

Package ‘mspath’

January 15, 2012

Title Multi-state Path-Dependent Models in Discrete Time

Version 0.9-9

Date 2010-01-23

Author Ross Boylan <ross@biostat.ucsf.edu> and Peter Bacchetti <peter@biostat.ucsf.edu>, building on the work of Christopher H. Jackson <chris.jackson@imperial.ac.uk>. Thorsten Ottosen <nesotto@cs.auc.dk> for the ptr_container library. Gennadiy Rozental <rogeeff@fusemail.com> for the Boost Test Library. And the authors of other Boost libraries used by the previous two.

Description Functions for fitting path-dependent (non-Markov) multi-state models to categorical processes observed at arbitrary times, optionally with misclassified responses, and covariates on transition or misclassification rates. Uses discrete-time approximation. Based on the Jackson’s msm package v 0.3.1, with an interface as compatible as possible.

Maintainer Ross Boylan <ross@biostat.ucsf.edu>

Depends methods, stats

Suggests Rmpi

SystemRequirements Boost C++ libraries (1.31 or greater)

License file LICENSE

Collate allGenerics.R checkPoint.R mspathCoefficients.R
mspathCalculator.R mspathDistributedCalculator.R mspath.R
readingError.R runTime.R subset.R trial.R utils.R zzz.R

Repository CRAN

Date/Publication 2012-01-15 10:43:49

R topics documented:

addResult	3
alldone	3
calculate	4
checkpoint	5
deltamethod	7
done	8
doTrials	9
e2	11
effort	12
estimateWork	13
expit	14
fixedSchedule	15
isAllFixed-methods	15
logit	16
master	17
MatrixExp	18
minus2loglik	19
mspath	20
mspath-class	26
mspath.subset	28
mspathCalculator	29
mspathCalculator-class	31
mspathCoefficients-class	33
mspathDistributedCalculator	36
mspathDistributedCalculator-class	38
mspathDistributedCalculatorFactory	39
mspathEstimatedCoefficients-class	40
q2	43
readingError-class	43
results	45
runAnalyzer-class	46
runeverywhere	48
runTime	49
runTime-class	50
simulated HCV data	52
slave	54
smoosh	55
trial-class	55

addResult	<i>Add a Job Run's Results to the Analyzer</i>
-----------	--

Description

Once you have timing information on a job, use this function to add the results to the analyzer.

Usage

```
addResult(analyzer, runTime)
```

Arguments

analyzer	An runAnalyzer
runTime	A runTime with the job and how long it took

Value

The analyzer, suitably updated. You *must* capture this value for the new data to be permanent.

Author(s)

Ross Boylan

See Also

[runAnalyzer](#) provides fuller discussion and example.

alldone	<i>Distributed Calculation Shut-Down</i>
---------	--

Description

Shut down the distributed calculation

Usage

```
alldone(comm = 0)
```

Arguments

comm	The MPI communicator to shut down.
------	------------------------------------

Details

Note that the distributed environment itself may remain and need to be shutdown separately. For example, if `lamboot` started the MPI session, you still need to execute `lamhalt`.

Only the master (rank 0) process should execute this command.

Value

The return value from `mpi.exit()`.

Author(s)

Ross Boylan

See Also

[master](#), [mspathDistributedCalculator](#), [Rmpi](#)

calculate

calculate

Description

Tell a calculator to perform its main calculation.

Usage

```
calculate(calc, params, activeCases, do.what)
```

Arguments

<code>calc</code>	The calculator object, mspathCalculator or an extension.
<code>params</code>	Optional "numeric" vector of the free parameters of the calculator. If missing, use the values already in the calculator.
<code>activeCases</code>	Optional "numeric" vector of ID's of cases over which the calculation will be performed. Note these will be converted to <code>integer</code> before use, and they are case ID's, not indices into the data. <code>activeCases</code> should be a subset of the cases in the calculator; ID's that are not will be ignored. As a special case, <code>integer()</code> for this argument means "use all cases." If missing, use the values already in the calculator.
<code>do.what</code>	Optional <code>integer</code> giving the code for the type of computation to perform. If missing, use the value already in the calculator.

Details

Performs the necessary calculation by invoking C++. The calculation may be time-consuming, which is why the computation must be invoked by this method rather than happening automatically.

Value

Returns the `calc` argument, updated with the return value and any state changes, including those from the optional arguments. You *must* use the returned value, not the one passed in as an argument, after the call.

`results` and `minus2loglik`, among other methods, will extract the results. The exact results depend upon the mode (`do.what`) of the calculation; sometimes only counts of paths and related parameters are returned.

Methods

`calc = "mspathCalculator", params = "ANY", activeCases = "ANY", do.what = "ANY"` Perform the multi-state path model calculation.

`calc = "mspathDistributedCalculator", params = "numeric", activeCases = "missing", do.what = "missing"` Distribute the multi-state calculation across processors. That calculation will be for all cases, and always computes the likelihood; thus the 2 related arguments *must* be omitted. Different subsets of cases are distributed to different processors.

Note

These methods are mostly for internal use. You *must* always and exclusively use the return value, not the original `calc`, after calling this function. As an incentive, this is the only way to get the results. Results are not in a separate object, because returning a separate results object would make it awkward to return an updated calculator.

Author(s)

Ross Boylan

See Also

[mspathCalculator](#), [mspathDistributedCalculator](#)

checkpoint

Checkpointing and progress information

Description

To checkpoint the optimization of function `f` use `checkpoint(f)` instead as an argument to the optimizer. Each time the optimizer calls the function the parameter values are recorded in a table in the caller's environment and on disk.

Usage

```
checkpoint(f, name = paste(substitute(f), ".trace", sep = ""), fileName = substitute(f), nCalls = 1, nTi
```

Arguments

<code>f</code>	The function to be optimized, as would be passed to the optimizer.
<code>name</code>	The name of the variable to be written into in the caller's environment. Defaults to <code>name.trace</code> .
<code>fileName</code>	The stem of the filename that will hold the R object name given above. By default it has the same name as that variable. The actual filenames will have '0' and '1' appended to them; the two files are written alternately.
<code>nCalls</code>	Writes to disk will occur every <code>nCalls</code>
<code>nTime</code>	Writes to disk will occur whenever at least <code>nTime</code> seconds have passed since the last write.
<code>frame</code>	This is the frame into which the results are written.

Details

If either the `nCalls` or `nTime` test is passed, the data will be written to disk.

Any variable named `name` in `frame` will be erased.

`name` is a `data.frame`, each of whose rows corresponds to a single invocation of the objective function `f`. The name of the row is iteration count. Its columns are

`time` The system time (`Sys.time`) of the call

`val` The value of `f` with the indicated parameters

parameters Each parameter value for the call

Value

A function that takes the same arguments as `f` and returns the same values, while recording information about the calls.

Note

Although this function is part of the `mspath` package, it is completely general and can be used in many other settings.

Writing a variable into the caller's environment violates the usual R scoping rules.

Each "iteration" of an optimizer may produce *many* calls to the objective function.

The determination of when to write to a file is not ideal. It should be sensitive to missing arguments, so that if one of the two tests is specified the other is ignored. Furthermore, the writeout caused by a time test will not reset the counter for the number of cases. For example, suppose you used `nCases=5`, `nTime=60`. If 60 seconds pass and the 4th case is written out, the fifth case will still be written out (and the 10'th, and so on). However, if the 5th case occurs before 60 seconds pass, the 60 second clock resets.

If the process is interrupted it is likely that the `val` of the last call will be unavailable, since it has not been computed.

Restarting currently consists of initializing the optimizer with the last recorded values of the previous run. Some optimizers build up information as they go; thus the result is not quite equivalent to

restarting in the middle. It might be desirable to have a function that could replay the value already computed.

In general, more insight into the optimization process would be desirable, e.g., measures of convergence. C code would need to be modified to achieve that.

Author(s)

Ross Boylan

See Also

linkoptim

deltamethod

The delta method

Description

Delta method for approximating the standard error of a transformation $g(X)$ of a random variable $X = (x_1, x_2, \dots)$, given estimates of the mean and covariance matrix of X .

Usage

```
deltamethod(g, mean, cov, ses=TRUE)
```

Arguments

g	A formula representing the transformation. It must have arguments labelled x_1, x_2, \dots . For example, $\sim 1 / (x_1 + x_2)$. If the transformation returns a vector, then a list of formulae g_1, g_2, \dots can be provided, for example <code>list(~ x1 + x2, ~ x1 / (x1 + x2))</code>
mean	The estimated mean of X
cov	The estimated covariance matrix of X
ses	If TRUE, then the standard errors of $g_1(X), g_2(X), \dots$ are returned. Otherwise the covariance matrix of $g(X)$ is returned.

Details

The delta method expands a differentiable function of a random variable about its mean, usually with a first-order Taylor approximation, and then takes the variance. For example, an approximation to the covariance matrix of $g(X)$ is given by

$$\text{Cov}(g(X)) = g'(mu)\text{Cov}(X)[g'(mu)]^T$$

where mu is an estimate of the mean of X .

Value

A vector containing the standard errors of $g_1(X), g_2(X), \dots$ or a matrix containing the covariance of $g(X)$.

Author(s)

C. H. Jackson <chris.jackson@imperial.ac.uk>

References

Oehlert, G. W. *A note on the delta method*. American Statistician 46(1), 1992

Examples

```
## Simple linear regression, E(y) = alpha + beta x
x <- 1:100
y <- rnorm(100, 4*x, 5)
toy.lm <- lm(y ~ x)
estmean <- coef(toy.lm)
estvar <- summary(toy.lm)$cov.unscaled

## Approximate standard error of (1 / (alphahat + betahat))
deltamethod (~ 1 / (x1 + x2), estmean, estvar)
```

done

Finished with a Calculator

Description

When you no longer need a calculator, call this function. It will perform any necessary cleanup.

Usage

```
done(calc)
```

Arguments

calc	The calculator you no longer need. Should be <code>mspathCalculator-class</code> or an extension.
------	---

Details

This is a generic function. You should call it even if you don't think any cleanup is necessary. It ensures proper cleanup of the C++ objects.

Value

Returns the updated `calc`, but you shouldn't really refer to either the old or new value after calling this function.

Methods

`calc = "mspathCalculator"` Clean up a regular calculator.

`calc = "mspathDistributedCalculator"` Clean up a distributed calculator, including letting all slaves no the computation is done. However, the overall session will remain up for future use.

Note

You should call this function even if you don't think any cleanup is necessary. You don't know what is happening behind the scenes.

Author(s)

Ross Boylan

See Also

[mspathCalculator](#), [mspathDistributedCalculator](#)

doTrials

Run trials until told to stop

Description

Run a user-defined analysis repeatedly with user-defined sets of parameters. Stops when it detects a particular file's existence on the disk. Returns results and writes them to disk; also writes interim results to disk.

Usage

```
doTrials(generator, executor, stopFileName = "cancel", resultsFile = "trials.RData", nTrials = 10, nTim
```

Arguments

<code>generator</code>	<code>generator</code> is a function taking the optional arguments ... and returning a list (ordinarily a named list) of arguments for <code>executor</code> . <code>generator</code> can return NULL to make <code>doTrials</code> finish.
<code>executor</code>	<code>executor</code> is a function which takes arguments produced by <code>generator</code> and performs the operation you are testing. It returns an object containing the results you want to preserve. The arguments are preserved automatically (see below).
<code>stopFileName</code>	Before every trial check if <code>stopFileName</code> exists, and stop if it does.

resultsFile	When done, write the results of the trials to resultsFile. Before then, this file name with .0 or .1 appended holds results so far.
nTrials	After every nTrials perform checkpointing.
nTime	After nTime seconds have passed, perform checkpointing.
...	Optional arguments passed to generator.

Details

Checkpointing writes the results so far to a file (named from resultsFile with, alternately, .0 and .1 appended) and prints a progress message on the controlling terminal. Every time either nTrials trials or nTime seconds have passed since the last checkpoint a new one will occur; whichever is hit first will control checkpointing.

Value

A list of `trial`'s, in the order they were performed. `trial` holds the arguments for the trial (produced by generator) and the results (produced by executor).

Note

This function was designed to check the robustness of estimates. The generator can produce varying initial values, algorithmic choices, and tuning parameters, for example. The executor then takes those and calls to a function of interest.

The generator can produce values randomly, using a deterministic scheme, or some mix of the two. The deterministic scheme may be infinite. If not, generator should return NULL when it is done, and the function will finish.

The executor is responsible for recording whatever information is of interest, apart from the arguments it receives. Relevant values could include convergence, parameter estimates, and run-time, among other things.

Results objects returned by estimators commonly include a lot of information, including the arguments of the function call. To save space (and thus time too) you may wish to record only the statistics you want, or blank out the data elements you don't need. The arguments returned by generator are likely to be more compact than those to the function being tested; for example, generator might return an index or name that executor translates into a large set of parameters.

The checkpoint files are deleted on exit from this function.

Author(s)

Ross Boylan <ross@biostat.ucsf.edu>

See Also

`trial`

Examples

```

## since the function runs forever, we don't want to really run it
## Not run:

# create a set of parameters
testgen <- function(...) {
  p <- rnorm(3, ...)
  list(a=p[1], b=p[2], c=p[3])
}

# estimate a function using those parameters
testexec <- function(a, b, c) {
  Sys.sleep(10) # some long computation
  a+b+c # result, sensitive to the parameters
}

## Note that parameters, as used above, are not the same as the
## parameters of the inner function the executor runs. The parameters
## here might be the initial value for the parameters of the inner model.

r <- doTrials(testgen, testexec, nTrials=20, nTime=30)

## End(Not run)

```

e2

Allowed Misclassifications: e2

Description

Matrix of allowed transitions (e for errors) among 5 states.

Format

A 5x5 matrix of 0's and 1's indicating allowed misclassifications among real and observed states. The diagonal is ignored.

Details

This matrix says that a given true state may be observed as being up to 2 states below the true state. However, the highest state is observed accurately. True states are never observed as being in a higher than true state.

Source

Made up.

Examples

```
library(mspath)
data(e2, q2, sim2)
r <- mspath(fib~time, misc=TRUE, ematrix=e2, qmatrix=q2, inits=rep(.5, 9), subject=id,
            data=sim2, stepnumerator=1, stepdenominator=1, initprobs=c(1.0, 0, 0, 0, 0),
            do.what=0)
```

effort

Get measures of effort to perform a calculation

Description

Generic function for calculators returning estimates of the amount of work involved in the computation. Exact details are likely to change.

Usage

```
effort(calc)
```

Arguments

calc The [mspathCalculator](#) or subclass for which we want the effort estimate.

Details

Only call this after invoking [calculate](#).

Value

A single number giving the best estimate of likely effort for the active cases. This is useful for gauging the *relative* times of different subsets; the absolute values have no particular meaning. In particular, they are not in seconds. Subject to change and the exact history of the calculator.

Methods

calc = "mspathCalculator" standard case

Note

In the future values disaggregated down to individual cases or some intermediate level may be returned.

Author(s)

Ross Boylan

See Also

[mspathCalculator](#), [link{estimateWork}](#) for a richer set of information on likely effort.

 estimateWork

Estimate Amount of Work for Computations

Description

Compute estimates of the amount of work a calculator will need to do for each individual case. These may be much quicker to compute than the actual likelihood evaluation.

Usage

```
estimateWork(calc)
```

Arguments

calc	The calculator for which we want the work estimate. Should be mspathCalculator or subclass.
------	---

Details

Subject to change. Various estimates of node and path counts, for the particular model and data currently in the calculator. I think this always does all cases, but need to check.

Value

A matrix whose rows are individual cases. The columns are

ID	ID of the case for this row.
Good Nodes	Presumed to be the best single predictor of effort, in particular of floating-point operations. This is the number of unique nodes on good paths.
Good Paths	Number of distinct paths for this case, consistent with the model and the data.
Bad Nodes	Number of nodes considered that fell outside of the good paths. May vary with the pruning strategy used, but at any rate note that these nodes need not be constructed.
Good Path Nodes	This is the sum of all the nodes in all the good paths, ignoring the fact that some nodes are shared between paths. This is a measure of the work effort that a naive implementation would require. The current implementation is not naive.
Steps	The average number of steps on the good paths. If some paths can end before others, this value may not be an integer.

Methods

calc = "mspathCalculator" standard case

Note

The returned value is not stored in the calculator.

[mspathDistributedCalculator](#) uses this function to schedule work.

Author(s)

Ross Boylan

See Also

[mspathCalculator](#), [results](#).

expit

Inverse logit function

Description

Inverse logit function

Usage

expit(x)

Arguments

x A non-negative vector

Value

$\exp(x)/(1 + \exp(x))$

See Also

[logit](#)

fixedSchedule	<i>fixedSchedule</i>
---------------	----------------------

Description

Return indices to cut input work into roughly equal chunks

Usage

```
fixedSchedule(work, ngroups)
```

Arguments

work	vector of work effort, sorted in descending order
ngroups	desired number of work chunks. If some of the individual work items are big, you may get less than ngroups back.

Details

Does simple fixed scheduling, dividing up cases into batches with roughly equal work effort. Dynamic scheduling would adjust to the run-time conditions; since we have uniform slave processors and expect run-time to be fairly repeatable, fixed scheduling should suffice.

However, this may initially be used with a proxy of actual run time in work; in that case, it may help to collect the actual run-times and repartition the work.

Value

The indices in work to divide it up. These are the right-hand endpoints, so [1] is the index of the last work item to include in the first group.

Author(s)

Ross Boylan

isAllFixed-methods	<i>Test for fixed objects</i>
--------------------	-------------------------------

Description

Return TRUE if coefficients have been fixed in advance so that there are no estimable values. Objects without estimates are assumed fixed, while those with classes indicating estimates but all parameters actually fixed are also fixed.

Methods

isAllFixed signature(object="mspathCoefficients"): TRUE because all values are fixed in advance.

isAllFixed signature(object="mspathCoefficients"): usually FALSE, but TRUE if all parameters are fixed.

Note

Motivation: if the user has fixed all the parameter values in advance, it's probably not worth showing them again when displaying the results. The anticipated clients are the internal print methods of this package.

logit

Logit function

Description

Logit function

Usage

logit(x)

Arguments

x A real vector

Value

$\log(x/(1-x))$

See Also

[expit](#) for inverse logit function

master	<i>Start Multi-State Path Model Distributed Calculations on Master Node</i>
--------	---

Description

This is the top-level function for the master (rank 0) process in a distributed computation. It does any necessary setup, and then executes the R commands given in the `channel` argument. After executing the commands it shuts down the slaves and itself, exiting R.

Usage

```
master(channel = "kickStart.R", comm = 0)
```

Arguments

<code>channel</code>	A source of R commands to execute within the distributed environment. They should create and use a mspathDistributedCalculator . Invoking mspath with the optional <code>comm</code> argument will satisfy that requirement.
<code>comm</code>	The MPI communicator to use. Usually the default will suffice.

Details

Before invoking this command you must establish the necessary MPI environment and launch appropriate slave processes. This routine does not spawn slave processes, in part so it's useful on systems without that capability.

The code you provide by the `channel` should not attempt to shut things down itself; `master` always does so at the end.

Value

The return value of the final `mpi.exit`.

Note

Requires package **Rmpi** exist and be loadable via `library(Rmpi)`. It will be loaded automatically; you should not load it yourself.

See the details section for more about the necessary setup.

Since the function exits R without saving, you will need to arrange to store or display your results.

Author(s)

Ross Boylan

See Also

Rmpi, [mspathDistributedCalculator](#), [mspath](#)

MatrixExp

*Matrix exponential***Description**

Calculates the exponential of a square matrix.

Usage

```
MatrixExp(mat, t = 1, n = 20, k = 3)
```

Arguments

mat	A square matrix
t	An optional scaling factor for the eigenvalues of mat
n	Number of terms in the series approximation to the exponential
k	Underflow correction factor

Details

The exponential E of a square matrix M is calculated as

$$E = U \exp(D) U^{-1}$$

where D is a diagonal matrix with the eigenvalues of M on the diagonal, $\exp(D)$ is a diagonal matrix with the exponentiated eigenvalues of M on the diagonal, and U is a matrix whose columns are the eigenvectors of M .

However, if M has repeated eigenvalues, then its eigenvector matrix is non-invertible. In this case, the matrix exponential is calculated using a power series approximation,

$$\exp(M) = I + M + M^2/2 + M^3/3! + M^4/4! + \dots$$

For a continuous-time homogeneous Markov process with transition intensity matrix Q , the probability of occupying state s at time $u + t$ conditional on occupying state r at time u is given by the (r, s) entry of the matrix $\exp(tQ)$.

The series approximation method was adapted from the corresponding function in Jim Lindsey's R package `rmutil`.

Value

The exponentiated matrix $\exp(mat)$.

Author(s)

C. H. Jackson <chris.jackson@imperial.ac.uk>

References

Cox, D. R. and Miller, H. D. *The theory of stochastic processes*, Chapman and Hall, London (1965)

minus2loglik

Retrieve Multi-State Path Model Likelihood

Description

Returns $-2 * \text{the calculator's previously computed log-likelihood, if any.}$

Usage

minus2loglik(x)

Arguments

x A previously evaluated calculator, which should be a [mspathCalculator](#) or subclass

Value

$-2 * \text{log-likelihood, a double.}$

Methods

x = "mspathCalculator" standard case

Note

Use this function, rather than a less-reliable unpacking of [results](#), to get the computed likelihood.

You must create and evaluate the calculator with appropriate `do.what` arguments to get a useful likelihood.

Author(s)

Ross Boylan

See Also

[mspathCalculator](#)

Description

Ordinarily, fit a path-dependent (non-Markov) multi-state model by maximum likelihood. Observations of the process can be made at arbitrary times. Covariates can be fitted to the transition intensities. When the true state is observed with error, a model can be fitted which simultaneously estimates the state transition intensities and misclassification probabilities, with optional covariates on both processes.

This function will also perform some auxiliary operations, including counting paths and generating data according to the model. See `do.what`.

Based on the Christopher Jackson's `msm` package v 0.3.3. The key differences are that this package can handle non-Markov models, and it does so using a discrete-time approximation. Note that while some of the inputs to `msm` were transition rates or misclassification probabilities, the corresponding terms here are the intercepts of multinomial logits. Finally, `fromto` and related options do not exist; they are not appropriate in a path-dependent setting and have also been dropped from later versions of `msm`.

The interface to `msm` changed with more recent releases to a style that combined specification of allowed transitions or observation errors and their values. This would not be appropriate here, since 0 is a legitimate value for a coefficient. `msm` also has considerably richer options for handling the different kinds of information conveyed by the times of different observations. There is no theoretical problem with handling those options here; there just hasn't been a need to implement those options.

Usage

```
mspath ( formula, qmatrix, misc = FALSE, ematrix, inits, subject,
         covariates = NULL, constraint = NULL, misccovariates = NULL,
         misconstraint = NULL, qconstraint=NULL, econstraint=NULL,
         pathvars = NULL, pathoffset = 0, pathconstraint = NULL,
         initprobs = NULL,
         data = list(),
         isexact = FALSE,
         fixedpars = NULL, stepnumerator = 1, stepdenominator = 1,
         do.what = 1, testing,
         comm, profile = FALSE,
         calcFactory = mspathCalculator,
         seed,
         trace,
         ... )
```

Arguments

`formula` A formula giving the vectors containing states and the corresponding observation times. For example,

states ~ times

Observed states should be in the set 1, . . . , n, where n is the number of states. Use state 0—*not* missing—when the state is not observed but the covariates are. Note that analysis will exclude any observation with missing covariates.

qmatrix Matrix of indicators for the allowed transitions. If a transition is allowed from state r to state s , then **qmatrix** should have (r, s) entry 1, otherwise it should have (r, s) entry 0. The diagonal of **qmatrix** is ignored. For example,

```
rbind( c( 0, 1, 1 ), c( 1, 0, 1 ), c( 0, 0, 0 ) )
```

represents a 'health - disease - death' model, with transitions allowed from health to disease, health to death, disease to health, and disease to death.

misc Set **misc** = TRUE if misclassification between observed and underlying states is to be modelled, FALSE if observation is assumed accurate, or SIMPLE (no quotes) if the values given in **inits**, in conjunction with **econstraint**, specify the probabilities of measurement error.

In the last case, those probabilities are fixed constants and you should omit **misccovariates** and **misconstraint**. All probabilities should be between 0 and 1, as should 1 - their sum, which the routine computes as the probability of accurate observation. For SIMPLE the values are interpreted directly, rather than via a multinomial logit transformation. You do not need to list them in **fixedpars**; the routine always considers them fixed.

ematrix (required when **misc** == TRUE) Matrix of indicators for the allowed misclassifications. The rows represent underlying states, and the columns represent observed states. If an observation of state s is possible when the subject occupies underlying state r , then **ematrix** should have (r, s) entry 1, otherwise it should have (r, s) entry 0. The diagonal of **ematrix** is ignored. For example,

```
rbind( c( 0, 1, 0 ), c( 1, 0, 1 ), c( 0, 1, 0 ) )
```

represents a model in which misclassifications are only permitted between adjacent states.

inits (required) Vector of initial parameter estimates for the optimisation. The "Details" section below gives full interpretation of the parameters. Parameters are in this order:

- transition intercepts (reading across first rows of intensity matrix, then second row ...)

- covariate effects on transitions

- history-dependent path variables effects on transitions

- misclassification intercepts (reading across first row of misclassification matrix, then second row ...)

- covariate effects on misclassification

Covariate effects are given in the following order,

- effects of first covariate on transition/misclassification matrix elements (reading across first row of matrix, then second row ...)

- effects of second covariate ...

subject	Vector of subject identification numbers, when the data are specified by formula. If missing, then all observations are assumed to be on the same subject. These must be sorted so that all observations on the same subject are adjacent.
covariates	Formula representing the covariates on the transition intensities, for example, <code>~ age + sex + treatment</code>
constraint	<p>A list of one vector for each named covariate. The vector indicates which covariate effects are constrained to be equal. Take, for example, a model with five transition intensities and two covariates. Specifying</p> <pre>constraint = list (age = c(1,1,1,2,2), treatment = c(1,2,3,4,5))</pre> <p>constrains the effect of age to be equal for the first three intensities, and equal for the fourth and fifth. The effect of treatment is assumed to be different for each intensity. Any vector of increasing numbers can be used as indicators. The intensity parameters are assumed to be ordered by reading across the rows of the transition matrix, starting at the first row.</p> <p>For categorical covariates, defined using <code>factor(covname)</code>, specify constraints as follows:</p> <pre>list(..., covnameVALUE1 = c(...), covnameVALUE2 = c(...), ...)</pre> <p>where VALUE1, VALUE2, ... are the levels of the factor. Make sure the <code>contrasts</code> option is set appropriately, for example, the default options (<code>contrasts=c(contr.treatment, contr.poly)</code>) sets the first (baseline) level of unordered factors to zero.</p> <p>To assume no covariate effect on a certain transition, set its initial value to zero and use the <code>fixedpars</code> argument to fix it during the optimisation.</p>
misccovariates	A formula representing the covariates on the misclassification probabilities, analogously to <code>covariates</code> .
misconstraint	A list of one vector for each named covariate on misclassification probabilities. The vector indicates which covariate effects on misclassification probabilities are constrained to be equal, analogously to <code>constraint</code> .
qconstraint	<p>A vector of indicators specifying which transition intercepts are equal. For example,</p> <pre>qconstraint = c(1,2,3,3)</pre> <p>constrains the third and fourth transition intercepts to be equal, in a model with four allowed instantaneous transitions.</p>
econstraint	A similar vector of indicators specifying which misclassification intercepts are constrained to be equal.
pathvars	A vector of the names of path-dependent variables to use as covariates on transition intensities. Allowed values currently include TIS, Time in State; TSO, Time Since Origin, TIP, Time in Previous states, and LN(TIS), LN(TIP) or LN(TSO) for natural logs of the same.
pathoffset	This value will be added to the origin time of path-dependent variables. Use it to pick where in the interval to use for your time values, or to avoid calculations with 0.

pathconstraint	Constraints on path variables' effects on transition rates.
initprobs	Currently only used in misclassification models. Vector of assumed underlying state occupancy probabilities at each individual's first observation. Defaults to <code>c(1, rep(0, nstates-1))</code> , that is, in state 1 with a probability of 1. The current implementation only supports a single initial state.
data	Optional data frame in which to interpret subject as well as the time, state, and covariates defined in <code>formula</code> , <code>covariates</code> , and <code>misccovariates</code> formulae.
isexact	By default, the transitions of the Markov process are assumed to take place at unknown occasions in between the observation times. However, <code>isexact = TRUE</code> implies observation times for entry to absorbing states are exact. Only paths with transitions at the observed time will be counted. Typically, use this for death. Note this option has no effect on non-absorbing states. It is an error to use this option when observation of the absorbing state is imprecise.
fixedpars	Vector of indices of parameters whose values will be fixed at their initial values during the optimisation. These correspond to indices of the <code>inits</code> vector, whose order is specified above. This can be useful for building complex models stage by stage.
stepnumerator	The calculation will use a time grid for each case with steps every <code>stepnumerator/stepdenominator</code> . This allows exact representation of rationals. The numerator and denominator must both be integers.
stepdenominator	See <code>stepnumerator</code> just above.
do.what	1, the default, calculates a maximumlikelihood. To evaluate a single likelihood, set all parameters to fixed. 0 counts the number of paths and related statistics without evaluating a likelihood. -1 get detailed counts, but not likelihoods, associated with each case. The return value is a matrix. 10 uses the model to simulate a random patch for each case, returning a <code>data.frame</code> with simulated observed states and times and all other data as observed.
testing	This argument is only for use by developers. Set it to a list with the expected low-level arguments to test if that's what is actually generated. The result will either be a <code>stop</code> or <code>true</code> ; the underlying C code is not called. See the code for the exact arguments.
comm	An optional MPI communicator. If you provide it, perform a distributed calculation. Ordinarily, <code>comm = 0</code> . Ordinarily a call using this argument will be invoked from the commands supplied via <code>master</code> 's <code>channel</code> argument. See that routine for more information on distributed calculation and the necessary setup.
profile	Profile the time taken in the distributed calculation.
calcFactory	Optional, advanced argument. This should be a function that, when called with the arguments to the standard <code>mspathCalculator</code> , produces a calculator for internal use. The <code>comm</code> and <code>profile</code> arguments will be ignored in this case.
seed	Optional seed to set for simulation (<code>do.what=10</code>). Use an integer for full precision; it will be cast to unsigned.

trace	Optional list of arguments to <code>checkpoint</code> . If present, the objective function will be wrapped by <code>checkpoint</code> and every function evaluation's parameters and value will be recorded in the caller's environment and in files on disk. See <code>checkpoint</code> for details.
...	Optional arguments to the general-purpose R optimization routine <code>optim</code> . Useful options include <code>method="BFGS"</code> for using a quasi-Newton optimisation algorithm, which can often be faster than the default Nelder-Mead. If the optimisation fails to converge, consider normalising the problem using, for example, <code>control=list(fnscale = 2500)</code> , for example, replacing 2500 by a number of the order of magnitude of the likelihood. If 'false' convergence is reported and the standard errors cannot be calculated due to a non-positive-definite Hessian, then consider tightening the tolerance criteria for convergence. If the optimisation takes a long time, intermediate steps can be printed using the trace argument of the control list. See <code>optim</code> for details.

Details

For each case, the model generates all possible paths consistent with the observed data, measurement model, and transition model. It computes and sums the likelihood of each path, including the initial probability specified in `initprobs`.

It is quite possible to specify a model which makes the observed data impossible; you will get an error in that case.

Both the transition and misclassification models use the multinomial logit. Self transition has assumed coefficients of 0; the other coefficients are multiplied by the covariates, summed and exponentiated. All coefficients, including the intercept or baseline terms, are so transformed. With J possible outcomes and p_j the probability of the j 'th outcome, the formula is

$$p_j = \frac{e^{\mathbf{X}\beta_j}}{\sum_{k=0}^J e^{\mathbf{X}\beta_k}}.$$

β_j is the vector of coefficients for outcome j and \mathbf{X} are the covariates. Calculations use the covariates at the endpoint of the transition to set the transition rate.

TIP, Time in Previous states, is the start time of the current state. Note it is time in all previous states, not just the previous state. It will remain constant as TIS, Time in State, changes. If `pathOffset=0`, $\text{TSO} = \text{TIP} + \text{TIS}$. TIP is a constant for the first state; we recommend constraining the corresponding coefficient to 0 if you allow effects of TIP to vary with stage.

There must be enough information in the data on each state to estimate each transition rate, otherwise the likelihood will be flat and the maximum will not be found. It may be appropriate to reduce the number of states in the model, or reduce the number of covariate effects, to ensure convergence.

Choosing an appropriate set of initial values for the optimisation can also be important. For flat likelihoods, 'informative' initial values will often be required.

If a single discrete time point includes multiple observations, one is selected. In the initial interval, the first point is used to get the correct start time. For later times, selection favors observations with (non-0) state, and picks the last possible observation. If any observations include state, the last among those is the one picked.

Value

Ordinarily a [mspath](#) object. See it for details. However, there are the following special cases:

testing non-empty	Either returns TRUE or stops with an error if the calculated variables do not those in the testing list. Specifying testing supersedes any actions given in do.what.
do.what=-1	Return a matrix with counts for each case.
do.what=10	Return a data frame with a random path as described above.

Simulation

do.what=10 requests generation of a simulated data frame based on the specified model and data. The first step of the simulation drops rows as in the likelihood estimation. Rows with missing data may be dropped depending upon the options you have requested globally or in data. Then rows in the same discrete time interval are dropped.

Second, all the data are collected: data and subject and any variables used in formula, covariates and misccovariates. Data not present in data will be pulled in from the appropriate environment and added to the output data frame. If this happens with subject, the name of the column will be "subject"; in these circumstances subject may not appear as a term in any formula. Note that variables in data that are not referenced in formulae *will* be retained for later analysis.

The variable names will be those used in the formulae. If the caller specifies subject with a name (e.g., `mspath(subject=id, ...)`) that name will be the name of the subject variable; otherwise the name will be "subject". In either case, that name should not be used elsewhere.

Third, for each case a true path and corresponding observed states will be simulated using the model. Ordinarily the simulated data will have the same observation times as the real data. However, if the simulated path enters an absorbing state and `isexact == TRUE` then the simulated data will show an observation at that time, using the then current covariates. Furthermore, the simulated data will never have more than one observation in an absorbing state, since otherwise `mspath` will refuse to estimate for the simulated data.

Note

The trace argument will cause tracing information to go into the caller's environment and onto the disk, as described in [checkpoint](#).

Author(s)

Ross D. Boylan <ross@biostat.ucsf.edu>

References

- Boylan, R.D. 2010 Multi-state Path-Dependent Models. Working paper. University of California, San Francisco. Available as `doc/mspath-program.*` within this package.
- Bacchetti, P., Boylan, R.D., Terrault, N.A., Monto, A. and Berenguer, Marina. Forthcoming Non-Markov multistate modeling using time-varying covariates, with application to progression of liver fibrosis due to hepatitis C following liver transplant. *The International Journal of Biostatistics*.

Jackson, C.H., Sharples, L.D., Thompson, S.G. and Duffy, S.W. and Couto, E. 2003 Multi-state Markov models for disease progression with classification error. *The Statistician*, **52**(2), 193–209.

Jackson, C.H. and Sharples, L.D. 2002 Hidden Markov models for the onset and progresison of bronchiolitis obliterans syndrome in lung transplant recipients *Statistics in Medicine*, **21**(1), 113–128.

Kay, R. 1986 A Markov model for analysing cancer markers and disease states in survival studies. *Biometrics* **42**, 855–865.

Gentleman, R.C., Lawless, J.F., Lindsey, J.C. and Yan, P. 1994 Multi-state Markov models for analysing incomplete disease history data with illustrations for HIV disease. *Statistics in Medicine* **13**(3), 805–821.

Satten, G.A. and Longini, I.M. 1996 Markov chains with measurement error: estimating the 'true' course of a marker of the progression of human immunodeficiency virus disease (with discussion) *Applied Statistics* **45**(3), 275-309.

See Also

[master](#), [checkpoint](#), [mspath](#) return object.

mspath-class

Results of a multi-state path calculation

Description

These classes hold the results of likelihood estimation from a call to [mspath](#), either with ([mspathFull](#)) or without ([mspath](#)) modeling observational measurement error.

Objects from the Class

Result of calls to [mspath](#).

Slots

transCoef: Estimated transition coefficients, class "[mspathCoefficients](#)".

errCoef: Only [mspathFull](#) has this slot, which gives the estimated observation error coefficients. Slot class [mspathCoefficients](#).

calc: The "[mspathCalculator](#)" used to produce this object.

overlap: Indices in original data frame that were eliminated because of overlapping observations (multiple observations in the same discrete time period. Object of class "[numeric](#)").

call: The call to [mspath](#) that created this object. Slot has class "[call](#)".

callArgs: A "[list](#)" with the resolved values specified by the user in the call. Each argument captured by ... will be present, but omitted arguments will be absent, even if they have defaults. To understand the difference between this slot and the previous one, suppose one invokes [mspath](#)([qmatrix](#)=[b4](#), ...), and then later redefines [b4](#). Then [call](#) will have [qmatrix](#)=[b4](#) but [callArgs](#)\$[qmatrix](#) will have the value of [b4](#) at the time of the call.

opt: The object returned by the call to [optim](#) that is used for the likelihood maximization. Slot class "[list](#)".

Methods

- coef** Estimated coefficients for the transitions, a [mspathCoefficients](#) object.
- matrixCoef** A list of transition matrices of coefficients. The optional `coeff` argument is a vector of strings, each being the name of a coefficient of interest; otherwise all are reported.
- sd** `signature(x="mspath", na.rm="ANY")`: Estimated standard deviation of coefficients, or 0 if they are fixed. `na.rm` is required for compatibility with the general interface for this function, but probably won't work properly.
- minus2loglik** `signature(x = "mspath")`: -2 time the estimated log-likelihood.
- nBadNodes** `signature(x = "mspath")`: Number of invalid nodes generated.
- nCases** `signature(x = "mspath")`: Number of cases, i.e., unique id's, in the input data. The number of individual observations will generally be greater than this.
- nGoodNodes** `signature(x = "mspath")`: Number of unique good nodes on these paths. Note that nodes may be shared between paths, and some "good" nodes turn out not to be on any good paths. Expected likelihood computation time is roughly proportional to this number.
- nGoodPathNodes** `signature(x = "mspath")`: Total number of nodes in all the good paths. `goodPathNodes/goodPaths` is the average path length. `goodNodes/goodPathNodes` estimates the speedup from sharing nodes between paths vs a naive implementation that treats each path independently.
- nGoodPaths** `signature(x = "mspath")`: Number of complete good paths on which to compute a likelihood.
- optresults** `signature(x = "mspath")`: The results of the inner call to [optim](#).
- optresults<-** `signature(x = "mspath")`: For internal use only.
- print** `signature(x="mspath")`: print results. If there is an error model, the default behavior is only to show the transition or error coefficients if they were not completely fixed in advance; use the optional `showAll=TRUE` argument to show the fixed coefficients as well. Other arguments, including general printing parameters and `coeff` are passed to the coefficient printing routines.
- printFooter** `signature(x = "mspath")`: For internal use only.

Note

The counts are counts over all cases, so the number good paths, for example, is the sum of all the good paths for each case. Currently the counts are not reliable for distributed calculations.

This class currently exists mostly to print out the estimated coefficients. It records other information to preserve the context of the computation, but doesn't currently do much with it. As need arises, the class will be enhanced.

In particular, the class only provides the raw coefficients and their mapping to cells in the matrices. It does not multiply by covariates, calculate implied transition or error probabilities, or otherwise work out the implications of the model.

The related `msm` package does work out many of the implications of the estimated coefficients (in fact, it does not present the raw coefficients at all in its standard print method). Almost none of the operations there (markov, continuous time models) translate directly into the context of this model (non-markov, discrete time). Also, `msm` uses S3 classes, while this package uses S4. The only user implication is that the S4 help system is available.

In principle this class could hold sets of coefficients that did not come from maximization; in particular, the coefficients do not have to be estimated ones. Neither the class nor its methods are designed for that use, however.

Author(s)

Ross Boylan <ross@biostat.ucsf.edu>

See Also

[mspath](#) creates these objects. The key values are in [mspathCoefficients](#), which will usually be the [mspathEstimatedCoefficients](#) subclass. See [optim](#) for details of its results object, which is preserved here. [mspathCalculator](#) describes the calculator.

mspath.subset

subset a dataset based on estimated work

Description

Produces a subset of a dataframe or matrix drawn from cases that will be quick to analyze. Ordinarily, there will be repeated observations per case.

Usage

```
mspath.subset(data, lop=.3, id="id", time="time", nCases = 50)
```

Arguments

data	The dataframe or matrix to be subsetted. Items should be sorted by id and then time.
lop	This fraction of cases will be dropped from eligibility for the subset. These are the cases with the highest amount of estimated work.
id	The character name of the column with the case identifier.
time	The function assumes that computation will be an increasing function of total time a case covers, as indicated by this column. The difference between the first and last time is the total case time.
nCases	Approximate number of cases to produce in the subset. The actual number may differ slightly. Note this is the number of cases, not the number of records.

Details

Case selection is deterministic, so that repeated calls will generate the same data.

Value

data subsetted

Note

This function is mostly for internal testing, though you may find it useful in other contexts. In situations in which evaluation can be very slow or memory intensive, and is strongly dependent on some feature of the data, analysis of a subset can be a lot easier.

A case with only one record will be assumed to take no time, since first and last time are identical.

Actual computation effort need not be linear in time; the subsetting is sensible for any monotonic increasing function of time. Time need not be a perfect predictor of effort for the subsetting to be useful.

Author(s)

Ross Boylan

Examples

```
data("sim3")
short <- mspath.subset(sim3)
```

mspathCalculator

Constructors for mspathCalculator Objects

Description

Make [mspathCalculator](#) objects. This is primarily for internal use.

Usage

```
mspathCalculator(do.what, params, allinits, misc, subject, time, state, qvector, evector, covvec,
  constrvec, misccovvec, misconstrvec, baseconstrvec, basemisconstrvec,
  pathvars, pathoffset, pathconstrvec,
  initprobs, nstates, nintens, nintenseffs, nmisc, nmisceffs, nobs, npts,
  ncovs, ncoveffs, nmisccovs, nmisccoveffs,
  npatheffs,
  isexact, fixedpars, stepnumerator, stepdenominator)
mspathCalculatorFromArgs(args, do.what=1)
```

Arguments

See [mspath](#) and the documentation on the C++ code for the full details.

Type of computation to perform, as interpreted by the C++ code.

`params`

Ordinarily, the precise values here may be irrelevant, since they can be reset on specific calls to [calculate](#). However, the size of this argument must be correct.

`allinits`

The “initial” values for all fixed and free parameters. Only the fixed values are used; the rest come from `params`, which may be reset during the life of the calculator.

<code>misc</code>	0 = no misclassification; 1 = full misclassification; 2 = simple, fixed misclassification.
<code>subject</code>	The ID, which must be an integer, for each row of data. The ID's should be sorted in ascending order.
<code>time</code>	time of each observation, in ascending order within cases.
<code>state</code>	The state of each observation. States should be numbered 1 through <code>nstates</code> .
<code>qvector</code>	Vectorized matrix of allowed transitions, 1 for allowed, 0 for not allowed.
<code>evector</code>	Vectorized matrix of allowed misclassifications (on relevant if <code>do.what</code> is not 0).
<code>covvec</code>	vectorized matrix of covariate values.
<code>constrvec</code>	constraints for each covariate
<code>misccovvec</code>	Vectorized matrix of covariate values for misclassification (only relevant if <code>misc</code> = 1).
<code>misconstrvec</code>	list of constraints for each misclassification covariate
<code>baseconstrvec</code>	constraints on baseline transition intensities
<code>basemisconstrvec</code>	constraints on baseline misclassification probabilities
<code>pathvars</code>	character vector of the names of history-dependent variables.
<code>pathoffset</code>	add this double to every time 0 in the paths
<code>pathconstrvec</code>	constraints on path effects on intensities
<code>initprobs</code>	initial state occupancy probabilities
<code>nstates</code>	number of states
<code>nintens</code>	number of intensity parameters
<code>nintenseffs</code>	number of distinct intensity parameters
<code>nmisc</code>	number of misclassification rates
<code>nmisceffs</code>	number of distinct misclassification rates
<code>nobs</code>	number of observations in the data set
<code>npts</code>	number of individuals/cases in the data set
<code>ncovs</code>	number of covariates on transition rates
<code>ncoveffs</code>	number of distinct covariate effect parameters
<code>nmisccovs</code>	number of distinct misclassification parameters
<code>nmiscoveffs</code>	number of distinct misclassification effects
<code>npatheffs</code>	number of distinct path (history) effects on transitions
<code>isexact</code>	non-0 if we observed time of entry to absorbing states exactly (e.g., death)
<code>fixedpars</code>	which parameters to fix
<code>stepnumerator</code>	integer This divided by <code>stepdenominator</code> give the maximum step size in the discrete approximation, in the same scale as time.
<code>stepdenominator</code>	integer see <code>stepnumerator</code>
<code>args</code>	These arguments for <code>mspathCalculatorFromArgs</code> , unlike those for <code>mspathCalculator</code> , must be processed with <code>as.integer</code> and other appropriate forms for the call to C++.

Value

Returns a new [mspathCalculator](#) object.

Note

To use this properly you need to use the correct 1 or 0-based indexing (which varies by argument) and know the exact rules for unrolling matrices into vectors.

Author(s)

Ross Boylan

See Also

[mspathCalculator](#), [mspath](#), and the documentation for the C++ code.

mspathCalculator-class

Calculator for Multi-State Path Models

Description

This calculator calculates the likelihood and related quantities for a multi-state path model with possible non-Markov properties. Alternately, it can generate a simulated data set. Though you can use a calculator directly, it is usually more convenient to invoke [mspath](#). If you use it, you must follow the rules about the proper sequence in which to invoke different methods, and you must always use the possibly updated calculator object that many methods return.

Objects from the Class

Create objects of this class by calling [mspathCalculator](#) or [mspathCalculatorFromArgs](#).

Slots

args: A list of arguments to the C function call. As such, they must be in the correct order, with all necessary `as.xxx`'s performed.

manager: Reference to the C++ object that performs the calculations. Object of class "externalptr". Much of the state of the `mspathCalculator` is really in this C++ object. The persistence of information between calls permits faster calculations, but raises the possibility for errors. Do *not* do anything with the manager.

do.what: "integer" with the code for the desired operation. This will be interpreted by the C++ routines, and is subject to being changed as a side-effect of user-level operations.

subset: "integer" of the ID's of cases for which the likelihood should be evaluated. The restriction to integers makes the interface with C easier. If this vector is empty, it means analyze all cases.

results: Object of class "numeric" holding the results of the calculation. In the future, it may hold other types. If there has not been a calculation with the current calculator values, this slot will have length 0. See [results](#) for details.

Methods

Setup:

`signature(calc = "mspathCalculator", value = "numeric")`: value is a vector of the ID's (which will be converted to integer) of the active subset. They should be a subset of the cases in the calculator; ID's that are not will be ignored. To reset to using all cases, do `activeCases(calc) <- integer()`.

activeCases `signature(calc = "mspathCalculator", value = "numeric")`: Sets the parameter values, but does not perform any calculations.

Calculation:

calculate `signature(calc = "mspathCalculator", params = "ANY", activeCases = "ANY", do.what = "ANY")`: Perform the requested calculation, return `calc` with the desired results. The other arguments are optional; if set, they will set the corresponding values in `calc` before the calculation. The returned calculator will have those new values.

estimateWork `signature(calc = "mspathCalculator")`: Return a matrix with each row being an individual case and each column some quantity that might be related to the amount of computation required for the case. See [estimateWork](#) for details.

simulateObservations `signature(object="mspathCalculator")`: Return a data.frame of simulated observations based on the actual observations and the model. See the discussion in [mspath](#) for details. This method uses R's random number generator; the caller must set that up appropriately before the simulation.

Analysis:

activeCases `signature(calc = "mspathCalculator")`: If a subset of the cases are active, returns an "integer" vector of the ID's of the active cases. Otherwise, return an empty vector.

effort `signature(calc = "mspathCalculator")`: Compute a single number giving the best estimate of likely effort. This is useful for gauging the *relative* times of different subsets; the absolute values have no particular meaning. In particular, they are not in seconds.

minus2loglik `signature(calc = "mspathCalculator")`: Returns -2 times the computed log-likelihood. The value is only meaningful if you have previously requested a computation.

nActiveCases `signature(calc = "mspathCalculator")`: count the number of cases currently under analysis, restricted to the active subset if any.

nCases `signature(calc = "mspathCalculator")`: The number of cases in the calculator. This counts all cases, not just the active subset. Each case consists of one or more consecutive records with the same ID.

nBadNodes `signature(x = "mspathCalculator")`: Number of invalid nodes generated.

nGoodNodes `signature(x = "mspathCalculator")`: Number of unique good nodes on these paths. Note that nodes may be shared between paths, and some "good" nodes turn out not to be on any good paths. Expected likelihood computation time is roughly proportional to this number.

nGoodPathNodes `signature(x = "mspathCalculator")`: Total number of nodes in all the good paths. `goodPathNodes/goodPaths` is the average path length. `goodNodes/goodPathNodes` estimates the speedup from sharing nodes between paths vs a naive implementation that treats each path independently.

nGoodPaths `signature(x = "mspathCalculator")`: Number of complete good paths on which to compute a likelihood.

results signature(calc = "mspathCalculator"): Returns the results of the calculations. You must have requested calculations first for this to be meaningful. Currently it is a numeric vector, but that may change. See [results](#) for details.

Cleanup:

done signature(calc = "mspathCalculator"): Call this when you are all done with the computations and this calculator. This ensures proper cleanup of the C++ objects.

Internal:

optimizeWork signature(calc = "mspathCalculator", effort = "numeric", chunks = "numeric"): for internal use only.

useActualTimes signature(calc = "mspathCalculator", chunks = "numeric"): for internal use only.

Note

Ordinarily one creates a calculator and then asks it to evaluate the likelihoods of different parameter sets, finally extracting the results. `calc <- mspathCalculator(...)` `calc <- calculate(calc, params)` `ll <- minus2loglik(calc)` `done(calc)`

Author(s)

Ross Boylan

See Also

Creation: [mspathCalculator](#), [mspathCalculatorFromArgs](#). Setup: [params<-](#), [estimateWork](#). Calculation: [calculate](#), [estimateWork](#). Analysis: [minus2loglik](#), [effort](#), [results](#). Cleanup: [done](#).

mspathCoefficients-class

Class "mspathCoefficients"

Description

This class contains coefficients for the multi-state path model. It knows about constraints. End users need not be aware of this class, except for the arguments to the print function.

Objects from the Class

Objects can be created by calls of the form `mspathCoefficients(permit, params, offset, baseConstrVec, covVars, constrVec, pathVars, pathConstrVec)`. The arguments are the same as for the slots, though the type restrictions are less severe; any numeric is fine. Note that "intercept" is not a legal name to pass in for `covVars` or `pathVars`, and that names should be unique across the last two arguments. I.e., don't give a covariate the same name as a path variable.

Slots

- permit:** "matrix" of "logical" values indicating which entries are permitted. The diagonal is always FALSE.
- params:** All the model parameters, in the canonical order for mspath. There is no distinction between free and fixed parameters in this class; use [mspathEstimatedCoefficients](#) if that's a concern. ("numeric")
- baseConstrVec:** "integer", starting at 1, for constraints on the intercepts.
- covVars:** Names of the covariates, if any ("character").
- constrVec:** "integer", starting at 1, with constraints on the covariates. Empty vector if no covariates.
- pathVars:** Names of path-dependent variables, if any ("character").
- pathConstrVec:** "integer", starting at 1, with constraints on the path variables. Empty if no path variables.
- iEff:** "integer" indices of effective parameters in params. This permits identification of the subset of params of interest.
- map:** map["vname"] gives a matrix of integer's. Each entry that is non-zero indicates that, for variable "vname" in that matrix position, the coefficient comes from the corresponding parameter. Use "intercept" to retrieve the intercepts, and the names of covariates or path-dependent variables to retrieve those values. This slot is a "list".

Methods

- coef** signature(object="mspathCoefficients"): Vector of all coefficients
- matrixCoef** A list of transition matrices of coefficients. The optional `coef` argument is a vector of names of coefficients of interest; otherwise all are reported.
- sd** signature(x="mspathCoefficients", na.rm="ANY"): Vector of 0's of same length as coefficients. This class is not for estimates. `na.rm` is ignored.
- matrixCoef** signature(x="mspathCoefficients", ...): A list of matrices with the coefficient values for each matrix element and 0 for those not allowed. The optional argument `coef` gives the names of terms to return; use "intercept" for the intercept.
- isAllFixed** signature(object="mspathCoefficients"): TRUE because all values are fixed in advance.
- print** signature(x="mspathCoefficients", ...): print out the coefficient values. The printout consists of blocks of coefficients related to a particular covariate. Each line indicates the matrix element(s) the coefficient applies to, the index in the parameter list from which the element comes, and the value of the coefficient. For the matrix element, `i -> j` indicates row `i`, column `j`, which is either the transition from `i` to `j` or true state `i` observed as state `j`. Parameters that apply to several covariates because of constraints will appear in several places. The optional named argument `coef` is a character vector of coefficient names. If provided, only those coefficients will be printed out, in the order given. Use "intercept" to get the intercept.
... allows one to use further arguments to control the details of the printout; see [printCoefmat](#) for details.
Returns `invisible(x)`.
- show** signature(object="mspathCoefficients"): print the object, returning `invisible(NULL)`.

Note

For full details of the ordering of the different values and the interpretation of the constraints, see [mspath](#).

In places above where the names of covariates or coefficients are called for, the exact meaning is names in the design matrix. So, for example, if `avar` is a factor, the actual names you will enter are `avar2`, `avar3`, etc (exact details depend on your choice of contrasts and the levels of the variables).

It is likely in the future this will include more methods, for example to multiply coefficients by covariates.

The terminology above is a bit loose and confusing. Sometimes covariate or coefficient refers to terms that include the intercept and the path variables. Sometimes coefficient refers to a particular estimated parameter. And sometimes, as in the “names of covariates” it refers to a covariate plus a factor addition.

Author(s)

Ross Boylan <ross@biostat.ucsf.edu>

See Also

[mspathEstimatedCoefficients](#) for estimated coefficients, [mspath](#) for general background, and [printCoefmat](#) for optional arguments to the print function.

Examples

```
permissible <- matrix(0, nrow=3, ncol=3)
permissible <- row(permissible)+1 == col(permissible)
covNames <- c("a", "b")
pathNames <- c("TIS", "TS0")
covConstraint <- seq(4)
covConstraint[4] <- 2
pathConstraint <- c(1, 1, 2, 3)
c <- mspathCoefficients(permissible,
                        params=seq(from=.1, by=.1, len=18),
                        offset=10,
                        baseConstrVec=seq(2),
                        covVars=covNames,
                        constrVec=covConstraint,
                        pathVars <- pathNames,
                        pathConstrVec <- pathConstraint
                        )
print(c)
```

mspathDistributedCalculator

Constructors for mspathCalculator Objects

Description

Generic function to compute the likelihood and or counts of the model described in the calculator under the given parameters. It does so by updating the calculator itself.

Usage

```
mspathDistributedCalculator(do.what, params, allinits, misc, subject, time, state, qvector, evector, c
    constrvec, misccovvec, misconstrvec, baseconstrvec, basemisconstrvec,
    pathvars, pathoffset, pathconstrvec,
    initprobs, nstates, nintens, nintenseffs, nmisc, nmisceffs, nobs, npts,
    ncovs, ncoveffs, nmisccovs, nmisccovseffs,
    npatheffs,
    isexact, fixedpars, stepnumerator, stepdenominator,
    comm=0,
    profile = FALSE)
```

Arguments

See [mspath](#) and the documentation on the C++ code for the full details.

Type of computation to perform, as interpreted by the C++ code. This may be ignored, or only apply to the non-distributed calculators created by this one (need to check).

<code>do.what</code>	Ordinarily, the precise values here may be irrelevant, since they can be reset on specific calls to calculate . However, the size of this argument must be correct.
<code>allinits</code>	The “initial” values for all fixed and free parameters. Only the fixed values are used; the rest come from <code>params</code> , which may be reset during the life of the calculator.
<code>misc</code>	0 = no misclassification; 1 = full misclassification; 2 = simple, fixed misclassification.
<code>subject</code>	The ID, which must be an integer, for each row of data. The ID’s should be sorted in ascending order.
<code>time</code>	time of each observation, in ascending order within cases.
<code>state</code>	The state of each observation. States should be numbered 1 through <code>nstates</code> .
<code>qvector</code>	Vectorized matrix of allowed transitions, 1 for allowed, 0 for not allowed.
<code>evector</code>	Vectorized matrix of allowed misclassifications (on relevant if <code>do.what</code> is not 0).
<code>covvec</code>	vectorized matrix of covariate values.
<code>constrvec</code>	constraints for each covariate

misccovvec	Vectorized matrix of covariate values for misclassification (only relevant if misc = 1).
misconstrvec	list of constraints for each misclassification covariate
baseconstrvec	constraints on baseline transition intensities
basemisconstrvec	constraints on baseline misclassification probabilities
pathvars	character vector of the names of history-dependent variables.
pathoffset	add this double to every time 0 in the paths
pathconstrvec	constraints on path effects on intensities
initprobs	initial state occupancy probabilities
nstates	number of states
nintens	number of intensity parameters
nintenseffs	number of distinct intensity parameters
nmisc	number of misclassification rates
nmisceffs	number of distinct misclassification rates
nobs	number of observations in the data set
npts	number of individuals/cases in the data set
ncovs	number of covariates on transition rates
ncoveffs	number of distinct covariate effect parameters
nmiscovs	number of distinct misclassification parameters
nmiscoveffs	number of distinct misclassification effects
npatheffs	number of distinct path (history) effects on transitions
isexact	non-0 if we observed time of entry to absorbing states exactly (e.g., death)
fixedpars	which parameters to fix
stepnumerator	integer This divided by stepdenominator give the maximum step size in the discrete approximation, in the same scale as time.
stepdenominator	integer see stepnumerator
comm	The MPI communicator to use. The default ordinarily suffices.
profile	TRUE to enable profiling.

Value

Returns a new *mspathDistributedCalculator* object.

Note

The **Rmpi** library is loaded on object creation, and must be accessible.

Author(s)

Ross Boylan

See Also

[mspathDistributedCalculator](#), [mspath](#), [Rmpi](#)

mspathDistributedCalculator-class

Distributed Calculator for Multi-State Path Models

Description

This calculator distributes the calculation of the likelihood to slave processes, which compute a multi-state path model with possible non-Markov properties. Though you can use this class directly, it is usually more convenient to invoke [mspath](#). This is a subclass of (aka "extends") [mspathCalculator](#); the slaves also use that class to perform their computations.

Objects from the Class

Create objects of this class by calling [mspathDistributedCalculator](#), and clean them up by calling [done](#). One can also create a factory object via [mspathDistributedCalculatorFactory](#) and create instances via that object.

Slots

comm: MPI communicator ("integer")

work: Work estimate, to aid scheduling ("matrix")

worki: Sorted indices into the rows of work such that worki[1] is the row with the most work, worki[2] has the 2nd biggest workload, and so on ("integer")

analyzer: Holds a [runAnalyzer](#), but initially empty. ("ANY")

mode: requestCode or timerequestCode, depending on whether we are profiling ("integer")

cuts: show how to cut the work into jobs. These are indices into worki ("numeric").

maxlen: The longest list of ID's that will be sent ("integer")

cutids: ID's of first case in each group ("integer")

evals: Number of evaluations of this calculator. Useful for determining if we need initial timings or setup. Object of class "integer"

From [mspathCalculator](#):

args: A list of arguments to the C function call. As such, they must be in the correct order, with all necessary as.xxx's performed.

manager: Reference to the C++ object that performs the calculations. Object of class "externalptr". Much of the state of the [mspathCalculator](#) is really in this C++ object. The persistence of information between calls permits faster calculations, but raises the possibility for errors. Do *not* do anything with the manager.

do.what: "integer" with the code for the desired operation. This will be interpreted by the C++ routines, and is subject to being changed as a side-effect of user-level operations.

subset: "integer" of the ID's of cases for which the likelihood should be evaluated. The restriction to integers makes the interface with C easier. If this vector is empty, it means analyze all cases.

results: Object of class "numeric" holding the results of the calculation. In the future, it may hold other types. If there has not been a calculation with the current calculator values, this slot will have length 0. See [results](#) for details.

Extends

Class "mspathCalculator", directly.

Methods

calculate signature(calc = "mspathDistributedCalculator", params = "numeric", activeCases = "missing", do.what = "missing"): Perform the requested calculation by distributing it across slave processes. Returns calc with the desired results. Notice that distribution evaluates a given set of parameters (if params are not given the existing ones will be used) for all cases. Different subsets of the cases go to different processors.

done signature(calc = "mspathDistributedCalculator"): Call this to cleanup. Note it will leave the overall MPI session running. Returns an updated calc, but you shouldn't refer to the old or new values after calling this method.

params<- signature(calc = "mspathDistributedCalculator", value = "numeric"): Sets the free parameters (including for the slaves), but performs no calculation.

Note

Using this class requires proper initialization of the MPI environment.

Author(s)

Ross Boylan

See Also

[mspathCalculator](#), [mspath](#)

mspathDistributedCalculatorFactory

Make a Constructor for mspathCalculator Objects

Description

This function returns a function that behaves like [mspathCalculator](#), except that it produces a distributed calculator as the result of the call. The object returned is sometimes referred to as a factory or a constructor.

Usage

```
mspathDistributedCalculatorFactory(
  comm=0,
  profile = FALSE)
```

Arguments

comm	The MPI communicator to use. The default ordinarily suffices.
profile	TRUE to enable profiling.

Value

Returns a function that, when called with the arguments to [mspathCalculator](#), produces a new [mspathDistributedCalculator](#) object. That object will have the distributed calculator-specific arguments set with the values used in the calling this function..

Author(s)

Ross Boylan

See Also

[mspathDistributedCalculator](#), [mspathDistributedCalculator](#), [mspathCalculator](#), [mspath](#), **Rmpi**

mspathEstimatedCoefficients-class

mspathEstimatedCoefficients

Description

This class describes estimated coefficients for the multi-state path model. It knows which coefficients are fixed before estimation and which are constrained to be equal. It knows about the errors of the estimates as well. Ordinary users will only care about the arguments to the print function.

Objects from the Class

Create these objects by calling `mspathCoefficients(permit, params, offset, baseConstrVec, covVars, constrVec, pathVars, pathConstrVec, fixed=integer(0), n=Inf, var=numeric(0))`. `var` is the estimated variance for all free parameters in this and other coefficients. For other parameters, see slot definitions for details, except that arguments to this function can use more general types like `numeric`.

Slots

- fixed:** Indices, if any, of parameters fixed at the outset ("integer").
- n:** Sample size on which estimates are based ("numeric").
- foundSE:** "logical" TRUE if the estimate appears to have converged.
- var:** "numeric" estimate of the variances of the effective parameters, with 0's for the fixed parameters. Only valid if foundSE == TRUE; otherwise it will be numeric(0). *Careful:* the variances passed in to the constructor are for all free parameters in all coefficients; this slot has variances for both free and fixed parameters for this coefficient set only.
The remaining items are inherited from [mspathCoefficients](#):
- permit:** "matrix" of "logical" values indicating which entries are permitted. The diagonal is always FALSE.
- params:** All the model parameters, both free and fixed, in the canonical order for mspath. ("numeric")
- baseConstrVec:** "integer", starting at 1, for constraints on the intercepts.
- covVars:** Names of the covariates, if any ("character").
- constrVec:** "integer", starting at 1, with constraints on the covariates. Empty vector if no covariates.
- pathVars:** Names of path-dependent variables, if any ("character").
- pathConstrVec:** "integer", starting at 1, with constraints on the path variables. Empty if no path variables.
- map:** map["vname"] gives a matrix of integer's. Each entry that is non-zero indicates that, for variable "vname" in that matrix position, the coefficient comes from the corresponding parameter. Use "intercept" to retrieve the intercepts, and the names of covariates or path-dependent variables to retrieve those values. This slot is a "list".

Extends

Class "mspathCoefficients", directly.

Methods

- isAllFixed** signature(object="mspathCoefficients"): usually FALSE, but TRUE if all parameters are fixed.
- coef** signature(object="mspathEstimatedCoefficients"): Vector of all coefficients, fixed and free. These are the effective coefficients, meaning that if constraints make two coefficients equal, only a single term will appear for them.
- sd** signature(x="mspathCoefficients"), na.rm="ANY": Vector of estimated standard deviations of the results of coef(). The na.rm probably won't work correctly.
- print** signature(x="mspathEstimatedCoefficients", coeff, ...): print out the coefficient values. The printout consists of blocks of coefficients related to a particular covariate. Each line indicates the matrix element(s) the coefficient applies to, the index in the parameter list from which the element comes (negative index if fixed), the estimated (or fixed) value of the coefficient, the estimated standard error of the estimate (or NA for fixed values), and various measures of statistical significance.
Parameters that apply to several covariates because of constraints will appear in several places.

The optional argument `coeff` is a character vector of coefficient names. If provided, only those coefficients will be printed out, in the order given.

... allows one to use further arguments to control the details of the printout; see [printCoefmat](#) for details.

Purists will note that neither `coeff` nor ... is part of the method signature. They are acceptable arguments to the function, however.

Returns `invisible(x)`.

show signature(object="mspathEstimatedCoefficients"): print the object, returning `invisible(NULL)`.

Note

`foundSE` will be TRUE if the hessian is positive definite. In that case, `var` has the diagonal of the inverse of the hessian.

Author(s)

Ross Boylan <ross@biostat.ucsf.edu>

See Also

[mspath](#) gives full information about parameter order and constraints. [mspathCoefficients](#) is my parent class. [printCoefmat](#) describes additional optional arguments to the print function.

Examples

```
# 5 x 5, history but no covariates
permissible <- matrix(c( 0, 1, 1, 0, 1,
                       1, 0, 1, 1, 1,
                       0, 0, 0, 0, 1,
                       1, 0, 0, 0, 1,
                       0, 0, 0, 0, 0), ncol=5, byrow=TRUE)
pathConstraint <- c(1, 1, 2, 3, 1, 1, 4, 5, 2, 6,
                   7, 8, 2, 8, 7, 8, 9, 10, 11, 12)
baseConstraint <- c(1, 1, 2, 3, 1, 1, 4, 5, 2, 6)

pathNames <- c("TIS", "TSO")
fixed <- c(3, 10, 11)
var <- seq(from=30, by=5, length.out=15)^2
c <- mspathEstimatedCoefficients(permissible,
                                 params= 101:118,
                                 baseConstrVec=baseConstraint,
                                 pathVars=pathNames,
                                 pathConstrVec = pathConstraint,
                                 fixed=fixed,
                                 n=50,
                                 var=var)

# default show method
c

# optional print arguments
```

```
print(c, coeff=c("intercept", "TIS"), digits=2)
```

q2

Allowed Transitions: q2

Description

Matrix of allowed transitions among 5 states.

Format

A 5x5 matrix of 0's and 1's indicating allowed transitions between states. The diagonal is ignored.

Details

This matrix allows only transitions up to the next-highest state. It is appropriate, for example, for progressive diseases in an analysis for which the time step on the path is sufficiently small that transitions across multiple states are not present in a single step.

Source

Made up.

Examples

```
library(mspath)
data(q2, e2, sim2)
r <- mspath(fib~time, misc=TRUE, ematrix=e2, qmatrix=q2, inits=rep(.5, 9), subject=id,
            data=sim2, stepnumerator=1, stepdenominator=1, initprobs=c(1.0, 0, 0, 0, 0),
            do.what=0)
```

readingError-class

Class "readingError"

Description

Describes errors in readings for discrete states. This is a special type of `matrix` in which rows are the true states and columns are observed states. The diagonal is mostly ignored, since it must be 1 - sum of other elements on the row.

Objects from the Class

Objects can be created by calls of the form `readingError(...)` where the arguments are the usual ones for `matrix`.

Slots

.Data: Object of class "matrix"

Extends

Class "matrix", from data part. Class "structure", by class "matrix", distance 2. Class "array", by class "matrix", distance 2. Class "vector", by class "matrix", distance 3, with explicit coerce. Class "vector", by class "matrix", distance 4, with explicit coerce.

Methods

bitMask signature(object = "readingError"): returns a matrix with 1's on off-diagonal positive elements, 0 elsewhere.

boolMask signature(object = "readingError"): return a matrix with code TRUE on off-diagonal positive elements, FALSE elsewhere.

coef signature(object = "readingError"): returns a vector with only the off-diagonal positive elements, starting with the first row, then the second, and so on.

mean signature(x = "readingError"): The arguments are one or more readingError's. This returns a readingError whose entries are the means of each of the cells in the arguments. Note that extra arguments to mean elsewhere in R provide optional tuning values for the calculation; that is not the case here.

Note

This class is designed to work with `mspath` and may not be appropriate for other uses. In particular, `coef` returns values in the order expected by the `inits` argument to `mspath`, which is the transpose of the usual order for R.

Author(s)

Ross Boylan

References

Bacchetti, Peter and Boylan, Ross (2009) "Estimating Complex Multi-State Misclassification Rates for Biopsy-Measured Liver Fibrosis in Patients with Hepatitis C," *The International Journal of Biostatistics*: Vol. 5 : Iss. 1, Article 5. DOI: 10.2202/1557-4679.1139 <http://www.bepress.com/ijb/vol5/iss1/5>

See Also

`matrix`, `mspath`, and sample data provided with this package.

Examples

```
library(mspath)
re <- readingError(c(.73, .27, 0, 0, 0,
                    .02, .73, .25, 0, 0,
                    0, .02, .90, .08, 0,
```

```

                                0, 0, 0, .88, .13,
                                0, 0, 0, .07, .93),
                                byrow=TRUE, nrow=5, ncol=5)
coef(re)

```

results *Retrieve Results of Multi-State Path Calculation*

Description

Get results of previous calculations.

Usage

```
results(calc)
```

Arguments

calc A previously evaluated calculator, which should be a [mspathCalculator](#) or subclass.

Value

Currently numeric of length 6 with the following entries, each being a sum over the active cases:

- 1: -2 Log-likelihood
Minus twice the log-likelihood. This will only be a meaningful value if the calculation had an appropriate `do.what`.
- 2: Number of Cases
This counts subjects, each of which may have several records.
- 3: Good Paths
Number of distinct paths consistent with the data and the model.
- 4: Good Nodes
Presumed to be the best single predictor of effort, in particular of floating-point operations. This is the number of unique nodes on good paths.
- 5: Bad Nodes
Number of nodes considered that fell outside of the good paths. May vary with the pruning strategy used, but at any rate note that these nodes need not be constructed.
- 6: Good Path Nodes
This is the sum of all the nodes in all the good paths, ignoring the fact that some nodes are shared between paths. This is a measure of the work effort that a naive implementation would require. The current implementation is not naive.

Which of those values has useful data depends in particular on the value of `do.what`. Other, more complex, types of return values are possible.

Methods

calc = "mspathCalculator" standard case

Author(s)

Ross Boylan

See Also

[mspathCalculator](#), [estimateWork](#).

runAnalyzer-class *Class to Analyze Runtimes*

Description

Assists the analysis of the factors affecting the performance of jobs. It requires individual jobs to be identified by the ID's of the cases the job computed.

Details

`runAnalyzer(estimate)` constructs an object of this class. `estimate` should be a dataframe or matrix with each row giving information for a single case. The 'ID' column, which must exist, is the ID of the case. Other columns give other values believed to predict how long the computation on the case will take. It should be sensible to sum a predictor across cases to predict how long the cases will take.

A `runTime` object includes information about a particular job, in particular various measures of time. To use the object with a `runAnalyzer` requires that the job be constructed with the list of ID's it will analyze, e.g., `aRunTime <- runTime(c(1, 56, 90))` for a job that will evaluate cases 1, 56, and 90. If `ra` is a `runAnalyzer`, then add the results of a run with `ra <- addResult(ra, aRunTime)`.

Once you are done `results <- smoosh(ra)` gets you a summary of the runs. Each row corresponds to a single job; the 'ID' column will have the ID of the first case in the job. Don't be fooled; the rest of the row applies to the whole job, and hence to all the cases in the job. The columns give the sum of the predictors for each case in the job, and the various performance measures, labelled appropriately: `cpu`, `wall`, `wait`, `start`, `end`, `rank`. See `runTime` for their meaning.

If you wish to repeat the exercise, use `ra <- newRun(ra)`.

Currently, evaluation of the performance of the predictors is up to you, and if you want to look at previous runs you will need to pull them out with `ra@prior`.

Objects from the Class

`runAnalyzer(estimate)` creates one of these objects. The `estimate` should contain estimates of the runtimes of the jobs that will later be added.

Slots

estimate: A "matrix" or "data.frame" with one row for each case to be analyzed. There must be a column named 'ID' whose values match those identifying a job in the `runTimes`. Other columns contain possible predictors of how much computation the case will require.

actual: An initially empty "list" that will hold the `runTime`'s of the jobs.

prior: Only relevant if you conduct more than one round of analysis, i.e., call `newRun`. In that case it will be a list of list's. The first entry holds the values in `actual` for the first run, the second entry holds the second run, and so on.

Methods

`addResult` signature(`analyzer = "runAnalyzer"`, `runTime = "runTime"`): to add a job and its time.

`newRun` signature(`analyzer = "runAnalyzer"`): Return the analyzer argument, ready for a new round of analysis.

`smoosh` signature(`analyzer = "runAnalyzer"`): ... `smoosh`

signature(`analyzer = "runAnalyzer"`): returns a `data.frame` or `matrix` summarizing the results.

Background

This class assists in the following general setting. You have computations to perform on cases; each job may calculate values for one or more cases, e.g., you may distribute the calculation of a likelihood.

You also have information that you think will predict how long the cases will take to compute.

You divide the cases into jobs, perform the computations, and then want to see how well your predictors explain various measures of performance.

Based on this analysis, or perhaps randomly, you may partition the cases in a different way and try again.

This class collects the results of all the runs (including prior rounds if you choose to do more than one round), and can provide a summary of the performance versus the predictors.

Note

Subject to change. Not terribly general at the moment, and rather hackish.

Author(s)

Ross Boylan

See Also

`addResult` to populate; `smoosh` to analyze; `runTime` for internal data.

Examples

```
# slightly contrived, since non-distributed
estimate <- data.frame(ID=c(30, 50), estimate=exp(c(30, 50)))
analyzer <- runAnalyzer(estimate)
for( x in c(30, 50) ) {
  rt <- runTime(x)
  mpirank(rt) <- 1
  tjob <- system.time(factorial(x))
  remoteTime(rt) <- c(tjob[1:3], 0) # last number is delay wait
  analyzer <- addResult(analyzer, rt)
}
result <- smoosh(analyzer)
```

runeverywhere

Top Level of Distributed Calculation for Multi-State Path Model

Description

Ordinarily every node in a distributed calculation invokes this function. It determines whether it is running on a master (rank 0) or slave, and invokes the appropriate function. This permits SPMD-style programming despite the difference between the master and slave processes.

Usage

```
runeverywhere(channel = "kickStart.R", comm = 0)
```

Arguments

channel	The source of commands to execute on the master process. See master for details.
comm	The MPI communicator to use. In most cases, the default should suffice.

Details

This is just a thin wrapper around [master](#) and [slave](#).

Value

Terminates the R session at the end.

Note

Do *not* load the Rmpi library before invoking this function. Do set up the MPI environment and launch all the necessary R processes.

Author(s)

Ross Boylan

See Also

[master](#), [slave](#)

runTime

runTime constructor

Description

Create a runTime object associated with a particular job.

Usage

```
runTime(job)
```

Arguments

job	Any object, though much of the rest of the system expects this to be a vector of the cases analyzed.
-----	--

Details

The constructor also records the current time; thus, when the object is created matters. Later calls will need to set information about when the job completed.

Value

An S4 [runTime](#) object.

Note

The newly created object is incomplete, since it doesn't have information about when the job finished.

Users are not likely to care about this object or function.

Author(s)

Ross Boylan

See Also

[runTime](#)

runTime-class

Class "runTime"

Description

This low-level class captures the time it takes to run a particular job. This is for further analysis by [runAnalyzer](#).

Objects from the Class

Create with `runTime(job)`. It automatically records the time it was created, so don't make it until you're ready.

The sequence of events is that the job is created locally, started remotely, finished remotely, and completed locally. Scheduling and transmission delays may occur.

Slots

job: Some indicator of the job whose time is being described. Could be the job object itself or, in current application, the id of the case being analyzed. Object of class "ANY"

start: time in seconds since start of process that job began. Object of class "numeric"

end: time in seconds since start of process that job ended (set when `remote<-` is called). Object of class "numeric"

remote: seconds of user, system, wall clock, and wait time on the remote system. wall clock time is between start of remote computation and end of main computations; wait time is time waiting for response from root. Object of class "numeric"

rank: MPI rank of job. Object of class "numeric"

Methods

addResult signature(`analyzer = "runAnalyzer"`, `runTime = "runTime"`): Once a `runTime` object is complete, use this method to add it to an analyzer for later analysis. This uses the R idiom in which the return value is the updated `runAnalyzer` object.

cpuTime signature(`runTime = "runTime"`): CPU seconds the job took. If the job executed remotely, this is only the remote CPU time.

endTime signature(`runTime = "runTime"`): time in seconds since start of process that job ended (set when `remote<-` is called). Object of class "numeric"

job signature(`runTime = "runTime"`): Get the job that goes with this `runTime`.

mpirank<- signature(`runTime = "runTime"`, `value = "numeric"`): Sets the MPI rank of the job, once it is known. The return value is the `runTime` object with the updated rank.

mpirank signature(`runTime = "runTime"`): Get the MPI rank of the job.

remoteTime<- signature(`runTime = "runTime"`, `value = "numeric"`): Return a `runTime` object updated with the vector of time values from the remote job. See slot `remote` for the exact form the argument should take. This also sets the `endTime` of the object as a side-effect.

startTime signature(`runTime = "runTime"`): time in seconds since start of process that job began. Object of class "numeric".

waitTime signature(`runTime = "runTime"`): Number of wall-clock seconds the remote jobs was waiting for a response from root after the remote job's completion.

wallTime signature(`runTime = "runTime"`): Number of seconds from the time the master process initiated the job to the time it was finished with it (i.e. `endTime - startTime`).

Note

Resolution of time is system-dependent; it's whatever `proc.time` can get.

The creator of the object is responsible for assuring that the method definitions above are true. In particular,

1. Create the `runTime` object only when the corresponding job is started.
2. Populate `mpirank` with a proper value.
3. Set appropriate values with `remoteTime<-` and make this call exactly when all processing on the job is done.

In order to retain the updated values, you must capture the return value of the update methods, in keeping with the general functional style of R.

Author(s)

Ross Boylan

See Also

[runTime](#) constructor, [runAnalyzer](#), [proc.time](#)

Examples

```
### Make a runTime object
# at the start of the job
rt <- runTime("Job1")
# in real life, remote computations would then ensue
mpirank(rt) <- 3
Sys.sleep(1.5) # time passes
# when we get the job back
# next line is c( user cpu, system cpu, wall clock time, wall clock wait)
remoteTime(rt) <- c(.05, .01, 1.0, .15)

### Use it
cat("Job", job(rt), "executed on MPI rank", mpirank(rt))
cat("It took", wallTime(rt),
    "seconds, out of which remote waited for master",
    waitTime(rt), "seconds.")
```

simulated HCV data *HCV Progression after liver transplant*

Description

Simulated data for individuals with liver transplants, giving covariates, time since transplant, and observed liver state. Cases generally have repeated biopsies.

Format

A data frame. `sim1` has 21 cases with 46 observations; `sim2` has 335 cases with 789 records, and `sim3` has 1,340 cases with 3,356 records.

`id` The patient id.

`fib` Fibrosis state from 1 through 5, using the Metavir scale (see the reference by Bedosa et al. We actually use the Metavir value +1 because the algorithm expects 1 as the first state, not 0.). 1 means a healthy liver, with higher numbers representing increasing liver damage. 5 means cirrhosis.

`time` Years since transplant.

`k` (`sim1` only) Standard Normal covariate intended to vary between cases but not across time within a case. The intention was not realized: it varies within cases as well.

`x1` (`sim1` only) Standard Normal covariate, drawn independently for each time of each case.

`C1` (`sim2` and `sim3`) Normally distributed continuous variable.

`C2` (`sim2` and `sim3`) Exponentially distributed continuous variable.

`D3` (`sim2` and `sim3`) binary variable

`C4` (`sim2` and `sim3`) age, from a uniform initial age and time of observation.

`K5` (`sim2` and `sim3`) Normally distributed continuous variable, constant within case.

`K6` (`sim2` and `sim3`) Binary variable, constant within case.

Details

The datasets include a time 0 observation of fibrosis stage 1 (uninfected) for each case. Note the times are time of observation, *not* time of entry into a state. Fibrosis is observed with error; errors are likely to produce a reading lower than the actual score.

As required by `mspath`, observations are sorted by `id` and then `time`.

It is unusual to know the exact time of infection and have repeated observations per case. We were interested in a study of transplant patients that had both features.

The observed fibrosis stage was generated by simulating the `mspath` model, using `do.what=10`. As an example, after generating the cases, covariates and observation times for `sim1` the outcomes came from

```

data(q2, e2, sim1)
cons1 <- list(k=c(1, 1, 2, 2), x1=rep(3, 4))
init1 <- c(seq(-1, by=.2, length=4),# intercepts
          0.5, 0.8, #k
          -1.0, # x1
          -0.5, .7, # history
          .2, .05 # misclassification
          )

r <- mspath(fib~time, q2, misc=SIMPLE, e2,
            inits=init1,
            subject=id,
            covariates=~k+x1,
            constraint=cons1,
            econstraint=c(1, 2, 1, 2, 1),
            pathvars=c("LN(TSO)", "TIS"),
            pathoffset=0.5,
            pathconstraint=list("LN(TSO)"=rep(1, 4), "TIS"=rep(2, 4)),
            data=sim1,
            isexact=TRUE,
            fixedpars=seq(along=init1),
            stepnumerator=1,
            stepdenominator=1,
            do.what=10)

```

The final `sim1` was extracted from the return value `r`.

Source

See the reference by Berenguer et al. for a published study using a dataset in the same format as this simulated one. That study did not use multi-state path methods.

References

- Bedosa P, Pynard T. An algorithm for the grading of activity in chronic hepatitis C. The METAVIR Cooperative Study Group. *Hepatology* 1996. 24:289-93.
- Berenguer M, Ferrell L, Watson J, et al. HCV-related fibrosis progression following liver transplantation: increase in recent years. *J Hepatology*, 32:673-84, 2000.

Examples

```

library(mspath)
data(q2, e2, sim2)
r <- mspath(fib~time, misc=TRUE, ematrix=e2, qmatrix=q2, inits=rep(.5, 9), subject=id,
            data=sim2, stepnumerator=1, stepdenominator=1, initprobs=c(1.0, 0, 0, 0, 0),
            do.what=0)

```

slave	<i>Prepare for Distributed Multi-State Path Model Computations on Slaves</i>
-------	--

Description

This function kicks off the main loop of distributed slave processes. The master process sends them work, and eventually an indication to shut down. At that point the function shuts down R.

Usage

```
slave(comm = 0)
```

Arguments

comm	An MPI communicator to use to talk to the master process
------	--

Details

The slave's outer loop receives the arguments from which to construct a local calculator. It will exit if it gets NULL arguments. Once the calculator is set up, the slave receives free parameters values. Finally, it repeatedly asks the master for cases to evaluate and returns the results of evaluating those cases to the master. Generally there will be many sets of cases evaluated for each set of parameters, and many sets of parameters for each set of arguments.

Value

Returns likelihoods to the master process. This function never returns, since it shuts down the R process when done.

Note

Usually invoked from [runeverywhere](#). You must set up the necessary MPI environment for the slave to do useful work. Do not load the Rmpi library, but be sure that `library(Rmpi)` will be able to access the library.

Author(s)

Ross Boylan

See Also

[runeverywhere](#), [master](#), [mspathDistributedCalculator](#), **Rmpi**

`smoosh`*Analyze Runtime Estimates and Actual Values*

Description

This method combines the observed runtimes of jobs with estimates of same. The analysis is left for the caller to do.

Usage

```
smoosh(analyzer)
```

Arguments

`analyzer` A [runAnalyzer](#) previously populated with the runtimes of jobs.

Value

A summary of the runs, as either a `link{matrix}` or `data.frame`. Each row corresponds to a single job; the 'ID' column will have the ID of the first case in the job. Don't be fooled; the rest of the row applies to the whole job, and hence to all the cases in the job. The columns give the sum of the predictors (from the estimate given when the [runAnalyzer](#) was constructed) for each case in the job, and the various performance measures from the [runTime](#) for the job, labelled appropriately: `cpu`, `wall`, `wait`, `start`, `end`, `rank`. See [runTime](#) for their meaning.

Author(s)

Ross Boylan

See Also

[runAnalyzer](#), [runTime](#)

`trial-class`*Class "trial"*

Description

Holds results of a trial done by [doTrials](#)

Objects from the Class

Objects can be created by calls of the form `trial(args, results)`.

Slots

args: The "list" of arguments for this trial, as produced by [doTrials](#)' generator.

result: The interesting information from the trial, as returned by [doTrials](#)' executor.

Methods

No methods defined with class "trial" in the signature.

Note

You may wish to provide methods that are meaningful for your particular analysis. You can access the raw results with the basic slot syntax, e.g., `aTrial@args`

Author(s)

Ross Boylan <ross@biostat.ucsf.edu>

See Also

To understand the context in which this is used, see [doTrials](#).

Index

- *Topic **array**
 - readingError-class, 43
- *Topic **classes**
 - addResult, 3
 - mspath-class, 26
 - mspathCalculator-class, 31
 - mspathCoefficients-class, 33
 - mspathDistributedCalculator-class, 38
 - mspathEstimatedCoefficients-class, 40
 - readingError-class, 43
 - runAnalyzer-class, 46
 - runTime-class, 50
 - smoosh, 55
 - trial-class, 55
- *Topic **datagen**
 - doTrials, 9
 - mspath.subset, 28
 - smoosh, 55
 - trial-class, 55
- *Topic **datasets**
 - e2, 11
 - q2, 43
 - readingError-class, 43
 - simulated HCV data, 52
- *Topic **environment**
 - alldone, 3
 - runTime, 49
- *Topic **file**
 - checkpoint, 5
- *Topic **interface**
 - calculate, 4
 - mspathCalculator-class, 31
- *Topic **iteration**
 - checkpoint, 5
 - doTrials, 9
 - fixedSchedule, 15
 - trial-class, 55
- *Topic **manip**
 - mspath.subset, 28
- *Topic **math**
 - deltamethod, 7
 - expit, 14
 - logit, 16
 - MatrixExp, 18
- *Topic **methods**
 - calculate, 4
 - done, 8
 - effort, 12
 - estimateWork, 13
 - isAllFixed-methods, 15
 - minus2loglik, 19
 - mspathCalculator-class, 31
 - results, 45
 - runTime-class, 50
- *Topic **models**
 - calculate, 4
 - doTrials, 9
 - effort, 12
 - minus2loglik, 19
 - mspath, 20
 - mspath-class, 26
 - mspathCalculator, 29
 - mspathCalculator-class, 31
 - mspathDistributedCalculator, 36
 - mspathDistributedCalculator-class, 38
 - mspathDistributedCalculatorFactory, 39
 - results, 45
 - runeverywhere, 48
 - slave, 54
 - trial-class, 55
- *Topic **multivariate**
 - calculate, 4
 - mspathCalculator-class, 31
- *Topic **nonlinear**

- calculate, 4
- mspathCalculator-class, 31
- *Topic **optimize**
 - checkpoint, 5
 - doTrials, 9
 - trial-class, 55
- *Topic **programming**
 - runTime, 49
- *Topic **survival**
 - calculate, 4
 - mspathCalculator-class, 31
- *Topic **univar**
 - estimateWork, 13
 - mspathCalculator-class, 31
 - results, 45
- *Topic **utilities**
 - checkpoint, 5
 - done, 8
 - doTrials, 9
 - effort, 12
 - estimateWork, 13
 - fixedSchedule, 15
 - master, 17
 - runAnalyzer-class, 46
 - runeverywhere, 48
 - slave, 54
 - smoosh, 55
 - trial-class, 55
- activeCases (mspathCalculator-class), 31
- activeCases, mspathCalculator-method (mspathCalculator-class), 31
- activeCases-method (mspathCalculator-class), 31
- activeCases<- (mspathCalculator-class), 31
- activeCases<-, mspathCalculator, numeric-method (mspathCalculator-class), 31
- activeCases<--methods (mspathCalculator-class), 31
- addResult, 3, 47
- addResult, runAnalyzer, runTime-method (runTime-class), 50
- alldone, 3
- array, 44
- bitMask (readingError-class), 43
- bitMask, readingError-method (readingError-class), 43
- boolMask (readingError-class), 43
- boolMask, readingError-method (readingError-class), 43
- calculate, 4, 12, 29, 33, 36
- calculate, mspathCalculator, ANY, ANY, ANY-method (calculate), 4
- calculate, mspathDistributedCalculator, numeric, missing, miss (calculate), 4
- calculate-methods (calculate), 4
- checkpoint, 5, 24–26
- coef, mspath-method (mspath-class), 26
- coef, mspathCoefficients-method (mspathCoefficients-class), 33
- coef, mspathEstimatedCoefficients-method (mspathEstimatedCoefficients-class), 40
- coef, mspathFull-method (mspath-class), 26
- coef, readingError-method (readingError-class), 43
- cpuTime (runTime-class), 50
- cpuTime, runTime-method (runTime-class), 50
- data.frame, 6, 55
- deltamethod, 7
- done, 8, 33, 38
- done, mspathCalculator-method (done), 8
- done, mspathDistributedCalculator-method (done), 8
- done-methods (done), 8
- doTrials, 9, 55, 56
- e2, 11
- effort, 12, 33
- effort, mspathCalculator-method (effort), 12
- effort-methods (effort), 12
- endTime (runTime-class), 50
- endTime, runTime-method (runTime-class), 50
- estimateWork, 13, 32, 33, 46
- estimateWork, mspathCalculator-method (estimateWork), 13
- estimateWork-methods (estimateWork), 13
- expit, 14, 16
- fixedSchedule, 15

- isAllFixed (isAllFixed-methods), 15
- isAllFixed, mspathCoefficients-method (mspathCoefficients-class), 33
- isAllFixed, mspathEstimatedCoefficients-method (isAllFixed-methods), 15
- isAllFixed-methods, 15

- job (runTime-class), 50
- job, runTime-method (runTime-class), 50

- logit, 14, 16

- master, 4, 17, 23, 26, 48, 49, 54
- matrix, 43, 44
- matrixCoef (mspathCoefficients-class), 33
- matrixCoef, mspath-method (mspath-class), 26
- matrixCoef, mspathCoefficients-method (mspathCoefficients-class), 33
- matrixCoef-methods (mspathCoefficients-class), 33
- MatrixExp, 18
- mean, readingError-method (readingError-class), 43
- minus2loglik, 5, 19, 33
- minus2loglik, mspath-method (mspath-class), 26
- minus2loglik, mspathCalculator-method (minus2loglik), 19
- minus2loglik-methods (minus2loglik), 19
- mpi.exit, 17
- mpirank (runTime-class), 50
- mpirank, runTime-method (runTime-class), 50
- mpirank<- (runTime-class), 50
- mpirank<-, runTime, numeric-method (runTime-class), 50
- mspath, 6, 17, 20, 25, 26, 28, 29, 31, 32, 35, 36, 38–40, 42, 44
- mspath-class, 26
- mspath.subset, 28
- mspathAbstractCalculator-class (mspathCalculator-class), 31
- mspathCalculator, 4, 5, 9, 12–14, 19, 23, 28, 29, 29, 31, 33, 38–40, 45, 46
- mspathCalculator-class, 31
- mspathCalculatorFromArgs, 31, 33
- mspathCalculatorFromArgs (mspathCalculator), 29
- mspathCoefficients, 26–28, 41, 42
- mspathCoefficients (mspathCoefficients-class), 33
- mspathCoefficients-class, 33
- mspathDistributedCalculator, 4, 5, 9, 14, 17, 36, 37, 38, 40, 54
- mspathDistributedCalculator-class, 38
- mspathDistributedCalculatorFactory, 38, 39
- mspathEstimatedCoefficients, 28, 34, 35
- mspathEstimatedCoefficients (mspathEstimatedCoefficients-class), 40
- mspathEstimatedCoefficients-class, 40
- mspathFull-class (mspath-class), 26
- nActiveCases (mspathCalculator-class), 31
- nActiveCases, mspathCalculator-method (mspathCalculator-class), 31
- nActiveCases-methods (mspathCalculator-class), 31
- nBadNodes (mspath-class), 26
- nBadNodes, mspath-method (mspath-class), 26
- nBadNodes, mspathCalculator-method (mspathCalculator-class), 31
- nBadNodes-methods (mspath-class), 26
- nCases (mspath-class), 26
- nCases, mspath-method (mspath-class), 26
- nCases, mspathCalculator-method (mspathCalculator-class), 31
- nCases-methods (mspath-class), 26
- newRun (runAnalyzer-class), 46
- newRun, runAnalyzer-method (runAnalyzer-class), 46
- nGoodNodes (mspath-class), 26
- nGoodNodes, mspath-method (mspath-class), 26
- nGoodNodes, mspathCalculator-method (mspathCalculator-class), 31
- nGoodNodes-methods (mspath-class), 26
- nGoodPathNodes (mspath-class), 26
- nGoodPathNodes, mspath-method (mspath-class), 26
- nGoodPathNodes, mspathCalculator-method (mspathCalculator-class), 31

- nGoodPathNodes-methods (mspath-class), 26
- nGoodPaths (mspath-class), 26
- nGoodPaths, mspath-method (mspath-class), 26
- nGoodPaths, mspathCalculator-method (mspathCalculator-class), 31
- nGoodPaths-methods (mspath-class), 26

- optim, 24, 26–28
- optimizeWork (mspathCalculator-class), 31
- optimizeWork, mspathCalculator, numeric, numeric-method (mspathCalculator-class), 31
- optimizeWork-methods (mspathCalculator-class), 31
- optresults (mspath-class), 26
- optresults, mspath-method (mspath-class), 26
- optresults-methods (mspath-class), 26
- optresults<- (mspath-class), 26
- optresults<-, mspath-method (mspath-class), 26
- optresults<--methods (mspath-class), 26

- params<- (mspathCalculator-class), 31
- params<-, mspathCalculator, numeric-method (mspathCalculator-class), 31
- params<-, mspathDistributedCalculator, numeric-method (mspathDistributedCalculator-class), 38
- params<--methods (mspathCalculator-class), 31
- params<- , 33
- print, mspath-method (mspath-class), 26
- print, mspathCoefficients-method (mspathCoefficients-class), 33
- print, mspathEstimatedCoefficients-method (mspathEstimatedCoefficients-class), 40
- print, mspathFull-method (mspath-class), 26
- printCoefmat, 34, 35, 42
- printFooter (mspath-class), 26
- printFooter, mspath-method (mspath-class), 26
- printFooter-methods (mspath-class), 26
- proc.time, 51

- q2, 43

- readingError (readingError-class), 43
- readingError-class, 43
- remoteTime<- (runTime-class), 50
- remoteTime<- , runTime, numeric-method (runTime-class), 50
- results, 5, 14, 19, 31, 33, 39, 45
- results, mspathCalculator-method (results), 45
- results-methods (results), 45
- runAnalyzer, 3, 38, 46, 50, 51, 55
- runAnalyzer (runAnalyzer-class), 46
- runAnalyzer-class, 46
- runeverywhere, 48, 54
- runTime, 3, 46, 47, 49, 49, 51, 55
- runTime-class, 50

- sd, mspath-method (mspath-class), 26
- sd, mspathCoefficients-method (mspathCoefficients-class), 33
- sd, mspathEstimatedCoefficients-method (mspathEstimatedCoefficients-class), 40
- sd, mspathFull-method (mspath-class), 26
- show, mspath-method (mspath-class), 26
- show, mspathCoefficients-method (mspathCoefficients-class), 33
- show, mspathEstimatedCoefficients-method (mspathEstimatedCoefficients-class), 40
- sim1 (simulated HCV data), 52
- sim2 (simulated HCV data), 52
- sim3 (simulated HCV data), 52
- simulated HCV data, 52
- simulateObservations (mspathCalculator-class), 31
- simulateObservations, mspathCalculator-method (mspathCalculator-class), 31
- simulateObservations-methods (mspathCalculator-class), 31
- slave, 48, 49, 54
- smoosh, 47, 55
- smoosh, runAnalyzer-method (runAnalyzer-class), 46
- startTime (runTime-class), 50
- startTime, runTime-method (runTime-class), 50
- structure, 44

`Sys.time`, 6

`trial`, 10
`trial` (`trial-class`), 55
`trial-class`, 55

`useActualTimes`
 (`mspathCalculator-class`), 31
`useActualTimes`, `mspathCalculator`, `numeric-method`
 (`mspathCalculator-class`), 31
`useActualTimes-methods`
 (`mspathCalculator-class`), 31

`vector`, 44

`waitTime` (`runTime-class`), 50
`waitTime`, `runTime-method`
 (`runTime-class`), 50
`wallTime` (`runTime-class`), 50
`wallTime`, `runTime-method`
 (`runTime-class`), 50