Package ‘tripEstimation’

October 14, 2022

Type Package

Title Metropolis Sampler and Supporting Functions for Estimating Animal Movement from Archival Tags and Satellite Fixes

Version 0.0-44

Imports lattice, mgcv, rgdal, sp, zoo

Description Data handling and estimation functions for animal movement estimation from archival or satellite tags. Helper functions are included for making image summaries binned by time interval from Markov Chain Monte Carlo simulations.

License GPL-3

NeedsCompilation no

ByteCompile yes

URL https://github.com/mdsumner/tripEstimation

Author Michael D. Sumner [aut, cre], Simon Wotherspoon [ctb]

Maintainer Michael D. Sumner <mdsumner@gmail.com>

Repository CRAN

Date/Publication 2016-01-31 12:22:31

R topics documented:

as.image.pimg ................................................................. 2
astro ................................................................. 3
behav.bin ............................................................... 5
bits ................................................................. 6
chain.read ............................................................... 7
elevation ................................................................. 7
get.mask ................................................................. 8
get.sst ................................................................. 9
initialize.x ............................................................. 11
julday ................................................................. 12
**as.image.pimg**

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

**Usage**

```r
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**

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

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

- `r` (rho component)
- `theta` (theta component - elevation)
- `phi` (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**

```r
@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**

```r
## 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))
```

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
Description

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

Usage

\[
\text{bits} \text{(object, bit)} \\
\text{bits} \text{(object, bit)} \leftarrow \text{value}
\]

Arguments

- **object**: a numeric value
- **bit**: the desired bit
- **value**: logical value to set bit to

Details

\text{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 \text{get.mask} for a higher level access of a mask object

Examples

\begin{verbatim}
a <- 1
bits(a, 0)  ## 1
bits(a, 2) <- 1
a  # 5
\end{verbatim}
Description

These functions read and write to cache files for storing long MCMC outputs from model functions, such as `solar.model` or `satellite.model`.

Usage

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

Arguments

filename       cache file for model chain
A              chain array
append         append to existing file or overwrite?

Author(s)

Michael D. Sumner and Simon Wotherspoon

See Also

pimg.list

elevation    Calculate elevation of astronomical objects

Description

Function to calculate elevation.

Usage

elevation(lon, lat, sun)

Arguments

lon       vector of longitude values
lat       vector of latitude values
sun       pre-stored values as returned by `solar` or `lunar`
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

```r
get.mask(masks, k)
mkSmall(lst, thin = 10)
set.mask(object, segment) <- value
mkMaskObject(xs, ys, nsegs)
```

Arguments

- `masks`: 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
- `object`: array Mask object
- `segment`: segment number to be modified in the mask
- `value`: individual mask to be set
- `xs`: x coordinates of mask cells
- `ys`: y coordinates of mask cells
- `nsegs`: number of segments to be represented

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) > 0)}

## End(Not run)
Arguments

xlim  Longitude range of SST to read.
ylim  Latitude range of SST to read.
tlim  Time range of SST to read.
server  Server to read from.

Details

This functionality was originally taken from the kfsst package by Anders Nielsen.
Longitude xlim range may be specified in [-180, 180] or [0, 360] convention. The result is given in [-180, 180].
Time range must be given in POSIXct form.

Value

A list object with the temperature data and coordinates.

x, y, t  Coordinates of the temperature data
z  Matrix or array of temperature data

Author(s)

Michael D. Sumner

References

Original concept taken from kfsst: http://www.soest.hawaii.edu/tag-data/tracking/kfsst/

See Also

See Also mkMaskObject for the use of these data for masks.

Examples

## Not run:
d0 <- read.url.sst(tlim = range(ISOdatetime(2009, 1:3, 1, 0, 0, 0)))
image(d0$x, d0$y, d0$z[,1], axes = TRUE)

## 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.logp` produces grids of simplistic position likelihood for each twilight and uses those to initialize positions for running estimations.

**Usage**

```r
position.logp(model, x1, x2, xrest = NULL, subset = 1:model$n,
initialize.x = TRUE, start = NULL, end = NULL, prob = 0.8, winoffset = 5)
```

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

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

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

**Arguments**

- `model`: estimation model object
- `x1`: vector of x-coordinates defining the prior grid
- `x2`: vector of y-coordinates defining the prior grid
- `xrest`: value for remaining parameters - default is light attenuation
- `subset`: evaluate subset of segments - default uses all
- `initialize.x`: logical - create initial points for x?
- `prob`: probability - threshold to apply to overlapping quantiles, defaults to 0.8
- `winoffset`: an odd-numbered window size to use when intersecting subsequent segments - defaults to 5
- `chain`: chain object from estimation
- `day`: POSIXct vector of date-times
- `seg`: desired segment
- `probl`: probability level for quantile
- `start`: known position of release
- `end`: known position of recapture
- `segment`: vector of segment data
- `light`: vector of light data
- `k`: desired segment to show
- `n`: length of vector to evaluate
- `...`: 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

Date values required by `solar`.

Usage

```r
julday(tm)
```

```r
julcent(time)
```

Arguments

- `tm` vector of date-times
- `time` vector of date-times

Author(s)

Michael D. Sumner

References

[http://www.srrb.noaa.gov/highlights/sunrise/azel.html](http://www.srrb.noaa.gov/highlights/sunrise/azel.html)
metropolis

**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**

```r
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**

- `model`: model for estimation, such as one created by `solar.model`
- `iters`: number of iterations to run
- `thin`: number of iterations to thin by
- `start.x`: starting points for the primary locations
- `start.z`: starting points for the intermediate locations (midpoints between the `start.x` points is a good first guess)

**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

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

**Author(s)**

Michael D. Sumner and Simon Wotherspoon

**References**


**See Also**

`solar.model`, `satellite.model`
**Description**

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

**Usage**

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

**Arguments**

- `x`: an xyz-list with matrix or array of masks
- `by.segment`: 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 `over` 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)
mvn.norm.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.
mvnnorm.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

Author(s)

Simon Wotherspoon
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)

k.prior(seg, ps)

Arguments

segments vector identifying the segment of each time and light value
day date-time values in POSIXct
light vector of light data
calib calibration function for light levels
x1 matrix of track locations
x2 matrix of track locations (optional second part)
mask image object of masked areas
nseg number of (twilight) segments
nlpost negative log posterior function
pars names of parameters
lookup lookup function for masked areas
p0 initial locations for sampler
cov0 covariance matrix for sampler
start known start parameters
end known end parameters
iter number of iterations
step number of thinning iterations per iter
x image-like object of matrix or array of binary masks
binArray logical: are the masks compressed into bits?
seg segment
ps light attenuation value

Details
These functions are included for legacy purposes, this was the original implementation.

Value
If it is a LIST, use

Author(s)
Michael D. Sumner

See Also
Please use the more up to date function metropolis, with the models such as solar.model or satellite.model.

---

pick Choose twilight segments interactively from light data.

Description
pick plots up series of light data agains 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
id index vector to identify records
val sequence of data (light levels) to choose segments from
nsee number of points to plot per screen
twind vector of index pairs generated by pick
n Number of segments values required - length of record
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

```r
## 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

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

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**: starting positions for the primary locations, see `position.logp`
- **start.z**: starting positions for the intermediate locations.
- **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

These are simple wrapper functions to create the desired model for use in metropolis. These models are structurally very simple and may be easily edited as required.

Author(s)

Michael D. Sumner

References


See Also

See also solar.model for the counterpart model for estimating positions for light tags.

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

http://www.srrb.noaa.gov/highlights/sunrise/azel.html

solar.model

Function to create a solar model object for metropolis location sampler

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?
solar.model

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calibration</td>
<td>calibration function for predicted light level for solar elevation</td>
</tr>
<tr>
<td>light.sigma</td>
<td>variance for light data</td>
</tr>
<tr>
<td>k.sigma</td>
<td>variance for light attenuation</td>
</tr>
<tr>
<td>behav</td>
<td>model distributions to be used for behaviour - defaults to &quot;speed&quot;</td>
</tr>
<tr>
<td>behav.dist</td>
<td>distribution to be used for behaviour</td>
</tr>
<tr>
<td>behav.mean</td>
<td>mean for behavioural distribution</td>
</tr>
<tr>
<td>behav.sd</td>
<td>variance for behavioural distribution</td>
</tr>
<tr>
<td>proj.string</td>
<td>PROJ.4 string for coordinate system used</td>
</tr>
<tr>
<td>ekstrom</td>
<td>parameters to use for ekstrom limit - min elevation, max elevation, sigma for outside that range</td>
</tr>
<tr>
<td>ekstrom.limit</td>
<td>mode of ekstrom limit to impose - defaults to &quot;light&quot;</td>
</tr>
</tbody>
</table>

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.behavioural(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.behavioural. 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 $(\text{lon}, \text{lat}, k)$ 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

* dplot
  initialize.x, 11
  mkCalibration, 14
  pick, 18

* manip
  as.image.pimg, 2
  astro, 3
  behav.bin, 5
  bits, 6
  chain.read, 7
  elevation, 7
  get.mask, 8
  get.sst, 9
  initialize.x, 11
  julday, 12
  metropolis, 13
  mkCalibration, 14
  mkLookup, 15
  norm.proposal, 16
  pick, 18
  pimg.list, 19
  satellite.model, 20
  solar, 21

* misc
  old.metropolis, 17

* models
  solar.model, 22

* spatial
  get.sst, 9

approxfun, 15
as.image.pimg, 2
as.local.pimg(as.image.pimg), 2
as.matrix.pimg(as.image.pimg), 2
astro, 3

behav.bin, 5
bin.pimg(behav.bin), 5
bits, 6, 9
bits<-(bits), 6

bmvnorm.proposal(norm.proposal), 16
chain.dim(chain.read), 7
chain.read, 7
chain.write(chain.read), 7
chunk.bin(behav.bin), 5
combine(as.image.pimg), 2
coords.pimg(as.image.pimg), 2
elevation, 4, 7
EQUHOR(astro), 3
FRAC(astro), 3
get.mask, 6, 8, 15
get.pimgs(as.image.pimg), 2
get.sst, 9
image, 8, 9
initialize.x, 11
julcent(julday), 12
julday, 12
k.prior(old.metropolis), 17
light.quantile(initialize.x), 11
LMST(astro), 3
locator, 19
lunar, 7
lunar(astro), 3
metropolis, 13, 13, 18, 21
metropolis0, 12, 13
metropolis0(metropolis), 13
mini.sun(astro), 3
MJD(astro), 3
mkCalibration, 14
mkElevationSeg(old.metropolis), 17
mkLookup, 9, 15
mkMaskObject, 10
INDEX

mkMaskObject (get.mask), 8
mkNLPrior (old.metropolis), 17
mkSmall (get.mask), 8
mvnorm.proposal (norm.proposal), 16

norm.proposal, 16

old.dist.gc (old.metropolis), 17
old.find.init (old.metropolis), 17
old.metropolis, 17
old.mkLookup (old.metropolis), 17
over, 15

pick, 18
picksegs (pick), 18
pimg (pimg.list), 19
pimg.list, 19
POLAR (astro), 3
position.logp, 20
position.logp (initialize.x), 11
project, 21, 23

satellite.model, 7, 14, 18, 20
set.mask<-(get.mask), 8
show.segment (initialize.x), 11
solar, 7, 12, 21
solar.model, 7, 12–14, 18, 19, 21, 22

unzipper (as.image.pimg), 2