Package 'crackR'

February 19, 2015

Type Package

Title Probabilistic damage tolerance analysis for fatigue cracking of metallic aerospace structures

Version 0.3-9

Date 2014-04-21

Author Keith Halbert

Maintainer Keith Halbert <keith.a.halbert@gmail.com>

Description Using a sampling-based approach (either sequential importance sampling or explicit Monte Carlo), this package can be used to perform a probabilistic damage tolerance for aircraft structures. It can model a single crack, or two simultaneously growing fatigue cracks (the so-called continuing damage problem). With a single crack, multiple types of future repairs are possible.

License GPL-3

Depends Hmisc (>= 3.14-3), evd (>= 2.3-0)

NeedsCompilation no

Repository CRAN

Date/Publication 2014-04-22 07:23:26

R topics documented:

crackR-package *Probabilistic damage tolerance analysis for fatigue cracking of metallic aerospace structures*

Description

Using a sampling-based approach (either sequential importance sampling explicit Monte Carlo), this package can be used to perform a probabilistic damage tolerance for aircraft structures. It can model a single crack, or two simultaneously growing fatigue cracks (the so-called continuing damage problem). With a single crack, multiple types of future repairs are possible.

Details

Package:	crackR
Type:	Package
Version:	0.3-9
Date:	2014-04-21
License:	GPL-3

To run a crackR analysis, first create an object of class crackR. This is accomplished by first creating an object of class crackRparameters. This can be done by modifying a supplied data set, such as with data(cp7). Use crackRinit() on the crackRparameters object to generate the crackR object. To perform the analysis, use analyze() or crackRmc().

Author(s)

Keith Halbert
Maintainer: Keith Halbert <keith.a.halbert@gmail.com>

References

MIL-STD-1530C (USAF) "General Guidelines For Aircraft Structural Integrity Program (ASIP)", Nov 2005
MIL-HDBK-1823A (Department of Defense, USA) "Nondestructive Evaluation System Reliability Assessment", Apr 2009
PRoF Ver. 3.0 PRobability of Fracture, Software Package, University of Dayton Research Institute, Dayton, OH, 2005
Halbert, K. "Estimation of Probability of Failure for Damage-Tolerant Aerospace Structures" PhD Thesis, Temple University Department of Statistics, Philadelphia, PA, Apr 2014

See Also

crackRinit
analyze
crackRmc

Examples

```
set.seed(327)
## cp7ext data is part of this package
data(cp7ext)
## prepare a crackR object for analysis
cp7ext.init <- crackRinit(cp7ext)
## perform sequential importance sampling analysis
cp7ext.out  <- analyze(cp7ext.init)
## Plot SFPOF and PCD results
plot(cp7ext.out)
```

`analyze` *Perform a probabilistic damage tolerance analysis on a crackR object*

Description

This is the main analysis function for the sequential importance sampling approach to probabilistic damage tolerance analysis in the crackR package. User can enter an object of class crackRparameters or class crackR. If a crackRparameters object is entered, crackRinit will be run to initialize a crackR object. The sample size for an existing crackR object can be increased by utilizing an existing crackR object and specifying add=TRUE.

Usage

```
analyze(obj,add=FALSE)
```

Arguments

`obj`	Object of class Sing, Mult, CD, or an appropriate list of input parameters
`add`	Logical, indicating whether this run will add trials to an existing run

Details

If add=TRUE, the number of particles specified in the input object parameter Np will be added to the existing particle count and the results will be weighted according to the existing particle count. The new set of results, containing only the results of the subset run, will be included in a new component of the output: 'new.results'. The cumulative results will as before be contained in component 'results'. Note, if the input object was already initialized using crackRinit, it's class will be either Sing, Mult, or CD. If an uninitialized list of crackR parameters is submitted, this function will attempt to initialize it.

Value

`state`	List containing vectors representing the final state of each particle
`parameters`	List of the analysis parameters (not altered by the running of analyze
`results`	List of the results of a single run (where add=FALSE) or of a cumulative set of runs (when add=TRUE)
`new.results`	List of the results for the latest run in a cumulative set of runs (when add=TRUE)

Author(s)

Keith Halbert <keith.a.halbert@gmail.com>

See Also

crackRinit

Examples

```
set.seed(327)

## cp7ext data is part of this package
data(cp7ext)

## prepare a crackR object for analysis
cp7ext.init <- crackRinit(cp7ext)

## perform analysis
cp7ext.out  <- analyze(cp7ext.init)

## Plot SFPOF and PCD results
plot(cp7ext.out)
```

---

analyzeParallel — *Perform multiple parallel runs of a probabilistic damage tolerance analysis on a crackR object*

Description

One powerful method for assessing convergence of a sequential importance sampling analysis is to run parallel sequences. This function does this, saving only the results (of class crackRresults). Results are returned in a list. Note parallelization is not performed, the analyses are run sequentially...thus this is a convenience function and a placeholder for the time being.

Usage

```
analyzeParallel(obj,n.parallel=3)
```

Arguments

obj	Object of class Sing, Mult, or CD
n.parallel	Number of parallel sequences to run

Value

List of crackRresults components

Author(s)

Keith Halbert <keith.a.halbert@gmail.com>

See Also

analyze

Examples

```
set.seed(327)
## cp7ext data is part of this package
data(cp7ext)
## prepare a crackR object for analysis
cp7ext.init <- crackRinit(cp7ext)
## perform 2 parallel runs
cp7ext.out.2  <- analyzeParallel(cp7ext.init, n.parallel=2)
## Plot SFPOF results
plot(cp7ext.out.2)
```

---

calcInterval *Advance crackR object to the next scheduled inspection*

Description

Advance in time through a single inspection interval, adjusting the particle weights and reporting failure probabilities along the way. Estimates of Single Flight Probability Of Failure (SFPOF) will be calculated at intervals specified in the parameter flt.calc.interval, along with the bootstrap estimates of quantiles of SFPOF (if bootstrap.sfpof is TRUE) and an estimate of the probability of failure for each subinterval.

Usage

```
calcInterval(obj, interval.flights)
## S3 method for class 'Sing'
calcInterval(obj,interval.flights)
## S3 method for class 'Mult'
calcInterval(obj,interval.flights)
## S3 method for class 'CD'
calcInterval(obj,interval.flights)
```

Arguments

obj	Object of class Sing, Mult, or CD (each of which is an initialized crackR object created with crackRinit)
interval.flights	Integer, number of flights in the interval

Details

When the parameter survival.updating="fbf", the analysis proceeds through the interval flight-by-flight (which takes longer to run and is slower to converge). With survival.updating="int", an approximate analysis is performed which proceeds in INTervals of several flights-at-a-time (the number of flights for each subinterval being determined by parameter flt.calc.interval). While approximate, the "int" version will often converge on a solution more quickly, particularly if failure due to reaching the so-called "critical crack length" is likely. The SFPOF (and various other) results are output at intervals of flt.calc.interval regardless of the value of survival.updating.

Value

Object of class crackR, advanced in time to the next scheduled inspection and including SFPOF (and various other) results of the interval

Author(s)

Keith Halbert <keith.a.halbert@gmail.com>

References

Halbert, K. "Estimation of Probability of Failure for Damage-Tolerant Aerospace Structures" PhD Thesis, Temple University Department of Statistics, Philadelphia, PA, Apr 2014

See Also

analyze

Examples

```
data(cp7ext)
cp7ext$Np <- 200
cp7ext.init <- crackRinit(cp7ext)
cp7ext.next <- calcInterval(cp7ext.init, interval.flights=1000)
## crack lengths have grown
head(cp7ext.init$state)
head(cp7ext.next$state)
## SFPOF estimates for the interval are now present
cp7ext.next$results$sfpof
```

---

calcSfpofFromPofInt *Calculate SFPOF results from interval failure probabilities*

Description

For the flight-by-flight sequential importance sampling routine of the crackR package, the default is to output SFPOF at the start and end of a number of subintervals of flights. Also output in the results object is the probability of failure of each interval: `pof.int`. This function calculates and "average" value of SFPOF for each flight in each subinterval using `pof.int`, yielding SFPOF results in the familiar format for plotting.

Usage

```
calcSfpofFromPofInt(pof.int)
```

Arguments

pof.int	data.frame of the subintervals of a crackR run with columns indicating the number of flights in each interval and the failure probability of each interval. pof.int is standard output in a crackR run.

Details

Like the interval version of the routine, SFPOF will converge more quickly for many problems when averaging estimates over a number of flights. Using this function to process flight-by-flight results will be approximate within each subinterval, but these will be less approximate in general than those obtained by the interval routine.

Value

data.frame of SFPOF results for printing or plotting

Author(s)

Keith Halbert <keith.a.halbert@gmail.com>

References

Halbert, K. "Estimation of Probability of Failure for Damage-Tolerant Aerospace Structures" PhD Thesis, Temple University Department of Statistics, Philadelphia, PA, Apr 2014

See Also

analyze
plot.crackRresults

---

cp4 *Examples CP4, CP6, and CP7, adapted from PRobability Of Fracture (PROF) v3.0 documentation (in various forms).*

Description

Examples CP4, CP6 and CP7 are adapted from the documentation of the software PRobability Of Fracture (PROF) v3.0. The data has been modifed from the documentation in the following ways: crack growth is indexed to number of flights instead of flight hours; probability of inspection is set to 1.0; and the PROF parameter 'number of similar locations' is not part of this routine (effectively always set equal to 1). For example CP7ext, the crack growth and geometry curves have been extrapolated to a larger final crack size. The objects cp4, cp6, cp7, and cp7ext are of analysis type 'single'; i.e., there is a single damage tolerance analysis (DTA) associated with each.

cp7ext.Mult.simple and cp7ext.Mult.complex are of the 'multiple' type; i.e., there are two or three DTAs associated, respectively. Finally, cp7.CD is of the continuing damage variety. Any one of these can be used as a starting point for creating a new probabilistic damage tolerance analysis problem, though it is recommended to use the appropriate type (single, multiple, or continuing damage) due to differences regarding the structure of each list. All parameters are discussed in detail below along with an indication of whether they apply to Sequential Importance Sampling (SIS) or Explicit Monte Carlo (EMC). This information is repeated from `crackRinit`.

Usage

```
data(cp4)
data(cp6)
data(cp7)
data(cp7ext)
data(cp7ext.Mult.simple)
data(cp7ext.Mult.complex)
data(cp7.CD)
```

Format

List (class Sing, Mult, or CD).

Details

- `Np` – integer – SIS/EMC – number of Monte Carlo samples to utilize
  - For SIS (e.g., using `analyze`) this is the number of particles in the set. In this case the complete set of particles is carried through the analysis, so `Np` > several million may use up available memory. If many millions of particles are needed, the runs can be split and re-combined later either by using `analyzeParallel` or by using the `add=TRUE` option in `analyze`.

    For EMC run (using `crackRmc`) this is the number of trials. Detailed results of each trial are not maintained, thus many millions is fine from a memory standpoint. `crackRmc` is NOT fast, so testing a run with `Np` around 1000 is recommended (see `system.time`).

- `analysis.type` – character – SIS/EMC – the type of probabilistic damage tolerance analysis being performed, be it a single-type at a single crack ('single'), multiple-type at a single crack ('multiple'), or a continuing damage analysis involving two cracks ('CD')
  - Must be either 'single', 'multiple', or 'CD'. For an SIS run, this determines the class of the object created by `crackRinit` (either `Sing`, `Mult`, or `CD`). For an EMC run, the appropriate method of `crackRmc` is selected according to this parameter.

- `survival.updating` – character – SIS – whether to proceed flight-by-flight ('fbf') or in intervals ('int')
  - Must be either 'fbf' or 'int' for an SIS run (specifically, in `calcInterval`). This parameter is not used in an EMC run (`crackRmc`).

- `flt.calc.interval` – integer – SIS/EMC – frequency of output of SFPOF calculations in number of flights for each sub-interval

  - Regardless of the value of survival.updating, SFPOF is calculated and output at the beginning and end of sub-intervals of length flt.calc.interval. The actual sub-interval lengths will often vary slightly since sub-intervals are forced to end at a scheduled inspection. The sub-intervals within an inspection interval are rounded to integers which minimize the variation from flt.calc.interval. A value of 1 may be used, but for long service lives the results object may become large and calculation speeds may suffer.

- sfpof.threshold – numeric – SIS/EMC – if specified, a horizontal line will appear on results plots at this value.
  - This parameter is not currently used for anything important.

- sfpof.min – numeric – SIS/EMC – minimum value of SFPOF to include in results
  - If > 0, SFPOF calculated below sfpof.min will be recorded as sfpof.min. If = 0, SFPOF will be allowed to be zero in results. In this case, when plotting SFPOF on the log scale, warnings will be issued and the plot will not appear at the zero values.

- bootstrap.sfpof – logical – SIS – whether or not to perform obtain bootstrap confidence intervals of SFPOF during a run
  - If bootstrap confidence intervals are desired, set this parameter to TRUE. The number of bootstrap samples to generate at each calculation of SFPOF is set by bootstrap.samples and the desired quantiles to estimate by bootstrap.quantiles. Results will be in additional column(s) with SFPOF in the SFPOF data.frame of the crackRresults object. Note that there are many SFPOF calculations and using a large number of bootstrap samples may significantly increase the run time.

- bootstrap.pcd – logical – SIS – whether or not to perform obtain bootstrap confidence intervals of PCD during a scheduled inspection
  - If bootstrap confidence intervals are desired, set this parameter to TRUE. The number of bootstrap samples to generate at each calculation of PCD is set by bootstrap.samples and the desired quantiles to estimate by bootstrap.quantiles. Results will be in additional column(s) with PCD in the PCD data.frame of the crackRresults object. Inspections occur less frequently than SFPOF calculations, but large values of bootstrap.samples may significantly increase the run time.

- bootstrap.samples – integer – SIS – number of bootstrap samples to take for either SFPOF or PCD
  - More samples = better bootstrap estimates of the quantiles of the sampling distribution(s), with more samples taking longer to run. Note the bootstrap run time is also a function of the number of particles (Np).

- bootstrap.quantiles – numeric – SIS – quantile(s) at which to estimate the sampling distribution of either SFPOF or PCD
  - this is simply the probs argument to the quantile function.

- ms.gumbel – numeric – SIS/EMC – vector of length two specifying the location (loc) and scale parameters for the Gumbel distribution which describes the maximum applied stress per flight
  - For example: `cp4$ms.gumbel <- c(loc=31.079,  scale=0.832)`

  - The names "loc" and "scale" are required!
  - crackR is currently hard-coded to utilize the Gumbel distribution for max stress per flight. Expanding to a generic distribution is not difficult; it just hasn't been done yet. Mostly this is because max stress per flight is almost always Gumbel distributed in the literature, so I haven't bothered.

- inspections – data.frame – SIS/EMC – specification of inspection intervals (in number of flights) and the inspection type to occur at the end of the interval
  - For example: `cp4$inspections <- data.frame(flt.interval = c(4615,2308,2308), type = c(1,2,1))`
  - The names "flt.interval" and "type" are required!
  - The above example specifies three inspection intervals over the `4615+2308+2308=9231` flights. The three inspections are of types 1, 2, and 1, respectively, which refers to parameter `pod.func`.

- pod.func – list – SIS/EMC – list of POD functions
  - For example: `cp7ext$pod.func <- list(); cp7ext$pod.func[[1]] <- function(a) lognormalPOD(a, 0.03, 1); cp7ext$pod.func[[2]] <- function(a) lognormalPOD(a, 0.01, 1)`
  - The above example specifies two POD curves. At inspection time the parameter `inspections` specifies which of these to use. If the inspection type calls for a POD curve that is not specified here, errors will occur. See `lognormalPOD` and `inspection`.

- pod.threshold – numeric – SIS/EMC – either a matrix or a vector specifying the crack sizes at which different severities of repair will occur (see below for details)
  - For a run of `analysis.type=="single"` or `analysis.type=="CD"`, this is a weakly increasing vector of thresholds at which to partition the PCD results. For example, if = 0, then no partitioning will occur; if = c(0.01, 0.05), then PCD will be split into three: a < 0.01, 0.01 < a < 0.05, and a > 0.05.
  - For `analysis.type=="multiple"`, there is more than one type of repair that is possible and the behavior of each is different. Because of this, `pod.threshold` is an upper triangular matrix of dimension `c(dta.types, dta.types)`, where each row is the POD threshold vector for that type. Note each row must be weakly increasing from left to right. It is upper triangular since it is assumed that a repair can only get more severe...unless the part is replaced because the crack grew to exceed the largest threshold value. This is admittedly confusing, so here's an example. Suppose there are two DTA types describing the possible condition at a single fastener hole, where the first represents the as-manufactured condition and the second represents an oversized fastener installation. Suppose `pod.threshold==matrix(c(0.005,NA,0.25,0.005), nrow=2, ncol=2)`. For a type 1 particle, the POD threshold values used are taken from the first row of this matrix: c(0.005, 0.25). If a crack is found smaller than 0.005, it can be repaired to as-manufactured condition. If 0.005 < a < 0.25, the oversized fastener is installed and the type switches to type 2 as a result of the inspection being performed. If a > 0.25, the part is replaced (back to type 1). If instead when heading into inspection the repair was of type 2, then `pod.threshold==0.005`. If the found crack is smaller than 0.005, it can be repaired back to the oversized fastener condition (but not back to type 1, a smaller fastener). If a > 0.005, the part must be replaced (since the oversized fastener repair has already been utilized and no other repair is possible). Note that zero and Inf values may be used. Also the NA values can optionally be zero as they are not used by the code.

- dta.types – integer – SIS/EMC – number of Damage Tolerance Analysis (DTA) data sets present in the analysis
  - This is used only for analysis.type=="multiple" in the methods corresponding to a crackR object of class Mult.
- ismc.bool – logical – EMC – whether or not to utilize importance sampling to set the initial state in an EMC run
  - If FALSE standard Monte Carlo sampling is used and the initial crack size will be generated using, for example, ifs.ractual.
  - If TRUE importance sampling is used and the initial size will be generated using ifs.rsamp instead. Subsequently ifs.dsamp and ifs.dactual are used to determine the importance weights.
  - See parameter dta for more information regarding specification of initial flaw size distributions
- cg.cc – numeric – SIS/EMC – critical crack length for a run of either analysis.type=="single" or analysis.type=="multiple"
  - The crack length at which failure is assumed to occur with certainty. Note that if cg.cc is larger than the largest values in the DTA data item cg, linear extrapolation will occur until reaching cg.cc. If cg.cc is smaller than the largest values of said table, failure will occur earlier.
  - Generally cg.cc should be set to the last value of crack length in the DTA data item cg.
- cg.cc.pc/cg.cc.ph/cg.cc.sc/cg.cc.sh – numeric – SIS/EMC – critical crack length for a run of analysis.type=="CD"
  - For a continuing damage analysis, there are two cracks involved (primary and secondary), each of which can be in either a 'cold' state or a 'hot' state, depending on whether the other crack has previously reached critical size. Thus there are four sets of DTA data: primary cold (dta.pc), primary hot (dta.ph), secondary cold (dta.sc), and secondary hot (dta.sh). See below for more details.
  - The various critical crack sizes associated with a continuing damage analysis are as follows: cg.cc.pc, the length at which the primary crack goes critical (secondary goes hot); cg.cc.sc, the length at which the secondary crack goes critical (primary goes hot); cg.cc.ph, the length at which a primary hot crack fails; and cg.cc.sh, the length at which a secondary hot crack fails.
- dta – list – SIS/EMC – deterministic damage tolerance analysis input data for analysis.type=="single" or analysis.type=="multiple". There are several components to this list, each discussed in turn below. Note that for analysis.type=="single", this is a list of the following parameters. For analysis.type=="multiple", dta is a "list of lists", where each item of the list (e.g., dta[[1]]) is a complete dta listing representing that type of part condition.
  - cg – data.frame – crack growth data with columns flight and crack; note this differs from some other probabilistic damage tolerance analysis codes which are indexed to flight hours instead of flight numbers. Linear extrapolation will occur using the last two rows of this table if cg.cc is larger than the last crack size in the table.
  - geo – data.frame – geometry data relating crack length to normalized stress intensity K/sigma with columns crack and ksig. The data in this table is not extrapolated; if a K/sigma value is needed for a crack size beyone the end of this table, the final value of K/sigma will be used.

  - kc.rsamp – function – generates values from fracture toughness distribution
  - ifs.rsamp – function – generates values from initial flaw size sampling distribution
  - ifs.dsamp – function – density function for initial flaw size sampling distribution
  - ifs.ractual – function – generates values from initial flaw size actual distribution
  - ifs.dactual – function – density function for initial flaw size actual distribution
  - rfs.rsamp – function – generates values from repair flaw size sampling distribution
  - rfs.dsamp – function – density function for repair flaw size sampling distribution
  - rfs.ractual – function – generates values from repair flaw size actual distribution
  - rfs.dactual – function – density function for repair flaw size actual distribution

- dta.pc/dta.ph/dta.sc/dta.sh – list – SIS/EMC – deterministic damage tolerance analysis input data for a continuing damage run
  - For a continuing damage run there are four distinct sets of DTA data, primary cold (dta.pc), primary hot (dta.ph), secondary cold (dta.sc), and secondary hot (dta.sh). The cracks become hot at cg.cc.pc or cg.cc.sc, and fail at cg.cc.ph or cg.cc.sh.
  - At manufacture and following a repair, cracks are always 'cold'. Hence dta.pc and dta.sc need to have the various distributions for fracture toughness and initial flaw size, but dta.ph and dta.sh don't require them.

Source

PRoF Ver. 3.0 PRobability of Fracture, Software Package, University of Dayton Research Institute, Dayton, OH, 2005

References

Halbert, K. "Estimation of Probability of Failure for Damage-Tolerant Aerospace Structures" PhD Thesis, Temple University Department of Statistics, Philadelphia, PA, Apr 2014

Examples

```
data(cp7)
head(cp7)
```

---

crackRcalcSfpofMc — *Calculates SFPOF by post-processing an explicit Monte Carlo analysis as performed by crackRmc.*

Description

crackRmc will calculate SFPOF on the intervals specified for that function, along with the raw results of the simulation. If a different set of flight intervals for SFPOF is desired, this function can be used to do so without re-running the simulation.

Usage

```
crackRcalcSfpofMc(fail.count,insp.intervals,calc.interval,sfpof.min=1e-16)
```

Arguments

`fail.count`	Vector of failure counts originating from a crackRmc run
`insp.intervals`	Vector of the inspection intervals which were used in the run (not cumulative); the subintervals for pooling SFPOF results will avoid overlapping an inspection flight
`calc.interval`	Length of the interval in flights to use to pool SFPOF estimates
`sfpof.min`	SFPOF estimates below this value will be raised to this value; this may be zero, though the estimates will not plot on the log-scale in that case

Value

Returns an object of class crackRresults

Author(s)

Keith Halbert <keith.a.halbert@gmail.com>

References

Halbert, K. "Estimation of Probability of Failure for Damage-Tolerant Aerospace Structures" PhD Thesis, Temple University Department of Statistics, Philadelphia, PA, Apr 2014

See Also

`crackRmc`

`crackRinit` *Initializes a crackR object (of class Sing, Mult, or CD) from an appropriate list of parameters*

Description

Creates an object of class Sing, Mult, or CD from an appropriate list of parameters (using parameter analysis.type to determine the output class). The list of parameters may or may not be of class crackRparameters (the function will attempt to do its thing regardless). User calling of this function is optional since an object of crackRparameters (or simply an appropriate list) may be directly passed to analyze, which will run crackRinit if this was not previously done.

Usage

```
crackRinit(parameters)
```

Arguments

`parameters`	Object of class crackRparameters

Details

- Np – integer – SIS/EMC – number of Monte Carlo samples to utilize
  - For SIS (e.g., using analyze) this is the number of particles in the set. In this case the complete set of particles is carried through the analysis, so Np > several million may use up available memory. If many millions of particles are needed, the runs can be split and re-combined later either by using analyzeParallel or by using the add=TRUE option in analyze.

    For EMC run (using crackRmc) this is the number of trials. Detailed results of each trial are not maintained, thus many millions is fine from a memory standpoint. crackRmc is NOT fast, so testing a run with Np around 1000 is recommended (see system.time).

- analysis.type – character – SIS/EMC – the type of probabilistic damage tolerance analysis being performed, be it a single-type at a single crack ('single'), multiple-type at a single crack ('multiple'), or a continuing damage analysis involving two cracks ('CD')
  - Must be either 'single', 'multiple', or 'CD'. For an SIS run, this determines the class of the object created by crackRinit (either Sing, Mult, or CD). For an EMC run, the appropriate method of crackRmc is selected according to this parameter.

- survival.updating – character – SIS – whether to proceed flight-by-flight ('fbf') or in intervals ('int')
  - Must be either 'fbf' or 'int' for an SIS run (specifically, in calcInterval). This parameter is not used in an EMC run (crackRmc).

- flt.calc.interval – integer – SIS/EMC – frequency of output of SFPOF calculations in number of flights for each sub-interval
  - Regardless of the value of survival.updating, SFPOF is calculated and output at the beginning and end of sub-intervals of length flt.calc.interval. The actual sub-interval lengths will often vary slightly since sub-intervals are forced to end at a scheduled inspection. The sub-intervals within an inspection interval are rounded to integers which minimize the variation from flt.calc.interval. A value of 1 may be used, but for long service lives the results object may become large and calculation speeds may suffer.

- sfpof.threshold – numeric – SIS/EMC – if specified, a horizontal line will appear on results plots at this value.
  - This parameter is not currently used for anything important.

- sfpof.min – numeric – SIS/EMC – minimum value of SFPOF to include in results
  - If > 0, SFPOF calculated below sfpof.min will be recorded as sfpof.min. If = 0, SFPOF will be allowed to be zero in results. In this case, when plotting SFPOF on the log scale, warnings will be issued and the plot will not appear at the zero values.

- bootstrap.sfpof – logical – SIS – whether or not to perform obtain bootstrap confidence intervals of SFPOF during a run
  - If bootstrap confidence intervals are desired, set this parameter to TRUE. The number of bootstrap samples to generate at each calculation of SFPOF is set by bootstrap.samples

and the desired quantiles to estimate by bootstrap.quantiles. Results will be in additional column(s) with SFPOF in the SFPOF data.frame of the crackRresults object. Note that there are many SFPOF calculations and using a large number of bootstrap samples may significantly increase the run time.

- bootstrap.pcd – logical – SIS – whether or not to perform obtain bootstrap confidence intervals of PCD during a scheduled inspection
  - If bootstrap confidence intervals are desired, set this parameter to TRUE. The number of bootstrap samples to generate at each calculation of PCD is set by bootstrap.samples and the desired quantiles to estimate by bootstrap.quantiles. Results will be in additional column(s) with PCD in the PCD data.frame of the crackRresults object. Inspections occur less frequently than SFPOF calculations, but large values of bootstrap.samples may significantly increase the run time.

- bootstrap.samples – integer – SIS – number of bootstrap samples to take for either SFPOF or PCD
  - More samples = better bootstrap estimates of the quantiles of the sampling distribution(s), with more samples taking longer to run. Note the bootstrap run time is also a function of the number of particles (Np).

- bootstrap.quantiles – numeric – SIS – quantile(s) at which to estimate the sampling distribution of either SFPOF or PCD
  - this is simply the probs argument to the quantile function.

- ms.gumbel – numeric – SIS/EMC – vector of length two specifying the location (loc) and scale parameters for the Gumbel distribution which describes the maximum applied stress per flight
  - For example: `cp4$ms.gumbel <- c(loc=31.079,  scale=0.832)`
  - The names "loc" and "scale" are required!
  - crackR is currently hard-coded to utilize the Gumbel distribution for max stress per flight. Expanding to a generic distribution is not difficult; it just hasn't been done yet. Mostly this is because max stress per flight is almost always Gumbel distributed in the literature, so I haven't bothered.

- inspections – data.frame – SIS/EMC – specification of inspection intervals (in number of flights) and the inspection type to occur at the end of the interval
  - For example: `cp4$inspections <- data.frame(flt.interval = c(4615,2308,2308), type = c(1,2,1))`
  - The names "flt.interval" and "type" are required!
  - The above example specifies three inspection intervals over the 4615+2308+2308=9231 flights. The three inspections are of types 1, 2, and 1, respectively, which refers to parameter pod.func.

- pod.func – list – SIS/EMC – list of POD functions
  - For example: `cp7ext$pod.func <- list(); cp7ext$pod.func[[1]] <- function(a) lognormalPOD(a, 0.03, 1); cp7ext$pod.func[[2]] <-  function(a) lognormalPOD(a, 0.01, 1)`
  - The above example specifies two POD curves. At inspection time the parameter inspections specifies which of these to use. If the inspection type calls for a POD curve that is not specified here, errors will occur. See lognormalPOD and inspection.

- pod.threshold – numeric – SIS/EMC – either a matrix or a vector specifying the crack sizes at which different severities of repair will occur (see below for details)
  - For a run of analysis.type=="single" or analysis.type=="CD", this is a weakly increasing vector of thresholds at which to partition the PCD results. For example, if = 0, then no partitioning will occur; if = c(0.01, 0.05), then PCD will be split into three: a < 0.01, 0.01 < 0.05, and a > 0.05.
  - For analysis.type=="multiple", there is more than one type of repair that is possible and the behavior of each is different. Because of this, pod.threshold is an upper triangular matrix of dimension c(dta.types, dta.types), where each row is the POD threshold vector for that type. Note each row must be weakly increasing from left to right. It is upper triangular since it is assumed that a repair can only get more severe...unless the part is replaced because the crack grew to exceed the largest threshold value. This is admittedly confusing, so here's an example. Suppose there are two DTA types describing the possible condition at a single fastener hole, where the first represents the as-manufactured condition and the second represents an oversized fastener installation. Suppose pod.threshold==matrix(c(0.005,NA,0.25,0.005), nrow=2, ncol=2). For a type 1 particle, the POD threshold values used are taken from the first row of this matrix: c(0.005, 0.25). If a crack is found smaller than 0.005, it can be repaired to as-manufactured condition. If 0.005 < a < 0.25, the oversized fastener is installed and the type switches to type 2 as a result of the inspection being performed. If a > 0.25, the part is replaced (back to type 1). If instead when heading into inspection the repair was of type 2, then pod.threshold==0.005. If the found crack is smaller than 0.005, it can be repaired back to the oversized fastener condition (but not back to type 1, a smaller fastener). If a > 0.005, the part must be replaced (since the oversized fastener repair has already been utilized and no other repair is possible). Note that zero and Inf values may be used. Also the NA values can optionally be zero as they are not used by the code.

- dta.types – integer – SIS/EMC – number of Damage Tolerance Analysis (DTA) data sets present in the analysis
  - This is used only for analysis.type=="multiple" in the methods corresponding to a crackR object of class Mult.

- ismc.bool – logical – EMC – whether or not to utilize importance sampling to set the initial state in an EMC run
  - If FALSE standard Monte Carlo sampling is used and the initial crack size will be generated using, for example, ifs.ractual.
  - If TRUE importance sampling is used and the initial size will be generated using ifs.rsamp instead. Subsequently ifs.dsamp and ifs.dactual are used to determine the importance weights.
  - See parameter dta for more information regarding specification of initial flaw size distributions

- cg.cc – numeric – SIS/EMC – critical crack length for a run of either analysis.type=="single" or analysis.type=="multiple"
  - The crack length at which failure is assumed to occur with certainty. Note that if cg.cc is larger than the largest values in the DTA data item cg, linear extrapolation will occur until reaching cg.cc. If cg.cc is smaller than the largest values of said table, failure will occur earlier.

  - Generally cg.cc should be set to the last value of crack length in the DTA data item cg.

- cg.cc.pc/cg.cc.ph/cg.cc.sc/cg.cc.sh – numeric – SIS/EMC – critical crack length for a run of analysis.type=="CD"
  - For a continuing damage analysis, there are two cracks involved (primary and secondary), each of which can be in either a 'cold' state or a 'hot' state, depending on whether the other crack has previously reached critical size. Thus there are four sets of DTA data: primary cold (dta.pc), primary hot (dta.ph), secondary cold (dta.sc), and secondary hot (dta.sh). See below for more details.
  - The various critical crack sizes associated with a continuing damage analysis are as follows: cg.cc.pc, the length at which the primary crack goes critical (secondary goes hot); cg.cc.sc, the length at which the secondary crack goes critical (primary goes hot); cg.cc.ph, the length at which a primary hot crack fails; and cg.cc.sh, the length at which a secondary hot crack fails.

- dta – list – SIS/EMC – deterministic damage tolerance analysis input data for analysis.type=="single" or analysis.type=="multiple"
- There are several components to this list, each discussed in turn below
- For analysis.type=="single", this is a list of the following parameters
- For analysis.type=="multiple", dta is a "list of lists", where each item of the list (e.g., dta[[1]]) is a complete dta list with the components described below, each of which corresponds to a repair type. The length of the list of lists should equal the value of the parameter dta.types (this is redundant at present).
  - cg – data.frame – crack growth data with columns flight and crack; note this differs from some other probabilistic damage tolerance analysis codes which are indexed to flight hours instead of flight numbers. Linear extrapolation will occur using the last two rows of this table if cg.cc is larger than the last crack size in the table.
  - geo – data.frame – geometry data relating crack length to normalized stress intensity K/sigma with columns crack and ksig. The data in this table is not extrapolated; if a K/sigma value is needed for a crack size beyone the end of this table, the final value of K/sigma will be used.
  - kc.rsamp – function – generates values from fracture toughness distribution
  - ifs.rsamp – function – generates values from initial flaw size sampling distribution
  - ifs.dsamp – function – density function for initial flaw size sampling distribution
  - ifs.ractual – function – generates values from initial flaw size actual distribution
  - ifs.dactual – function – density function for initial flaw size actual distribution
  - rfs.rsamp – function – generates values from repair flaw size sampling distribution
  - rfs.dsamp – function – density function for repair flaw size sampling distribution
  - rfs.ractual – function – generates values from repair flaw size actual distribution
  - rfs.dactual – function – density function for repair flaw size actual distribution

- dta.pc/dta.ph/dta.sc/dta.sh – list – SIS/EMC – deterministic damage tolerance analysis input data for a continuing damage run
  - For a continuing damage run there are four distinct sets of DTA data, primary cold (dta.pc), primary hot (dta.ph), secondary cold (dta.sc), and secondary hot (dta.sh). The cracks become hot at cg.cc.pc or cg.cc.sc, and fail at cg.cc.ph or cg.cc.sh.

– At manufacture and following a repair, cracks are always 'cold'. Hence `dta.pc` and `dta.sc` need to have the various distributions for fracture toughness and initial flaw size, but `dta.ph` and `dta.sh` don't require them.

Value

`state`	List containing vectors representing the initial state of each particle, consisting of several of the following (depending on the analysis type): the crack lengths (a), crack lengths of primary and secondary cracks (a.p and a.s), fracture toughnesses (kc), importance weights (w), and particle types (typ)
`parameters`	List of the analysis parameters
`results`	List of blank data frames as a placeholder for results

Author(s)

Keith Halbert <keith.a.halbert@gmail.com>

References

Halbert, K. "Estimation of Probability of Failure for Damage-Tolerant Aerospace Structures" PhD Thesis, Temple University Department of Statistics, Philadelphia, PA, Apr 2014

See Also

`analyze`

Examples

```
data(cp7ext)
cp7ext.init <- crackRinit(cp7ext)
head(cp7ext.init$state)
```

---

crackRmc *Performs a probabilistic damage tolerance analysis using explicit Monte Carlo sampling*

Description

The main focus of the crackR package is on the sequential importance sampling approach to probabilistic damage tolerance analysis. As part of the work creating that approach, an explicit sampling routine was created for validation of results. It proceeds by repeatedly simulating the life cycle, flight-by-flight, and finding the first flight to failure for each trial. This approach requires many millions of samples to yield useful SFPOF estimates, but provided a sanity check for the results of the sequential importance sampling routine. Scheduled inspections may be included. If there are no scheduled inspections, the user may utilize importance sampling to set the initial state and drastically speed up convergence of the SFPOF estimates. The parameters for running this analysis are the same as those of the sequential importance sampling routine.

Usage

```
crackRmc(parameters)
## S3 method for class 'Sing'
crackRmc(parameters)
## S3 method for class 'Mult'
crackRmc(parameters)
## S3 method for class 'CD'
crackRmc(parameters)
```

Arguments

parameters	Object of class parametersSing, parametersMult, or parametersCD (or a list that looks like one of these classes, which are simply lists themselves.)

Details

Several items in the input list of parameters are either specific to the explicit MC approach, or important for it. ismc.bool indicates whether importance sampling should be used to set the initial state, in which case the function will look for, for example, the distribution sampling function ifs.rsamp to set the initial state instead of ifs.ractual. Np is the number of trials in this case; be warned, millions will take a VERY long time, so I suggest starting with 1,000 to check the speed on your machine. For a detailed description of each input parameter, see the provided examples (cp4).

Value

A crackRresults object, along with an additional list of the raw data from the MC run for later re-processing.

Note

This function could benefit strongly from parallelization, so let me know if you intend to use this function and we can probably get that going.

Author(s)

Keith Halbert <keith.a.halbert@gmail.com>

References

Halbert, K. "Estimation of Probability of Failure for Damage-Tolerant Aerospace Structures" PhD Thesis, Temple University Department of Statistics, Philadelphia, PA, Apr 2014

See Also

crackRmc

Examples

```
data(cp7ext)
cp7ext.ismc <- cp7ext
## importance sampling run (much faster)
cp7ext.ismc$ismc.bool <- TRUE
## this is very few samples for demo only
cp7ext.ismc$Np <- 2000
## only one inspection interval may be included with importance sampling
cp7ext.ismc$inspections <- data.frame(flt.interval=3000, type=1)
## set a low SFPOF minimum so we can see results
cp7ext.ismc$sfpof.min <- 1e-30
out.ismc <- crackRmc( cp7ext.ismc )
plot(out.ismc)
```

`inspection` *Predict the results of a future scheduled inspection and update the state accordingly*

Description

Predict the results of a future scheduled inspection and update the state accordingly. The state prior to inspection is utilized to determine the likelihood of finding each particle, and the state after inspection consists of a combination of missed particles and repaired particles. The Probability of Crack Detection (PCD) results of this inspection are appended to the previously existing PCD results (if any).

Usage

```
inspection(obj,inspection.type=1)
## S3 method for class 'Sing'
inspection(obj,inspection.type=1)
## S3 method for class 'Mult'
inspection(obj,inspection.type=1)
## S3 method for class 'CD'
inspection(obj,inspection.type=1)
```

Arguments

`obj`	Object of class crackR
`inspection.type`	Integer, index of which Probability Of Detection (POD) function from the parameters to utilize for this inspection.

Details

The likelihood of finding each particle at a future inspection depends only on the crack length at that time and the POD function (as specified by inspection.type). The overall Probability of Crack Detection (PCD) is found by taking a weighted average of the probability of detection for each particle and the importance weights.
Suppose there is a 40% chance of finding a particular particle. That particle will remain in the state, but to reflect the possibility that it was not found, the weight is reduced to 60% of the weight prior to inspection. The remaining weight will be used when generating new repaired particles (by sampling from the repair flaw size distribution). The total weight that is found for all particles is the estimate of PCD for this inspection (optionally partitioned into several crack length ranges using pod.threshold). After inspection, the weight of the repaired particles will sum to PCD, and the weight of the missed particles will sum to (1-PCD). Note also that the set of particles will be re-sampled without replacement to reduce the particle count back to parameters$Np.

Value

Object of class crackR.

Author(s)

Keith Halbert <keith.a.halbert@gmail.com>

References

Halbert, K. "Estimation of Probability of Failure for Damage-Tolerant Aerospace Structures" PhD Thesis, Temple University Department of Statistics, Philadelphia, PA, Apr 2014

See Also

crackRinit
analyze
calcInterval

Examples

```
set.seed(327)
data(cp7ext)

## initialize crackR object
cp7ext.init <- crackRinit(cp7ext)

## advance through time 6000 flights
cp7ext.before.insp <- calcInterval(cp7ext.init, interval.flights=6000)

## conduct inspection
cp7ext.after.insp <- inspection(cp7ext.before.insp)

## print inspection results
cp7ext.after.insp$results$pcd
```

`lognormalPOD` *Define Probability Of Detection (POD) curve for Non-Destructive Inspection (NDI) based on a Log-Normal CDF formulation*

Description

The POD curve in probabilistic damage tolerance analysis is often defined using the formulation of a Log-Normal CDF, with several optional modifying parameters. This function can be used to generate the appropriate POD function for use in the crackRparameters component "pod.func".

Usage

```
lognormalPOD(a,median,slope,a.min.detectable=0,poi=1,far=0)
```

Arguments

`a`	Crack length; the sole input to the POD function generated by this function.
`median`	Median detectable crack length.
`slope`	Slope parameter (aka sdlog).
`a.min.detectable`	Crack size below which detection is assumed to be impossible. Below this size the generated POD curve returns a zero probability. Note the far parameter will override this if far > 0.
`poi`	Probability Of Inspection. In practice an analyst may assume that some percentage of scheduled inspections will not occur as planned or will be conducted incorrectly. This can be represented by setting poi less than 1, in which case all detection probabilities are factored down.
`far`	Often in Non-Destructive Inspection (NDI) it is assumed there is no false alarm rate. This is most likely untrue. Setting a false alarm rate in this function forces a minimum value of POD that will be returned from the generated function.

Note

To define a custom POD function based on another type of curve or otherwise, the code for `crackRlognormalPOD()` is a good place to start.

Author(s)

Keith Halbert <keith.a.halbert@gmail.com>

References

MIL-HDBK-1823A (Department of Defense, USA) "Nondestructive Evaluation System Reliability Assessment", Apr 2009
Halbert, K. "Estimation of Probability of Failure for Damage-Tolerant Aerospace Structures" PhD Thesis, Temple University Department of Statistics, Philadelphia, PA, Apr 2014

See Also

crackRinit
inspection

Examples

```
myPODcurve <- function(a) lognormalPOD(a, median=0.01, slope=0.5, a.min.detectable =
0, poi = 0.95, far = 0.001)
myPODcurve(c(0, 0.005, 0.01, 0.05, 1))
```

---

plot.Sing *Plotting methods for various crackR objects*

Description

These methods generate plots for the various crackR object classes

Usage

```
## S3 method for class 'Sing'
plot(x,sfpof.int=FALSE,...)
## S3 method for class 'Mult'
plot(x,sfpof.int=FALSE,...)
## S3 method for class 'CD'
plot(x,sfpof.int=FALSE,...)
## S3 method for class 'crackRresults'
plot(x,boot=FALSE,thresh=NA,sfpof.int=FALSE,...)
## S3 method for class 'crackRparallel'
plot(x,thresh=NA,sfpof.int=FALSE,...)
## S3 method for class 'Sing'
lines(x,sfpof.int=FALSE,...)
## S3 method for class 'Mult'
lines(x,sfpof.int=FALSE,...)
## S3 method for class 'CD'
lines(x,sfpof.int=FALSE,...)
## S3 method for class 'crackRresults'
lines(x,sfpof.int=FALSE,...)
## S3 method for class 'crackRparallel'
lines(x,sfpof.int=FALSE,...)
```

Arguments

x	Object of appropriate class
boot	Logical. Whether to also plot bootstrap SFPOF estimates
thresh	Numeric. Plots a horizontal line on SFPOF plots at this value. If NA, no line is plotted.

`sfpof.int`	Logical. Whether SFPOF estimates should be approximated from the `pof.int` component of the results instead of the usual `sfpof` component
`...`	Optional additional arguments. Note the line type is hard-coded to \"l\"; trying to set a different type will currently cause an error.

Value

NULL

Author(s)

Keith Halbert <keith.a.halbert@gmail.com>

See Also

`crackRinit`
`analyze`
`analyzeParallel`

`print.Sing` *Printing methods for various crackR objects*

Description

These methods do the expected printings of crackR objects

Usage

```
## S3 method for class 'Sing'
print(x,...)
## S3 method for class 'Mult'
print(x,...)
## S3 method for class 'CD'
print(x,...)
## S3 method for class 'stateSing'
print(x,...)
## S3 method for class 'stateMult'
print(x,...)
## S3 method for class 'stateCD'
print(x,...)
## S3 method for class 'crackRresults'
print(x,sfpof.int=FALSE,...)
## S3 method for class 'crackRparallel'
print(x,sfpof.int=FALSE,...)
## S3 method for class 'Sing'
head(x,...)
## S3 method for class 'Mult'
```

```
head(x,...)
## S3 method for class 'CD'
head(x,...)
## S3 method for class 'parametersSing'
head(x,...)
## S3 method for class 'parametersMult'
head(x,...)
## S3 method for class 'parametersCD'
head(x,...)
## S3 method for class 'stateSing'
head(x,...)
## S3 method for class 'stateMult'
head(x,...)
## S3 method for class 'stateCD'
head(x,...)
## S3 method for class 'crackRresults'
head(x,sfpof.int=FALSE,...)
## S3 method for class 'crackRparallel'
head(x,sfpof.int=FALSE,...)
## S3 method for class 'Sing'
tail(x,...)
## S3 method for class 'Mult'
tail(x,...)
## S3 method for class 'CD'
tail(x,...)
## S3 method for class 'parametersSing'
tail(x,...)
## S3 method for class 'parametersMult'
tail(x,...)
## S3 method for class 'parametersCD'
tail(x,...)
## S3 method for class 'stateSing'
tail(x,...)
## S3 method for class 'stateMult'
tail(x,...)
## S3 method for class 'stateCD'
tail(x,...)
## S3 method for class 'crackRresults'
tail(x,sfpof.int=FALSE,...)
## S3 method for class 'crackRparallel'
tail(x,sfpof.int=FALSE,...)
```

Arguments

x	Object of appropriate class
...	Optional additional arguments
sfpof.int	Logical. Whether SFPOF estimates should be approximated from the pof.int component of the results instead of the usual sfpof component

Value

NULL

Author(s)

Keith Halbert <keith.a.halbert@gmail.com>

See Also

crackRinit
analyze
analyzeParallel

Index