Package 'PtProcess'

July 21, 2025

Version 3.3-17

Date 2025-06-03

Title Time Dependent Point Process Modelling

Description Fits and analyses time dependent marked point process models with an emphasis on earthquake modelling. For a more detailed introduction to the package, see the topic ``PtProcess". A list of recent changes can be found in the topic ``Change Log".

Suggests parallel

LazyData no

License GPL (>= 2)

URL https://www.statsresearch.co.nz/dsh/sslib/

NeedsCompilation no

Author David Harte [aut, cre]

Maintainer David Harte <d.s.harte@gmail.com>

Repository CRAN

Date/Publication 2025-06-03 05:50:01 UTC

Contents

`PtProcess-package` *Overview of PtProcess Package*

Description

This topic gives an introductory overview to the package **PtProcess**. Links are given to follow up topics where more detail can be found.

Introduction

This package contains routines for the fitting of *time dependent* point process models, particularly marked processes with "jumps". These models have particular application to earthquake data. A detailed theoretical background to these and other point process models can be found in Daley & Vere-Jones (2003, 2008). An overview of the package structure is given by Harte (2010).

The direction of the development of the package has been influenced by our research on the application of point process models to seismology. The package was originally written for S-PLUS, being part of the Statistical Seismology Library (Harte, 1998; Brownrigg & Harte, 2005). The package **ptproc** by Peng (2002, 2003) analyses multi-dimensional point process models, and the package **spatstat** by Baddeley et al (2005, 2005a, 2008) analyses spatial point processes.

The topic `Changes` *lists recent changes made to the package. Version 3 of the package has some major changes from Version 2, and code for Version 2 will not work in Version 3 without modification. Some examples giving the old code and the required new code are given in the topic* `Changes`*. Changes made in Version 3 enable one to fit a more general class of model.*

Classes of Point Process Models Analysed

The classes of models currently fitted by the package are listed below. Each are defined within an object that contains the data, current parameter values, and other model characteristics.

Marked Point Process Model: is described under the topic `mpp`. This model can be simulated or fitted to data by defining the required model structure within an object of class `"mpp"`.

Linked Stress Release Model: is described under the topic `linksrm`. This model is slightly peculiar, and doesn't fit naturally in the `mpp` framework.

Main Tasks Performed by the Package

The main tasks performed by the package are listed below. These can be achieved by calling the appropriate generic function.

Simulation: can be performed by the function `simulate`.

Parameter Estimation: can be achieved by using the function `neglogLik`.

Model Residuals: can be calculated with the function `residuals`.

Model Summary: can be extracted with the function `summary`.

Log-Likelihood: can be calculated with the function `logLik`.

Ground Intensity Plot: can be performed by the function `plot`.

The method function conforms to the following naming convention, for example, the function `logLik.mpp` provides the method to calculate the log-likelihood for `mpp` objects. The function code can be viewed by entering `PtProcess:::logLik.mpp` on the R command line.

If you want to modify such a function, `dump` the code to your local directory, modify in a text editor, then use `source` at the beginning of your program script, but after `library(PtProcess)`. Your modified version will then be used in preference to the version in the **PtProcess** package.

Organisation of Topics in the Package

Cited References: anywhere in the manual are only listed within this topic.

General Documentation: topics summarising general structure are indexed under the keyword "documentation" in the Index.

Acknowledgements

The package is based on an S-PLUS package which was commenced at Victoria University of Wellington in 1996. Contributions and suggestions have been made by many, including: Mark Bebbington, Ray Brownrigg, Edwin Choi, Robert Davies, Michael Eglinton, Dongfeng Li, Li Ma, Alistair Merrifield, Andrew Tokeley, David Vere-Jones, Wenzheng Yang, Leon Young, Irina Zhdanova and Jiancang Zhuang.

References

Aalen, O.O. & Hoem, J.M. (1978). Random time changes for multivariate counting processes. *Scandinavian Journal of Statistics* **5**, 81–101. doi:10.1080/03461238.1978.10419480

Baddeley, A. (2008). *Open source software for spatial statistics.* URL: `http://spatstat.org/`.

Baddeley, A. & Turner, R. (2005). Spatstat: an R package for analyzing spatial point patterns. *Journal of Statistical Software* **12(6)**, 1–42. doi:10.18637/jss.v012.i06

Baddeley, A.; Turner, R.; Moller, J. & Hazelton, M. (2005a). Residual analysis for spatial point processes (with discussion). *J. R. Statist. Soc. B* **67(5)**, 617–666. doi:10.1111/j.14679868.2005.00519.x

Bebbington, M.S. & Harte, D.S. (2001). On the statistics of the linked stress release model. *Journal of Applied Probability* **38A**, 176–187. doi:10.1239/jap/1085496600

Bebbington, M.S. & Harte, D.S. (2003). The linked stress release model for spatio-temporal seismicity: formulations, procedures and applications. *Geophysical Journal International* **154**, 925–946. doi:10.1046/j.1365246X.2003.02015.x

Brownrigg, R. & Harte, D.S. (2005). Using R for statistical seismology. R *News* **5(1)**, 31–35. URL: https://cran.r-project.org/doc/Rnews/Rnews_2005-1.pdf.

Daley, D.J. & Vere-Jones, D. (2003). *An Introduction to the Theory of Point Processes. Volume I: Elementary Theory and Methods. Second Edition.* Springer-Verlag, New York. doi:10.1007/b97277

Daley, D.J. & Vere-Jones, D. (2008). *An Introduction to the Theory of Point Processes. Volume II: General Theory and Structure. Second Edition.* Springer-Verlag, New York. doi:10.1007/9780387-498355

Harte, D. (1998). Documentation for the Statistical Seismology Library. School of Mathematical and Computing Sciences Research Report No. 98–10 (Updated Edition June 1999), Victoria University of Wellington. (ISSN 1174–4545)

Harte, D. (2010). PtProcess: An R package for modelling marked point processes indexed by time. *Journal of Statistical Software* **35(8)**, 1–32. doi:10.18637/jss.v035.i08

Kagan, Y. & Schoenberg, F. (2001). Estimation of the upper cutoff parameter for the tapered Pareto distribution. *Journal of Applied Probability* **38A**, 158–175. doi:10.1239/jap/1085496599

Lewis, P.A.W. & Shedler, G.S. (1979). Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistics Quarterly* **26(3)**, 403–413. doi:10.1002/nav.3800260304

Ogata, Y. (1981). On Lewis' simulation method for point processes. *IEEE Transactions on Information Theory* **27(1)**, 23–31. doi:10.1109/TIT.1981.1056305

Ogata, Y. (1988). Statistical models for earthquake occurrences and residual analysis for point processes. *J. Amer. Statist. Assoc.* **83(401)**, 9–27. doi:10.2307/2288914

Ogata, Y. (1998). Space-time point-process models for earthquake occurrences. *Ann. Instit. Statist. Math.* **50(2)**, 379–402. doi:10.1023/A:1003403601725

Ogata, Y. (1999). Seismicity analysis through point-process modeling: a review. *Pure and Applied Geophysics* **155**, 471–507. doi:10.1007/s000240050275

Ogata, Y. & Zhuang, J.C. (2006). Space-time ETAS models and an improved extension. *Tectonophysics* **413(1-2)**, 13–23. doi:10.1016/j.tecto.2005.10.016

Peng, R. (2002). Multi-dimensional Point Process Models. Package "ptproc", URL: https://rdpeng.org/.

Peng, R. (2003). Multi-dimensional point process models in R. *Journal of Statistical Software* **8(16)**, 1–27. doi:10.18637/jss.v008.i16

Reid, H.F. (1910). The mechanism of the earthquake. In *The California Earthquake of April 18, 1906, Report of the State Earthquake Investigation Commission* **2**, 16–28. Carnegie Institute of Washington, Washington D.C.

Utsu, T. and Ogata, Y. (1997). Statistical analysis of seismicity. In: *Algorithms for Earthquake Statistics and Prediction* (Edited by: J.H. Healy, V.I. Keilis-Borok and W.H.K. Lee), pp 13–94. IASPEI, Menlo Park CA.

Vere-Jones, D. (1978). Earthquake prediction - a statistician's view. *Journal of Physics of the Earth* **26**, 129–146. doi:10.4294/jpe1952.26.129

Vere-Jones, D.; Robinson, R. & Yang, W. (2001). Remarks on the accelerated moment release model: problems of model formulation, simulation and estimation. *Geophysical Journal International* **144(3)**, 517–531. doi:10.1046/j.1365246x.2001.01348.x

Zheng, X.-G. & Vere-Jones, D. (1991). Application of stress release models to historical earthquakes from North China. *Pure and Applied Geophysics* **135(4)**, 559–576. doi:10.1007/BF01772406

Zhuang, J.C. (2006). Second-order residual analysis of spatiotemporal point processes and applications in model evaluation. *J. R. Statist. Soc. B* **68(4)**, 635–653. doi:10.1111/j.14679868.2006.00559.x

---

`Change Log` *Changes Made to the Package*

Description

This page contains a listing of recent changes made to functions, and known general problems.

Recent Changes

1. Version 3 contains major changes, and code that worked in Version 2 will no longer work in Version 3. The models included in Version 2 are also contained in Version 3, but the framework has been extended so that the original models can now contain a variety of mark distributions. This has been achieved by giving a more general structure and utilising the object orientated aspects of the R language. *Examples are given below that show how models were defined in Version 2 and how the corresponding models are now defined in Version 3.* (28 Apr 2008)
2. Naming changes to the `*.cif` functions. In Version 2, these were referred to as "conditional intensity functions", which is really a slightly more general class. In keeping with Daley & Vere-Jones (2003) we now call them ground intensity functions, with a suffix of "gif". Further, the dot has been replaced by an underscore, e.g. `etas.cif` to `etas_gif`. This is to lessen the possibility of future conflicts with object orientated naming conventions in the R language. (28 Apr 2008)
3. Arguments `eval.pts` and `t.plus` in the ground intensity functions have been renamed to `evalpts` and `tplus`, respectively. This is to lessen the possibility of future conflicts with object orientated naming conventions in the R language. (28 Apr 2008)
4. `logLik`: the log-likelihood calculated in package Versions before Version 3 did not have the sum over the mark density term (see topic `logLik`, under "Details"). This term can also be excluded in this Version of the package by placing `NULL` for the mark density in the `mpp` object, see example below. (28 Apr 2008)
5. Version 2 had a framework to assign prior densities to the estimated parameters. This has not been retained in Version 3. However, some of the features like holding a parameter at a fixed value, and restricting it to an open or closed interval can be achieved in Version 3; see `neglogLik` for further details. (28 Apr 2008)
6. `neglogLik`: the format of this function has been changed to be consistent with that in package **HiddenMarkov**. Argument `updatep` renamed as `pmap`. (07 Aug 2008)
7. `simulate`: manual page revised to include more information about controlling the length of the simulated series. (18 Nov 2008)
8. `mpp`: example modified due to warning messages caused by negative $\lambda_g(t|\mathcal{H}_t)$. (18 Nov 2008)
9. `marks`: manual page revised to include more information. (18 Nov 2008)
10. `mpp`: fuller description to argument `marks` on manual page. (19 Nov 2008)

11. Phuket: new dataset added. (4 Dec 2008)
12. linksrm_gif, marks: remove some LaTeX specific formatting to be compatible with R 2.9.0. (26 Jan 2009)
13. Phuket: clarify magnitude scale used in the dataset. (11 Jul 2009)
14. Attribute type is no longer required on the gif functions, removed. (7 Oct 2009)
15. logLik, neglogLik: Parallel processing support, using package **snow**, has been added. (8 Oct 2009)
16. plot: Correct hyperlink to generic plot function. (10 Oct 2009)
17. etas_normal0: New function. Test version of a spatial ETAS conditional intensity function. (12 Oct 2009)
18. logLik: Fixed bug when using parallel processing on only two nodes. (22 Oct 2009)
19. Tidied HTML representation of equations in manual pages. Removal of "synopsis" on manual pages of functions with multiple forms of usage. (26 Jan 2010)
20. logLik.mpp, summary.mpp: Changed to inherits to determine class. (27 Jan 2010)
21. Phuket: Additional data, until the beginning of 2009, have been added. The magnitude is now the maximum of the body wave and surface wave magnitudes, m_b and M_s, respectively. Earlier it was simply m_b. (01 Feb 2010)
22. simulate.linksrm, simulate.mpp, logLik.mpp: Inconsistency in nomenclature between "mark" and "marks", will standardise on the plural. (07 May 2010)
23. simulate.mpp: Two bugs:
    use <- (data[, "time"] < TT[1]) changed to use <- (data[, "time"] <= TT[1]),
    and else data <- data[use, c("time", "magnitude")] changed to
    else data <- data[use,]. (18 Jun 2010)
24. etas_normal0: Errors in some terms involving beta. (18 Jun 2010)
25. Minor citation and reference inclusion changes to manual pages. (19 Jul 2010)
26. simulate.mpp: Bug fix on 18 June 2010 induced another bug;
    data <- rbind(data, newevent) changed to
    data <- rbind(data[, names(newevent)], newevent). (11 Dec 2010)
27. Implement very basic NAMESPACE. (5 Nov 2011)
28. List functions explicitly in NAMESPACE; "LazyData: no" and "ZipData: no" in DESCRIPTION file. (9 Dec 2011)
29. logLik.mpp: Enable one to specify the relative CPU speeds of the nodes when parallel processing. (9 Dec 2011)
30. mpp and etas_normal0: Restrict the number of iterations in examples on manual pages to minimise time during package checks. (13 Dec 2011)
31. residuals and linksrm: Include example using cusum of residuals on manual page. (15 Dec 2011)
32. dpareto, dtappareto, ltappareto (etc): Include parameter consistency checks. (6 Jan 2014)
33. etas_gif: Documentation example error: marks=list(rmagn_mark, rmagn_mark) should be marks=list(dmagn_mark, NULL). (23 Jan 2014)

34. `linksrm1_gif`: Function deleted, alternative discussed on manual page of `linksrm_gif`. (19 Mar 2014)
35. Correct html problem in '`inst/doc/index.html`'. (14 Aug 2014)
36. `logLik.mpp`: Call to `clusterApply` changed to `snow::clusterApply`. (20 Aug 2014)
37. `logLik.mpp`: The package **snow** has been superseded by **parallel**. Change `snow` to `parallel`, also in file '`DESCRIPTION`'. (15 Oct 2014)
38. `makeSOCKcluster`: This function is in **snow** but not in **parallel**. This function points to the closest eqivalent in **parallel**, `makePSOCKcluster`. `makeSOCKcluster` will eventually become deprecated. Was added to the export list in file '`NAMESPACE`' too. (15 Oct 2014)
39. `logLik.mpp`, `neglogLik`: Update consistent with changes from **snow** to **parallel**. (17 Oct 2014)
40. `logLik.mpp`: Change `require(parallel)` to `requireNamespace("parallel")`. (21 Jan 2015)
41. Added to NAMESPACE:
    `importFrom(graphics, plot)`
    `importFrom(stats, dexp, integrate, logLik, pnorm,`
    `qexp, rexp, runif, simulate, ts)`
    (03 Jul 2015)
42. `PtProcess`: Add DOI to some references, rename topic to appear first in table of contents. (16 Oct 2015)
43. `plot.mpp`: Activate argument `ylim`. (17 Aug 2016)
44. `etas_normal0`: This has been removed. Adding a spatial dimension requires more generality in other package functions like `logLik.mpp`. For a reasonable amount of generality, it requires the addition of new model class, currently under development. (01 Sep 2016)
45. `simulate.mpp`: Did not allow argument `marks = list(NULL, NULL)` in `mpp` object. `simulate.mpp` now tests to see if `NULL` marks. (17 Nov 2017)
46. `fourier_gif`: Example added on manual page with `NULL` marks. (17 Nov 2017)
47. `Phuket`: Hyperlink to data source updated, others updated to https where possible. (24 Apr 2021)
48. Updated '`DESCRIPTION`' file. (03 Jun 2025)

Future Development

1. Currently spatial versions of the ETAS model are being written and tested.
2. In the model object, allow one to alternatively specify the `name` of the gif function.
3. Function `linksrm_gif`: Use of `St1` and `St2`. Is there a tidier way? Also utilise this feature in `srm_gif`.
4. Want a generic function, possibly called `forecast`, to produce probability forecasts. This would be based on simulating empirical probability distributions.
5. Want a function like `linksrm_convert` to map between the two main parametrisations of the ETAS model.
6. Add general forms of the truncated exponential and gamma distributions as marks for the magnitude of the event.
7. A tidy way to pass the values of the `gif` function into the mark distributions, if required.

References

Cited references are listed on the PtProcess manual page.

Examples

```
#    SRM: magnitude is iid exponential with bvalue=1
#    simulate and calculate the log-likelihood

TT <- c(0, 1000)
bvalue <- 1
params <- c(-1.5, 0.01, 0.8, bvalue*log(10))

#   --- Old Method ---
# x <- pp.sim(NULL, params[1:3], srm.cif, TT, seed=5, magn.sim=1)
# print(pp.LL(x, srm.cif, params[1:3], TT))
# [1] -601.3941

#   --- New Method, no mark density ---
x1 <- mpp(data=NULL,
          gif=srm_gif,
          marks=list(NULL, rexp_mark),
          params=params,
          gmap=expression(params[1:3]),
          mmap=expression(params[4]),
          TT=TT)
x1 <- simulate(x1, seed=5)
print(logLik(x1))

#  An advantage of the object orientated format is that it
#  simplifies further analysis, e.g. plot intensity function:
plot(x1)
#  plot the residual process:
plot(residuals(x1))

#-----------------------------------------------------
#    SRM: magnitude is iid exponential with bvalue=1
#    simulate then estimate parameters from data

#   --- Old Method ---
# TT <- c(0, 1000)
# bvalue <- 1
# params <- c(-2.5, 0.01, 0.8)
#
# x <- pp.sim(NULL, params, srm.cif, TT, seed=5, magn.sim=1)
#
# posterior <- make.posterior(x, srm.cif, TT)
#
# neg.posterior <- function(params){
#     x <- -posterior(params)
#     if (is.infinite(x) | is.na(x)) return(1e15)
#     else return(x)
# }
```

```
#
# z <- nlm(neg.posterior, params, typsize=abs(params),
#          iterlim=1000, print.level=2)
#
# print(z$estimate)
# [1] -2.83900091  0.01242595  0.78880647

#   --- New Method, no mark density ---
#   maximise only SRM parameters (like old method)

TT <- c(0, 1000)
bvalue <- 1
params <- c(-2.5, 0.01, 0.8, bvalue*log(10))

x1 <- mpp(data=NULL,
          gif=srm_gif,
          marks=list(dexp_mark, rexp_mark),
          params=params,
          gmap=expression(params[1:3]),
          mmap=expression(params[4]),
          TT=TT)
#  note that dexp_mark above is not used below
#  and could alternatively be replaced by NULL

x1 <- simulate(x1, seed=5)

#  maximise only SRM parameters
onlysrm <- function(y, p){
    #  maps srm parameters into model object
    #  the exp rate for magnitudes is unchanged
    y$params[1:3] <- p
    return(y)
}

params <- c(-2.5, 0.01, 0.8)

z1 <- nlm(neglogLik, params, object=x1, pmap=onlysrm,
          print.level=2, iterlim=500, typsize=abs(params))
print(z1$estimate)
```

`distribution` *General Notes on Distribution Fitting*

Description

This page contains general notes about fitting probability distributions to datasets.

Details

We give examples of how the maximum likelihood parameters can be estimated using standard optimisation routines provided in the R software (`nlm` and `optim`). We simply numerically maximise

the sum of the logarithms of the density evaluated at each of the data points, i.e. log-likelihood function. In fact, by default, the two mentioned optimizers find the *minimum*, and hence we minimise the negative log-likelihood function.

Both optimization routines require initial starting values. The optimisation function `optim` uses a grid search technique, and is therefore more robust to poor starting values. The function `nlm` uses derivatives and the Hessian to determine the size and direction of the next step, which is generally more sensitive to poor initial values, but faster in the neighbourhood of the solution. One possible strategy is to start with `optim` and then use its solution as a starting value for `nlm`. This is done below in the example for the tapered Pareto distribution.

The function `nlm` numerically calculates the Hessian and derivatives, by default. If the surface is very flat, the numerical error involved may be larger in size than the actual gradient. In this case the process will work better if analytic derivatives are supplied. This is done in the tapered Pareto example below. Alternatively, one could simply use the Newton-Raphson algorithm (again, see the tapered Pareto example below).

We also show that parameters can be constrained to be positive (or negative) by transforming the parameters with the exponential function during the maximisation procedure. Similarly, parameters can be restricted to a finite interval by using a modified logit transform during the maximisation procedure. The advantage of using these transformations is that the entire real line is mapped onto the positive real line or the required finite interval, respectively; and further, they are differentiable and monotonic. This eliminates the "hard" boundaries which are sometimes enforced by using a penalty function when the estimation procedure strays into the forbidden region. The addition of such penalty functions causes the function that is being optimised to be non-differentiable at the boundaries, which can cause considerable problems with the optimisation routines.

Examples

```
#    Random number generation method
RNGkind("Mersenne-Twister", "Inversion")
set.seed(5)

#-------------------------------------------
#    Exponential Distribution

#    simulate a sample
p <- 1
x <- rexp(n=1000, rate=p)

#    Transform to a log scale so that -infty < log(p) < infty.
#    Hence no hard boundary, and p > 0.
#    If LL is beyond machine precision, LL <- 1e20.

neg.LL <- function(logp, data){
    x <- -sum(log(dexp(data, rate=exp(logp))))
    if (is.infinite(x)) x <- 1e20
    return(x)
}

p0 <- 5
logp0 <- log(p0)
z <- nlm(neg.LL, logp0, print.level=0, data=x)
```

```
print(exp(z$estimate))

#    Compare to closed form solution
print(exp(z$estimate)-1/mean(x))

#-------------------------------------------
#    Normal Distribution

#    simulate a sample
x <- rnorm(n=1000, mean=0, sd=1)

neg.LL <- function(p, data){
    x <- -sum(log(dnorm(data, mean=p[1], sd=exp(p[2]))))
    if (is.infinite(x)) x <- 1e20
    return(x)
}

p0 <- c(2, log(2))
z <- nlm(neg.LL, p0, print.level=0, data=x)
p1 <- c(z$estimate[1], exp(z$estimate[2]))
print(p1)

#    Compare to closed form solution
print(p1 - c(mean(x), sd(x)))

#-------------------------------------------
#    Gamma Distribution
#    shape > 0 and rate > 0
#    use exponential function to ensure above constraints

#    simulate a sample
x <- rgamma(n=2000, shape=1, rate=5)

neg.LL <- function(p, data){
    #   give unreasonable values a very high neg LL, i.e. low LL
    if (any(exp(p) > 1e15)) x <- 1e15
    else{
        x <- -sum(log(dgamma(data, shape=exp(p[1]), rate=exp(p[2]))))
        if (is.infinite(x)) x <- 1e15
    }
    return(x)
}

p0 <- c(2, 2)
z <- optim(p0, neg.LL, data=x)
print(exp(z$par))

z <- nlm(neg.LL, p0, print.level=0, data=x)
print(exp(z$estimate))

#-------------------------------------------
#    Beta Distribution
#    shape1 > 0 and shape2 > 0
```

```
#    use exponential function to ensure above constraints

#    simulate a sample
x <- rbeta(n=5000, shape1=0.5, shape2=0.2)

#    exclude those where x=0
x <- x[x!=1]

neg.LL <- function(p, data)
    -sum(log(dbeta(data, shape1=exp(p[1]), shape2=exp(p[2]))))

p0 <- log(c(0.1, 0.1))

z <- optim(p0, neg.LL, data=x)
print(exp(z$par))

z <- nlm(neg.LL, p0, typsize=c(0.01, 0.01), print.level=0, data=x)
print(exp(z$estimate))

#-------------------------------------------
#    Weibull Distribution
#    shape > 0 and scale > 0
#    use exponential function to ensure above constraints

#    simulate a sample
x <- rweibull(n=2000, shape=2, scale=1)

neg.LL <- function(p, data)
    -sum(log(dweibull(data, shape=exp(p[1]), scale=exp(p[2]))))

p0 <- log(c(0.1, 0.1))
z <- optim(p0, neg.LL, data=x)
print(exp(z$par))

#-------------------------------------------
#    Pareto Distribution
#    lambda > 0
#    Use exponential function to enforce constraint

#    simulate a sample
x <- rpareto(n=2000, lambda=2, a=1)

neg.LL <- function(p, data){
    #   give unreasonable values a very high neg LL, i.e. low LL
    if (exp(p) > 1e15) x <- 1e15
    else x <- -sum(log(dpareto(data, lambda=exp(p), a=1)))
    if (is.infinite(x)) x <- 1e15
    return(x)
}

p0 <- log(0.1)
z <- nlm(neg.LL, p0, print.level=0, data=x)
print(exp(z$estimate))
```

```
#-------------------------------------------
#    Tapered Pareto Distribution
#    lambda > 0  and  theta > 0

# simulate a sample
x <- rtappareto(n=2000, lambda=2, theta=4, a=1)

neg.LL <- function(p, data){
    x <- -ltappareto(data, lambda=p[1], theta=p[2], a=1)
    attr(x, "gradient") <- -attr(x, "gradient")
    attr(x, "hessian") <- -attr(x, "hessian")
    return(x)
}

#   use optim to get approx initial value
p0 <- c(3, 5)
z1 <- optim(p0, neg.LL, data=x)
p1 <- z1$par
print(p1)
print(neg.LL(p1, x))

#   nlm with analytic gradient and hessian
z2 <- nlm(neg.LL, p1, data=x, hessian=TRUE)
p2 <- z2$estimate
print(z2)

#    Newton Raphson Method
p3 <- p1
iter <- 0
repeat{
    LL <- ltappareto(data=x, lambda=p3[1], theta=p3[2], a=1)
    p3 <- p3 - as.numeric(solve(attr(LL,"hessian")) %*%
                matrix(attr(LL,"gradient"), ncol=1))
    iter <- iter + 1
    if ((max(abs(attr(LL,"gradient"))) < 1e-8) |
        (iter > 100)) break
}
print(iter)
print(LL)
print(p3)
```

---

dpareto *Pareto and Tapered Pareto Distributions*

Description

Density, cumulative probability, quantiles and random number generation for the Pareto and tapered Pareto distributions with shape parameter λ, tapering parameter θ and range $a \leq x < \infty$; and log-likelihood of the tapered Pareto distribution.

Usage

```
dpareto(x, lambda, a, log=FALSE)
ppareto(q, lambda, a, lower.tail=TRUE, log.p=FALSE)
qpareto(p, lambda, a, lower.tail=TRUE, log.p=FALSE)
rpareto(n, lambda, a)

dtappareto(x, lambda, theta, a, log=FALSE)
ltappareto(data, lambda, theta, a)
ptappareto(q, lambda, theta, a, lower.tail=TRUE, log.p=FALSE)
qtappareto(p, lambda, theta, a, lower.tail=TRUE, log.p=FALSE,
           tol=1e-8)
rtappareto(n, lambda, theta, a)

ltappareto(data, lambda, theta, a)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
data	vector of sample data.
n	number of observations to simulate.
lambda	shape parameter, see Details below.
theta	tapering parameter, see Details below..
a	the random variable takes values on the interval $a \leq x < \infty$. This is a scalar and is assumed to be a constant for all values in a given function call.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $\Pr\{X \leq x\}$, otherwise, $\Pr\{X > x\}$.
tol	convergence criteria for the Newton Raphson algorithm for solving the quantiles of the tapered Pareto distribution.

Details

For all functions except ltappareto, arguments lambda and theta can either be scalars or vectors of the same length as x, p, or q. If a scalar, then this value is assumed to hold over all cases. If a vector, then the values are assumed to have a one to one relationship with the values in x, p, or q. The argument a is a scalar.

In the case of ltappareto, all data are assumed to be drawn from the same distribution and hence lambda, theta and a are all scalars.

Let Y be an exponential random variable with parameter $\lambda > 0$. Then the distribution function of Y is

$$F_Y(y) = \Pr\{Y < y\} = 1 - \exp(-\lambda y),$$

and the density function is

$$f_Y(y) = \lambda \exp(-\lambda y).$$

Further, the mean and variance of the distribution of Y is $1/\lambda$ and $1/\lambda^2$, respectively.

Now transform Y as

$$X = a \exp(Y),$$

where $a > 0$. Then X is a Pareto random variable with shape parameter λ and distribution function

$$F_X(x) = \Pr\{X < x\} = 1 - \left(\frac{a}{x}\right)^{\lambda},$$

where $a \leq x < \infty$, and density function

$$f_X(x) = \frac{\lambda}{a}\left(\frac{a}{x}\right)^{\lambda+1}.$$

We simulate the Pareto deviates by generating exponential deviates, and then transforming as described above.

As above, let X be Pareto with shape parameter λ, and define $W - a$ to be exponential with parameter $1/\theta$, i.e.

$$\Pr\{X > x\} = \left(\frac{a}{x}\right)^{\lambda}$$

and

$$\Pr\{W > w\} = \exp\left(\frac{a-w}{\theta}\right),$$

where $a \leq w < \infty$. Say we sample one independent value from each of the distributions X and W, then

$$\Pr\{X > z \,\&\, W > z\} = \Pr\{X > z\}\Pr\{W > z\} = \left(\frac{a}{z}\right)^{\lambda}\exp\left(\frac{a-z}{\theta}\right).$$

We say that Z has a tapered Pareto distribution if it has the above distribution, i.e.

$$F_Z(z) = \Pr\{Z < z\} = 1 - \left(\frac{a}{z}\right)^{\lambda}\exp\left(\frac{a-z}{\theta}\right).$$

The above relationship shows that a tapered Pareto deviate can be simulated by generating independent values of X and W, and then letting $Z = \min(X, W)$. This minimum has the effect of "tapering" the tail of the Pareto distribution.

The tapered Pareto variable Z has density

$$f_Z(z) = \left(\frac{\lambda}{z} + \frac{1}{\theta}\right)\left(\frac{a}{z}\right)^{\lambda}\exp\left(\frac{a-z}{\theta}\right).$$

Given a sample of data $z_1, z_2, \cdots, z_n$, we write the log-likelihood as

$$\log L = \sum_{i=1}^{n} \log f_Z(z_i).$$

Hence the gradients are calculated as

$$\frac{\partial \log L}{\partial \lambda} = \theta \sum_{i=1}^{n} \frac{1}{\lambda\theta + z_i} - \sum_{i=1}^{n} \log(z_i/a)$$

and

$$\frac{\partial \log L}{\partial \theta} = \frac{-1}{\theta} \sum_{i=1}^{n} \frac{z_i}{\lambda\theta + z_i} - \frac{1}{\theta^2} \sum_{i=1}^{n} (a - z_i).$$

Further, the Hessian is calculated using

$$\frac{\partial^2 \log L}{\partial \lambda^2} = -\theta^2 \sum_{i=1}^{n} \frac{1}{(\lambda\theta + z_i)^2},$$

$$\frac{\partial^2 \log L}{\partial \theta^2} = \frac{1}{\theta^2} \sum_{i=1}^{n} \frac{z_i(2\lambda\theta + z_i)}{(\lambda\theta + z_i)^2} - \frac{2}{\theta^3} \sum_{i=1}^{n} (a - z_i),$$

and

$$\frac{\partial^2 \log L}{\partial \theta \, \partial \lambda} = \frac{\partial^2 \log L}{\partial \lambda \, \partial \theta} = \sum_{i=1}^{n} \frac{z_i}{(\lambda\theta + z_i)^2}.$$

See the section "Seismological Context" (below), which outlines its application in Seismology.

Value

`dpareto` and `dtappareto` give the densities; `ppareto` and `ptappareto` give the distribution functions; `qpareto` and `qtappareto` give the quantile functions; and `rpareto` and `rtappareto` generate random deviates.

`ltappareto` returns the log-likelihood of a sample using the tapered Pareto distribution. It also calculates, using analytic expressions (see "Details"), the derivatives and Hessian which are attached to the log-likelihood value as the attributes `"gradient"` and `"hessian"`, respectively.

Seismological Context

The Gutenberg-Richter (GR) Law says that if we plot the base 10 logarithm of the number of events with magnitude greater than M (vertical axis) against M (horizontal axis), there should be a straight line. This is equivalent to magnitudes having an exponential distribution.

Assume that the magnitude cutoff is M_0, and let $Y = M - M_0$. Given that Y has an exponential distribution with parameter λ, it follows that

$$\log_{10}(1 - F_Y(y)) = \frac{-\lambda y}{\log_e 10}.$$

The coefficient $\lambda/(\log_e 10)$ is often referred to as the b-value, and its negative value is the slope of the line in the GR plot.

Now define S as

$$S = 10^{\gamma(M - M_0)} = 10^{\gamma Y}.$$

When $\gamma = 0.75$, S is the "stress"; and when $\gamma = 1.5$, S is the "seismic moment". Still assuming that Y is exponential with parameter λ, then $Y\gamma \log_e 10$ is also exponential with parameter $\lambda/(\gamma \log_e 10)$. Hence, by noting that S can be rewritten as

$$S = \exp\{Y\gamma \log_e 10\},$$

it is seen that S is Pareto with parameter $\lambda/(\gamma \log_e 10)$, and $1 \leq S < \infty$.

While the empirical distribution of magnitudes appears to follow an exponential distribution for smaller events, it provides a poor approximation for larger events. This is because it is not physically possible to have events with magnitudes much greater than about 9.5. Consequently, the tail of the Pareto distribution will also be too long. Hence the tapered Pareto distribution provides a more realistic description.

See Also

See dexp for the exponential distribution. Generalisations of the exponential distribution are the gamma distribution dgamma and the Weibull distribution dweibull.

See the topic distribution for examples of estimating parameters.

Examples

```
#    Simulate and plot histogram with density for Pareto Distribution

a0 <- 2
lambda0 <- 2
x <- rpareto(1000, lambda=lambda0, a=a0)
x0 <- seq(a0, max(x)+0.1, length=100)
hist(x, freq=FALSE, breaks=x0, xlim=range(x0),
     main="Pareto Distribution")
points(x0, dpareto(x0, lambda0, a0), type="l", col="red")

#-----------------------------------------------
#    Calculate probabilities and quantiles for Pareto Distribution

a0 <- 2
lambda0 <- 2
prob <- ppareto(seq(a0, 8), lambda0, a0)
quan <- qpareto(prob, lambda0, a0)
print(quan)

#-----------------------------------------------
#    Simulate and plot histogram with density for tapered Pareto Distribution

a0 <- 2
lambda0 <- 2
theta0 <- 3
x <- rtappareto(1000, lambda=lambda0, theta=theta0, a=a0)
x0 <- seq(a0, max(x)+0.1, length=100)
hist(x, freq=FALSE, breaks=x0, xlim=range(x0),
     main="Tapered Pareto Distribution")
points(x0, dtappareto(x0, lambda0, theta0, a0), type="l", col="red")

#-----------------------------------------------
#    Calculate probabilities and quantiles for tapered Pareto Distribution

a0 <- 2
lambda0 <- 2
theta0 <- 3
prob <- ptappareto(seq(a0, 8), lambda0, theta0, a0)
```

```
quan <- qtappareto(prob, lambda0, theta0, a0)
print(quan)

#-----------------------------------------------
#    Calculate log-likelihood for tapered Pareto Distribution
#    note the Hessian and gradient attributes

a0 <- 2
lambda0 <- 2
theta0 <- 3
x <- rtappareto(1000, lambda=lambda0, theta=theta0, a=a0)
LL <- ltappareto(x, lambda=lambda0, theta=theta0, a=a0)
print(LL)
```

---

etas_gif *Ground Intensity for ETAS Model*

Description

This function calculates the value of the ground intensity of a time-magnitude Epidemic Type Aftershock Sequence (ETAS) model. Spatial coordinates of the events are not taken into account.

Usage

```
etas_gif(data, evalpts, params, TT=NA, tplus=FALSE)
```

Arguments

data	a data frame containing the event history, where each row represents one event. There must be columns named "time", usually the number of days from some origin; and "magnitude" which is the event magnitude less the magnitude threshold, i.e. $M_i - M_0$.
evalpts	a vector, matrix or data.frame. If a vector, the elements will be assumed to represent the required evaluation times. Other objects must include a column named "time" that can be referred to as evalpts[,"time"], at which the intensity function will be evaluated.
params	vector of parameter values in the following order: (μ, A, α, c, p).
TT	vector of length 2, being the time interval over which the integral of the ground intensity function is to be evaluated.
tplus	logical, $\lambda_g(t\|\mathcal{H}_t)$ is evaluated as $\lambda_g(t^+\|\mathcal{H}_t)$ if TRUE, else $\lambda_g(t^-\|\mathcal{H}_t)$.

Details

The ETAS model was proposed by Ogata (1988, 1998, 1999) for the modelling of earthquake mainshock-aftershock sequences. The form of the ground intensity function used here is given by

$$\lambda_g(t|\mathcal{H}_t) = \mu + A \sum_{i:t_i<t} e^{\alpha(M_i - M_0)} \left(1 + \frac{t - t_i}{c}\right)^{-p},$$

where t_i denotes the event times and the summation is taken over those i such that $t_i < t$.

Value

Two usages are as follows.

```
etas_gif(data, evalpts, params, tplus=FALSE)
etas_gif(data, evalpts=NULL, params, TT)
```

The first usage returns a vector containing the values of $\lambda_g(t)$ evaluated at the specified points. In the second usage, it returns the value of the integral.

Function Attributes

`rate` is `"decreasing"`.

References

Cited references are listed on the PtProcess manual page.

See Also

General details about the structure of ground intensity functions are given in the topic `gif`.

Examples

```
#  Tangshan: ground intensity and magnitude time plots

data(Tangshan)
p <- c(0.007, 2.3, 0.98, 0.008, 0.94)
bvalue <- 1
TT <- c(0, 4018)

x <- mpp(data=Tangshan,
         gif=etas_gif,
         marks=list(dexp_mark, NULL),
         params=p,
         gmap=expression(params),
         mmap=expression(bvalue*log(10)),
         TT=TT)

par.default <- par(mfrow=c(1,1), mar=c(5.1, 4.1, 4.1, 2.1))
par(mfrow=c(2,1), mar=c(4.1, 4.1, 0.5, 1))
```

```
plot(x, log=TRUE, xlab="")

plot(Tangshan$time, Tangshan$magnitude+4, type="h",
     xlim=c(0, 4018),
     xlab="Days Since 1 January 1974", ylab="Magnitude")

par(par.default)
```

---

gif *General Notes on Ground Intensity Functions*

Description

This page contains general notes about the required structure of ground intensity functions (including those that are not conditional on their history) to be used with this package.

Forms of Usage

The usage of a ground intensity function takes two forms, one to evaluate the `gif` at specified `evalpts`, or to evaluate the integral of the `gif` on the interval `TT`, each shown below, respectively.

```
gif(data, evalpts, params, tplus=FALSE)
gif(data, NULL, params, TT)
```

Arguments

All ground intensity functions should be defined to contain the following arguments, in the order below, even though they may not be required (see Details below).

`data` a data frame containing the history of the process, denoted below as $\mathcal{H}_t$. It should contain all variables that are required to evaluate the `gif` function, though can contain others too. No history is represented as `NULL`.

`evalpts` a object containing the values at which the `gif` function is to be evaluated, consistent with what is required by the `gif` function.

`params` vector containing values of the parameters required by the `gif` function.

`TT` vector of length 2, being the time interval over which the integral of the ground intensity function is to be evaluated.

`tplus` logical, $\lambda_g(t|\mathcal{H}_t)$ is evaluated as $\lambda_g(t^+|\mathcal{H}_t)$ if `TRUE`, else $\lambda_g(t^-|\mathcal{H}_t)$. It is important if a "jump" occurs at t.

Details

Note that the `gif` functions not only evaluate values of $\lambda_g(t_i|\mathcal{H}_t)$, but also the integral. The value of the ground intensity function is returned at each time point specified in `evalpts` when `TT==NA`. If `TT` is not missing, the integral between `TT[1]` and `TT[2]` of the ground intensity function is calculated. In this last situation, anything assigned to the argument `evalpts` will have no effect.

At the moment, we have the following types of processes: those jump processes that are conditional on their history (`etas_gif`, `srm_gif`, `linksrm_gif`), and non-homogeneous Poisson processes that

are not conditional on their history (simple_gif). Another case is where we have a collection of point like "regions" (or lattice nodes), each with their own ground intensity function, but where each is also dependent on what is happening in the other regions (linksrm_gif).

Functions have been given an attribute "rate", taking the values of "bounded", "decreasing" or "increasing". This is used within the simulation function simulate.mpp which uses the thinning method. This method requires a knowledge of the maximum of $\lambda_g(t|\mathcal{H}_t)$ in a given interval. The argument tplus is also used by the simulation routine, where it is necessary to determine the value of the intensity immediately after a simulated event.

Value

The returned value is either $\lambda_g(t_i|\mathcal{H}_t)$, where the t_i are specified within evalpts; or

$$\int \lambda_g(t|\mathcal{H}_t)dt$$

where the limits of the integral are specified by the function argument TT.

Function Attributes

Each function should have some of the following attributes if it is to be used in conjunction with residuals.mpp or simulate.mpp:

rate must be specified if the default method for simulate.mpp is to be used. Takes the values "bounded", "decreasing" or "increasing"; see Details.

regions an expression giving the number of regions; required with linksrm_gif.

See Also

etas_gif, expfourier_gif, exppoly_gif, fourier_gif, linksrm_gif, poly_gif, simple_gif, srm_gif

Examples

```
#  Ogata's Data: ground intensity function
#  evaluate lambda_g(t) at certain times

data(Ogata)

p <- c(0.02, 70.77, 0.47, 0.002, 1.25)
times <- sort(c(seq(0, 800, 0.5), Ogata$time))
TT <- c(0, 800)

plot(times, log(etas_gif(Ogata, times, params=p)), type="l",
     ylab=expression(paste(log, " ", lambda[g](t))),
     xlab=expression(t), xlim=TT)

#  Evaluate the integral
#   The first form below is where the arguments are in their
#   default positions, the 2nd is where they are not, hence
#   their names must be specified
```

```
print(etas_gif(Ogata, NULL, p, TT))
#  or
print(etas_gif(Ogata, params=p, TT=TT))
```

`linksrm` *Linked Stress Release Model Object*

Description

Creates a point process model object with class `"linksrm"`.

Usage

```
linksrm(data, gif, marks, params, gmap, mmap, TT)
```

Arguments

`data`	a `data.frame` containing the history of the process, denoted below as $\mathcal{H}_t$. It should contain all variables that are required to evaluate the `gif` function and the mark distribution, though can contain others too. No history is represented as `NULL`.
`gif`	ground intensity function. At this stage, this can only be `linksrm_gif` or modifications of that function; see "Details" below.
`marks`	mark distribution. See topic `marks` for further details.
`params`	numeric vector of *all* model parameters.
`gmap`	`expression`, maps the model parameters (`params`) into the parameter sub-space of the ground intensity function; see "Details" below.
`mmap`	`expression`, maps the model parameters (`params`) into the parameter sub-space of the mark distribution; see "Details" below.
`TT`	vector of length 2, being the time interval over which the integral of the ground intensity function is to be evaluated.

Details

The linked stress release model has a slightly peculiar structure which makes it difficult to fit into the `mpp` class. While the region should be thought of as a mark, it is completely defined by the function `linksrm_gif`, and hence from the programming perspective the `region` mark is really tied in with the `gif` function. Hence at the moment, the linked stress release model is treated as a special case. There may be other models that could be grouped into this class.

Examples

```
p <- c(-1.5, -1.5, 0.01, 0.03, 2, -0.5, 0.2, 1, 1*log(10), 3)
TT <- c(0, 1000)

rexptrunc_mark <- function(ti, data, params){
    x <- rexp(n=1, params[1])
    x[x > params[2]] <- params[2]
    names(x) <- "magnitude"
    return(x)
}

x <- linksrm(data=NULL,
             gif=linksrm_gif,
             marks=list(NULL, rexptrunc_mark),
             params=p,
             gmap=expression(params[1:8]),
             mmap=expression(params[9:10]),
             TT=TT)

x <- simulate(x, seed=5)
print(logLik(x))

#   estimate parameters
temp_map <- function(y, p){
    #    map only gif parameters into model object
    y$params[1:8] <- p
    return(y)
}

weight <- c(0.1, 0.1, 0.005, 0.005, 0.1, 0.1, 0.1, 0.1)

#   see manual page for linksrm_gif for modifications to
#   make calculations faster

#   for testing, restrict to 5 iterations
z <- nlm(neglogLik, p[1:8], object=x, pmap=temp_map,
         hessian=TRUE, gradtol=1e-08, steptol=1e-10,
         print.level=2, iterlim=5, typsize=weight)

param.names <- c("a1", "a2", "b1", "b2", "c11", "c12", "c21", "c22")
param.est <- cbind(p[1:8], z$estimate, sqrt(diag(solve(z$hessian))))
dimnames(param.est) <- list(param.names,
                            c("Actual", "Estimate", "StdErr"))
print(param.est)

#   place parameter estimates into model object
x <- temp_map(x, z$estimate)

#   plot ground intensity function
par.default <- par(mfrow=c(2,1), mar=c(4.1, 4.1, 0.5, 1))
x$gif <- linksrm_gif
plot(x, 1, xlab="")
```

```
plot(x, 2)
par(par.default)

#   plot "residuals" for each region
tau <- residuals(x)
par(mfrow=c(2,1))
for (i in 1:2){
    plot(tau[[i]], ylab="Transformed Time",
         xlab="Event Number", main=paste("Region", i))
    abline(a=0, b=1, lty=2, col="red")
}

#   plot cusum of "residuals" for each region
for (i in 1:2){
    plot(tau[[i]] - 1:length(tau[[i]]), ylab="Cusum of Transformed Time",
         xlab="Event Number", main=paste("Region", i))
    abline(h=0, lty=2, col="red")
}

par(mfrow=c(1,1))
```

`linksrm_convert` *Parameter Conversion for Linked Stress Release Model*

Description

Converts parameter values between two different parameterisations (described in Details below) of the linked stress release model.

Usage

```
linksrm_convert(params, abc=TRUE)
```

Arguments

`params`	a vector of parameter values of length n^2+2n, where n is the number of regions in the model.
`abc`	logical. If `TRUE` (default), then the input value of params is that of the abc parameterisation. See Details for further explanation.

Details

If `abc == TRUE`, the conditional intensity for the ith region is assumed to have the form

$$\lambda_g(t,i|\mathcal{H}_t) = \exp\left\{a_i + b_i\left[t - \sum_{j=1}^{n} c_{ij}S_j(t)\right]\right\}$$

with `params`$= (a_1,\cdots,a_n,b_1,\cdots,b_n,c_{11},c_{12},c_{13},\cdots,c_{nn})$.

If abc == FALSE, the conditional intensity for the ith region is assumed to have the form

$$\lambda_g(t, i|\mathcal{H}_t) = \exp\left\{\alpha_i + \nu_i\left[\rho_i t - \sum_{j=1}^{n}\theta_{ij}S_j(t)\right]\right\}$$

where $\theta_{ii} = 1$ for all i, $n = \sqrt{\texttt{length(params)} + 1} - 1$, and params

$$= (\alpha_1, \cdots, \alpha_n, \nu_1, \cdots, \nu_n, \rho_1, \cdots, \rho_n, \theta_{12}, \theta_{13}, \cdots, \theta_{1n}, \theta_{21}, \theta_{23}, \cdots, \theta_{n,n-1}).$$

Value

A list object with the following components is returned:

params	vector as specified in the function call.
a	vector of length n as in the abc parameterisation.
b	vector of length n as in the abc parameterisation.
c	n by n matrix as in the abc parameterisation.
alpha	vector of length n as in the alternative parameterisation.
nu	vector of length n as in the alternative parameterisation.
rho	vector of length n as in the alternative parameterisation.
theta	n by n matrix with ones on the diagonal as in the alternative parameterisation.

See Also

linksrm_gif

---

linksrm_gif *Ground Intensity for Linked Stress Release Model*

Description

Calculates the value of the ground intensity of a Linked Stress Release Model (LSRM). This model allows for multiple linked regions, where the stress can be transferred between the regions.

Usage

```
linksrm_gif(data, evalpts, params, TT=NA, tplus=FALSE, eta=0.75)
```

Arguments

`data`	a data frame containing the event history, where each row represents one event. There must be columns named `"time"`, usually the number of days from some origin; `"magnitude"` which is the event magnitude less the magnitude threshold, i.e. $M_k - M_0$; and `"region"` which are consecutively numbered starting at 1.
`evalpts`	a `matrix` or `data.frame`. It must include two columns named `"time"` and `"region"` that can be referred to as `evalpts[,"time"]` and `evalpts[,"region"]`, respectively. The function will be evaluated at these points.
`params`	vector of parameters of length $n^2 + 2n$, where n is the number of regions, for the proposed LSRM in the following order: $$(a_1, \cdots, a_n, b_1, \cdots, b_n, c_{11}, c_{12}, c_{13}, \cdots, c_{nn}).$$
`TT`	vector of length 2, being the time interval over which the integral of the ground intensity function is to be evaluated.
`tplus`	logical, $\lambda_g(t, i\|\mathcal{H}_t)$ is evaluated as $\lambda_g(t^+, i\|\mathcal{H}_t)$ if `TRUE`, else $\lambda_g(t^-, i\|\mathcal{H}_t)$.
`eta`	a scalar used in the stress calculations, see Details below.

Details

The ground intensity for the ith region is assumed to have the form

$$\lambda_g(t, i|\mathcal{H}_t) = \exp\left\{a_i + b_i\left[t - \sum_{j=1}^{n} c_{ij} S_j(t)\right]\right\}$$

with `params`$= \texttt{c}(a_1, \cdots, a_n, b_1, \cdots, b_n, c_{11}, c_{12}, c_{13}, \cdots, c_{nn})$; and

$$S_j(t) = \sum_k 10^{\eta(M_k - M_0)},$$

where the summation is taken over those events in region j with time $t_k < t$. This model has been discussed by Bebbington & Harte (2001, 2003). The default value of $\eta =$ `eta` $= 0.75$.

Value

Two usages are as follows.

```
linksrm_gif(data, evalpts, params, tplus=FALSE, eta=0.75)
linksrm_gif(data, evalpts=NULL, params, TT, eta=0.75)
```

The first usage returns a vector containing the values of $\lambda_g(t, i)$ evaluated at the specified "time-region" points. In the second usage, it returns a vector containing the value of the integral for each region.

Function Attributes

`rate` is `"increasing"`.

`regions` is `expression(sqrt(length(params) + 1) - 1)`.

Modify Function to Decrease Calculation Time

The function `linksrm_gif` calculates the stress reduction matrices `St1` and `St2` every time that the function is called. Ideally, these should be calculated once and be included within the model object. Currently, the structure of the model object is not sufficiently flexible. However, the user can create a new function to calculate `St1` and `St2` once. This will only work if the event *history* is not changing between successive calls (e.g. parameter estimation). However, in a simulation, the history changes with the addition of each new event, and in this situation `St1` and `St2` need to be calculated with every function call.

The modified function, as described below, will write the objects `St1` and `St2` to a temporary database (position 2 in the search path). Consequently, it cannot be defined within the package itself because this violates the CRAN rules. The function `linksrm_gif` contains markers indicating the beginning and ending of the parts where `St1` and `St2` are calculated. The modified function is made by editing the function `linksrm_gif`. We firstly `deparse` the function `linksrm_gif` (i.e. put the contents into a character vector). We initially create a temporary database called `PtProcess.tmp` in which to write `St1` and `St2`. We then search for the line numbers that mark the beginning and ending of the parts where `St1` and `St2` are calculated. We replace the beginning of each with a conditional statement so that the contents are only run if these two objects do not already exist. We then `parse` the lines of code in the character vector back into a function, and call this new function `linksrm1_gif`. The same thing can be achieved by dumping `linksrm_gif` to a text file and editing manually.

```
#   define linksrm1_gif by modifying linksrm_gif

#   put function linksrm_gif into a character vector
tmp <- deparse(linksrm_gif)

#   remove "if (FALSE)" lines
linenum <- grep("if \(FALSE\)", tmp)
tmp <- tmp[-linenum]

#   attach new database at pos=2 in search path called PtProcess.tmp
linenum <- grep("attach new database to search path", tmp)
tmp[linenum] <- "if (!any(search()==\"PtProcess.tmp\")) attach(NULL,
                     pos=2L, name=\"PtProcess.tmp\", warn.conflicts=TRUE)"

#   calc St1 if St1 does not exist
linenum <- grep("this loop calculates St1", tmp)
tmp[linenum] <- "if (!exists(\"St1\", mode = \"numeric\")) {"
linenum <- grep("assign statement for St1", tmp)
tmp[linenum] <- "assign(\"St1\", St1, pos=\"PtProcess.tmp\")"
linenum <- grep("end loop St1", tmp)
tmp[linenum] <- "}"


#   calc St2 if St2 does not exist
linenum <- grep("this loop calculates St2", tmp)
tmp[linenum] <- "if (!exists(\"St2\", mode = \"numeric\")) {"
linenum <- grep("assign statement for St2", tmp)
```

```
tmp[linenum] <- "assign(\"St2\", St2, pos=\"PtProcess.tmp\")"
linenum <- grep("end loop St2", tmp)
tmp[linenum] <- "}"

linksrm1_gif <- eval(parse(text=tmp))
```

Warning: The function `linksrm1_gif` checks to see whether the matrices `St1` and `St2` exist. If so, these existing matrices are used, and new ones are not calculated. Therefore when using `linksrm1_gif` for parameter estimation, one **must** check for the existence of such matrices, and delete upon starting to fit a new model:

```
if (exists("St1")) rm(St1)
if (exists("St2")) rm(St2)
```

or detach the database as `detach(2)`. The objects `St1` and `St2` will exist for the duration of the current R session, so should be deleted when no longer required.

References

Cited references are listed on the PtProcess manual page.

See Also

General details about the structure of ground intensity functions are given in the topic `gif`.

`logLik` *Log Likelihood of a Point Process Model*

Description

Calculates the log-likelihood of a point process. Provides methods for the generic function `logLik`.

Usage

```
## S3 method for class 'mpp'
logLik(object, SNOWcluster=NULL, ...)
## S3 method for class 'linksrm'
logLik(object, ...)
```

Arguments

`object`	an object with class `"mpp"` or `"linksrm"`.
`SNOWcluster`	an object of class `"cluster"` created by the package **parallel**; default is `NULL`. Enables parallel processing if not `NULL`. See "Parallel Processing" below for further details.
`...`	other arguments.

Value

Value of the log-likelihood.

Parallel Processing

Parallel processing can be enabled to calculate the term $\sum_i \log \lambda_g(t_i|\mathcal{H}_{t_i})$. Generally, the amount of computational work involved in calculating $\lambda_g(t|\mathcal{H}_t)$ is much greater if there are more events in the process history prior to t than in the case where there are fewer events. Given m nodes, the required evaluation points are divided into m groups, taking into account the amount of "history" prior to each event and the CPU speed of the node (see below).

We have assumed that communication between nodes is fairly slow, and hence it is best to allocate the work in large chunks and minimise communication. If the dataset is small, then the time taken to allocate the work to the various nodes may in fact take more time than simply using one processor to perform all of the calculations.

The required steps in initiating parallel processing are as follows.

```
#   load the "parallel" package
library(parallel)

#   define the SNOW cluster object, e.g. a SOCK cluster
#   where each node has the same R installation.
cl <- makeSOCKcluster(c("localhost", "horoeka.localdomain",
                        "horoeka.localdomain", "localhost"))

#   A more general setup: Totara is Fedora, Rimu is Debian:
#   Use 2 processors on Totara, 1 on Rimu:
totara  <- list(host="localhost",
                rscript="/usr/lib/R/bin/Rscript",
                snowlib="/usr/lib/R/library")
rimu    <- list(host="rimu.localdomain",
                rscript="/usr/lib/R/bin/Rscript",
                snowlib="/usr/local/lib/R/site-library")
cl <- makeCluster(list(totara, totara, rimu), type="SOCK")

#   NOTE: THE STATEMENTS ABOVE WERE APPROPRIATE FOR THE snow PACKAGE.
#   I HAVE NOT YET TESTED THEM USING THE parallel PACKAGE.

#   Relative CPU speeds of the nodes can be added as an attribute
#   Say rimu runs at half the speed of totara
#   (default assumes all run at same speed)
attr(cl, "cpu.spd") <- c(1, 1, 0.5)

#   then define the required model object, e.g. see topic "mpp"
#   say the model object is called x

#   then calculate the log-likelihood as
print(logLik(x, SNOWcluster=cl))
```

```
#   stop the R jobs on the slave machines
stopCluster(cl)
```

Note that the communication method does not need to be SOCKS; see the **parallel** package documentation, topic makeCluster, for other options. Further, if some nodes are on other machines, the firewalls may need to be tweaked. The master machine initiates the R jobs on the slave machines by communicating through port 22 (use of security keys are needed rather than passwords), and subsequent communications use random ports. This port can be fixed, see makeCluster.

Examples

```
#    SRM: magnitude iid exponential with bvalue=1

TT <- c(0, 1000)
bvalue <- 1
params <- c(-2.5, 0.01, 0.8, bvalue*log(10))

#   calculate log-likelihood excluding the mark density term
x1 <- mpp(data=NULL,
          gif=srm_gif,
          marks=list(NULL, rexp_mark),
          params=params,
          gmap=expression(params[1:3]),
          mmap=expression(params[4]),
          TT=TT)
x1 <- simulate(x1, seed=5)
print(logLik(x1))

#   calculate log-likelihood including the mark density term
x2 <- mpp(data=x1$data,
          gif=srm_gif,
          marks=list(dexp_mark, rexp_mark),
          params=params,
          gmap=expression(params[1:3]),
          mmap=expression(params[4]),
          TT=TT)
print(logLik(x2))

#  contribution from magnitude marks
print(sum(dexp(x1$data$magnitude, rate=bvalue*log(10), log=TRUE)))
```

---

makeSOCKcluster *Parallel Processing: Transition Functions*

Description

Package **snow** has become deprecated and replaced by **parallel**. Some functions in **snow** used by package **PtProcess** do not appear in **parallel** under the same name. Below are transition functions to map some functions in **snow** to the most comparable functions in **parallel**. These transition functions will ultimately be deprecated.

Usage

```
makeSOCKcluster(names, ...)
```

Arguments

names	character vector of node names.
...	cluster option specifications.

Details

makeSOCKcluster calls makePSOCKcluster.

---

marks *Mark Distributions*

Description

Contains densities and random number generators for some example mark distributions. The mark distributions can be multi-dimensional. Users can write their own functions, and general rules are given under "Details".

Usage

```
dexp_mark(x, data, params)
rexp_mark(ti, data, params)
```

Arguments

ti	scalar, time of an event.
x	a data.frame of mark values at given times, often a subset of the history.
data	a data.frame containing the history of the process, denoted below as $\mathcal{H}_t$.
params	numeric vector of parameters.

Details

The example functions listed under "Usage" calculate the *logarithm* of the (mark) density and simulate earthquake magnitudes assuming an exponential distribution that is independent of the history of the process. This corresponds to the Gutenberg-Richter law. They assume that the history contains a variable named "magnitude".

All mark densities and random number generators must have the three arguments as shown in the examples above. Multi-parameter distributions have their parameters specified as a vector in the params argument. Other ancillary data or information can be passed into the function non formally, though one needs to be careful about possible conflict with names of other objects.

Value

Mark density functions must return a vector with length being equal to the number of rows in `x`. Each element contains the *logarithm* of the joint density of the marks corresponding to each time (row) in `x`.

The random number generator simulates each mark for a *single value* of `ti`. It must return a `list` of simulated marks corresponding to the specified time `ti`. Further, the list must have its elements named the same as those in the history. Note that each component in the list will be of length one. A list is used (rather than a vector) because it allows marks to be character as well as numeric.

Example 1

This is an example where the density of the magnitude distribution is dependent on the value of the ground intensity function (assumed to be `etas_gif`), and in this case, the history of the process. The history is assumed to contain a variable named `"magnitude"`. In this mark distribution, it is assumed that after large events, there is a deficit of smaller magnitude events with more larger magnitude events. It has seven parameters with parameters $p_1, \cdots, p_5$ relating to `etas_gif`. It assumes that the magnitude distribution is gamma (`GammaDist`), with a shape parameter given by

$$\text{shape} = 1 + \sqrt{\lambda_g(t|\mathcal{H}_t)}\, p_7\,,$$

where p_7 ($p_7 > 0$) is a free estimable parameter, and parameter p_6 is the scale parameter. Hence when $\lambda_g(t|\mathcal{H}_t)$ is small, the magnitude distribution returns to an approximate exponential distribution with an approximate rate of p_6 (i.e. Gutenberg Richter law).

```
dexample1_mark <- function(x, data, params){
    lambda <- etas_gif(data, x[,"time"], params=params[1:5])
    y <- dgamma(x[,"magnitude"], rate=params[6],
                shape=1+sqrt(lambda)*params[7], log=TRUE)
    return(y)
}

rexample1_mark <- function(ti, data, params){
    #  Gamma distribution
    #  exponential density when params[7]=0
    lambda <- etas_gif(data, ti, params=params[1:5])
    y <- rgamma(1, shape=1+sqrt(lambda)*params[7],
                rate=params[6])
    return(list(magnitude=y))
}
```

Example 2

This an example of a 3-D mark distribution. Each component is independent of each other and the history, hence the arguments `ti` and `data` are not utilised in the functions. The history is assumed to contain the three variables `"magnitude"`, `"longitude"` and `"latitude"`. The event magnitudes are assumed to have an exponential distribution with rate `params[1]`, and the longitudes and latitudes to have normal distributions with means `params[2]` and `params[3]`, respectively.

```
dexample2_mark <- function(x, data, params)
    return(dexp(x[,"magnitude"], rate=params[1], log=TRUE) +
           dnorm(x[,"longitude"], mean=params[2], log=TRUE) +
           dnorm(x[,"latitude"], mean=params[3], log=TRUE))

rexample2_mark <- function(ti, data, params)
    return(list(magnitude=rexp(1, rate=params[1]),
                longitude=rnorm(1, mean=params[2]),
                latitude=rnorm(1, mean=params[3])))
```

---

mpp *Marked Point Process Object*

Description

Creates a marked point process model object with class "mpp".

Usage

```
mpp(data, gif, marks, params, gmap, mmap, TT)
```

Arguments

data	a data.frame containing the history of the process, denoted below as $\mathcal{H}_t$. It should contain all variables that are required to evaluate the gif function and the mark distribution, though can contain others too. No history is represented as NULL.
gif	ground intensity function. See topic gif for further details.
marks	a list containing the mark distribution. The first component (i.e. marks[[1]]) is the mark density and the second (i.e. marks[[2]]) is the random number generator. If either of these functions are not required, the particular component can be set to NULL. See topic marks for further details.
params	numeric vector of *all* model parameters.
gmap	expression, maps the model parameters (params) into the parameter sub-space of the ground intensity function; see "Details" below.
mmap	expression, maps the model parameters (params) into the parameter sub-space of the mark distribution; see "Details" below.
TT	vector of length 2, being the time interval over which the integral of the ground intensity function is to be evaluated.

Details

Let $\lambda_g(t|\mathcal{H}_t)$ denote the ground intensity function and $f(y|\mathcal{H}_t)$ denote the joint mark densities, where $y \in \mathcal{Y}$. The log-likelihood of a marked point process is given by

$$\log L = \sum_i \log \lambda_g(t_i|\mathcal{H}_{t_i}) + \sum_i \log f(y_i|\mathcal{H}_{t_i}) - \int \lambda_g(t|\mathcal{H}_t)dt,$$

where the summation is taken over those events contained in the interval `(TT[1], TT[2])`, and the integral is also taken over that interval. However, all events in the data frame `data` before t, even those before `TT[1]`, form the history of the process $\mathcal{H}_t$. This allows an initial period for the process to reach a "steady state" or "equilibrium".

The parameter spaces of the ground intensity function and mark distribution are not necessarily disjoint, and can have common parameters. Hence, when the model parameters are estimated, these relationships must be known, and are specified by the arguments `gmap` and `mmap`. The mapping expressions can also contain arithmetic expressions. The ith element in the `params` argument is addressed in the expressions as `params[i]`. Here is an example of a five parameter model, where the `gif` has 4 parameters, and the mark distribution has 2, with mappings specified as:

```
gmap = expression(c(params[1:3], exp(params[4]+params[5])))

mmap = expression(c(log(params[2]/3), params[5]))
```

Note the inclusion of the combine (`c`) function, because the `expression` must create a vector of parameters. Care must be taken specifying these expressions as they are embedded directly into the code of various functions.

Examples

```
data(Tangshan)

#   increment magnitudes a fraction so none are zero
Tangshan[,"magnitude"] <- Tangshan[,"magnitude"] + 0.01

dmagn_mark <- function(x, data, params){
    #  Gamma distribution
    #  exponential density when params[7]=0
    #   See topic "marks" for further discussion
    lambda <- etas_gif(data, x[,"time"], params=params[1:5])
    y <- dgamma(x[,"magnitude"], shape=1+sqrt(lambda)*params[7],
                rate=params[6], log=TRUE)
    return(y)
}

TT <- c(0, 4018)
# params <- c(0.0067, 1.1025, 1.0794, 0.0169, 0.9506, 1.9159, 0.4704)
params <- c(0.007, 1.1, 1.08, 0.02, 0.95, 1.92, 0.47)

x <- mpp(data=Tangshan,
         gif=etas_gif,
         marks=list(dmagn_mark, NULL),
```

```
          params=params,
          gmap=expression(params[1:5]),
          mmap=expression(params[1:7]),
          TT=TT)

allmap <- function(y, p){
    #    one to one mapping, all p positive
    y$params <- exp(p)
    return(y)
}

#    Parameters must be positive. Transformed so that nlm
#    can use entire real line (no boundary problems, see
#    topic "neglogLik" for further explanation).
#    Argument "iterlim" has been restricted to 2 to avoid
#    excessive time in package checks, set much larger to
#    ensure convergence.
z <- nlm(neglogLik, log(params), object=x, pmap=allmap,
          print.level=2, iterlim=2, typsize=abs(params))

x1 <- allmap(x, z$estimate)

#    print parameter estimates
print(x1$params)

print(logLik(x))
print(logLik(x1))
plot(x1, log=TRUE)
```

---

neglogLik *Negative Log-Likelihood*

Description

Calculates the log-likelihood multiplied by negative one. It is in a format that can be used with the functions nlm and optim.

Usage

```
neglogLik(params, object, pmap = NULL, SNOWcluster=NULL)
```

Arguments

params	a vector of revised parameter values.
object	an object of class "mpp".
pmap	a user provided function mapping the revised parameter values params into the appropriate locations in object. If NULL (default), an untransformed one to one mapping is used.
SNOWcluster	an object of class "cluster" created by the package **parallel**; default is NULL. Enables parallel processing if not NULL. See logLik for further details.

Details

This function can be used with the two functions nlm and optim (see "Examples" below) to maximise the likelihood function of a model specified in object. Both nlm and optim are *minimisers*, hence the "negative" log-likelihood. The topic distribution gives examples of their use in the relatively easy situation of fitting standard probability distributions to data assuming independence.

The maximisation of the model likelihood function can be restricted to be over a subset of the model parameters. Other parameters will then be fixed at the values stored in the model object. Let Θ_0 denote the full model parameter space, and let Θ denote the parameter sub-space ($\Theta \subseteq \Theta_0$) over which the likelihood function is to be maximised. The argument params contains values in Θ, and pmap is assigned a function that maps these values into the full model parameter space Θ_0. See "Examples" below.

The mapping function assigned to pmap can also be made to impose restrictions on the domain of the parameter space Θ so that the minimiser cannot jump to values such that $\Theta \not\subseteq \Theta_0$. For example, if a particular parameter must be positive, one can work with a transformed parameter that can take any value on the real line, with the model parameter being the exponential of this transformed parameter. Similarly a modified logit like transform can be used to ensure that parameter values remain within a fixed interval with finite boundaries. Examples of these situations can be found in the topic distribution and the "Examples" below.

Value

Value of the log-likelihood times negative one.

See Also

nlm, optim

Examples

```
#    SRM: magnitude is iid exponential with bvalue=1
#    maximise exponential mark density too

TT <- c(0, 1000)
bvalue <- 1
params <- c(-2.5, 0.01, 0.8, bvalue*log(10))

x <- mpp(data=NULL,
         gif=srm_gif,
         marks=list(dexp_mark, rexp_mark),
         params=params,
         gmap=expression(params[1:3]),
         mmap=expression(params[4]),
         TT=TT)
x <- simulate(x, seed=5)

allmap <- function(y, p){
    #    map all parameters into model object
    #    transform exponential param so it is positive
    y$params[1:3] <- p[1:3]
    y$params[4] <- exp(p[4])
```

```
    return(y)
}

params <- c(-2.5, 0.01, 0.8, log(bvalue*log(10)))

z <- nlm(neglogLik, params, object=x, pmap=allmap,
         print.level=2, iterlim=500, typsize=abs(params))
print(z$estimate)

#   these should be the same:
print(exp(z$estimate[4]))
print(1/mean(x$data$magnitude))
```

NthChina *Historical Earthquakes of North China*

Description

Contains 65 large historical earthquakes in North China between 1480 and 1997, as given by Bebbington & Harte (2003). Events are divided into 4 regions using the regionalisations given by Zheng & Vere-Jones (1991).

Usage

```
data(NthChina)
```

Format

A data frame with 65 rows, each representing an earthquake event, with the following variables:

time number of years since 1480 AD.

latitude number of degrees north.

longitude number of degrees east.

magnitude number of magnitude units *above* 6.

region 1, 2, 3, or 4; being the region of the event.

References

Cited references are listed on the PtProcess manual page.

Ogata *Ogata's ETAS Test Data*

Description

A data frame containing the test data from Utsu and Ogata's (1997) software contained in the file testetas.dat. The first column is named `"time"`, and the second column is named `"magnitude"`.

Usage

```
data(Ogata)
```

Format

A data frame with 100 rows (earthquake events) in the time interval (0, 800). It contains the following variables:

time number of time units since time zero.

magnitude number of magnitude units *above* 3.5.

References

Cited references are listed on the PtProcess manual page.

Examples

```
data(Ogata)
plot(Ogata$time, Ogata$magnitude + 3.5, type="h")
```

Phuket *Phuket Earthquake and Aftershock Sequence*

Description

The Phuket earthquake occurred on 26 December 2004 at 00:58:53.45 GMT. The `Phuket` data frame contains this event and its aftershock sequence.

Usage

```
data(Phuket)
```

Format

This data frame contains the following columns:

latitude number of degrees north.

longitude number of degrees east.

depth depth of event in kilometres.

mb body wave magnitude (m_b) rounded to one decimal place.

Ms surface wave magnitude (M_s) rounded to one decimal place.

magnitude event magnitude ($\max(m_b, M_s)$) rounded to one decimal place.

year year of event (numeric vector).

month month of event, 1 ... 12 (numeric vector).

day day of event, 1 ... 31 (numeric vector).

hour hour of event, 0 ... 23 (numeric vector).

minute minute of event, 0 ... 59 (numeric vector).

second second of event, 0 ... 59 (numeric vector).

time number of days (and fractions) from midnight on 1 January 2004.

Details

The `Phuket` data frame contains those events (1248) from the PDE Catalogue, within the spatial region 89°E–105°E and 5°S–16°N, with magnitude 5 or greater, occurring between midnight on 1 January 2004 and midnight on 1 January 2009 (1827 days later). The body wave magnitudes are determined by the amplitude of the initial primary wave, and these magnitudes tend to saturate for higher values. Consequently, the tabulated `magnitude` is taken as the maximum of the body wave magnitude (m_b) and surface wave magnitude (M_s).

Source

The data were extracted from the PDE (Preliminary Determination of Epicentres) catalogue provided by the US Geological Survey (https://earthquake.usgs.gov/data/comcat/catalog/us/).

Examples

```
data(Phuket)
print(Phuket[1:10,])
```

plot *Plot Point Process Ground Intensity Function*

Description

Provides methods for the generic function plot.

Usage

```
## S3 method for class 'mpp'
plot(x, log=FALSE, ...)
## S3 method for class 'linksrm'
plot(x, region, log=FALSE, ...)
```

Arguments

x	an object with class "mpp" or "linksrm".
region	scalar, specifies the required region.
log	plot $\log \lambda_g(t|\mathcal{H}_t)$, default is FALSE.
...	other arguments.

Examples

```
data(Ogata)

p <- c(0.02, 70.77, 0.47, 0.002, 1.25)
TT <- c(0, 800)
bvalue <- 1

#   Note that the plot function does not utilise the
#   information about mark distributions, hence these
#   arguments can be NULL

x <- mpp(data=Ogata,
         gif=etas_gif,
         marks=list(NULL, NULL),
         params=p,
         gmap=expression(params[1:5]),
         mmap=NULL,
         TT=TT)

plot(x, log=TRUE)
```

`residuals` *Residuals of a Point Process Model*

Description

Provides methods for the generic function `residuals`.

Usage

```
## S3 method for class 'mpp'
residuals(object, ...)
## S3 method for class 'linksrm'
residuals(object, ...)
```

Arguments

`object`	an object with class `mpp` or `linksrm`.
`...`	other arguments.

Details

Let t_i be the times of the observed events. Then the transformed times are defined as

$$\tau_i = \int_0^{t_i} \lambda_g(t|\mathcal{H}_t)dt.$$

If the proposed point process model is correct, then the transformed time points will form a stationary Poisson process with rate parameter one. A plot of transformed time points versus the cumulative number of events should then roughly follow the straight line $y = x$. Significant departures from this line indicate a weakness in the model. Further details can be found in Ogata (1988) and Aalen & Hoem (1978).

See Baddeley et al (2005) and Zhuang (2006) for extensions of these methodologies.

Value

Returns a time series object with class "`ts`" in the case of `mpp`. In the case of `linksrm` a list is returned with the number of components being equal to the number of regions, and with each component being a time series object.

References

Cited references are listed on the PtProcess manual page.

Examples

```
TT <- c(0, 1000)
bvalue <- 1
params <- c(-2.5, 0.01, 0.8, bvalue*log(10))

x <- mpp(data=NULL,
         gif=srm_gif,
         marks=list(NULL, rexp_mark),
         params=params,
         gmap=expression(params[1:3]),
         mmap=expression(params[4]),
         TT=TT)
x <- simulate(x, seed=5)

tau <- residuals(x)

plot(tau, ylab="Transformed Time", xlab="Event Number")
abline(a=0, b=1, lty=2, col="red")

#   represent as a cusum
plot(tau - 1:length(tau), ylab="Cusum of Transformed Time", xlab="Event Number")
abline(h=0, lty=2, col="red")
```

---

simple_gif *Non-Homogeneous Poisson Processes*

Description

The functions listed here are intensity functions that are not conditional on the history of the process. Each has exactly the same "Usage" and calling format (see section "Value") as the function `simple_gif`. They are: `expfourier_gif`, `exppoly_gif`, `fourier_gif`, `poly_gif`, and `simple_gif`.

Usage

```
simple_gif(data, evalpts, params, TT=NA, tplus=FALSE)
```

Arguments

`data`	`NULL` or a data frame. The contents of this object are not used by these functions, though they retain this argument for consistency with other `gif` functions.
`evalpts`	a `vector`, `matrix` or `data.frame`. If a vector, the elements will be assumed to represent the required evaluation times. Other objects must include a column named `"time"` that can be referred to as `evalpts[,"time"]`, at which the intensity function will be evaluated.
`params`	vector of parameter values as required by the particular intensity function, see Details below.

TT	vector of length 2, being the time interval over which the integral of the intensity function is to be evaluated.
tplus	logical, $\lambda_g(t|\mathcal{H}_t)$ is evaluated as $\lambda_g(t^+|\mathcal{H}_t)$ if TRUE, else $\lambda_g(t^-|\mathcal{H}_t)$. Included for compatibility with others conditional intensity functions.

Details

The models are parameterised as follows.

`expfourier_gif` The vector of parameters is

$$(p, a_0, a_1, a_2, \cdots, a_n, b_1, b_2, \cdots, b_n)$$

and the intensity function is

$$\lambda_g(t) = \exp\left\{ a_0 + \sum_{j=1}^{n} a_j \cos\left(\frac{2j\pi t}{p}\right) + \sum_{j=1}^{n} b_j \sin\left(\frac{2j\pi t}{p}\right) \right\}.$$

The length of `params` is $2n + 2$, and determines the order of the fitted Fourier series. The numbers of specified sine and cosine coefficients must be the same. The integral is evaluated using numerical integration, using the R function `integrate`.

`exppoly_gif` The vector of parameters is $(b_0, b_1, b_2, \cdots, b_n)$ and the intensity function is

$$\lambda_g(t) = \exp\left\{ b_0 + \sum_{j=1}^{n} b_j t^j \right\}.$$

The length of `params` determines the order of the fitted polynomial. The integral is evaluated using numerical integration, using the R function `integrate`.

`fourier_gif` The Fourier intensity function is the same as `expfourier_gif`, except the intensity function omits the exponential, and the integration is performed explicitly.

`poly_gif` The polynomial intensity function is the same as `exppoly_gif`, except the intensity function omits the exponential, and the integration is performed explicitly.

`simple_gif` The intensity function is $\lambda_g(t) = a + bt^g$ and the vector of parameters is (a, b, g).

Value

Two usages are as follows.

```
simple_gif(data, evalpts, params, tplus=FALSE)
simple_gif(data, evalpts=NULL, params, TT=NA)
```

The first usage returns a vector containing the values of $\lambda_g(t)$ evaluated at the specified points. In the second usage, it returns the value of the integral.

Function Attributes

`rate` is "bounded".

See Also

General details about the structure of conditional intensity functions are given in the topic gif.

Examples

```
expfourier_gif(NULL, c(1.1,1.2,1.3), c(2,3,1,2,3,4), TT=NA)
#  Evaluates:  lambda_g(t) = exp(3 + 1*cos(2*pi*t/2) + 2*cos(4*pi*t/2) +
#                                3*sin(2*pi*t/2) + 4*sin(4*pi*t/2))
#  lambda_g(1.1) = 162.56331
#  lambda_g(1.2) = 127.72599
#  lambda_g(1.3) =  23.83979

expfourier_gif(NULL, NULL, c(2,3,1,2,3,4), TT=c(3,4))
#  Let:  lambda_g(t) = exp(3 + 1*cos(2*pi*t/2) + 2*cos(4*pi*t/2) +
#                            3*sin(2*pi*t/2) + 4*sin(4*pi*t/2))
#  Evaluates: integral_3^4 lambda_g(t) dt = 46.21920


#-----------------------------------------------------------
#   Plot intensity function: lambda(t) = 3 + 3*sin(t)
#   on interval (0, 6*pi), no marks

params <- c(2*pi, 3, 0, 3)
TT <- c(0, 6*pi)
x <- seq(TT[1], TT[2], length.out=500)

plot(x, fourier_gif(NULL, x, params, TT=NA),
     ylim=c(0, 6), type="l", axes=FALSE,
     xlab="t",
     ylab=expression(lambda(t) == 3 + 3*phantom(.)*plain(sin)*phantom(.)*t),
     main="Sinusoidal Intensity Function", font.main=1)
abline(h=params[2], lty=2, col="red")
box()
axis(2)
axis(1, at=0, labels=0)
axis(1, at=2*pi, labels=expression(2*pi))
axis(1, at=4*pi, labels=expression(4*pi))
axis(1, at=6*pi, labels=expression(6*pi))

#   Now define a model object
#   note NULL "marks" argument, see manual page for "mpp"
z <- mpp(data=NULL,
         gif=fourier_gif,
         marks=list(NULL, NULL),
         params=params,
         gmap=expression(params),
         mmap=NULL,
         TT=TT)

#   Simulate event times
z <- simulate(z, seed=3, max.rate=6)
```

```
#   Plot simulated times on sine curve
x <- z$data$time
points(x, fourier_gif(NULL, x, params, TT=NA), col="blue", lwd=5)

#   Number of simulated events
print(nrow(z$data))

#   Estimate parameters based on simulated data
parmap <- function(y, p){
    #    fix parameters 1 and 3
    y$params <- c(2*pi, p[1], 0, p[2])
    return(y)
}

initial <- c(3, 3)
y <- nlm(neglogLik, initial, object=z, pmap=parmap,
         print.level=2, iterlim=20, stepmax=0.1)
print(y$estimate)
```

`simulate` *Simulate a Point Process*

Description

Provides methods for the generic function `simulate`.

Usage

```
## S3 method for class 'mpp'
simulate(object, nsim=1, seed=NULL, max.rate=NA,
         stop.condition=NULL, ...)
## S3 method for class 'linksrm'
simulate(object, nsim=1, seed=NULL, max.rate=NA,
         stop.condition=NULL, ...)
```

Arguments

`object`	an object with class `"mpp"` or `"linksrm"`.
`nsim`	has no effect, and is only included for compatibility with the generic function `simulate`. See section "Length of Simulated Series" below for control information.
`seed`	seed for the random number generator.
`max.rate`	maximum rate, only used if the attribute of `object$gif` is `"bounded"`. It is the maximum value of `object$gif` on the simulation interval `object$TT`.
`stop.condition`	a function returning a logical value. It is called after the addition of each simulated event. The simulation continues until either `object$TT[2]` is exceeded or `stopping.condition` returns `TRUE`. See section "Length of Simulated Series" below for further information.
`...`	other arguments.

Details

The *thinning method* (Ogata, 1981; Lewis & Shedler, 1979) is used to simulate a point process with specified ground intensity function. The method involves calculating an upper bound for the intensity function, simulating a value for the time to the next *possible* event using a rate equal to this upper bound, and then calculating the intensity at this simulated point; hence these "events" are simulated too frequently. The ratio of this rate with the upper bound is compared with a uniform random number to randomly determine whether the simulated time is retained or not (i.e. thinned).

The functions need to calculate an upper bound for the intensity function. The ground intensity functions will usually be discontinuous at event times, but may be monotonically increasing or decreasing at other times. The ground intensity functions have an attribute called `rate` with values of `"bounded"`, `"increasing"` or `"decreasing"`. This information is used to determine the required upper bounded.

The function `simulate.linksrm` is currently only used in conjunction with `linksrm_gif`, or a variation of that function. It expects the `gif` function to have an attribute called `regions`, which may be an expression, being the number of regions. The method used by the function `simulate.linksrm` also assumes that the function is "increasing" (i.e. rate, summed over all regions, apart from discontinuous jumps), hence a positive tectonic input over the whole system.

Value

The returned value is an object of the same class as `object`. It will contain all events prior to `object$TT[1]` in `object$data` and all subsequently simulated events. Variables (columns) in `object$data` will be restricted to `"time"` and those for which a mark is simulated.

Length of Simulated Series

The interval of time over which events are simulated is determined by `object$TT`. Simulation starts at `object$TT[1]` and stops at `object$TT[2]`. The "current" dataset will consist of all events prior to `object$TT[1]` in `object`, plus subsequently simulated events. A more complicated stopping condition can be formulated by using the argument `stop.condition`.

The argument `stop.condition` can be assigned a function that returns a logical value. The assigned function is a function of the "current" dataset. It is executed near the bottom of `simulate.mpp` (check by printing the function). Simulation will then continue until either the stopping condition has been met or the current time exceeds `object$TT[2]`.

For example, we may want to simulate until the first earthquake with a magnitude of 8. Assume that the current dataset contains a variable with name `"magnitude"` (untransformed). We would then assign `Inf` to `object$TT[2]`, and write this condition as a function:

```
stop.cond <- function(data){
    n <- nrow(data)
    #   most recent event is the nth
    return(data$magnitude[n] >= 8)
}
```

References

Cited references are listed on the PtProcess manual page.

Examples

```
TT <- c(0, 1000)
bvalue <- 1
params <- c(-2.5, 0.01, 0.8, bvalue*log(10))

x <- mpp(data=NULL,
         gif=srm_gif,
         marks=list(NULL, rexp_mark),
         params=params,
         gmap=expression(params[1:3]),
         mmap=expression(params[4]),
         TT=TT)
x <- simulate(x, seed=5)

y <- hist(x$data$magnitude, xlab="Magnitude", main="")

#   overlay with an exponential density
magn <- seq(0, 3, length.out=100)
points(magn, nrow(x$data)*(y$breaks[2]-y$breaks[1])*
             dexp(magn, rate=1/mean(x$data$magnitude)),
       col="red", type="l")
```

`srm_gif` *Conditional Intensity for Stress Release Model*

Description

This function calculates the value of the conditional intensity of a Stress Release Model (SRM). Spatial coordinates of the events are not taken into account.

Usage

```
srm_gif(data, evalpts, params, TT=NA, tplus=FALSE)
```

Arguments

`data`	a data frame containing the event history, where each row represents one event. There must be columns named "time", usually the number of days from some origin; and "magnitude" which is the event magnitude less the magnitude threshold, i.e. $M_i - M_0$.
`evalpts`	a `vector`, `matrix` or `data.frame`. If a vector, the elements will be assumed to represent the required evaluation times. Other objects must include a column named `"time"` that can be referred to as `evalpts[,"time"]`, at which the intensity function will be evaluated.
`params`	vector of parameters for the proposed SRM model in the order (a, b, c).
`TT`	vector of length 2, being the time interval over which the integral of the conditional intensity function is to be evaluated.
`tplus`	logical, $\lambda_g(t\|\mathcal{H}_t)$ is evaluated as $\lambda_g(t^+\|\mathcal{H}_t)$ if `TRUE`, else $\lambda_g(t^-\|\mathcal{H}_t)$.

Details

Vere-Jones (1978) proposed the stress release model, being a stochastic version of elastic rebound theory (Reid, 1910). The SRM assumes a deterministic increase in stress over time, and a stochastic release through earthquake events. The conditional intensity function is

$$\lambda_g(t) = \exp\{a + b[t - cS(t)]\},$$

where

$$S(t) = \sum_i 10^{0.75(M_i - M_0)}$$

and the summation is taken over those i such that $t_i < t$, where t_i denotes the event times.

Value

Two usages are as follows.

```
srm_gif(data, evalpts, params, tplus=FALSE)
srm_gif(data, evalpts=NULL, params, TT)
```

The first usage returns a vector containing the values of $\lambda_g(t)$ evaluated at the specified points. In the second usage, it returns the value of the integral.

Function Attributes

`rate` is `"increasing"`.

Problems and Inconsistencies

Runs much slower than `linksrm_gif`. Should set up matrices `St1` and `St2` as in `linksrm_gif`.

References

Cited references are listed on the PtProcess manual page.

See Also

General details about the structure of conditional intensity functions are given in the topic `gif`.

Examples

```
#  Treating North China as one region

data(NthChina)
p <- c(-2.46, 0.0113, 0.851)
times <- seq(0, 517, 0.5)

par.default <- par(mfrow=c(2,1), mar=c(4.1, 4.1, 0.5, 1))
plot(times+1480, srm_gif(NthChina, times, params=p), type="l",
     ylab=expression(lambda[g](t)),
     xlab="", xlim=c(1480, 2000))
plot(NthChina$time+1480, NthChina$magnitude+6, type="h",
```

```
     xlim=c(1480, 2000), ylim=c(5.8, 8.6),
     xlab="Year", ylab="Magnitude")

par(par.default)
```

---

summary *Summary of a Point Process Model*

Description

Provides methods for the generic function summary.

Usage

```
## S3 method for class 'mpp'
summary(object, ...)
## S3 method for class 'linksrm'
summary(object, ...)
```

Arguments

object	an object with class "mpp" or "linksrm".
...	other arguments.

Value

A list object with a reduced number of components, mainly the parameter values.

Examples

```
TT <- c(0, 1000)
bvalue <- 1
params <- c(-2.5, 0.01, 0.8, bvalue*log(10))

x <- mpp(data=NULL,
         gif=srm_gif,
         marks=list(NULL, rexp_mark),
         params=params,
         gmap=expression(params[1:3]),
         mmap=expression(params[4]),
         TT=TT)
x <- simulate(x, seed=5)

print(summary(x))
```

Tangshan *Tangshan Earthquake and Aftershock Sequence*

Description

The Tangshan earthquake occurred on 28 July 1976 at 03:42:53, with a magnitude of 7.9. The Tangshan data frame contains those events (455) from the Beijing Catalogue, within 100 km of the epicentre and with magnitude 4 or greater, from the beginning of 1974 to the end of 1984.

Usage

```
data(Tangshan)
```

Format

This data frame contains the following columns:

latitude number of degrees north.

longitude number of degrees east.

magnitude number of magnitude units *above* 4.

year year of event (numeric vector).

month month of event, 1 . . . 12 (numeric vector).

day day of event, 1 . . . 31 (numeric vector).

hour hour of event, 0 . . . 23 (numeric vector).

minute minute of event, 0 . . . 59 (numeric vector).

second second of event, 0 . . . 59 (numeric vector).

time number of days (and fractions) from the beginning of 1974.

Source

These data originate from the Beijing Catalogue which is administered by the China Seismological Bureau, Beijing.

Examples

```
data(Tangshan)
print(Tangshan[1:10,])
```

Index