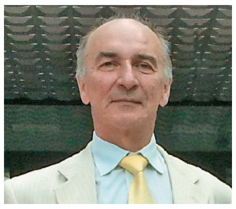
Original article
DOI: 10.21683/1729-2646-2018-18-1-4-13

Fault tree analysis in the R programming environment

Alexander V. Antonov, *JSC RASU, Moscow, Russia*
Evgeny Yu. Galivets, *JSC RASU, Moscow, Russia*
Valery A. Chepurko, *JSC RASU, Moscow, Russia*
Alexey N. Cherniaev, *JSC RASU, Moscow, Russia*


Alexander V. Antonov

Evgeny Yu. Galivets

Valery A. Chepurko

Alexey N. Cherniaev

Abstract. Aim. *Fault tree analysis (FTA) is one of the primary methods of dependability analysis of complex technical systems. This process often employs commercial software tools like Saphire, Risk Spectrum, Arbitr, etc. Each of them has both advantages and drawbacks. It must be noted that the primary purpose of the above software tools consists in performing qualitative fault tree analysis. The software systems additionally feature a number of statistical methods that, among other things, enable uncertainty analysis, interval estimation of indicators, as well as other statistical research. The number of such procedures is not large and is strictly limited by a certain array of proposed distributions and functions. In this paper, let us consider the possibility of solving the task of fault tree analysis by means of the R programming language. R was primarily created and is continuing to evolve as a statistical data processing tool. FTA in this environment is just one of 10 thousand packages. In other words, if compared to commercial packages with the FTA as the main function, the functionality of R is much wider and enables significantly higher quality of analysis. One of R's undeniable advantages is the freeware open-source environment. This paper aims to present a small number of primary procedures of R's FaultTree package that enable FTA: construction and display of fault trees, calculation of probability per nodes, determination of minimum cross-sections.* **Methods.** *R's FaultTree scripts were used for the calculations and FTA capabilities demonstration.* **Conclusions.** *Three examples are examined in detail. First, a tree is calculated based on known probabilities, then the time to failure distribution function of a technical system is identified. In the last example, FTA is performed for systems with elements that are described by different functional and service models. In conclusion, FTA capabilities of R are described that allow, for instance, taking into consideration common cause failures.*

Keywords: *fault tree, fault tree analysis, non-availability factor, evident failures, hidden failures, mean time between failure.*

For citation: *Antonov AV, Galivets EYu, Chepurko VA, Cherniaev AN. Fault tree analysis in the R programming environment. Dependability 2018;18(1): 4-13. DOI: 10.21683/1729-2646-2018-18-1-4-13*

Introduction

R is a programming language for statistical computing and graphics, as well as a free open-source computing software environment developed as part of the GNU project. Primarily, R was created by Ross Ihaka and Robert Gentleman (R is the first letter of their names), statistics researchers from the University of Auckland. The programming language and environment are supported and developed by the *R Foundation Company*. R entered into the winners list of the InfoWorld magazine contest in the nomination for the best open software for applications development in 2010.

R supports a wide range of statistical and numerical methods and is highly extensible through the use of packages. Packages are libraries that provide specific functions and subprograms or specific applications. A package is a set of functions, files with reference information and examples collected together in one archive used as an additional extension based on R. R is a programming language, thus own programs, or scripts, can be designed and specialized extensions, or packages, can be used.

A number of graphical interfaces such as RStudio, JGR, etc. were developed to simplify the process of working with R. R can perform simple calculations, edit tables with data and make simple statistical analysis (e.g. t-test, ANOVA or regression analysis) and more complex time-consuming computations, test hypotheses, construct vector graphics. Yet this is not the full list of R functions.

This article deals with the package for "constructing, calculating and displaying" fault trees, the FaultTree package. The author of this package is David J. Silkworth and the maintainer is Jacob T. Ormerod. FaultTree provides a set of functions for building tree structures as Dataframe objects. The fault tree includes logic nodes (primarily "AND" and "OR") that process input data and may direct output data "upwards" through the tree structure.

The method of fault tree analysis (FTA) was first used by Bell Labs for the US Air Force in 1962. Today, FTA is widely used to analyze failures of static systems.

FTA is a method for failures analysis in complex systems, in which undesired states of system failures are analyzed using Boolean algebra methods to combine a series of lower-level events (failures of the lowest level), which lead to the failure of the whole system. Fault tree analysis is widely used in various industries to calculate technical system dependability, identify the most unreliable elements subject to frequent failures. In this case chains of random events are defined, under which the system may fail, methods for reducing risks are identified and the frequencies of system failures are determined. Fault tree analysis is effectively used in the aerospace, nuclear power, chemical and processing industries, pharmaceutical, oil and gas and other high-hazard sectors.

The FaultTree is one of a huge number of packages of the R language, among which most are related to the qualitative statistical processing with easy application of different probability distributions. This abundance and at the same time flexibility of the language combined with the simplicity of scripting, make it quite an interesting solution for not only FTA, but for additional research that cannot be performed in known software products. The second example shows a method of FTA with the capability of dynamic analysis of object behavior when probability of events is functions of time. This probability allows obtaining not just a static (at a certain time) estimate of a system's failure probability but some time dependence of failure probability which allows, for example, reasonably determining a system's residual life not limited to exponential distribution. Not only standard laws, but also nonparametric distribution estimates can be used as initial distribution of system elements, which sets the R language apart.

Let us consider some basic features of the FaultTree package.

Calculation of the fault tree with known event probabilities

As an example, let us calculate with R the failure probability of a chain of 8 elements, each of which refers to non-recoverable objects and has specific failure probability (Fig. 1, Table 1).

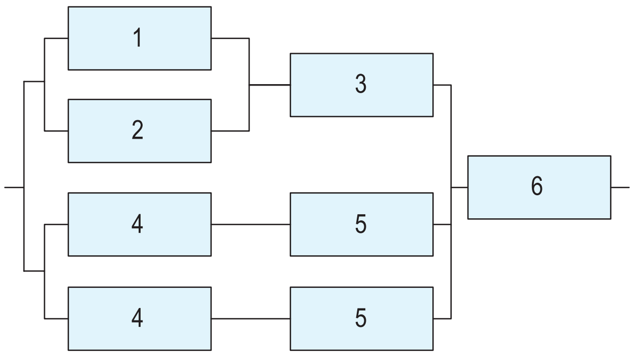


Figure 1. Chain diagram

Table 1. Failure probabilities of chain elements

Element	1	2	3	4	5	6
q_i	0,01	0,02	0,03	0,04	0,05	0,06

Note that the continuous set 4-5 is single redundant, i.e. in the lower part of the diagram there are two 4-5 sections connected in parallel.

In order to calculate the diagram it is required to connect the FaultTree package. This can be done in at least two ways: by directly connecting the package in the RStudio environment or by typing the following command in the generated script:

library(FaultTree).

Then, the tree is to be built by calling the free.make script which has several arguments, and the most important of which is the type of the generated tree rather than the type of the upper gate in the fault tree. In the current version of the FaultTree library, there are two basic types of the gate:

"or" and "and". Let us focus on Figure 1. The chain is a serial connection of the element 6 and the complex subchain 1-5 consisting of elements 1, 2, …, 5. Therefore, the chain failure will occur if the element 6 of the subchain 1-5 fails (or both occur immediately). Thus, it necessary to choose the type of the upper gate "or":

1. Tree1 <- ftree.make(type="or", name="Example 1. Calculation", name2="on probabilities").

In this script a data structure (dataframe) is created, i.e. a fault tree with the name (identifier) Tree1, which is essentially an "or" type gate. When the event tree is displayed graphically, this gate will be called "Example 1. Calculation in probabilities" (Fig. 2). Due to the fact that there is a limitation on the number of symbols in the title, long names sometimes have to be divided into two lines "name" and "name2". The upper gate will be assigned the number 1 (red value inside the gate). Let us note, that the line numbering of the script is given only for the convenience of making comments in the article and is not used in the syntax of the R language.

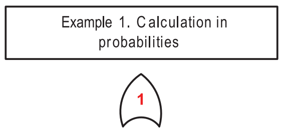


Figure 2. First step of the fault tree constructing. Upper gate of chain

The next step is to add two structures to gate 1: the event of "failure of element 6" and the "and" gate, which will be the upper gate in relation to the structure 1-5. The choice of this gate type is due to the fact that the structure 1-5 fails if both channels "1-3" and "4,5" simultaneously fail. Thus, the script will have the following commands:

2. Tree1 <- addProbability(Tree1, at=1, prob=.06, name=» Failure 6»),
3. Tree1 <- addLogic(Tree1, at=1, type= "and", name= "Failure 1-3",name2="and (4-5)*2").

In the first line the event "failure 6" is added to gate 1. Let us analyze in detail the arguments of the addProbability sub-program. The first argument is tree identifier to which the event will be built. The second argument "at" indicates the number of the parent node of the tree Tree1 to which this structure will be added. The third argument sets the numerical value of the failure probability. The fourth value, as before, is the name of the event. Let us note that the arguments of any sub-programs in the R language are permutable and do not have to obey a certain order. However, from the point of view of program debugging and simplifying of errors search it is reasonable to adhere to a certain order of arguments. In the second line above, an "and" gate is added to gate 1 of Tree1, which will be a conjunction of the "1-3" failure channel and both "4, 5" channels.

Verifying the fault tree requires making a draft of the future fault tree and then occasionally check the correspondence of the tree with the draft. For graphic output of the tree the FaultTree package suggests using the ftree2html script, which obviously creates an image in the html format:

4. ftree2html(Tree1, write_file=TRUE),
5. browseURL("Tree1.html").

The first argument of the ftree2html sub-program is the identifier of the output fault tree; the second argument indicates that the file will be overwritten after each following run. The browseURL sub-program allows seeing the constructed tree using a browser, which by default is called up by the R environment. Figure 3 shows the constructed fault tree with two added structures.

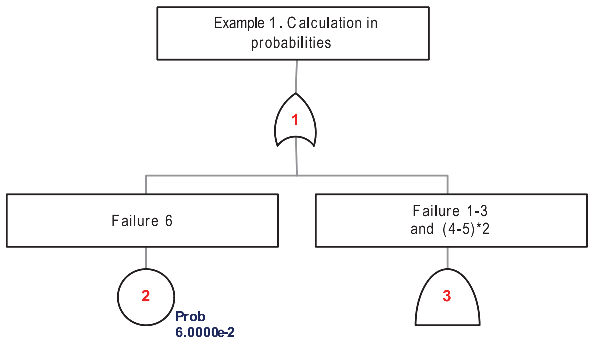


Figure 3. The second step of fault tree building

Note that the numbers 2 and 3 were automatically assigned to the simple event "failure of element 6" and the "and" gate respectively. During the permutation of script lines it is important to control the assigned numbers and remember that the permutation of lines can lead to a change of event numbers and possible occurrence of failure in the program code.

Next to the "failure of the element 6" event the numerical probability value of this event is shown. However, the other two gates are shown without the event probability.

Figure 4 shows the final version of the event tree with the calculated probabilities of each tree node.

The full version of the script for building such tree is shown below.

Example 1.
1. library(FaultTree)
2. Tree1 <- ftree.make(type="or", name="Example 1. Calculation", name2="on probabilities")
3. Tree1 <- addProbability(Tree1, at=1, prob=.06, name="Failure 6")
4. Tree1 <- addLogic(Tree1, at=1, type= "and", name=" Failure 1-3",name2="and (4-5)*2")
5. Tree1 <- addLogic(Tree1, at=3, type= "or", name=" Failure 1 и 2",name2="or 3")
6. Tree1 <- addProbability(Tree1, at=4, prob=.03, name=" Failure 3")

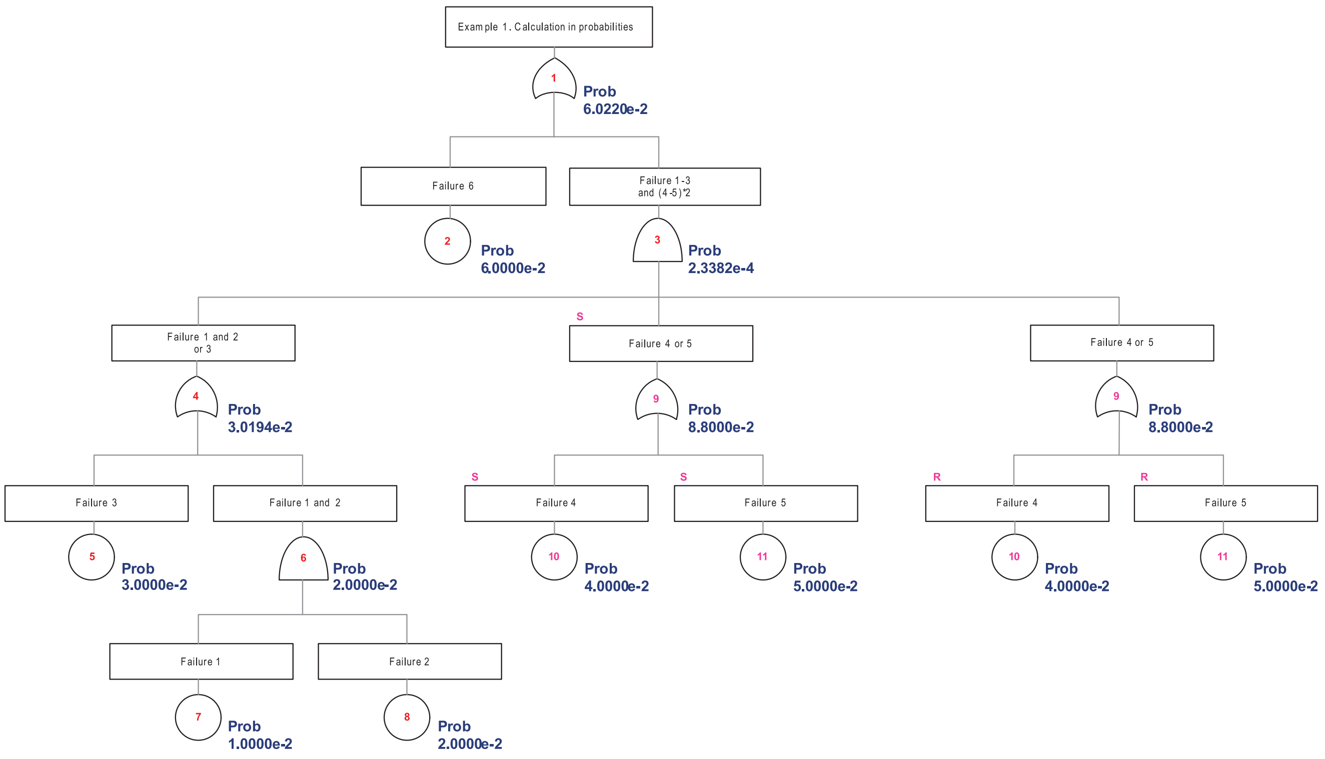


Figure 4. Final version of the fault tree with calculation in probabilities per nodes

7. Tree1 <- addLogic(Tree1, at=4, type= “and”, name=” Failure 1 and 2”)
8. Tree1 <- addProbability(Tree1, at=6, prob=.01, name=” Failure 1”)
9. Tree1 <- addProbability(Tree1, at=6, prob=.02, name=” Failure 2”)
10. Tree1 <- addLogic(Tree1, at=3, type= “or”, name=” Failure 4 or 5”)
11. Tree1 <- addProbability(Tree1, at=9, prob=.04, name=” Failure 4”)
12. Tree1 <- addProbability(Tree1, at=9, prob=.05, name=” Failure 5”)
13. Tree1 <- addDuplicate(Tree1, at=3, dup_id=9)
14. Tree1 <- ftree.calc(Tree1)
15. ftree2html(Tree1, write_file=TRUE)
16. browseURL(“Tree1.html”)

Let us pay attention to the 13th line. The FaultTree library has a very convenient function, which allows duplicating both the repeated event and whole data structure, i.e. event branches. As in our chain channel 4-5 was duplicated, the addDuplicate(Tree1, at=3, dup_id=9) structure duplication function was to be used. The first argument of this function, as before, is the identifier of the tree being built, at=3 is the number of the tree node to which the repeated data structure will be added, dup_id=9 is the upper node of the added data structure. Figure 4 shows letters S and R which mean branch-source (Source) and repeated branch (Repeat), accordingly. After the fault tree has been built, the sub-program ftree.calc(Tree1) is called up, which calculates the failure probability of each tree node according to the well-known probability calculation methods through transformation of logical dependencies.

Then let us consider the feasibility of identifying the failure probability of non-recoverable chain over time.

Calculation of the distribution function of the chain’s time to failure

An analysis of the previous script allows assuming that it is not necessary to specify the probabilities as fixed numbers, i.e. they can be made time dependent. To do that, it is necessary to recalculate the probability of each elementary event inside the time loop and calculate the fault tree inside the loop estimating the probability of chain failure.

In order to identify the failure probability dynamics (essentially, the distribution function of time to failure of the chain element), three different approaches may be used, i.e. parametric, non-parametric and combined.

In the first case time between failures of each chain element is defined by a parametrical law of distribution with a specific set of parameters. Let us note that different random values distribution laws are widely used in R language that is designed for “statistical data processing” [1-3].

The second approach consists in non-parametric estimation of the distribution function of time to failure of each chain element. If a researcher has statistical information, which contains full time to failure (the experiment resulted in the failure of each test sample), it suffices to estimate the empirical distribution function. For this purpose the R language has the ecdf() function. In case of censored information it is possible to use the Kaplan-Meier estimate, the script of which is also contained in the packages of the R language associated with survival analysis. It is possible to use kernel estimate of distribution function.

The third approach involves combined application of both the first (parametric) and the second (non-parametric) approaches, when the initial information is non-homogeneous, i.e. there are both statistical information and parametric estimate of the distribution law. In the dependability theory, the exponential law of distribution of time to failure has been widely used, and the failure rate is estimated based on the results of statistical tests. Methods of estimation are best developed for this distribution.

This article is not to question which is better, parameterization of distribution or non-parametric approach. We think that it is sufficient to use all information to the maximum extent without making unfounded conclusions regarding the distribution law.

Let us consider an example of finding the distribution function of time to failure for a structure. Let us first set forth the script and then analyze it.

Example 2.

```
1. library(FaultTree)
2. T=50
3. h=1
4. c1=”green4”
5. c2=”red2”
6. c3=”blue2”
7. t=seq(h,T,h)
8. n=length(t)
9. p1=pexp(t,0.01)
10. p2=pexp(t,0.02)
11. p3=pexp(t,0.03)
12. p4=pexp(t,0.04)
13. p5=pexp(t,0.05)
14. p6=pgamma(t,3,0.06)
15. p0=array(dim=n)
16. p7=array(dim=n)
17. TreePBF<-function(p1,p2,p3,p4,p5,p6){
18. Tree2 <- ftree.make(type=”or”, name=”Example 2. Calculation”, name2=”on probabilities”)
19. Tree2 <- addProbability(Tree2, at=1, prob=p6, name=”Failure 6”)
20. Tree2 <- addLogic(Tree2, at=1, type= “and”, name=” Failure 1-3”,name2=”and (4-5)*2”)
21. Tree2 <- addLogic(Tree2, at=3, type= “or”, name=” Failure 1 and 2”,name2=”or 3”)
22. Tree2 <- addProbability(Tree2, at=4, prob=p3, name=” Failure 3”)
23. Tree2 <- addLogic(Tree2, at=4, type= “and”, name=” Failure 1 и 2”)
24. Tree2 <- addProbability(Tree2, at=6, prob=p1, name=” Failure 1”)
25. Tree2 <- addProbability(Tree2, at=6, prob=p2, name=” Failure 2”)
26. Tree2 <- addLogic(Tree2, at=3, type= “or”, name=” Failure 4 or 5”)
27. Tree2 <- addProbability(Tree2, at=9, prob=p4, name=” Failure 4”)
28. Tree2 <- addProbability(Tree2, at=9, prob=p5, name=” Failure 5”)
29. Tree2 <- addDuplicate(Tree2, at=3, dup_id=9)
30. Tree2 <- ftree.calc(Tree2)
31. return(Tree2$PBF)
32. }
33. for (i in 1:n){
34. q=TreePBF(p1[i],p2[i],p3[i],p4[i],p5[i],p6[i])
35. p0[i]=q[1]
```

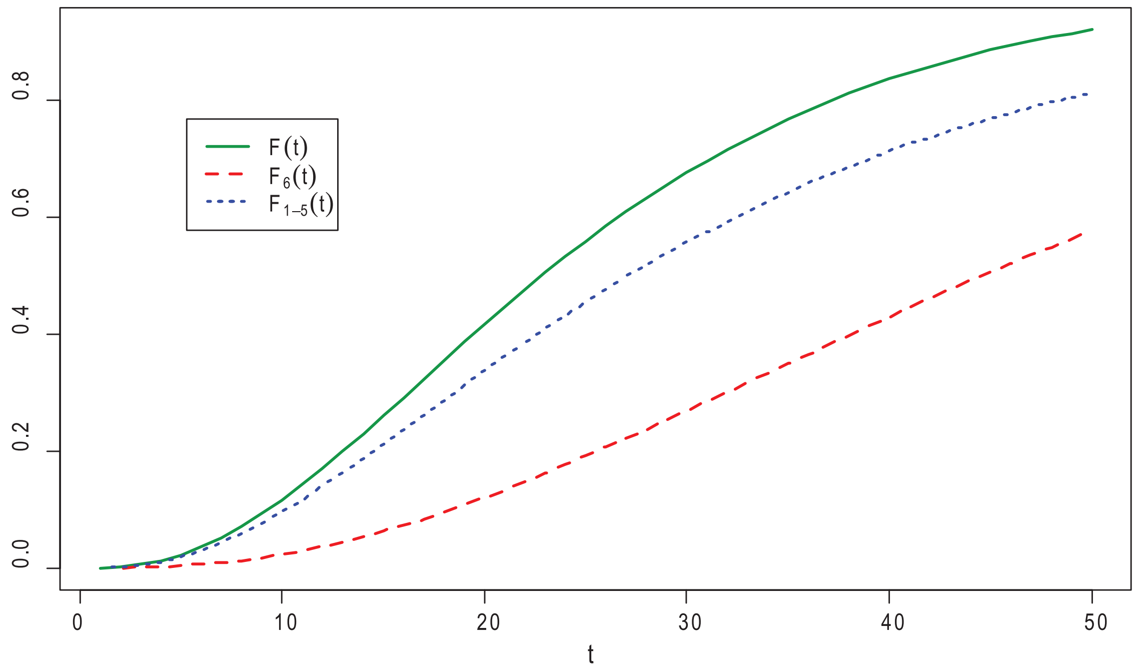


Figure 5. Distribution function of chain and its components

```
36. p7[i]=q[3]
37. }
38. plot(t,p0,type="l",lwd=2,col=c1,ylab="")
39. lines(t,p6,type="l",lwd=2,lty=2,col=c2)
40. lines(t,p7,type="l",lwd=2,lty=3,col=c3)
41. legend(locator(1), c(expression (F(t)), expression
(F[6](t)), expression (F[1-5](t))),
42. col = c(c1, c2, c3),
43. lty = c(1, 2, 3),
44. lwd = c(2, 2, 2))
```

The first line, as before, connects the FaultTree library. The maximum time T and the grid step h with which the distribution functions will be determined, are assigned on the time axis in the second and third lines. The colors of the displayed graphics are indicated in the 4-6 lines (Fig. 5). Note that there are several ways to specify a color in the R language and the color palette has a lot of colors. The quality of the applied vector graphics is undeniable, and it is possible to save pictures in different formats.

The numerical "temporal" array is created in the 7th line:

$$\vec{t} = (h, 2h, \ldots, T) = (1, 2, \ldots, 50).$$

The length of this array n=50 is determined in the 8th line.

The arrays of distribution functions of time to failure of each chain element (Fig. 1) are built in the 9th – 14th lines. So, time to failure of the first element has an exponential distribution with the rate λ=0.01. λ=rate=1/scale. The rate for the second element is 0.02, 0.03 for the third one, etc. The distribution function of time to failure of the 6th element corresponds to a gamma-distribution with the shape parameter 3 and rate λ=0.06.

Auxiliary arrays with the dimension n=50 are created in the 15th and 16th lines. The values of the distribution function of time to failure for the whole chain will be stored in array p0, and time to failure of the chain section 1-5 will be stored, for example, in array p7.

Lines 17 to 32 contain a function which computes the array of node probabilities of the fault tree. Six failure probabilities are the input parameters. Let us pay attention to the 31st line. The data structure of Tree2 contains many different factors, which becomes evident if we type the command Tree2 in the command prompt. So, the PBF factor contains the calculated failure probabilities of each tree node (Fig. 4). Thus, Tree2$PBF is an array of probabilities. Lines from 33 to 37 contain a simple loop, in which the fault tree will be calculated on a time grid with the step h=1. In this case, inside the loop, the failure probability of each element of the structure on the same time grid will be calculated. All calculated probabilities will be contained in the array q. Node 1 corresponds to the failure of the whole chain, node 3 corresponds to the failure the chain part 1-5, therefore q[1] and q[3] are the failure probabilities of the whole chain and its section 1-5 respectively. The graphical output of the results is in the 38th-44th lines (Fig. 5).

The advantage of the R language is the capability to build both parametric and non-parametric confidence intervals for the failure probability of the structure. In the parametric case it is necessary, as parameters of the law, to use such values of the confidence interval which provide the maximum and the minimal value of the failure probability. In the non-parametric case it is necessary to substitute the lower and upper bounds of the distribution function in accordance with the Clopper-Pearson method or their approximants, particularly, with the help of quintiles of the standard normal law. To determine the interval estimation of the failure probability of the structure, an insignificant script modification is required.

The next part of the article deals with the most important, in our opinion, situation related to the feasibility of calculation for the repairable as well as periodically maintainable equipment.

Calculation of the unavailability factor of the structure with repairable elements

Let us consider the following diagram (Fig. 6).

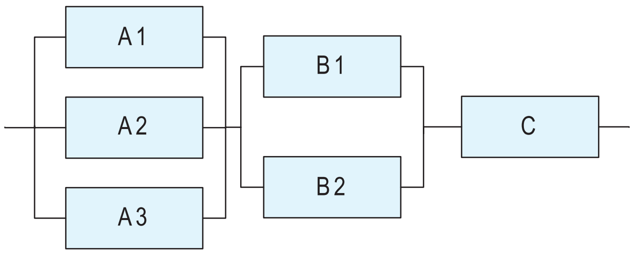


Figure 6. System diagram

Let us assume that the chain elements A1, A2, A3 are identical and operated under the same conditions; the same assumptions concern the elements B1, B2. All elements of the chain are repairable. The recovery time is T_r=8 h and does not depend on the number of repairable elements. In addition, let us suppose that the system has built-in diagnostic capability and the elements of the chain are under control. As a result, diagnostics can detect 90% of failures. Maintenance activities (preventive measures) are periodically carried out in a system with a period of τ=100 h, during which 10% of remaining "hidden" failures are detected. Let us assume that the mean time of the elements A is T_f(A)=1000 h, the mean time of elements B is T_f(B)=1500 h, the element C is T_f(C)=4000 h. Common cause failures (CCFs) are not taken into account.

To calculate an unavailability factor due to evident failures the addActive(DF, at, mttf, name) script is provided in the FaultTree library, where DF means the name of the fault tree, at is the tree node to which the event is attached, mttf is the mean time to failure, mttr is the mean time to recover, name is the name of the event in the failure tree.

In this case the non-asymptotic unavailability factor is used when calculating the failure probability [4-7]:

$$q = \frac{T_r}{T_r + T_f}, \quad (1)$$

where T_r=mttr is the mean time to recover, T_f=mttf is the mean time to failure.

To calculate an unavailability factor due to hidden failures the addLatent(DF, at, mttf, mttr, inspect, pzero, name) function is provided in the FaultTree library, in the arguments of which DF means the name of the fault tree, at is the tree node to which the event is attached, mttf is the mean time to failure, mttr is the mean time to recover, inspect is the control period, pzero is the optional argument equal to the probability that an element is restored at random times. As a default pzero=q means the asymptotic nonavailability factor (1). The name argument, as before, is the name of the event in the fault tree. In this case for calculating the failure probability the following approximate formula is used:

$$q = 1-(1-pzero)\cdot\left(1-K_{nonav}\right), \quad (2)$$

where, as a default, pzero= $\frac{T_r}{T_r + T_f}$ T_r = mttr is the mean recovery time, T_f=mttf is the mean time to failure, $K_{nonav} = 1-\frac{1-e^{-\lambda\tau}}{\lambda\tau} = 1+\frac{e^{-\lambda\tau}-1}{\lambda\tau}$ is the asymptotic nonavailability factor for a periodically controlled element with instant recovery [6].

The program script will be as follows:

Example 3.

```
1.  library(FaultTree)
2.  Tree3 <- ftree.make(type=»or», name=»Example 3»)
3.  Tree3 <- addLogic(Tree3, at=1, type= “or”, name= ”Failure C”)
4.  Tree3 <- addLatent(Tree3, at=2, mttf=4000/0.1, mttr=8, inspect=100, name=”hidden”)
5.  Tree3 <- addActive(Tree3, at=2, mttf=4000/0.9, mttr=8, name=”evident”)
6.  Tree3 <- addLogic(Tree3, at=1, type= “and”, name= ”Failure B1,B2”)
7.  Tree3 <- addLogic(Tree3, at=5, type= “or”, name= ”Failure B”)
8.  Tree3 <- addLatent(Tree3, at=6, mttf=1500/0.1, mttr=8, inspect=100, name=”hidden”)
9.  Tree3 <- addActive(Tree3, at=6, mttf=1500/0.9, mttr=8, name=”evident”)
10. #Tree3 <- addDuplicate(Tree3, at=5, dup_id=6)
11. Tree3 <- addLogic(Tree3, at=5, type= “or”, name= ”Failure B”)
12. Tree3 <- addLatent(Tree3, at=9, mttf=1500/0.1, mttr=8, inspect=100, name=”hidden”)
13. Tree3 <- addActive(Tree3, at=9, mttf=1500/0.9, mttr=8, name=”evident”)
14. Tree3 <- addLogic(Tree3, at=1, type= “and”, name= ”Failure A1,A2,A3”)
15. Tree3 <- addLogic(Tree3, at=12, type= “or”, name= ”Failure A”)
16. Tree3 <- addLatent(Tree3, at=13, mttf=1000/0.1, mttr=8,inspect=100, name=”hidden”)
17. Tree3 <- addActive(Tree3, at=13, mttf=1000/0.9, mttr=8, name=”evident”)
18. #Tree3 <- addDuplicate(Tree3, at=12, dup_id=13)
19. Tree3 <- addLogic(Tree3, at=12, type= “or”, name= ”Failure A”)
20. Tree3 <- addLatent(Tree3, at=16, mttf=1000/0.1, mttr=8,inspect=100, name=”hidden”)
21. Tree3 <- addActive(Tree3, at=16, mttf= 1000/0.9,mttr=8, name=”evident”)
22. #Tree3 <- addDuplicate(Tree3, at=12, dup_id=13)
23. Tree3 <- addLogic(Tree3, at=12, type= “or”, name= ”Failure A”)
24. Tree3 <- addLatent(Tree3, at=19, mttf=1000/0.1, mttr=8,inspect=100, name=”hidden”)
25. Tree3 <- addActive(Tree3, at=19, mttf=1000/0.9, mttr=8, name=”evident”)
26. Tree3 <- ftree.calc(Tree3)
27. Tree3_cs<-cutsets(Tree3)
28. ftree2html(Tree3, write_file=TRUE)
29. browseURL(“Tree3.html”)
```

Let us note that evident and hidden failures of the same element are connected by logical element “or”, while they should be connected with mutually exclusive element “or” – “xor”. Such logical element is not yet available in the current package version. However, the application of the simple “or” will lead to a minor failure in this situation.

Also note, that in the 4th line the addLatent function has the mttf=4000/0.1 argument, but in the 5th line the addActive function has mttf=4000/0.9 argument. This is due to the assumption that 90% of failures are evident failures and 10% are hidden failures, i.e.

$$\lambda = \lambda_{evid} + \lambda_{hidd}, \ \lambda_{evid} = 0,9\lambda, \ \lambda_{hidd} = 0,1\lambda.$$

In this case the mean time to failures of evident and hidden failures will be equal, accordingly:

$$T_{f,evid} = \frac{1}{\lambda_{evid}} = \frac{1}{0,9\lambda} = \frac{T_f}{0,9} \text{ and}$$

$$T_{f,hidd} = \frac{1}{\lambda_{hidd}} = \frac{1}{0,1\lambda} = \frac{T_f}{0,1}. \quad (3)$$

Lines 10, 18 and 22 are commented out. The 10th line duplicates lines from 11th to 13th, the 18th line duplicates lines from 19th to 21st, etc. In terms of script shortness, the variant with lines commented out is preferable. In line 27, the possibility of constructing minimal cut sets is first shown. The application of addDuplicate adds to the fault tree nodes with existing values, and in this case in the current package version cutsets sometimes yields incorrect cuts. Therefore, if it is intended to analyze the minimal cuts, it is better to duplicate events through repeated commands.

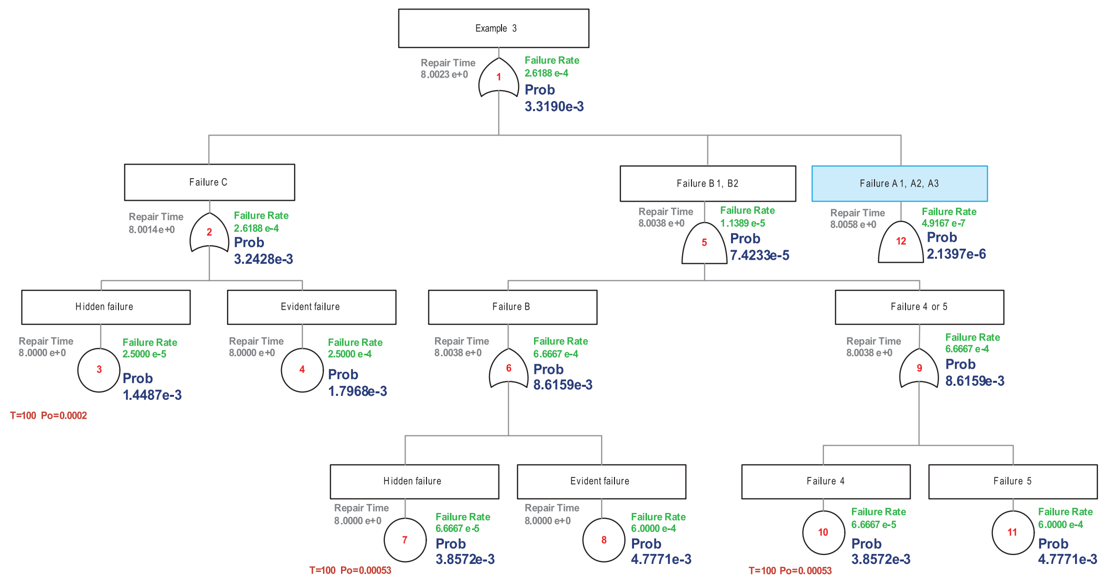


Figure 7. The fault tree for the example 3

Figure 7 shows the fault tree with "minimized" failures of the A1, A2, A3 elements due to large size of the fault tree. The created tree is interactive and allows minimizing branches, and in this case the complex event is highlighted in blue.

Note that now next to nodes, additional information has appeared along with the deduced failure probability.

Failure Rate is the rate which for such nodes as addLatent and addActive is calculated using the formula $\lambda_f = \dfrac{1}{T_f}$. For the logical node "or", corresponding to the series chain, the failure rate is the sum of failure rates of the descending node, i.e. $\lambda_{f-S} = \lambda_{f,1} + \lambda_{f,2} + \ldots$. For the logical element "or" the known approximate formula for the mean time between failures is used [11]:

$$T_{f-S} = \frac{K_{av}(p)}{\sum_{i=1}^{n} \lambda_{f-i} p_i \dfrac{\partial K_{av}(p)}{\partial p_i}}, \qquad (4)$$

from which follows $\lambda_{f-S} = \dfrac{1}{T_{f-S}} = \dfrac{\sum_{i=1}^{n} \lambda_{f-i} p_i \dfrac{\partial K_{av}(p)}{\partial p_i}}{K_{av}(p)}$,

where $K_{av}(p)$ is the quorum-function or availability factor of the structure, λ_{f-i}, p_i is the failure rate and the probability of no failure (PNF) of the ith component of the complex system which consists from n elements. So, for n=2 the following formula is used

$$\lambda_{f-S} = \frac{\lambda_{f-1} p_1 (1-p_2) + \lambda_{f-2} p_2 (1-p_1)}{p_1 + p_2 - p_1 p_2}.$$

Repair time is the time which is set by the user for the addLatent and addActive nodes. For the logical node "and" in R the repair time should be equal to the repair time of the first attached node. For the logical element "or" the same formula (4) for the mean repair time is used [11]:

$$T_{R-S} = \frac{1-K_{av}(p)}{\sum_{i=1}^{n} \lambda_{f-i} p_i \dfrac{\partial K_{av}(p)}{\partial p_i}} = \frac{1-K_{av}(p)}{K_{av}(p) \sum_{i=1}^{n} \lambda_{f-i}}. \qquad (5)$$

In this case the availability factor in the FaultTree is calculated by the following formula:

$$K_{av} = \prod_{i=1}^{n} \frac{T_{f-i}}{T_{f-i} + T_{R-i}}. \qquad (6)$$

In formula (6) hidden failures are not taken into consideration. The probability of the hidden failure is defined by formula (2).

Let us note that in Figure 7, next to the nodes corresponding to the hidden failures, the control period T=100 and the probability value are indicated, which is defined by the following formula: pzero=$P_0 = \dfrac{T_r}{T_r + T_f}$.

Thus, the value of the unavailability factor of the chain in the Example 3 is 3.319E-03, the failure rate is λ_{f-S} = 2.6188E-04 1/h, and the mean time between failures is $T_{f-S} = 1/\lambda_{f-s}$ =1/Tree$CFR=3818.535 h. To calculate the last value it is required to type the following command in the command prompt:

```
> 1/Tree3$CFR[1]
```

To define the minimal cuts in the command prompt it is required to type the following command:

```
> Tree3_cs
```

The minimal cut sets with one event, which lead to the chain failure, are given in the following matrix [[1]]:

$$\text{Matrix [[1]]: } \begin{pmatrix} 3 \\ 4 \end{pmatrix}$$

This matrix contains one column and two rows. The number of columns is equal to the number of nodes in the cut, i.e. in the unit, and the number of rows determines the number of minimal cuts with one event; there are two such cuts. Matrix elements are the number of nodes, which lead to the chain failure. In this case, these are nodes 3 and 4 – hidden and evident failures of the C element.

Minimal cut sets with two events are given in matrix [[2]]. This matrix contains two columns and four rows. Minimal cut sets form a combination of events 7,10 is the hidden failure B1 and hidden failure B2, and 7,11 is the hidden failure B1 and evident failure B2, etc.

$$\text{Matrix [[2]]: } \begin{pmatrix} 7 & 10 \\ 7 & 11 \\ 8 & 10 \\ 8 & 11 \end{pmatrix}$$

In matrix [[3]] different combinations of three events are given, which lead to the chain failure; obviously, these events are associated with elements A1, A2, A3.

$$\text{Matrix [[3]]: } \begin{pmatrix} 14 & 17 & 20 \\ 14 & 17 & 21 \\ 14 & 18 & 20 \\ 14 & 18 & 21 \\ 15 & 17 & 20 \\ 15 & 17 & 21 \\ 15 & 18 & 20 \\ 15 & 18 & 21 \end{pmatrix}$$

In specialized packages such as Saphire, Arbitr etc., along with the minimal cut sest, the probabilities of these cut sets and the impact on the general failure probability of the system are displayed. There is no such capability in the FaultTree package, but it can be done with the help of the R language tools. For example, using the following script:

```
m=length(Tree3_cs)
pr=Tree3_cs
for (i in 1:m){
n1=length(Tree3_cs[[i]][,1])
n2=length(Tree3_cs[[i]][1,])
for (j1 in 1:n1){
q1=Tree3_cs[[i]][j1,1]
prob=Tree3$PBF[q1]
if (n2>1) {
for (j2 in 2:n2){
q2=Tree3_cs[[i]][j1,j2]
prob=prob*Tree3$PBF[q2]
pr[[i]][j1,j2]=NA}}
pr[[i]][j1,1]=prob}}
```

If you type pr in the command prompt after executing script, the following information will be displayed:

```
> pr
[[1]]
[,1]
[1,] 0.001448669
[2,] 0.001796766

[[2]]
[,1] [,2]
[1,] 1.487810e-05 NA
[2,] 1.842618e-05 NA
[3,] 1.842618e-05 NA
[4,] 2.282040e-05 NA

[[3]]
[,1] [,2] [,3]
[1,] 1.929755e-07 NA NA
[2,] 2.387178e-07 NA NA
[3,] 2.387178e-07 NA NA
[4,] 2.953028e-07 NA NA
[5,] 2.387178e-07 NA NA
[6,] 2.953028e-07 NA NA
[7,] 2.953028e-07 NA NA
[8,] 3.653006e-07 NA NA
```

The first column shows the probability of the corresponding cut set. Additional columns have no important information. Analyzing the results, it may be concluded that two events negatively affect the system's dependability, i.e. evident and hidden failures of the C element.

In conclusion it should be noted that addLatent, addActive functions and others, which calculate the probability in accordance with some dependability model, can be uniformly replaced by addProbability with the failure probability calculated in advance.

Conclusion

This article demonstrates the capabilities of the actively developing R language for statistical data processing and its FaultTree package related to the construction and analysis of the fault trees. Fault trees are used to analyze the reliability of complex systems. Three examples are examined and analyzed in detail in this article. First, FTA is calculated with known failure probabilities of elements. Second, the dynamic fault tree is analyzed, i.e. the distribution function of time to failure of chain is identified. In the last example, FTA of the chain with repairable elements are examined.

References

1. Crawley MJ. The R Book. 2nd ed. Wiley Publishing; 2012.

2. Shipunov AB, Baldin EM, Volkov PA, Korobeynikov AI, Nazarov SA, Petrov SV et al. Nagliadnaya statistika. Ispolzuem R! [Illustrative statistics. Using R!]. Moscow: DMK Press; 2012 [in Russian].

3. Kabakov RI. R v deystvii. Analiz i vizualizatsiya dannykh v programme R [R in action. Analysis and visualization of data in R]. Moscow: DMK Press; 2014 [in Russian].

4. Antonov AV, Nikulin MS, Nikulin AM, Chepurko VA. Teoria nadiozhnosti. Statisticheskie modeli; Ouchebnoie posobie [Dependability theory. Statistical models: A study guide. Moscow: INFRA-M; 2015 [in Russian].

5. Gnedenko BV, Beliaev YuK, Soloviev AD. Matematicheskie metody v teorii nadiozhnosti [Mathematical methods in the dependability theory]. Moscow: Nauka; 1965 [in Russian].

6. Beliaev YuK, Bogatyrev VA, Bolotin VV et al. Ushakov IA, editor. Nadiozhnost tekhnicheskikh system: Spravochnik [Dependability of technical systems: Reference book]. Moscow: Radio i sviaz; 1985 [in Russian].

7. Antonov AV, Nikulin MS. Statisticheskie modeli v teorii nadiozhnosti: Ouchebnoie posobie [Statistical models in the dependability theory: A study guide]. Moscow: Abris; 2012 [in Russian].

8. <https://ru.wikipedia.org/wiki/R_(язык_программирования)>

9. <http://www.openreliability.org/fault-tree-analysis-on-r/>

10. Mosleh A et al. Procedures Guidelines in Modeling Common Cause Failures in Probabilistic Risk Assessment (NUREG/CR-5485); 1998.

11. Programmnyy kompleks avtomatizirovannogo strukturno-logicheskogo modelirovaniya i rascheta nadezhnosti i bezopasnosti ASUTP na stadii proektirovaniya (PK ASM SZMA) [Software system for automated structural and logical modeling and dependability and safety calculation of ACS at the design stage (PK ASM SZMA)]. Technical documentation. Saint-Petersburg: OAO SPIK SZMA; 2003 [in Russian].

12. Smith CL, Wood ST, Galyean WJ, Schroeder JA, Sattison MB. Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE). Version 8. Vol. 2, NUREG/CR-7039 INL/EXT-09-17009; June 2011.

About the authors

Alexander V. Antonov, Doctor of Engineering, Professor, Head of Division for Justifying Calculations of Design Solutions, JSC RZSU, Russia, Moscow, e-mail: AlVlaAntonov@rasu.ru

Evgeny Yu. Galivets, Deputy Director of Department, Head of Design Division, JSC RASU, Russia, Moscow, e-mail: EYGalivets@rasu.ru

Valery A. Chepurko, Candidate of Physics and Mathematics, Associate Professor, Chief Specialist of Division for Justifying Calculations of Design Solutions, JSC RZSU, Russia, Moscow, e-mail: VAChepurko@rasu.ru

Alexey N. Cherniaev, Candidate of Engineering, Deputy Technical Director, Director of Design Department, JSC RASU, Russia, Moscow, e-mail: AlNChernyaev@rasu.ru

Received on 04.08.2017