tolerance analysis,” *Collect. Tech. Pap. - AIAA/ASME/ASCE/AHS/ASC Struct. Struct. Dyn. Mater. Conf.*, no. April, pp. 1–9, 2010, doi: 10.2514/6.2010-2686.

[9] D. Broek, “Introduction To Fracture Mechanics.,” *Fract Mech Des Methodol, Struct Mater Panel, Delft, Oct 5-6, Munich, Oct 9-10*, no. 97, 1978, doi: 10.13140/RG.2.1.1444.2408.

[10] J. Harter, *AFGROW USER GUIDE AND TECHNICAL MANUAL*. LexTech, Inc, 1999.

[11] D. Liu and D. Pons, “Crack Propagation Mechanisms for Creep Fatigue: A Consolidated Explanation of Fundamental Behaviours from Initiation to Failure,” *Metals (Basel).*, vol. 8, no. 8, p. 623, Aug. 2018, doi: 10.3390/met8080623.

[12] A. M. Johansen, “Monte Carlo Methods,” *Int. Encycl. Educ.*, pp. 296–303, Jan. 2010, doi: 10.1016/B978-0-08-044894-7.01543-8.

[13] S. Theodoridis, “Probability and Stochastic Processes,” *Mach. Learn.*, pp. 19–65, Jan. 2020, doi: 10.1016/B978-0-12-818803-3.00011-8.

[14] E. Heredia-Zavoni and R. Montes-Iturrizaga, “A Bayesian model for the probability distribution of fatigue damage in tubular joints,” *J. Offshore Mech. Arct. Eng.*, vol. 126, no. 3, pp. 243–249, 2004, doi: 10.1115/1.1782645.

[15] A. Lye, A. Cicirello, and E. Patelli, “Sampling methods for solving Bayesian model updating problems: A tutorial,” *Mech. Syst. Signal Process.*, vol. 159, p. 107760, Oct. 2021, doi: 10.1016/J.YMSSP.2021.107760.

APPENDIX

This section presents the MATLAB graphical user interface code used for the software application created for this work including the Bayesian updating methodology.

APPENDIX A. CODE

```
classdef BayesianUpdatingApp < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        GridLayout                      matlab.ui.container.GridLayout
        LeftPanel                       matlab.ui.container.Panel
        LoaddatButton                   matlab.ui.control.Button
        ButtonGroup                     matlab.ui.container.ButtonGroup
        FIELDDATALabel                  matlab.ui.control.Label
        CurvetoinspectLabel             matlab.ui.control.Label
        DropDown                        matlab.ui.control.DropDown
        ImportDatatxtLabel              matlab.ui.control.Label
        ImportCS                        matlab.ui.control.Button
        AddCrackSizesButton             matlab.ui.control.Button
        NumberofCrackSizesSpinner       matlab.ui.control.Spinner
        NumberofCrackSizesSpinnerLabel  matlab.ui.control.Label
        AddInspectionButton             matlab.ui.control.Button
        PODin1                          matlab.ui.control.NumericEditField
        MeanLabel                       matlab.ui.control.Label
        PODin2                          matlab.ui.control.NumericEditField
        StdDevLabel                     matlab.ui.control.Label
        Iftherewasnotcrackfoundinput0inthetableLabel  matlab.ui.control.Label
        CrackSizeFoundLabel             matlab.ui.control.Label
        PODdd                           matlab.ui.control.DropDown
        ProbabilityofDetectionLabel     matlab.ui.control.Label
        MARin                           matlab.ui.control.NumericEditField
        FlighthoursButton               matlab.ui.control.RadioButton
        MaximumAllowableRiskButton      matlab.ui.control.RadioButton
        RightPanel                      matlab.ui.container.Panel
        PROBABILISTICDAMAGETOLERANCEWITHBAYESIANUPDATINGLabel
matlab.ui.control.Label
        NI                              matlab.ui.control.NumericEditField
        FlighthoursforinspectionLabel   matlab.ui.control.Label
        POFPlot                         matlab.ui.control.UIAxes
    end

    % Properties that correspond to apps with auto-reflow
```

```
properties (Access = private)
    onePanelWidth = 576;
end


properties (Access = private)
    Txt; % Legend text
    file; % Imported file name
    file1; % Imported file name without extention
    risk; % Maximum allowable risk
    matrix; % Matrix with flights and probability of failure
    matrix1; % Matrix with flights and probability of failure for the first .dat file
    timeinsp; % Time for next inspection
    riskLine; % Risk Line curve object
    InsType; % Inspection Type
    InsID; % Number of Inspection to write new .dat file
    Inspections; % Time of repair
    RowInsp; % Row of inspection time
    timeinsplocal; % Local inspection time
    insp; % Number of inspection
    FieldCS; % List of crack sizes found
    curve; % list of curves to do inspections
    hplotPOF; % Handle for uninspected curve
end

methods (Access = public)
    % Method to create or update the crack sizes found
    function LoadCrackSize(app,CrackSizes)
        app.FieldCS=CrackSizes(:);
    end
end


% Callbacks that handle component events
methods (Access = private)

    % Button pushed function: LoaddatButton
    function LoaddatButtonPushed(app, event)
        app.AddInspectionButton.Enable='off';    % Disables all the commands
        app.LoaddatButton.Enable='off';
        app.ButtonGroup.Enable='off';
        hold(app.POFPlot, 'off')    % Hold off plot to erase everything when the new pdf is plotted
        datSearched = uigetfile('.dat');  %Search for a .dat file and imports it
        if datSearched==0   % datSearched is 0 when nothing was selected
            figure(app.UIFigure)    % Focuses the gui
            waitfor(msgbox('Did not select any file, try again.'))  % warning when nothing is
selected
```

```
            figure(app.UIFigure)    % Focuses the gui
            if ~isempty(app.file)   % app.file is not empty when there was a .dat file already
selected
               app.AddInspectionButton.Enable='on';    % Enables all the commands again
               app.LoaddatButton.Enable='on';
               app.ButtonGroup.Enable='on';
            end
            return  % Exits the function
         end
         app.file=datSearched;   % Saves the file name searched in app.file
         figure(app.UIFigure)    % Focuses the gui
         f=waitbar(0.2, 'Running SMART...'); % Creates the loading bar in 20%
         Smart=strcat("smartdta.exe ",app.file); % Creates a string with smartdta.exe and the .dat
file name
         app.file1=app.file(1:end-4);    % Saves the .dat file name without extension
         system(Smart);  % Runs SMART|DT with file imported
         waitbar(.8,f, 'Plotting...');   % Updates the loading bar to 80%
         input=readtable(strcat(app.file1,'_pof.csv'));  % Reads a table from .csv file from SMART
         app.matrix= input{:,1:end}; % Converts the table to matrix
         app.matrix1=app.matrix; % Saves the first matrix in matrix1
         semilogy(app.POFPlot,app.matrix(:,2), app.matrix(:,3), 'b') %Plots POF from simulation
         app.Txt={'Uninspected'};    % Adds Uninspected to txt variable for the plot legend
         legend(app.POFPlot,app.Txt) % Displays the legend
         hold(app.POFPlot, 'on') % Holds the plot
         app.insp=0; % Sets number of inspection to 0
         app.timeinsp(1)=0;  % Sets the first time of inspection to 0
         app.timeinsplocal=0;    % Sets the local time of inspection to 0
         app.curve="Uninspected";    % Adds uninspected to curve variable
         app.DropDown.Items=app.curve;   % Sets the drop down list to curve variable
         waitbar(1,f, 'Finishing...');   % Updates the loading bar to 100%
         pause(0.5)  % Waits 0.5 seconds
         close(f)    % Closes the loading bar
         figure(app.UIFigure)    % Focuses the gui
         app.FieldCS=[];  % Sets crack sizes found as empty
         app.NumberofCrackSizesSpinner.Value=1;
         app.AddInspectionButton.Enable='on';    % Enables all the commands again
         app.LoaddatButton.Enable='on';
         app.ButtonGroup.Enable='on';
      end

      % Value changed function: PODdd
      function PODddValueChanged(app, event)
         % Changes the labels for text boxs when it is selected deterministic or
         % Lognormal in the probability of detection drop down list
         value = app.PODdd.Value;
         switch value
            case 'DETERMINISTIC'    % If deterministic is selected
```

```
                app.MeanLabel.Text='Depth (a)'; % Crack size a
                app.StdDevLabel.Text='Depth (c)';   % Crack size b
            case 'LOGNORMAL'    % If lognormal is selected
                app.MeanLabel.Text='Mean (µ)';  % Mean
                app.StdDevLabel.Text='Std Dev(?)';  % Standard deviation
        end

    end

    % Button pushed function: AddInspectionButton
    function AddInspectionButtonPushed(app, event)
        if isempty(app.FieldCS) % If Crack sizes found is empty
            figure(app.UIFigure)    % Focuses the gui
            waitfor(msgbox('There are not crack sizes, try again.'))    % Displays a warning when
crack sizes found is empty
            figure(app.UIFigure)    % Focuses the gui
            return  % Exits the function
        end
        app.AddInspectionButton.Enable='off';   % Disables all the commands
        app.LoaddatButton.Enable='off';
        app.ButtonGroup.Enable='off';

        f=waitbar(0, 'Searching for inspections...');   % Creates the loading bar
        % Reads the pof table required to do the inspection based on the dropdown
        if app.DropDown.Value=="Uninspected"    % If Uninspected is selected in the dropdown
list
            app.matrix= app.matrix1;    % Uses the matrix from the first .dat file to do the process
            Ninspection="0";    % Sets the reference inspection as 0
        else    % If Uninspected is not selected in the dropdown list
            Ninspection=split(app.DropDown.Value);  % Splits the selected text from the
dropdown list by spaces
            Ninspection=string(Ninspection(2)); % Gets the number of the inspection from the
selected inspection in the dropdown list
            input=readtable(strcat(app.file1,'_Insp_Repair',Ninspection,'_pof.csv'));   % Reads the
table from .csv file to analyze
            app.matrix= input{:,1:end-1};   % Saves the table in matrix and deletes the last column

        end
        if (app.matrix(end,2)+app.timeinsp(str2num(Ninspection)+1))<app.MARin.Value % If
the time desired to do the inspection is greater than the last time from the matrix plus its initial
time
            waitfor(msgbox('Flight hours selected are outside probability of failure boundaries')) %
Warning that flight time is higher than the upper limit
            close(f);   % Closes the loading bar
            app.AddInspectionButton.Enable='on'; % Enables all the commands again
            app.LoaddatButton.Enable='on';
            app.ButtonGroup.Enable='on';
```

```
            return  % Exits the function
        end


        %% Creates and runs a .dat file to get the CDF of crack size
        waitbar(.1,f, 'Predicting crack size...');  % Updates the loading bar
        selectedButton = app.ButtonGroup.SelectedObject.Text;   % Saves the selected button,
Maximum allowable risk or fligh hours
        if app.insp==0  % If it is the first inspection
            if selectedButton == "Flight hours" % If flight hours is selected
                app.risk=app.MARin.Value;   % Risk will be represented by the flight time typed

                for i=1:length(app.matrix)  % Loop to go through the matrix

                    if app.matrix(i,2)> app.risk    % If time value at (i,2) is greater than the time input
                        app.timeinsp(app.insp+2)=app.matrix(i-1,2); % Saves the time at i-1 in the
position of inspection number + 2
                        app.RowInsp=i-1;    % Saves the row number

                        % Writes the .dat file in a matlab variable, crack size at timeinsp
                        fileName=strcat(app.file1,'.dat');  % Saves the file name from dat file
                        insp1=fopen(fileName,'r');  % Opens the initial dat file
                        i = 1;
                        tline = fgetl(insp1);

                        % Writes the initial file in a matlab variable
                        A{i} = tline;
                        while ischar(tline)
                            i = i+1;
                            tline = fgetl(insp1);
                            A{i} = tline;
                        end
                        fclose(insp1);  % Closes the dat file
                        % Writes a line to extract the crack size distribution at time of
                        % inspection
                        CrackSize=strcat("WRITE_CRACK_SIZE_CDF_AT =
",int2str(app.timeinsp(app.insp+2)));
                        % Writes in a variable the body for a new dat file with the line to extract the
crack
                        % size distribution
                        CrackS={A{1:14},CrackSize,A{15:end-1}};
                        % Creates a new dat file to write the body created
                        fileCS=strcat(app.file1,'_',int2str(app.timeinsp(app.insp+2)),'_Cs','.dat');
                        fileID = fopen(fileCS,'w'); % Opens the file
                        formatSpec = '%s\n';
                        [~,columns]=size(CrackS);
                        for i = 1:columns
```

```
                fprintf(fileID,formatSpec,CrackS{i});   % Writes the body
              end
              fclose(fileID); % Closes the file
              Smart=strcat("smartdta.exe ",fileCS);   % Creates a string with smartdta.exe and
the .dat file name
              system(Smart);  % Runs SMART|DT with file imported

              break   % Exits the loop
            end
          end
        else  % when maximum allowable risk is selected
          app.risk=app.MARin.Value;   % Saves the input risk in the variable
          for i=1:length(app.matrix)  % Loop to go through the matrix

            if app.matrix(i,end-1)> app.risk    % If risk value at (i, last-1) is greater than the
time input

              app.timeinsp(app.insp+2)=app.matrix(i-1,2); % Saves the time at i-1 in the
position of inspection number + 2
              app.RowInsp=i-1;    % Saves the row number

              % Writes the .dat file in a matlab variable, crack size at timeinsp
              fileName=strcat(app.file1,'.dat');  % Saves the file name from dat file
              insp1=fopen(fileName,'r');  % Opens the initial dat file
              i = 1;
              tline = fgetl(insp1);

              % Writes the initial file in a matlab variable
              A{i} = tline;
              while ischar(tline)
                i = i+1;
                tline = fgetl(insp1);
                A{i} = tline;
              end
              fclose(insp1);  % Closes the dat file
              % Writes a line to extract the crack size distribution at time of
              % inspection
              CrackSize=strcat("WRITE_CRACK_SIZE_CDF_AT =
",int2str(app.timeinsp(app.insp+2)));
              % Writes in a variable the body for a new dat file with the line to extract the
crack
              % size distribution
              CrackS={A{1:14},CrackSize,A{15:end-1}};
              % Creates a new dat file to write the body created
              fileCS=strcat(app.file1,'_',int2str(app.timeinsp(app.insp+2)),'_Cs','.dat');
              fileID = fopen(fileCS,'w'); % Opens the file
              formatSpec = '%s\n';
```

```
                    [~,columns]=size(CrackS);
                    for i = 1:columns
                        fprintf(fileID,formatSpec,CrackS{i}); % Writes the body
                    end
                    fclose(fileID); % Closes the file
                    Smart=strcat("smartdta.exe ",fileCS);   % Creates a string with smartdta.exe and
the .dat file name
                    system(Smart);  % Runs SMART|DT with file imported

                    break   % Exits the loop
                end
            end
        end

    else    % If it is not the first inspection
        if selectedButton == "Flight hours" % If flight hours is selected
            app.risk=app.MARin.Value-app.timeinsp(str2double(Ninspection)+1);   % Converts
the input time into local time substracting the initial time of the inspection
            for i=1:length(app.matrix)  % Loop to go through the matrix

                if app.matrix(i,2)> app.risk    % If time value at (i,2) is greater than the time input

                    if i==1 % If time in the first row is greater than input time
                        msgbox("Higher hours were already selected")    % Warning to input greater
flight hours
                        app.AddInspectionButton.Enable='on';    % Enables all the commands again
                        app.LoaddatButton.Enable='on';
                        app.ButtonGroup.Enable='on';
                        return  % Exits the function
                    end

                    app.timeinsplocal=app.matrix(i-1,2);    % Saves selected time from the matrix
in timeinsplocal
                    % Computes the initial time for next inspection

app.timeinsp(app.insp+2)=app.timeinsp(str2double(Ninspection)+1)+app.timeinsplocal;
                    app.RowInsp=i-1;    % Saves the row number

                    % Writes the .dat file in a matlab variable, crack size at timeinsp
                    if Ninspection=="0" % If selected curve for inspection is uninspected
                        fileName=app.file; % Uses the inital .dat file
                    else % If selected curve for inspection is not uninspected
                        fileName=strcat(app.file1,'_Updated',Ninspection,'.dat');   % Uses the
according .dat file
                    end
                    insp1=fopen(fileName,'r');  % Opens the that file
                    i = 1;
```

```
                tline = fgetl(insp1);
                % Writes the initial file in a matlab variable
                A{i} = tline;
                while ischar(tline)
                    i = i+1;
                    tline = fgetl(insp1);
                    A{i} = tline;
                end
                fclose(insp1);  % Closes the dat file
                % Writes a line to extract the crack size distribution at time of
                % inspection
                CrackSize=strcat("WRITE_CRACK_SIZE_CDF_AT =
",int2str(app.timeinsplocal));
                % Writes in a variable the body for a new dat file with the line to extract the
crack
                % size distribution
                CrackS={A{1:14},CrackSize,A{15:end-1}};
                if Ninspection=="0" % If selected curve for inspection is uninspected
                    % Creates new file name based on uninspected
                    fileCS=strcat(app.file1,'_',int2str(app.timeinsp(app.insp+2)),'_Cs','.dat');
                else    % If selected curve for inspection is not uninspected
                    % Creates new file name based on selected inspection

fileCS=strcat(app.file1,'_Updated',Ninspection,'_',int2str(app.timeinsp(app.insp+2)),'_Cs','.dat');
                end
                fileID = fopen(fileCS,'w'); % Opens the file
                formatSpec = '%s\n';
                [~,columns]=size(CrackS);
                for i = 1:columns
                    fprintf(fileID,formatSpec,CrackS{i});   % Writes the body
                end
                fclose(fileID); % Closes the file
                Smart=strcat("smartdta.exe ",fileCS);   % Creates a string with smartdta.exe and
the .dat file name
                system(Smart);  % Runs SMART|DT with file imported

                break   % Exits the loop
            end
        end
    else    % when maximum allowable risk is selected
        app.risk=app.MARin.Value;   % Saves the input risk
        for i=1:length(app.matrix)  % Loop to go through the matrix

            if app.matrix(i,end-1)> app.risk    % If time value at (i,last-1) is greater than the
time input

                if i==1 % If risk in the first row is greater than input risk
```

```
            msgbox("Higher risk was already selected")  % Warning to input greater risk
            app.AddInspectionButton.Enable='on';    % Enables all the commands again
            app.LoaddatButton.Enable='on';
            app.ButtonGroup.Enable='on';
            return  % Exits the function
        end

        app.timeinsplocal=app.matrix(i-1,2);    % Saves selected time from the matrix
in timeinsplocal
        app.RowInsp=i-1;    % Saves the row number
        % Computes the initial time for next inspection

app.timeinsp(app.insp+2)=app.timeinsp(str2double(Ninspection)+1)+app.timeinsplocal; % Uses
the according .dat file

        % Writes the .dat file in a matlab variable, crack size at timeinsp
        if Ninspection=="0" % If selected curve for inspection is uninspected
            fileName=app.file;  % Uses the inital .dat file
        else    % If selected curve for inspection is not uninspected
            fileName=strcat(app.file1,'_Updated',Ninspection,'.dat');   % Uses the
according .dat file
        end
        insp1=fopen(fileName,'r');  % Opens the that file
        i = 1;
        tline = fgetl(insp1);
        % Writes the initial file in a matlab variabl
        A{i} = tline;
        while ischar(tline)
            i = i+1;
            tline = fgetl(insp1);
            A{i} = tline;
        end
        fclose(insp1);  % Closes the dat file
        % Writes a line to extract the crack size distribution at time of
        % inspection
        CrackSize=strcat("WRITE_CRACK_SIZE_CDF_AT =
",int2str(app.timeinsplocal))
        % Writes in a variable the body for a new dat file with the line to extract the
crack
        % size distribution
        CrackS={A{1:14},CrackSize,A{15:end-1}};
        if Ninspection=="0" % If selected curve for inspection is uninspected
            % Creates new file name based on uninspected
            fileCS=strcat(app.file1,'_',int2str(app.timeinsp(app.insp+2)),'_Cs','.dat');
        else    % If selected curve for inspection is not uninspected
            % Creates new file name based on selected inspection
```

```
fileCS=strcat(app.file1,'_Updated',Ninspection,'_',int2str(app.timeinsp(app.insp+2)),'_Cs','.dat');
                end
                fileID = fopen(fileCS,'w'); % Opens the file
                formatSpec = '%s\n';
                [~,columns]=size(CrackS);
                for i = 1:columns
                   fprintf(fileID,formatSpec,CrackS{i});   % Writes the body
                end
                fclose(fileID); % Closes the file
                Smart=strcat("smartdta.exe ",fileCS);   % Creates a string with smartdta.exe and
the .dat file name
                system(Smart);  % Runs SMART|DT with file imported

                break   % Exits the loop
             end
          end
       end
    end

    app.NI.Value=app.timeinsp(app.insp+2);  % Sets flight hours for next inspection textbox
to time found in loop

    waitbar(.2,f, 'Updating crack size distribution...');   % Updates the waiting bar to 20%
    if app.insp==0  % If inspection number is the first one
       %   Saves the following csv file name in namecs

namecs=strcat(app.file1,'_',int2str(app.timeinsp(app.insp+2)),'_Cs_JCSDPrior_',int2str(app.timei
nsp((app.insp+2))),'.csv');

    else    % If inspection number is not the first one
       if Ninspection=="0" % If inspection is based on uninspected
          %   Saves the following csv file name in namecs

namecs=strcat(app.file1,'_',int2str(app.timeinsp(app.insp+2)),'_Cs_JCSDPrior_',int2str(app.timei
nsp((app.insp+2))),'.csv');
       else    % If inspection is not based on uninspected
          %   Saves the following csv file name in namecs

namecs=strcat(app.file1,'_Updated',Ninspection,'_',int2str(app.timeinsp(app.insp+2)),'_Cs_JCSD
Prior_',int2str(app.timeinsplocal),'.csv')
       end
    end
    app.insp=app.insp+1;    % Adds a new inspection

    fid = fopen(namecs, 'r');   % Opens the csv file
```

```
% Counts the number of columns of the file
n = csvread(namecs,4,1,[4,1,4,1]);
N = n + 1;

% Reads the sixth and n-th row
% row 6 has the crack sizes
% row n has the cumulative density function
readData_1 = textscan(fid, '%f' ,N ,'EmptyValue', 0,'HeaderLines',6,'Delimiter',',');
readData_2 = textscan(fid, '%f' ,N ,'EmptyValue', 0,'HeaderLines',n, 'Delimiter',',');
fclose(fid);

% Transforms the read data into numbers
crack_length_r = readData_1{1};
cdf_value_r = readData_2{1};

% Deletes the first cell of each variable
crackSize = crack_length_r(2:N);
cdf =  cdf_value_r(2:N);

%________________________________________________________________________

%% Prior

% fits the values of crack size and cdf to a lognormal distribution and gets the parameters

fiteq=fittype( @(mu,h,x) 1./(x*h*sqrt(2*pi)).*exp((-1/2)*((log(x)-mu)/h).^2));  % Fitting equation for Lognormal PDF
fiteq2=fittype( @(mu,h,x) 1/2+1./2*erf((log(x)-mu)./(sqrt(2)*h)));  % Fitting equation for Lognormal CDF

priorms=fit(crackSize,cdf,fiteq2,'StartPoint',[-1.5 0.2]);  % Fits the prior distribution CDF to a lognormal distribution to get its parameters

priornorm = coeffvalues(priorms);  % priornorm contains the parameters of the lognormal distribution

% priorsd contains the standar deviation of the distribution
priorsd=sqrt(exp(2*priornorm(1)+priornorm(2)^2)*(exp(priornorm(2)^2)-1));
% priormn contains the mean of the distribution
priormn=exp(priornorm(1)+priornorm(2)^2/2);

syms x      % Symbolic variable x
% prior contains the prior distribution equation in terms of x
% prior --> P-(?)
prior=1/(x*priornorm(2)*sqrt(2*pi))*exp((-1/2)*((log(x)-priornorm(1))/priornorm(2))^2);
```

```
%______________________________________________________________________________

    %% Likelihood
    waitbar(.3,f, 'Creating likelihood function...');   % Updates the loading bar to 30%

    % Mean and standard deviation of detection method
    mean=app.PODin1.Value;
    sigma=app.PODin2.Value;

    % phi and h are the lognormal parameters
    phi=log(mean^2/sqrt(mean^2+sigma^2));
    h=sqrt(log(sigma^2/mean^2+1));

    %POD is probability of detection and has a lognormal cumulative distribution equation
    PODeq=1/2+1/2*erf((log(x)-phi)/(sqrt(2)*h));

    %PND is probability of no detection
    PNDeq=1-PODeq;
    % phi1 and h1 are the parameters for the f(D|theta) lognormal distrubion,
    % and the prior standard deviation is assumed as the standard deviation for
    % this distribution
    phi1=log(x^2/sqrt(x^2+priorsd^2));
    h1=sqrt(log(priorsd^2/x^2+1));

    %Variable to identify if there are no detections
    Ndet=false;

    L_final=1;  % Initialization of variable to be replaced by likelihood function
    for i=1:length(app.FieldCS) % Loop for each crack size found

        if app.FieldCS(i)==0    % If there is not crack size found

            syms Dn % Symbolic variable Dn represents possible crack sizes.

            % f_D is the equation of f(D|theta)
            f_D=1/(Dn*h1*sqrt(2*pi))*exp((-1/2)*((log(Dn)-phi1)/h1)^2);

            % L_ND is the integral of PND times f_D with respect to Dn
            L=int(PNDeq*f_D,Dn,0,inf);

            Ndet=true;  % Identifies that a no detection exists

        else

            % It is assumed that a crack of 0.3in is detected
            D=app.FieldCS(i);
```

```
            % f_D is the equation of f(D|theta)
            f_D=1/(D*h1*sqrt(2*pi))*exp((-1/2)*((log(D)-phi1)/h1)^2);
            % L_D is the likelihood function for one detected crack in one inspection
            L=PODeq*f_D;

        end
        L_final=L_final*L;  % L_final value is replaced by likelihood function

    end
    if Ndet==true   % If a no detection exists
        x=0.001:0.001:0.5;  % Creates an array of numbers for x
        Likelihood=subs(L_final);   % Substitutes values of x in L_final function
    end

%_____________________________________________________________________________

    %% Posterior
    waitbar(.4,f, 'Creating new crack size distribution...');   % Updates the loading bar to 40%
    %Equation of posterior distribution
    post=L_final*prior;
    %Duplicate of the posterior distribution to apply the normalization factor later on
    postF=post;

    %Assignation of values to x from 0.001 to 2
    x=0.001:0.001:2;
    NormFact=0;  %Normalization factor

    % Substitution of the posterior distribution with the values of x
    Postval=double(subs(post));

    % Applying integral by definition to obtain the value of the normalization factor
    for i=1:(length(x)-1)
        NormFact=NormFact+((x(i+1)-x(i))*Postval(i+1));
    end

    % Posterior distribution divided by the normalization factor
    postF=postF/NormFact;

    % Substitution of the new posterior distrubtion with the values of x
    postFVal=double(subs(postF));
    x=transpose(x);
    postFVal=transpose(postFVal);

    % Fitting of posterior distribution to a lognormal distribution equation to obtain the
    % parameters
    postFms=fit(x,postFVal,fiteq,'StartPoint',[priornorm(1) priornorm(2)]);
    postFnorm = coeffvalues(postFms);
```

```
        % postmn contains the mean of the posterior distribution and postsd the standard
deviation
        postsd=sqrt(exp(2*postFnorm(1)+postFnorm(2)^2)*(exp(postFnorm(2)^2)-1));
        postmn=exp(postFnorm(1)+postFnorm(2)^2/2);



%______________________________________________________________________
___
        waitbar(.5,f, 'Creating new .dat file...'); % Updates the loading bar to 50%

        % Reads and writes the initial .dat file in a matlab variable
        insp1=fopen(app.file,'r'); % Opnes the initial file
        i = 1;
        tline = fgetl(insp1);
        IniDat{i} = tline;
        while ischar(tline)
           i = i+1;
           tline = fgetl(insp1);
           IniDat{i} = tline; % Saves the body from the initial file in IniDat
        end
        fclose(insp1);      % Closes the initial file

        POI=strcat("PROB_OF_INSPECTION = DETERMINISTIC 1.0");   % Writes a line for
probability of inspection
        POD=strcat("POD = ",app.PODdd.Value," ",num2str(app.PODin1.Value),"
",num2str(app.PODin2.Value));   % Writes a line for probability of Detection with input
parameters
        CSrow=split(IniDat{19});    % Saves in CSrow the initial crack size distribution
        CS=strcat("REPAIR_CRACK_SIZE = LOGNORMAL ",CSrow(end-1)," ",CSrow(end));
% Writes the repaired crack size as the initial crack size
        ICS=strcat("INITIAL_CRACK_SIZE = LOGNORMAL ",num2str(postmn),"
",num2str(postsd));  % Writes the initial crack size as the parameters of posterior distribution
from bayesian updating

        % Writes the inspection
        app.Inspections="INSPECTIONS = 1";
        app.InsType="INSPECTION_TYPE = 1";
        app.InsID="INSPECTION_ID = 1";
        if Ninspection=="0"        % If uninspected was selected
           % Writes a comment to clarify what curve was selected to perform bayesian
           % updating
           comment=strcat("!  Inspection ",int2str(app.insp)," at
",int2str(app.timeinsp(app.insp+1)),"h"," based on Uninspected");
        else    % If uninspected was not selected
           % Writes a comment to clarify what curve was selected to perform bayesian
```

```
        % updating
        comment=strcat("!  Inspection ",int2str(app.insp)," at
",int2str(app.timeinsp(app.insp+1)),"h"," based on inspection ",Ninspection);
    end

B={IniDat{1:18},ICS,IniDat{20:24},app.Inspections,app.InsType,'',app.InsID,POI,POD,CS,IniD
at{26:end-1},comment}; % Writes a dat file for inspection and repair in a variable
    C={IniDat{1:18},ICS,IniDat{20:end-1},comment}; % Writes a dat file for inspection in a
variable

    % Input name of .dat file, Writes the new .dat file in a
    % variable and run smart with it
    filename=strcat(app.file1,'_Insp_Repair',int2str(app.insp),'.dat');
    fileID = fopen(filename,'w');   % Opens a new file to write
    formatSpec = '%s\n';
    [~,columns]=size(B);
    for i = 1:columns
        fprintf(fileID,formatSpec,B{i});    % Writes information for inspection and repair
    end
    fclose(fileID); % Closes the file
    Smart=strcat("smartdta.exe ",filename);     % Creates a string with smartdta.exe and the
.dat file name
    waitbar(.6,f, 'Running SMART...');      % Updates the loading bar to 60%
    system(Smart);  % Runs SMART|DT with file imported

    %-----------
    waitbar(.7,f, 'Creating new .dat files...');    % Updates the loading bar to 70%
    filename=strcat(app.file1,'_Updated',int2str(app.insp),'.dat');
    fileID = fopen(filename,'w');   % Opens a new file to write
    formatSpec = '%s\n';
    [~,columns]=size(C);
    for i = 1:columns
        fprintf(fileID,formatSpec,C{i});    % Writes information for inspection and update
    end
    fclose(fileID); % Closes the file
    Smart=strcat("smartdta.exe ",filename); % Creates a string with smartdta.exe and the .dat
file name
    waitbar(.8,f, 'Running SMART...');  % Updates the loading bar to 80%
    system(Smart);  % Runs SMART|DT with file imported

    %-----------
    waitbar(.9,f, 'Plotting...');   % Updates the loading bar to 90%
    if app.insp==1      % If it is the first inspection
        hold(app.POFPlot, 'off')    % Stops holding the plot
        % Creates a handle for the first POF curve (Uninspected)
        app.hplotPOF=semilogy(app.POFPlot,app.matrix1(1:app.RowInsp,2),
app.matrix1(1:app.RowInsp,3), 'b')
```

```
            app.Txt={'Uninspected'};    % Adds the legend
            hold(app.POFPlot, 'on') % Holds the plot
        end
        if Ninspection=="0" && app.insp~=1  % If the curve to inspect selected was Uninspected
and it is not the first inspection
            % Updates the POF curve for uninspected

set(app.hplotPOF,'XData',app.matrix1(1:app.RowInsp,2),'YData',app.matrix1(1:app.RowInsp,3))
;
        end

        % Creates a horizontal line at input risk
        app.riskLine=yline(app.matrix(app.RowInsp,3),'--r',strcat("Risk at
",int2str(app.timeinsp(app.insp+1))," for Upd",Ninspection),'Parent',app.POFPlot);
        % Omits the line for the legend
        app.Txt{(app.insp*3)-1}='';

        %% Prior, Likelihood and Posterior plot
        figure  % Creates a new figure
        fplot(prior,[0,2],'b')  % Plots the prior PDF between 0 and 2
        hold on        % Holds the plot
        yyaxis right    % Selects right y axis
        if Ndet==true   % If there is at least one no detection
            x=0.001:0.001:0.5;
            plot(x,Likelihood,'r')  % Plots the likelihood between 0 and 0.5
        else    % If there are not no detection
            fplot(L_final,[0 postmn*2],'r') % Plots the likelihood between 0 and 2 times the
posterior distribution mean
        end
        set(gca,'ycolor','r') % Sets the left y axis color to black and the right one to red

        yyaxis left % Selects left y axis
        x=0.001:0.001:postmn*2;
        postPDF=lognpdf(x,postFnorm(1),postFnorm(2));
        plot(x,postPDF','g')    % Plots the posterior distribution between 0 and 2 times the
posterior distribution mean

        title('Prior, Likelihood, Post','FontSize',14,'FontWeight','bold')  % Adds a title to the plot
        % Creates a text for subtitle
        txt = strcat("Bayesian updating at ", int2str(app.timeinsp(app.insp+1)), "h based on Upd",
Ninspection);
        subtitle(txt)   % Adds a subtitle
        xlim([0 postmn*2])  % Sets the x limits between 0 and two times the posterior
distribution mean
        xlabel('Crack size','FontSize',14,'FontWeight','bold')  % Creates the label and sets font to x
axis
        ylabel('PDF','FontSize',14,'FontWeight','bold') % Creates the label and sets font to y axis
```

```
grid on % Activates the grid
grid minor
legend(["Prior","Post","Likelihood (\mu of cs)"],'FontSize',11) % Adds legend


%Likelihood plot
figure  % Creates a new figure
if Ndet==true   % If there is at least one no detection
   x=0.001:0.001:0.5;
   plot(x,Likelihood,'r')  % Plots the likelihood between 0 and 0.5
else    % If there are not no detection
   fplot(L_final,[0 postmn*2],'r') % Plots the likelihood between 0 and 2 times the
posterior distribution mean
end

title('Likelihood','FontSize',14,'FontWeight','bold')   % Adds a title to the plot
% Creates a text for subtitle
txt = strcat("Bayesian updating at ", int2str(app.timeinsp(app.insp+1)), "h based on Upd",
Ninspection);
subtitle(txt)   % Adds a subtitle

xlabel('\mu of Crack size','FontSize',14,'FontWeight','bold')   % Creates the label and sets
font to x axis
ylabel('PDF','FontSize',14,'FontWeight','bold') % Creates the label and sets font to y axis
grid on % Activates the grid
grid minor


input1=readtable(strcat(app.file1,'_Updated',int2str(app.insp),'_pof.csv'));    % Reads
updated distribution POF
app.matrix= input1{:,1:end-1};  % Converts the table into matrix
semilogy(app.POFPlot,app.matrix(:,2)+app.timeinsp(app.insp+1), app.matrix(:,3)) %
Plots the updated distribution POF
% Adds label of previous curve to txt variable
app.Txt{app.insp*3}=strcat("Upd",int2str(app.insp), " w/o Repair at
",int2str(app.timeinsp(app.insp+1)),"h"," based on Upd",Ninspection);

input2=readtable(strcat(app.file1,'_Insp_Repair',int2str(app.insp),'_pof.csv'));    % Reads
POF of inspected and repaired distribution
app.matrix= input2{:,1:end-1};  % Converts the table into matrix
semilogy(app.POFPlot,app.matrix(:,2)+app.timeinsp(app.insp+1), app.matrix(:,5)) %
Plots the probability of failure for inspected and repaired distribution
% Adds label of previous curve to txt variable
app.Txt{(app.insp*3)+1}=strcat("Upd",int2str(app.insp)," w Repair at
",int2str(app.timeinsp(app.insp+1)),"h"," based on Upd",Ninspection);
```

```
            legend(app.POFPlot,app.Txt,'Location','Best')   % Updates the legend and locates it at the best position
            % Creates a line for drop down list
            var=strcat("Updated ",int2str(app.insp), " w/o Repair at ",int2str(app.timeinsp(app.insp+1)),"h");
            app.curve=horzcat(app.curve,var);
            app.DropDown.Items=app.curve;   % Adds the line to drop down list
            app.DropDown.Value=var; % Sets the drop down list into the new line
            waitbar(1,f, 'Finishing...');   % Updates the loading bar to 100%
            pause(0.5)  % Waits 0.5 seconds
            close(f)    % Closes the loading bar
            app.AddInspectionButton.Enable='on';    % Enables all the commands again
            app.LoaddatButton.Enable='on';
            app.ButtonGroup.Enable='on';

        end

        % Selection changed function: ButtonGroup
        function ButtonGroupSelectionChanged(app, event)
            selectedButton = app.ButtonGroup.SelectedObject.Text;
            if selectedButton == "Flight hours" % If Flight hours is selected
                app.MARin.Limits=[0,inf];   % Sets the limits to 0 and the infinte
                app.MARin.Value=20000;  % Sets the value of the textbox to 20000
                app.MARin.ValueDisplayFormat = '%d';    % Changes the text box format to integer


            else    % If Risk is selected

                app.MARin.ValueDisplayFormat = '%11.5g';    % Changes the text box format to scientific
                app.MARin.Value=1e-7;   % Sets the value of the textbox to 1e-7
                app.MARin.Limits=[0,1]  % Sets the limits to 0 and 1

            end
        end

        % Button pushed function: AddCrackSizesButton
        function AddCrackSizesButtonPushed(app, event)
            % Creates the table to input crack sizes based on the number of the spinner
            CrackSizeList(app,app.NumberofCrackSizesSpinner.Value,app.FieldCS);

        end

        % Button pushed function: ImportCS
        function ImportCSPushed(app, event)
            [CSfile,path,~]=uigetfile('.txt');  % Opens a window to select file with crack sizes
            if CSfile==0    % If nothing was selected
```

```
            figure(app.UIFigure)    % Focuses the gui
            waitfor(msgbox('Did not select any file, try again.'))  % Displays a warning
            figure(app.UIFigure)    % Focuses the gui
            return  % Exits the function
        end
        figure(app.UIFigure)    % Focuses the gui
        filename=[path CSfile]; % Saves the path and name of file selected
        A=importdata(filename); % Imports the crack sizes from file selected into A
        A=double(A);    % Converts the imported data into double format numbers
        LoadCrackSize(app,A)    % Calls the function to create or update crack sizes
        app.NumberofCrackSizesSpinner.Value=size(A,1);  % Sets the number of the spinner into
the number of crack sizes imported
    end

    % Close request function: UIFigure
    function UIFigureCloseRequest(app, event)
        % If the gui is closed with the close button, every plot that was created with
        % that instance is also closed
        delete(app)
        close all
    end

    % Changes arrangement of the app based on UIFigure width
    function updateAppLayout(app, event)
        currentFigureWidth = app.UIFigure.Position(3);
        if(currentFigureWidth <= app.onePanelWidth)
            % Change to a 2x1 grid
            app.GridLayout.RowHeight = {543, 543};
            app.GridLayout.ColumnWidth = {'1x'};
            app.RightPanel.Layout.Row = 2;
            app.RightPanel.Layout.Column = 1;
        else
            % Change to a 1x2 grid
            app.GridLayout.RowHeight = {'1x'};
            app.GridLayout.ColumnWidth = {262, '1x'};
            app.RightPanel.Layout.Row = 1;
            app.RightPanel.Layout.Column = 2;
        end
    end
end

% Component initialization
methods (Access = private)

    % Create UIFigure and components
    function createComponents(app)
```

```
        % Create UIFigure and hide until all components are created
        app.UIFigure = uifigure('Visible', 'off');
        app.UIFigure.AutoResizeChildren = 'off';
        app.UIFigure.Position = [100 100 838 543];
        app.UIFigure.Name = 'MATLAB App';
        app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);
        app.UIFigure.SizeChangedFcn = createCallbackFcn(app, @updateAppLayout, true);
        app.UIFigure.Pointer = 'hand';

        % Create GridLayout
        app.GridLayout = uigridlayout(app.UIFigure);
        app.GridLayout.ColumnWidth = {262, '1x'};
        app.GridLayout.RowHeight = {'1x'};
        app.GridLayout.ColumnSpacing = 0;
        app.GridLayout.RowSpacing = 0;
        app.GridLayout.Padding = [0 0 0 0];
        app.GridLayout.Scrollable = 'on';

        % Create LeftPanel
        app.LeftPanel = uipanel(app.GridLayout);
        app.LeftPanel.Layout.Row = 1;
        app.LeftPanel.Layout.Column = 1;

        % Create ButtonGroup
        app.ButtonGroup = uibuttongroup(app.LeftPanel);
        app.ButtonGroup.SelectionChangedFcn = createCallbackFcn(app,
@ButtonGroupSelectionChanged, true);
        app.ButtonGroup.Enable = 'off';
        app.ButtonGroup.BorderType = 'none';
        app.ButtonGroup.Position = [20 25 222 462];

        % Create MaximumAllowableRiskButton
        app.MaximumAllowableRiskButton = uiradiobutton(app.ButtonGroup);
        app.MaximumAllowableRiskButton.Text = 'Maximum Allowable Risk';
        app.MaximumAllowableRiskButton.WordWrap = 'on';
        app.MaximumAllowableRiskButton.Position = [11 396 110 44];
        app.MaximumAllowableRiskButton.Value = true;

        % Create FlighthoursButton
        app.FlighthoursButton = uiradiobutton(app.ButtonGroup);
        app.FlighthoursButton.Text = 'Flight hours';
        app.FlighthoursButton.Position = [128 407 85 22];

        % Create MARin
        app.MARin = uieditfield(app.ButtonGroup, 'numeric');
        app.MARin.Limits = [0 Inf];
        app.MARin.Position = [71 375 81 22];
```

```
app.MARin.Value = 1e-07;

% Create ProbabilityofDetectionLabel
app.ProbabilityofDetectionLabel = uilabel(app.ButtonGroup);
app.ProbabilityofDetectionLabel.HorizontalAlignment = 'center';
app.ProbabilityofDetectionLabel.FontWeight = 'bold';
app.ProbabilityofDetectionLabel.Position = [44 348 140 22];
app.ProbabilityofDetectionLabel.Text = 'Probability of Detection';

% Create PODdd
app.PODdd = uidropdown(app.ButtonGroup);
app.PODdd.Items = {'DETERMINISTIC', 'LOGNORMAL', 'TABULAR'};
app.PODdd.ValueChangedFcn = createCallbackFcn(app, @PODddValueChanged, true);
app.PODdd.Enable = 'off';
app.PODdd.Position = [47 318 133 22];
app.PODdd.Value = 'LOGNORMAL';

% Create CrackSizeFoundLabel
app.CrackSizeFoundLabel = uilabel(app.ButtonGroup);
app.CrackSizeFoundLabel.HorizontalAlignment = 'center';
app.CrackSizeFoundLabel.FontWeight = 'bold';
app.CrackSizeFoundLabel.Position = [59 224 106 22];
app.CrackSizeFoundLabel.Text = 'Crack Size Found';

% Create IftherewasnotcrackfoundinputOinthetableLabel
app.Iftherewasnotcrackfoundinput0inthetableLabel = uilabel(app.ButtonGroup);
app.Iftherewasnotcrackfoundinput0inthetableLabel.HorizontalAlignment = 'center';
app.Iftherewasnotcrackfoundinput0inthetableLabel.WordWrap = 'on';
app.Iftherewasnotcrackfoundinput0inthetableLabel.FontSize = 10;
app.Iftherewasnotcrackfoundinput0inthetableLabel.Position = [27 99 170 41];
app.Iftherewasnotcrackfoundinput0inthetableLabel.Text = '*If there was not crack found,
input 0 in the table.';

% Create StdDevLabel
app.StdDevLabel = uilabel(app.ButtonGroup);
app.StdDevLabel.HorizontalAlignment = 'right';
app.StdDevLabel.Position = [23 260 63 22];
app.StdDevLabel.Text = 'Std Dev(?)';

% Create PODin2
app.PODin2 = uieditfield(app.ButtonGroup, 'numeric');
app.PODin2.Limits = [0 Inf];
app.PODin2.Position = [101 260 100 22];
app.PODin2.Value = 0.07;

% Create MeanLabel
app.MeanLabel = uilabel(app.ButtonGroup);
```

```
        app.MeanLabel.HorizontalAlignment = 'right';
        app.MeanLabel.Position = [30 287 56 22];
        app.MeanLabel.Text = 'Mean (µ)';

        % Create PODin1
        app.PODin1 = uieditfield(app.ButtonGroup, 'numeric');
        app.PODin1.Limits = [0 Inf];
        app.PODin1.Position = [101 287 100 22];
        app.PODin1.Value = 0.06;

        % Create AddInspectionButton
        app.AddInspectionButton = uibutton(app.ButtonGroup, 'push');
        app.AddInspectionButton.ButtonPushedFcn = createCallbackFcn(app,
@AddInspectionButtonPushed, true);
        app.AddInspectionButton.FontWeight = 'bold';
        app.AddInspectionButton.Position = [60 8 104 31];
        app.AddInspectionButton.Text = 'Add Inspection';

        % Create NumberofCrackSizesSpinnerLabel
        app.NumberofCrackSizesSpinnerLabel = uilabel(app.ButtonGroup);
        app.NumberofCrackSizesSpinnerLabel.WordWrap = 'on';
        app.NumberofCrackSizesSpinnerLabel.Position = [115 182 98 49];
        app.NumberofCrackSizesSpinnerLabel.Text = 'Number of Crack Sizes';

        % Create NumberofCrackSizesSpinner
        app.NumberofCrackSizesSpinner = uispinner(app.ButtonGroup);
        app.NumberofCrackSizesSpinner.Limits = [1 Inf];
        app.NumberofCrackSizesSpinner.Position = [113 168 100 22];
        app.NumberofCrackSizesSpinner.Value = 1;

        % Create AddCrackSizesButton
        app.AddCrackSizesButton = uibutton(app.ButtonGroup, 'push');
        app.AddCrackSizesButton.ButtonPushedFcn = createCallbackFcn(app,
@AddCrackSizesButtonPushed, true);
        app.AddCrackSizesButton.Position = [109 136 104 22];
        app.AddCrackSizesButton.Text = 'Add Crack Sizes';

        % Create ImportCS
        app.ImportCS = uibutton(app.ButtonGroup, 'push');
        app.ImportCS.ButtonPushedFcn = createCallbackFcn(app, @ImportCSPushed, true);
        app.ImportCS.Icon = 'import-data-icon-20.jpg';
        app.ImportCS.Position = [39 177 44 37];
        app.ImportCS.Text = '';

        % Create ImportDatatxtLabel
        app.ImportDatatxtLabel = uilabel(app.ButtonGroup);
        app.ImportDatatxtLabel.HorizontalAlignment = 'center';
```

```
app.ImportDatatxtLabel.Position = [27 148 68 27];
app.ImportDatatxtLabel.Text = {'Import Data'; '(.txt)'};

% Create DropDown
app.DropDown = uidropdown(app.ButtonGroup);
app.DropDown.Items = {};
app.DropDown.Position = [1 53 222 22];
app.DropDown.Value = {};

% Create CurvetoinspectLabel
app.CurvetoinspectLabel = uilabel(app.ButtonGroup);
app.CurvetoinspectLabel.HorizontalAlignment = 'center';
app.CurvetoinspectLabel.FontWeight = 'bold';
app.CurvetoinspectLabel.Position = [63 80 99 22];
app.CurvetoinspectLabel.Text = 'Curve to inspect';

% Create FIELDDATALabel
app.FIELDDATALabel = uilabel(app.ButtonGroup);
app.FIELDDATALabel.HorizontalAlignment = 'center';
app.FIELDDATALabel.FontWeight = 'bold';
app.FIELDDATALabel.Position = [54 438 104 22];
app.FIELDDATALabel.Text = 'FIELD DATA :';

% Create LoaddatButton
app.LoaddatButton = uibutton(app.LeftPanel, 'push');
app.LoaddatButton.ButtonPushedFcn = createCallbackFcn(app, @LoaddatButtonPushed,
true);
app.LoaddatButton.FontWeight = 'bold';
app.LoaddatButton.Position = [78 498 94 24];
app.LoaddatButton.Text = 'Load .dat';

% Create RightPanel
app.RightPanel = uipanel(app.GridLayout);
app.RightPanel.Layout.Row = 1;
app.RightPanel.Layout.Column = 2;

% Create POFPlot
app.POFPlot = uiaxes(app.RightPanel);
title(app.POFPlot, 'POF')
xlabel(app.POFPlot, 'Flight Hours')
ylabel(app.POFPlot, 'Probability of Failure')
zlabel(app.POFPlot, 'Z')
app.POFPlot.XGrid = 'on';
app.POFPlot.YGrid = 'on';
app.POFPlot.Position = [22 66 526 391];

% Create FlighthoursforinspectionLabel
```

```
            app.FlighthoursforinspectionLabel = uilabel(app.RightPanel);
            app.FlighthoursforinspectionLabel.HorizontalAlignment = 'right';
            app.FlighthoursforinspectionLabel.Position = [184 36 143 22];
            app.FlighthoursforinspectionLabel.Text = 'Flight hours for inspection';

            % Create NI
            app.NI = uieditfield(app.RightPanel, 'numeric');
            app.NI.ValueDisplayFormat = '%.0f';
            app.NI.Editable = 'off';
            app.NI.Position = [342 36 100 22];

            % Create
PROBABILISTICDAMAGETOLERANCEWITHBAYESIANUPDATINGLabel
            app.PROBABILISTICDAMAGETOLERANCEWITHBAYESIANUPDATINGLabel =
uilabel(app.RightPanel);

app.PROBABILISTICDAMAGETOLERANCEWITHBAYESIANUPDATINGLabel.Horizontal
Alignment = 'center';

app.PROBABILISTICDAMAGETOLERANCEWITHBAYESIANUPDATINGLabel.FontSize =
13;

app.PROBABILISTICDAMAGETOLERANCEWITHBAYESIANUPDATINGLabel.FontWeig
ht = 'bold';

app.PROBABILISTICDAMAGETOLERANCEWITHBAYESIANUPDATINGLabel.Position =
[45 467 485 32];

app.PROBABILISTICDAMAGETOLERANCEWITHBAYESIANUPDATINGLabel.Text =
'PROBABILISTIC DAMAGE TOLERANCE WITH BAYESIAN UPDATING';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = ProgramMultiInspV3_exported

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)
```

```
            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```