

Package ‘tripEstimation’

October 12, 2009

Type Package

Title Metropolis sampler and supporting functions for estimating animal movement from archival tags and satellite fixes

Version 0.0-29

Depends zoo, mgcv, lattice

Suggests rgdal

Author Michael Sumner and Simon Wotherspoon

Maintainer Michael Sumner <mdsumner@utas.edu.au>

Description Data handling and estimation functions for animal movement estimation by light level and satellite data. Image summaries from MCMC simulations of point data, binned by time interval. Various convenience functions for creating generic summary images for periods defined by the location estimation and combining those in arbitrarily specified ways.

License GPL-2

URL <http://staff.acecrc.org.au/~mdsumner/Rutas/>

Repository CRAN

Date/Publication 2009-10-12 13:31:00

R topics documented:

as.image.pimg	2
astro	3
behav.bin	5
bits	6
chain.read	7
elevation	8
get.mask	8
initialize.x	10
julday	11

k.prior	12
metropolis	12
mkCalibration	13
mkLookup	14
norm.proposal	15
old.metropolis	16
pick	18
pimg.list	19
satellite.model	20
solar	21
solar.model	22

Index 25

as.image.pimg	<i>Convert to image list</i>
---------------	------------------------------

Description

Converts Probability image (pimage) component to standard R xyz list image.

Usage

```
as.image.pimg(pimg)
combine(pimgs, subset = 1:length(pimgs))
coords.pimg(pimg)
get.pimgs(rootdir = NULL, f2load = "p_ImageList.Rdata")
unzipper(px)
as.local.pimg(pimg)
## S3 method for class 'pimg':
as.matrix(x, ...)
```

Arguments

pimg	Probability image component
pimgs	pimgs
subset	subset
rootdir	rootdir
f2load	f2load
px	px
x	x
...	...

Author(s)

Michael D. Sumner

`astro`*Calculations for position of the sun and moon*

Description

This set of functions provides simple position calculations for the sun and moon, taken from Pascal routines published in Montenbruck and Pfleger (1994, Dunlop).

These are completely independent from the (specifically optimized) solar elevation calculations available via [elevation and solar].

Usage

`astro(lon, lat, astro.calc)``EQUHOR(DEC, TAU, PHI)``FRAC(x)``LMST(MJDay, LAMBDA)``lunar(time)``mini.sun(time)``MJD(date)``POLAR(X, Y, Z)`

Arguments

<code>lon</code>	vector of longitudes
<code>lat</code>	vector of latitudes
<code>astro.calc</code>	list object containing RA right ascension
<code>DEC</code>	declination
<code>TAU</code>	TAU
<code>PHI</code>	PHI
<code>x</code>	number
<code>MJDay</code>	modified julian day
<code>LAMBDA</code>	LAMBDA
<code>time</code>	vector of date-times in POSIXct format
<code>date</code>	vector of date-times in POSIXct format
<code>X</code>	x-coordinate
<code>Y</code>	y-coordinate
<code>Z</code>	z-coordinate

Value

`astro` returns a list object with the components of the moon or sun's position,

<code>r</code>	rho component
<code>theta</code>	theta component - elevation
<code>phi</code>	phi component - azimuth

Warning

Some of this could be faster (particularly the use of LMST in "astro" is not precalculated)

Note

Thanks to Nick.Ellis@csiro.au for pointing out a mistake pre-0.0-27

Author(s)

Michael D. Sumner

References

```
@BOOK{,
  title = {Astronomy on the Personal Computer},
  publisher = {Springer-Verlag, Berlin},
  year = {1994},
  author = {Oliver Montenbruck and Thomas Pfleger},
  edition = {2 (translated from German by Storm Dunlop)},
}
```

See Also

See Also [elevation](#)

Examples

```
## the moon
tm <- Sys.time() + seq(by = 3600, length = 100)
moon <- lunar(tm)
rtp <- astro(147, -42, moon)
op <- par(mfrow = c(2,1))
plot(tm, rtp$theta, main = "lunar elevation, Hobart")
plot(tm, rtp$phi, main = "lunar azimuth, Hobart")
par(op)

## the sun
tm <- Sys.time() + seq(by = 3600, length = 100)
sun <- mini.sun(tm)
rtp <- astro(147, -42, sun)
op <- par(mfrow = c(2,1))
```

```
plot(tm, rtp$theta, main = "solar elevation, Hobart")
plot(tm, rtp$phi, main = "solar azimuth, Hobart")
par(op)
## Not run:
  elev.gmt <- mkElevationSeg(1, tm)
  plot(tm, rtp$theta, main = "solar elevation mini.sun versus NOAA")
  lines(tm, elev.gmt(1, 147, -42))
## End(Not run)
```

behav.bin

Bin MCMC chains.

Description

Bin MCMC chains in probability image summaries.

Usage

```
behav.bin(z, pimgs, weights = NULL)
bin.pimg(pimg, xy, w = 1)
chunk.bin(filename, pimgs, weights = NULL, chunk = 2000, proj = NULL)
```

Arguments

z	z
pimgs	pimgs
weights	weights
pimg	pimg
xy	xy
w	w
filename	filename
chunk	chunk
proj	proj

bits *Set and get bits from binary masks.*

Description

Utility functions to access bits from numeric values, for the efficient storage of spatial masks.

Usage

```
bits(object, bit)

bits(object, bit) <- value
```

Arguments

object	a numeric value
bit	the desired bit
value	logical value to set bit to

Details

R uses 32-bit integers, so we can (easily) access 31 binary matrices in each numeric matrix. This is very useful for storing long time-series of spatial masks, required for track-location estimation from archival tags.

Value

A numeric object with the given bit set, or a logical value designating the status of the given bit.

Note

The 32nd bit is harder to access, so we ignore it.

Author(s)

Michael D. Sumner

See Also

See Also [get.mask](#) for a higher level access of a mask object

Examples

```
a <- 1
bits(a, 0) ## 1
bits(a, 2) <- 1
a ## 5
```

chain.read *~~function to do ... ~~*

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

```
chain.read(filename)
chain.dim(filename)
chain.write(filename, A, append = FALSE)
```

Arguments

```
filename        ~~Describe filename here~~
A               ~~Describe A here~~
append         ~~Describe append here~~
```

Note

~~further notes~~

Author(s)

~~who you are~~

References

~put references to the literature/web site here ~

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==> Define data, use random,
##--     or do help(data=index) for the standard data sets.

## The function is currently defined as
function(filename) {
  con <- file(filename,open="rb")
  ## Read the dimensions of the array
  seek(con,0,origin="start",rw="r")
  dm <- readBin(con,"integer",3)
  ## Read the data and set dimensions
  n <- dm[1]*dm[2]*dm[3]
  A <- readBin(con,"double",n)
  if(length(A)!=n) stop("Read Error\n")
  close(con)
}
```

```
array(A, dm)
}
```

```
elevation          ~~ data name/kind ... ~~
```

Description

~~ A concise (1-5 lines) description of the dataset. ~~

Usage

```
elevation(lon, lat, sun)
```

Arguments

lon	vector of longitude values
lat	vector of latitude values
sun	solar calculations as returned by solar

Author(s)

Michael D. Sumner

References

[NOAA webpage]

```
get.mask          Create, access and manipulate spatial masks
```

Description

Spatial masks are stored using the xyz-list structure used by [image](#) or as a series of masks stored as bits in the z-component as matrix or array object. `get.mask` is used to extract a specific mask from the binary storage, and `mkSmall` can be used to quickly down-sample an existing mask or `image`.

Usage

```
get.mask(masks, k)

mkSmall(lst, thin = 10)
```

Arguments

mask	A list object with components x, y, and z containing spatial masks
k	specifies the k-th mask
lst	an xyz-list structure with z containing either a matrix or array
thin	integer factor to down-sample grid

Author(s)

Michael D. Sumner

See Also

[mkLookup](#) for the use of these masks to query individual locations and locations measured over time. See [bits](#) for the underlying mechanism to set and get mask bits.

For the use of the xyz-list structure see [image](#).

Examples

```

data(volcano)
d <- list(x = seq(-10, 10, length = nrow(volcano)),
         y = seq(-5, 5, length = ncol(volcano)),
         z = array(0, c(nrow(volcano), ncol(volcano), 2)) )
mv <- min(volcano)

for (i in 0:61) {
  blk <- (i %% 31) + 1
  bit <- (i - 1) %% 31
  bits(d$z[,blk], bit) <- volcano > (mv + i*1.6 )
}
for (i in 0:61) image(get.mask(d, i))

## an object with 62 masks is only twice the size of the source data
object.size(d) / object.size(volcano)

## Not run:
## plot a smaller version
image(get.mask(d, 20), 5)

## pretend we have only one masks
lookup <- mkLookup(get.mask(d, 30), by.segment = FALSE)

## interactive to show use of lookup function
image(get.mask(d, 30), main = "Click on the red (FALSE) and cream (TRUE) areas")
for (i in 1:10) {x <- matrix(unlist(locator(1)), ncol = 2);text(x[1], x[2], lookup(x))}

## End(Not run)

```

```
initialize.x           Diagnose and initialize light level estimation.
```

Description

Primarily for the purposes of initializing the estimation, these functions can also be used for diagnostic purposes. `position.loggp` both produces diagnostic images for each twilight and uses those to initialize position.

Usage

```
position.loggp(model, x1, x2, xrest = NULL, subset = 1:model$n,
initialize.x = TRUE, start = NULL, end = NULL, prob = 0.8)

initialize.x(model, x1, x2, xrest = NULL)

light.quantile(model, chain, day, seg, probl = c(0.025, 0.5, 0.975))

show.segment(model, chain, segment, day, light, k, n = 50, ...)
```

Arguments

<code>model</code>	estimation model object
<code>x1</code>	vector of x-coordinates defining the prior grid
<code>x2</code>	vector of y-coordinates defining the prior grid
<code>xrest</code>	value for remaining parameters - default is light attenuation
<code>subset</code>	evaluate subset of segments - default uses all
<code>initialize.x</code>	logical - create initial points for x?
<code>prob</code>	probability - threshold to apply to overlapping quantiles, defaults to 0.8
<code>chain</code>	chain object from estimation
<code>day</code>	POSIXct vector of date-times
<code>seg</code>	desired segment
<code>probl</code>	probability level for quantile
<code>start</code>	known position of release
<code>end</code>	known position of recapture
<code>segment</code>	vector of segment data
<code>light</code>	vector of light data
<code>k</code>	desired segment to show
<code>n</code>	length of vector to evaluate
<code>...</code>	additional arguments to be passed to plot

Details

The primary function here is `position.logp`, for initializing the estimation for `solar.model` and `metropolis0`.

Author(s)

Michael D. Sumner

julday

Julian day and Julian century calculations from date-time values

Description

[definition and rationale of Julian]

Usage

`julday(tm)`

`julcent(time)`

Arguments

`tm` vector of date-times

`time` vector of date-times

Author(s)

Michael D. Sumner

References

[Astro and NOAA]

<code>k.prior</code>	<i>simple prior</i>
----------------------	---------------------

Description

This function is used by the metropolis sampler

Usage

```
k.prior(seg, ps)
```

Arguments

seg

ps

<code>metropolis</code>	<i>Metropolis-Hastings sampler for location estimation for archival and satellite tag</i>
-------------------------	---

Description

These functions provide a direct implementation of the Metropolis-Hastings algorithm, for calculating marginal posterior (locations and full-track estimates) properties using Markov Chain Monte Carlo. The sampler is written completely in R, vectorized to be as fast as possible. The sampler can include likelihood functions for large data records (including light and water temperature), as well as *mask* functions for simpler rejection sources. Behavioural constraints are implemented using a red/black update, so that location estimates X and Z may be estimated in an efficient manner. The parameter estimates may be cached and later queried arbitrarily.

Usage

```
metropolis(model, iters = 1000, thin = 10, start.x = NULL, start.z = NULL)
```

```
metropolis0(model, iters = 1000, thin = 10, start.x = NULL, start.z = NULL)
```

Arguments

<code>model</code>	~~Describe model here~~
<code>iters</code>	~~Describe iters here~~
<code>thin</code>	~~Describe thin here~~
<code>start.x</code>	~~Describe start.x here~~
<code>start.z</code>	~~Describe start.z here~~

Details

`metropolis0` is a slightly different version of `metropolis` that enables an initialization step, required to find parameter estimates that are consistent with any masks used. It is difficult to make this step more elegant, and so we live with the two versions.

In terms of the estimates, X's have m records with n parameters, where m is the number of data records in time (twilights for archival tags, Argos estimates for satellite tags) and n is at least x-coordinate, y-coordinate and maybe k-attenuation for light. Z's have $m-1$ records with 2 parameters for 'x' and 'y' (which are usually Longitude and Latitude). These parameters may be increased or changed, they are tied only to the likelihood functions used, not the sampler itself. Also, coordinate transformations may be used inside the model and likelihood functions, in order to use an appropriate map projection. Solar calculations rely on lon/lat and so this step does slow down light level geo-location.

Value

A MCM *Chain* stored as a list containing

<code>model</code>	The model object used by the sampler
<code>x</code>	The last <code>iters</code> X-samples accepted, stored as an <code>c(m, n, iters)</code> array
<code>z</code>	The last <code>iters</code> Z-samples accepted, stored as an <code>c(m - 1, 2, iters)</code>
<code>last.x</code>	The last accepted X-sample, stored as a <code>c(m, n)</code> matrix
<code>last.z</code>	The last accepted Z-sample, stored as a <code>c(m, 2)</code> matrix

Author(s)

Michael D. Sumner and Simon Wotherspoon

References

Sumner, Wotherspoon and Hindell (2009). Bayesian Estimation of Animal Movement from Archival and Satellite Tags, PLoS ONE. <http://dx.plos.org/10.1371/journal.pone.0007324>

See Also

[solar.model](#), [satellite.model](#)

mkCalibration

Create calibration of solar elevation to measured light level.

Description

Using a set of light level data from a known location create a calibration function to return the expected light level given solar elevation.

Usage

```
mkCalibration(x, known = NULL, elim = c(-36, 12), choose = TRUE)
```

Arguments

<code>x</code>	a data frame containing at least <code>gmt</code> and <code>light</code>
<code>known</code>	a known position - as a 2-element <code>c(x, y)</code> coordinate
<code>elim</code>	a 2-element vector of the range of solar elevation to define
<code>choose</code>	logical - choose segments from a plot or use all the data?

Details

It is assumed that the data frame `x` has columns "gmt" with POSIXct date-times and "light" with numeric light level data.

Value

A function, defined by `approxfun`.

Author(s)

Michael D. Sumner

References

~

See Also

[approxfun](#)

mkLookup

Create a lookup function to query locations against spatial masks

Description

Simple pixel spacing is used to overlay point locations on a spatial grid, or a series of grids.

Usage

```
mkLookup(x, by.segment = TRUE)
```

Arguments

<code>x</code>	an xyz-list with matrix or array of masks
<code>by.segment</code>	logical - is the mask to be queried separately for each time step?

Value

A function, with one argument - a matrix of points - that returns a logical vector indicating the overlay of each point against the masks.

Note

Very little error checking is done.

Author(s)

Michael D. Sumner

See Also

[get.mask](#) and related examples for creating and using masks.

See [overlay](#) for more general capabilities for overlays.

 norm.proposal

Manage proposal functions tune variance for metropolis sampler

Description

Generate new proposals for the x from the current. Generates all x at once.

Usage

```
norm.proposal(m, n, sigma)
```

```
mvnorm.proposal(m, n, Sigma)
```

```
bmvnorm.proposal(m, n, Sigma)
```

Arguments

m	number of records
n	number of parameters
sigma	variance
Sigma	variance

Details

norm.proposal - Independent Normal proposal - every component is independent, with variances of individual components determined by sigma. The recycling rule applies to sigma, so sigma may be a scalar, an m vector or a m by n matrix.

mvnorm.proposal - Multivariate Normal proposal - all components of all points are correlated. In this case Sigma is the joint covariance of the m*n components of the proposal points.

bmvnorm.proposal - Block Multivariate Normal proposal - components of points are correlated, but points are independent. Here Sigma is an array of m covariance matrices that determine the covariance of the m proposal points.

Value

An list object with get, set and tune functions to manage the state of the proposals.

proposal	propose new set of parameters from last
get	get variance values
set	set variance values
tune	tune the variance for proposal functions

Note

The mv and bmv versions are included for archival purposes - they are not tested.

Author(s)

Simon Wotherspoon

old.metropolis *Older versions of solar location estimation*

Description

Some deprecated functions, originally used purely for light level estimation before the sampling algorithm was generalized for satellite models as well.

Usage

```
mkElevationSeg(segments, day)

mkNLPosterior(segments, day, light, calib)

old.dist.gc(x1, x2 = NULL)

old.find.init(mask, nseg, nlpost, pars = c("Lon", "Lat", "k"))

old.metropolis(nlpost, lookup, p0, cov0, start, end, iter = 1000, step = 100)

old.mkLookup(x, binArray = TRUE)
```

Arguments

segments	~~Describe segments here~~
day	~~Describe day here~~
light	vector of light data
calib	calibration function for light levels
x1	matrix of track locations

<code>x2</code>	matrix of track locations (optional second part)
<code>mask</code>	image object of masked areas
<code>nseg</code>	number of (twilight) segments
<code>nlpost</code>	negative log posterior function
<code>pars</code>	names of parameters
<code>lookup</code>	lookup function for masked areas
<code>p0</code>	initial locations for sampler
<code>cov0</code>	covariance matrix for sampler
<code>start</code>	known start parameters
<code>end</code>	known end parameters
<code>iter</code>	number of iterations
<code>step</code>	number of thinning iterations per <code>iter</code>
<code>x</code>	image-like object of matrix or array of binary masks
<code>binArray</code>	logical: are the masks compressed into bits?

Details

These functions are included for archival purposes and are not fully supported.

Value

If it is a LIST, use

<code>comp1</code>	Description of 'comp1'
<code>comp2</code>	Description of 'comp2'

Author(s)

Michael D. Sumner

See Also

Please use the supported function `metropolis`, with the `solar.model` or `satellite.model`.

pick *Choose twilight segments interactively from light data.*

Description

`pick` plots up series of light data against record ID, allowing the user to click on the beginnings and ends of twilight in sequence. `picksegs` generates a vector of segment IDs for each record.

Usage

```
pick(id, val, nsee = 10000)
```

```
picksegs(twind, n)
```

Arguments

<code>id</code>	index vector to identify records
<code>val</code>	sequence of data (light levels) to choose segments from
<code>nsee</code>	number of points to plot per screen
<code>twind</code>	vector of index pairs generated by <code>pick</code>
<code>n</code>	Number of segments values required - length of record

Details

Choosing twilight segments interactively seems far easier than trying to automate it. Mark Hindell makes the point that you get to see the data in detail, which is good.

Value

`pick` returns a vector where each value (obtained using `locator` is the x coordinate for the begin or end of a twilight.

`picksegs` uses these paired indexes to return a vector of segment IDs, with NAs for non-twilight periods.

Warning

Segments are expected to be chosen as non-overlapping.

Note

It seems best to choose more of the light data than less, using the `ekstrom` keyword to `solar.model` we can limit the solar elevation used.

Author(s)

Michael D. Sumner

Examples

```
## Not run:

d <- sin(seq(0, 10, by = 0.01))
id <- 1:length(d)
## choose a series of start-begin pairs
pk <- pick(id, d, 1000)
## your start/ends should be marked as blue versus red
plot(id, d, col = c("red", "blue")[is.na(picksegs(pk, 1000))+1])

## End(Not run)
```

pimg.list

Create a collection of probability images, for MCMC binning.

Description

Pimage lists.

Usage

```
pimg(xmin, xmax, xn, ymin, ymax, yn)
pimg.list(times, xlim, ylim, img.dim, Z = TRUE)
```

Arguments

xmin	xmin
xmax	xmax
xn	xn
ymin	ymin
ymax	ymax
yn	yn
times	times
xlim	xlim
ylim	ylim
img.dim	img.dim
Z	Z

satellite.model *Function to create a satellite model object for metropolis location sampler*

Description

A model to manage likelihood functions, environmental masks and behavioural likelihood functions for pre-derived satellite locations. There are some options for configuration, but this may be considered a template for any given model. The model *function* exists simply to make the object construction simple.

Arguments

day	vector of date-times for each light level
X	matrix of pre-derived satellite locations
proposal.x	function from object managing X proposals
proposal.z	function from object managing Z proposals
mask.x	lookup function for X's against masks
mask.z	lookup function for Z's against masks
fix.release	logical - is the release point known?
fix.recapture	logical - is the recapture point known?
start.x	~~Describe start.x here~~
start.z	~~Describe start.z here~~
posn.sigma	variance for locations
behav.dist	distribution to use for behavioural constraint
behav.mean	mean to use for behavioural distribution
behav.sd	variance for behavioural distribution
proj.string	PROJ.4 string for coordinate system used

Details

posn.sigma may be a single value for all estimates, or a vector of values for each position estimate.

Transformation of coordinates is supported via a simple function that only performs coordinate transforms if proj.string is not longlat. See [project](#) for the underlying functionality.

Value

See solar.model for some related detail.

Note

This function is not completely consistent with its solar counterpart `solar.model`. These are working functions that are in constant development, but they are structurally simple and may be easily edited as required.

Author(s)

Michael D. Sumner

References

[Argos manual]

See Also

See also [solar.model](#) for the counterpart model for solar problems

solar

Calculate solar position parameters

Description

Pre-calculates astronomical solar position components for Earth-location sampling functions.

Usage

```
solar(day)
```

Arguments

day vector of date-time values

Value

A list of the following values for each input time:

```
solarTime     solar time  
sinSolarDec   sine solar declination  
cosSolarDec   cosine solar declination
```

Note

No account is made for horizon refraction, but this was available in the original (Javascript) code.

Author(s)

Michael D. Sumner

References

[NOAA website]

solar.model	<i>Function to create a solar model object for metropolis location sampler</i>
-------------	--

Description

A solar model to manage likelihood functions, environmental masks and behavioural likelihood functions. There are several options for configuring the model, and this may be considered a template for any given model. The model *function* exists simply to make the object construction simple.

Usage

```
solar.model(segments, day, light,
            proposal.x, proposal.z, mask.x, mask.z,
            fix.release = TRUE, fix.recapture = TRUE,
            calibration,
            light.sigma = 7, k.sigma = 10,
            behav = "speed", behav.dist = "gamma",
            behav.mean, behav.sd,
            proj.string = "+proj=longlat",
            ekstrom = c(-5, 3, light.sigma),
            ekstrom.limit = "light")
```

Arguments

segments	vector identifying twilight segment
day	vector of date-times for each light level
light	vector of light levels
proposal.x	function from object managing X proposals
proposal.z	function from object managing Z proposals
mask.x	lookup function for X's against masks
mask.z	lookup function for Z's against masks
fix.release	logical - is the release point known?
fix.recapture	logical - is the recapture point known?
calibration	calibration function for predicted light level for solar elevation
light.sigma	variance for light data
k.sigma	variance for light attenuation
behav	model distributions to be used for behaviour - defaults to "speed"
behav.dist	distribution to be used for behaviour

<code>behav.mean</code>	mean for behavioural distribution
<code>behav.sd</code>	variance for behavioural distribution
<code>proj.string</code>	PROJ.4 string for coordinate system used
<code>ekstrom</code>	parameters to use for ekstrom limit - min elevation, max elevation, sigma for outside that range
<code>ekstrom.limit</code>	mode of ekstrom limit to impose - defaults to "light"

Details

The vectors of `segments`, `day` and `light` are expected to be of the same length.

Fixed recapture and release points are treated specially for ease of sampling, but the sampling is written to be general for any fixed locations.

Behavioural models may be specified either as lognormal or log-gamma. By editing the function created as `logp.behavioural` this may be specified differently.

Transformation of coordinates is supported via a simple function that only performs coordinate transforms if `proj.string` is not `longlat`. See [project](#) for the underlying functionality.

Value

`proposal.x(x)` - generates new proposals for the `x` from the current `x`. Generates all `x` at once.

`proposal.z(z)` - generates new proposals for the `x` from the current `z`. Generates all `z` at once.

`mask.x(x)` - mask function for the `x`. Simultaneously tests all `x` and returns a vector of booleans indicating which are acceptable.

`mask.z(z)` - mask function for the `z`. Simultaneously tests all `z` and returns a vector of booleans indicating which are acceptable.

`logp.position(x)` - Given the set of `x`, returns a vector that gives the contribution each `x` make to the log posterior based on position alone.

`logp.behaviourial(k,xa,z,xb)` - Computes the contribution to the log posterior from the behavioural model on a subset of segments that make up the path. Here `k` is a vector of the segment numbers, where the segments pass from `xa` to `z` to `xb`, and the function returns the contribution to the log posterior from each segment. This is the only function expected to work with only a subset of the `x` and `z`.

`start.x` - suggested starting points for the `x`

`start.z` - suggested starting points for the `z`

The only function that must operate on a subset of the `x/z` is `logp.behaviourial`. All the other functions operate on all `x` or `z` simultaneously, simplifying the implementation for the user.

Note that `x` can consist of several parameters, not just the locations, but we assume the first two components of each `x` specify the location. For example, in the light level models each `x` is `(lon,lat,k)` where `k` is the attenuation of the light level.

Some details of this implementation are not as nice as they could be. First, it would be better if did not calculate the contributions to the posterior for points the mask rejects. Also, it may be better to separate the specification of the functions that generate proposals from the other functions, so that we can tune the proposal distributions without re-generating the whole model specification.

Author(s)

Simon Wotherspoon and Michael Sumner

Index

*Topic **dplot**

initialize.x, 9
mkCalibration, 13
pick, 17

*Topic **manip**

as.image.pimg, 2
astro, 2
behav.bin, 5
bits, 5
chain.read, 6
elevation, 7
get.mask, 8
initialize.x, 9
julday, 11
k.prior, 11
metropolis, 12
mkCalibration, 13
mkLookup, 14
norm.proposal, 15
pick, 17
pimg.list, 18
satellite.model, 19
solar, 20

*Topic **misc**

old.metropolis, 16

*Topic **models**

solar.model, 21

approxfun, 14

as.image.pimg, 2
as.local.pimg (as.image.pimg), 2
as.matrix.pimg (as.image.pimg), 2
astro, 2

behav.bin, 5
bin.pimg (behav.bin), 5
bits, 5, 8
bits<- (bits), 5
bmvnorm.proposal (norm.proposal),
15

chain.dim (chain.read), 6
chain.read, 6
chain.write (chain.read), 6
chunk.bin (behav.bin), 5
combine (as.image.pimg), 2
coords.pimg (as.image.pimg), 2

elevation, 4, 7
EQUHOR (astro), 2

FRAC (astro), 2

get.mask, 6, 8, 14
get.pimgs (as.image.pimg), 2

image, 8
initialize.x, 9

julcent (julday), 11
julday, 11

k.prior, 11

light.quantile (initialize.x), 9
LMST (astro), 2
locator, 18
lunar (astro), 2

metropolis, 12, 12, 17
metropolis0, 10, 12
metropolis0 (metropolis), 12
mini.sun (astro), 2
MJD (astro), 2
mkCalibration, 13
mkElevationSeg (old.metropolis),
16
mkLookup, 8, 14
mkNLPosterior (old.metropolis), 16
mkSmall (get.mask), 8
mvnorm.proposal (norm.proposal),
15

`norm.proposal`, 15

`old.dist.gc` (*old.metropolis*), 16

`old.find.init` (*old.metropolis*), 16

`old.metropolis`, 16

`old.mkLookup` (*old.metropolis*), 16

`overlay`, 14

`pick`, 17

`picksegs` (*pick*), 17

`pimg` (*pimg.list*), 18

`pimg.list`, 18

`POLAR` (*astro*), 2

`position.logp` (*initialize.x*), 9

`project`, 20, 22

`satellite.model`, 13, 17, 19

`show.segment` (*initialize.x*), 9

`solar`, 7, 20

`solar.model`, 10, 13, 17, 18, 20, 21

`unzipper` (*as.image.pimg*), 2