Package ‘hadron’

September 10, 2020

Version 3.1.2

Date 2020-09-09

Title Analysis Framework for Monte Carlo Simulation Data in Physics

SystemRequirements Gnu Scientific Library version >= 1.8

Description Toolkit to perform statistical analyses of correlation
functions generated from Lattice Monte Carlo simulations. In
particular, a class 'cf' for correlation functions and
methods to analyse those are defined. This includes (blocked)
bootstrap (based on the 'boot' package) and jackknife, but also an
automatic determination of integrated autocorrelation
times. 'hadron' also provides a very general function
bootstrap.nlsfit() to bootstrap a non-linear least squares fit.
More specific functions are provided to extract hadronic quantities
from Lattice Quantum Chromodynamics simulations, a particular Monte
Carlo simulation,(see e.g. European Twisted Mass Collaboration, P. Boucaud et
al. (2008) <doi:10.1016/j.cpc.2008.06.013>). Here, to determine
energy eigenvalues of hadronic states, specific fitting routines
and in particular the generalised eigenvalue method (see
e.g. B. Blossier et al. (2009) <doi:10.1088/1126-6708/2009/04/094>
and M. Fischer et al. (2020)
<https://inspirehep.net/literature/1792113>) are implemented.
In addition, input/output and plotting routines are available.

Imports abind, boot, dplyr, Rcpp, R6, stringr

LinkingTo Rcpp

Suggests minpack.lm, parallel, rhdf5, knitr, testthat, tictoc,
tikzDevice, hash, numDeriv, staplr, markdown, rmarkdown, errors

License GPL-3

URL https://github.com/HISKP-LQCD/hadron

BugReports https://github.com/HISKP-LQCD/hadron/issues

LazyData true

RoxygenNote 7.1.1

Encoding UTF-8

VignetteBuilder knitr

NeedsCompilation yes

Author Bartosz Kostrzewa [aut],
Johann Ostmeyer [aut],
Martin Ueding [aut],
Carsten Urbach [aut, cre],
Nikolas Schlage [ctb],
Markus Werner [ctb],
Ferenc Pittler [ctb],
Matthias Fischer [ctb],
Vittorio Lubicz [ctb]

Maintainer Carsten Urbach <urbach@hiskp.uni-bonn.de>

Repository CRAN

Date/Publication 2020-09-10 06:40:02 UTC

R topics documented:

`*.cf` *Divide two cf objects by each other measurement by measurement*

Description

Note that no complex arithmetic is used, real and imaginary parts are treated as seperate and indepenent, such that the real part of one is the divided by the real part of the other and similarly for the imaginary parts.

Usage

```
## S3 method for class 'cf'
cf1 * cf2

## S3 method for class 'cf'
cf1 / cf2
```

Arguments

`cf1, cf2`	`cf_orig` objects.

Details

Note that this is generally only allowed on bootstrap samples and mean values, although it makes sense in some exeptional circumstances. Don't use this function unless you're certain that you should!

Value

The value is

$$cf1/cf2\,.$$

`*.raw_cf` *multiply two `raw_cf` objects*

Description

multiply two `raw_cf` objects

Usage

```
## S3 method for class 'raw_cf'
cf1 * cf2
```

Arguments

`cf1`	first 'raw_cf' container with data and meta-data to be multiplied
`cf2`	second 'raw_cf' container with data and meta-data to be multiplied

Value

`raw_cf` object with `cf$data == cf1$data * cf2$data`

`+.cf` *Arithmetically add correlators*

Description

Arithmetically add correlators

Usage

```
## S3 method for class 'cf'
cf1 + cf2
```

Arguments

`cf1, cf2`	`cf_orig` objects.

Value

The value is

$$cf1 + cf2 \,.$$

`+.raw_cf` *add two `raw_cf` objects*

Description

add two `raw_cf` objects

Usage

```
## S3 method for class 'raw_cf'
cf1 + cf2
```

Arguments

`cf1`	first 'raw_cf' container to be added
`cf2`	second 'raw_cf' container to be added

Value

`raw_cf` object with `cf$data == cf1$data + cf2$data`

`-.cf` *Arithmetically subtract correlators*

Description

Arithmetically subtract correlators

Usage

```
## S3 method for class 'cf'
cf1 - cf2
```

Arguments

`cf1, cf2`	`cf_orig` objects.

Value

The value is

$$cf1 - cf2 .$$

`-.raw_cf` *add two `raw_cf` objects*

Description

add two `raw_cf` objects

Usage

```
## S3 method for class 'raw_cf'
cf1 - cf2
```

Arguments

`cf1`	first 'raw_cf' container to be subtracted
`cf2`	second 'raw_cf' container to be subtracted

Value

`raw_cf` object with `cf$data == cf1$data -cf2$data`

`/.raw_cf` *divide two* `raw_cf` *objects*

Description

divide two `raw_cf` objects

Usage

```
## S3 method for class 'raw_cf'
cf1 / cf2
```

Arguments

`cf1`	'raw_cf' container with data and meta-data to be the dividend
`cf2`	'raw_cf' container with data and meta-data to be the divisor

Value

`raw_cf` object with `cf$data` == `cf1$data` / `cf2$data`

`add.cf` *Arithmetically adds two correlation functions*

Description

Arithmetically adds two correlation functions

Usage

```
add.cf(cf1, cf2, a = 1, b = 1)
```

Arguments

`cf1, cf2`	`cf_orig` object.
`a, b`	Numeric. Factors that multiply the correlation function before the addition. Since addition is associative, this operates also on the bootstrap samples and these are thus not invalidated in the process.

Value

The value is

$$aC_1 + bC_2\,.$$

`add.raw_cf` *add two* `raw_cf` *objects*

Description

add two `raw_cf` objects

Usage

```
add.raw_cf(cf1, cf2, a = 1, b = 1)
```

Arguments

`cf1`	first 'raw_cf' container with data and meta-data
`cf2`	second 'raw_cf' container with data and meta-data
`a`	Numeric or complex, scaling factor applied to `cf1`.
`b`	Numeric or complex, scaling factor applied to `cf2`.

Value

`a*cf1$data` + `b*cf2$data`

`addConfIndex2cf` *add a configuration index to an* `cf` *object*

Description

add a configuration number index to `cf` object.

Usage

```
addConfIndex2cf(cf, conf.index)
```

Arguments

`cf`	and object of class `cf`
`conf.index`	a configuration index of the same length as `cf`.

Value

Returns an object of class `cf` equal to the input but with element `conf.index` added

Author(s)

Carsten Urbach, `<urbach@hiskp.uni-bonn.de>`

See Also

cf

Examples

```
data(samplecf)
conf.index <- c(1:1018)
samplecf <- addConfIndex2cf(samplecf, conf.index=conf.index)
```

`addStat.cf` *Combine statistics of two cf objects*

Description

`addStat.cf` takes the raw data of two `cf` objects and combines them into one

Usage

```
addStat.cf(cf1, cf2)
```

Arguments

`cf1`	the first of the two `cf` objects to be combined
`cf2`	the second of the two `cf` objects to be combined

Details

Note that the two `cf` objects to be combined need to be compatible. Otherwise, `addStat.cf` will abort with an error.

Value

an object of class `cf` with the statistics of the two input `cf` objects combined

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

See Also

`cf`

Examples

```
data(samplecf)
## the following is not useful, but
## explains the usage
cfnew <- addStat.cf(cf1=samplecf, cf2=samplecf)
```

`addStat.raw_cf` *Extend statistics of an existing* `raw_cf` *container*

Description

Extend statistics of an existing `raw_cf` container

Usage

```
addStat.raw_cf(cf1, cf2)
```

Arguments

cf1	`raw_cf` container with or without 'data' and 'meta' mixins
cf2	`raw_cf` container with or without 'data' and 'meta' mixins

Details

When either of `cf1` or `cf2` does not contain any data, the other object is returned. (allows empty `raw_cf` to be extended). If the dimensions (except for the measurements) of the `data` fields of the two containers match, they are concatenated along the measurement dimension.

Value

An object of S3 class `raw_cf` identical to the input object but with extended statistics.

`alphas` *compute alpha strong at given scale*

Description

compute alpha strong (α_s) at given scale μ up to N3LO in PT in the RI' renormalisation scheme.

Usage

```
alphas(mu, nl = 3, lam0 = 0.25, Nc = 3, Nf = 2, use.cimpl = TRUE)
```

Arguments

`mu`	the renormalisation scale μ in GeV
`nl`	order in PT, range 0 to 3
`lam0`	Λ_{QCD} in GeV
`Nc`	number of colours N_c, defaults to 3
`Nf`	number of flavours N_f, default is 2
`use.cimpl`	Use the C implementation instead of the R implementation, which might improve speed.

Value

returns the value of alpha strong α_s at scale μ

Author(s)

Carsten Urbach, `<curbach@gmx.de>`, Vittorio Lubicz (of the original Fortran code)

See Also

`zetazp`

Examples

```
alphas(mu=2.0, nl=3)
```

`analysis_gradient_flow`
analysis_gradient_flow

Description

function to analyse the gradient flow output files generated by the tmLQCD software, see references.

Usage

```
analysis_gradient_flow(path, outputbasename, basename = "gradflow",
  read.data = TRUE, pl = FALSE, plotsize = 4, skip = 0, start = 0,
  scale = 1, dbg = FALSE)
```

Arguments

`path`	string. path to data files
`outputbasename`	string. basename of output files
`basename`	string. basename of input files, for example "gradflow"
`read.data`	boolean. Indicates whether to read data fresh from data files or to use `basename.raw.gradflow.Rdata` instead
`pl`	boolean. If set to `TRUE` plots will be generated
`plotsize`	numeric. Plot sidelength, this is passed to `tikz.init`.
`skip`	integer. number of measurements to skip
`start`	integer. start value for time
`scale`	numeric. scale factor for the MD time, should be set to the stridelength (in units of trajectories or configurations) which was used to produce the gradient flow files, such that the distance between measurements can be interpreted correctly and the reported autocorrelation times scaled appropriately.
`dbg`	boolean. If set to `TRUE` debugging output will be provided.

Value

Nothing is returned.

References

K. Jansen and C. Urbach, Comput.Phys.Commun. 180 (2009) 2717-2738

`analysis_online` *analysis_online*

Description

`analysis_online` is a function to analyse the online measurements and output files of the tmLQCD software, see references. The function operates on a subdirectory either passed via `rundir` or automatically constructed from the various function arguments. Depending on which parts of the analysis are requested, this subdirectory is expected to contain onlinemeas.%06d files with online correlator measurements, `output.data` containing the plaquette and energy violation, amongst others and monomial-%02d.data with measurements of the extremal eigenvalues of the

Usage

```
analysis_online(L, Time, t1, t2, beta, kappa, mul, cg_col, evals_id, rundir,
  cg.ylim, type = "", csw = 0, musigma = 0, mudelta = 0, muh = 0,
  addon = "", skip = 0, rectangle = TRUE, plaquette = TRUE,
  dH = TRUE, acc = TRUE, trajtime = TRUE, omeas = TRUE, plotsize = 5,
  debug = FALSE, trajlabel = FALSE, title = FALSE, pl = FALSE,
  method = "uwerr", fit.routine = "optim", oldnorm = FALSE, S = 1.5,
  stat_skip = 0, omeas.samples = 1, omeas.stride = 1, omeas.avg = 1,
  omeas.stepsize = 1, evals.stepsize = 1, boot.R = 1500, boot.l = 2,
  outname_suffix = "", verbose = FALSE)
```

Arguments

`L`	integer. spatial lattice extent
`Time`	integer. temporal lattice extent
`t1`	integer. initial time of fit range
`t2`	integer. end time of fit range
`beta`	numeric. inverse squared gauge coupling
`kappa`	numeric. hopping parameter
`mul`	numeric. light sea twisted quark mass
`cg_col`	integer. column of CG iteration counts from `output.data` to use
`evals_id`	Integer. Monomial ID of the monomial for which eigenvalues are measured. Function will attempt to open monomial-%02d.data.
`rundir`	string. run directory. If not specified, run directory will be constructed automatically. See construct_onlinemeas_rundir for details.
`cg.ylim`	numeric. y-limits for CG iteration counts
`type`	string. Type specifier for the gauge action, this might be 'iwa' for Iwasaki, for example.
`csw`	numeric. clover coefficient
`musigma`	numeric. average 1+1 sea twisted quark mass
`mudelta`	numeric. splitting 1+1 sea twisted quark mass
`muh`	numeric. "heavy" twisted mass in the case of a `n_f=2+2` run
`addon`	string. addon to output filenames
`skip`	integer. number of initial measurements to skip in analysis
`rectangle`	boolean. If true, rectangle plaquettes are analysed
`plaquette`	boolean. If true, square plaquettes are analysed
`dH`	boolean. If true, delta H is analysed
`acc`	boolean. If true, the acceptance rate is analysed
`trajtime`	boolean. If true, the time per trajectory is analysed
`omeas`	boolean. If true, online measurements are analysed (onlinemeas.%06d)
`plotsize`	numeric. size of plots being generated
`debug`	boolean. provide debug information
`trajlabel`	boolean or string. If not `FALSE`, use as trajectory labels
`title`	bolean or string. If not `FALSE`, use as main title of plots
`pl`	boolean. If set to `TRUE` plots will be generated
`method`	string. method to compute errors, can be "uwerr", "boot" or "all"
`fit.routine`	string. minimisation routine for chisq, can be "optim"
`oldnorm`	boolean. If `TRUE`, the function assumes that the onlinemeas.%06d are in old tmLQCD normalisation.
`S`	numeric. `S` parameter of uwerr

`stat_skip`	integer. By passing this parameter, the various timeseries plots will include `stat_skip` measurements, but these will be skipped in the corresponding statistical analysis. This maybe useful, for example, to visualise thermalisation.
`omeas.samples`	integer. number of stochastic samples per online measurement
`omeas.stride`	integer. stride length in the reading of online measurements
`omeas.avg`	integer. Block average over this many subsequent measurements.
`omeas.stepsize`	integer. Number of trajectories between online measurements. Autocorrelation times of online measurement data will be scaled by this factor.
`evals.stepsize`	integer. Numer of trajectories between (strange-charm Dirac opertoar) eigenvalue measurements. Autocorrelation times of eigenvalues will be scaled by this factor.
`boot.R`	integer. number of bootstrap samples to use in bootstrap-based parts of analysis.
`boot.l`	integer. bootstrap block size
`outname_suffix`	string. suffix for output files
`verbose`	boolean. If `TRUE`, function produces verbose output. #'

Value

a list is returned with all the accumulated results. Moreover, a PDF file with statistics and analytics is created and the results are written into .Rdata files. On the one hand, the result of the call to the onlinemeas function is written to onlineout.%s.Rdata, where %s is replaced with a label built from meta information based on the arguments above. On the other hand, summary data across many calls of this function is silently accumulated in the file `omeas.summary.Rdata` which contains the named list 'resultsum' with element names based on `rundir`.

References

K. Jansen and C. Urbach, Comput.Phys.Commun. 180 (2009) 2717-2738

`avg.cbt.cf` *average close-by-times in a correlation function*

Description

"close-by-times" averaging replaces the value of the correlation function at t with the "hypercubic" average with the values at the neighbouring time-slices with weights 0.25, 0.5 and 0.25 C(t') = 0.25 C(t-1) + 0.5 C(t) + 0.25 C(t+1) where periodic boundary conditions are assumed in shift.cf

Usage

```
avg.cbt.cf(cf)
```

Arguments

`cf`	object of type cf

Value

Returns an object of class `cf`.

`block.raw_cf` *Block average correlation function data*

Description

Block `block_length` sequential measurements of the correlation function together. This occurs, for example, when multiple stochastic noise vectors are used per measurement or multiple source locations. Alternatively, it can also be used to account for auto-correlations in the data. If the total number of measurements is not divisible by `block_length`, the last measurements are discarded.

Usage

```
block.raw_cf(cf, block_length)
```

Arguments

`cf`	`raw_cf` object
`block_length`	Integer, number of successive measurements to average over.

Value

cf `raw_cf` object with the data member reduced in its first dimension by a factor of `block_length` and restricted (at the end) to the number of measurements divisible by `block_length`.

`bootstrap.analysis` *Performs a Bootstrap with Blocking Analysis of a Timeseries*

Description

Performs a Bootstrap with Blocking Analysis of a Timeseries

Usage

```
bootstrap.analysis(data, skip = 0, boot.R = 100, tsboot.sim = "geom",
  pl = FALSE, boot.l = 2)
```

Arguments

`data`	a numerical vector containing the time series
`skip`	integer value providing the warm up phase length.
`boot.R`	number of bootstrap samples. See also boot, and tsboot.
`tsboot.sim`	the `sim` parameter of tsboot.
`pl`	logical, indicating whether or not to plot the result.
`boot.l`	block length for blocked bootstrap.

Details

the routine will compute the error, the error of the error and the integrated autocorrelation time for different block size using a bootstrap analysis. The blocksize is systematically increased starting from 1 until `(length(data)-skip)/blocksize < 20`. Note that only data is kept in exact multiples of the block length.

Value

returns a data frame containing the mean value, the error approximation, the estimate of the error of the error, the value of tau int and the bias for all block sizes.

Author(s)

Carsten Urbach, `<carsten.urbach@liverpool.ac.uk>`

See Also

for an alternative way to analyse such time series see `uwerr` and `computeacf`

Examples

```
data(plaq.sample)
plaq.boot <- bootstrap.analysis(plaq.sample, pl=TRUE)
```

`bootstrap.cf` *bootstrap a set of correlation functions*

Description

bootstrap a set of correlation functions

Usage

```
bootstrap.cf(cf, boot.R = 400, boot.l = 2, seed = 1234, sim = "geom",
  endcorr = TRUE)
```

Arguments

`cf`	correlation matrix of class `cf` e.g. obtained with a call to `extrac.obs`.
`boot.R`	number of bootstrap samples.
`boot.l`	block size for autocorrelation analysis
`seed`	seed for the random number generation used for boostrapping.

`sim`	The type of simulation required to generate the replicate time series. The possible input values are ‘"fixed"’ (block resampling with fixed block lengths of ‘boot.l’) and ‘"geom"’ (block resampling with block lengths having a geometric distribution with mean ‘boot.l’). Default is ‘"geom"’. See tsboot for details.
`endcorr`	A logical variable indicating whether end corrections are to be applied when ‘sim’ is ‘"fixed"’. When ‘sim’ is ‘"geom"’, ‘endcorr’ is automatically set to ‘TRUE’; ‘endcorr’ is not used when ‘sim’ is ‘"model"’ or ‘"scramble"’. See tsboot for details.

Value

returns an object of class `cf` with bootstrap samples added for th correlation function called `cf.tsboot`. Moreover, the original average of `cf` is returned as `cf0` and the bootstrap errors as `tsboot.se`. We also copy the input parameters over and set `bootstrap.samples` to `TRUE`.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

See Also

`tsboot`, `jackknife.cf`

Examples

```
data(samplecf)
samplecf <- bootstrap.cf(cf=samplecf, boot.R=99, boot.l=2, seed=1442556)
plot(samplecf, log=c("y"))
```

`bootstrap.effectivemass` *Computes effective masses with bootstrapping errors*

Description

Generates bootstrap samples for effective mass values computed from an object of class `cf` (a correlation function)

Usage

```
bootstrap.effectivemass(cf, type = "solve")
```

Arguments

`cf`	a correlation function as an object of type `cf`, preferably after a call to `bootstrap.cf`. If the latter has not been called yet, it will be called in this function.
`type`	The function to be used to compute the effective mass values. Possibilities are "acosh", "solve", "log", "temporal", "shifted" and "weighted". While the first three assume normal cosh behaviour of the correlation function, "temporal" is desigend to remove an additional constant stemming from temporal states in two particle correlation functions. The same for "shifted" and "weighted", the latter for the case of two particle energies with the two particle having different energies. In the latter case only the leading polution is removed by `removeTemporal.cf` and taken into account here.

Details

A number of types is implemented to compute effective mass values from the correlation function:

"solve": the ratio
$C(t+1)/C(t) = \cosh(-m*(t+1))/\cosh(-m*t)$
is numerically solved for m.

"acosh": the effective mass is computed from
$m = acosh((C(t-1)+C(t+1))/(2C(t)))$
Note that this definition is less tolerant against noise.

"log": the effective mass is defined via
$m = \log(C(t)/C(t+1))$
which has artifacts of the periodicity at large t-values.

"temporal": the ratio
$[C(t)-C(t+1)]/[C(t-1)-C(t)] = [\cosh(-m*(t)) - \cosh(-m*(t+1))]/[\cosh(-m*(t-1)) - \cosh(-m(t))]$
is numerically solved for $m(t)$.

"shifted": like "temporal", but the differences $C(t)-C(t+1)$ are assumed to be taken already at the correlator matrix level using `removeTemporal.cf` and hence the ratio
$[C(t+1)]/[C(t)] = [\cosh(-m*(t)) - \cosh(-m*(t+1))]/[\cosh(-m*(t-1)) - \cosh(-m(t))]$
is numerically solved for $m(t)$.

"weighted": like "shifted", but now there is an additional weight factor w from `removeTemporal.cf` to be taken into account, such that the ratio
$[C(t+1)]/[C(t)] = [\cosh(-m*(t)) - w*\cosh(-m*(t+1))]/[\cosh(-m*(t-1)) - w*\cosh(-m(t))]$
is numerically solved for $m(t)$ with w as input.

Value

An object of class `effectivemass` is invisibly returned. It has objects: `effMass`:
The computed effective mass values as a vector of length `Time/2`. For `type="acosh"` also the first value is `NA`, because this definition requires three time slices.

`deffMass`:
The computed bootstrap errors for the effective masses of the same length as `effMass`.

`effMass.tsboot`:
The boostrap samples of the effective masses as an array of dimension RxN, where `R=boot.R` is the number of bootstrap samples and `N=(Time/2+1)`.

and `boot.R`, `boot.l`, `Time`

Author(s)

Carsten Urbach, <curbach@gmx.de>

References

arXiv:1203.6041

See Also

`fit.effectivemass`, `bootstrap.cf`, `removeTemporal.cf`

Examples

```
data(samplecf)
samplecf <- bootstrap.cf(cf=samplecf, boot.R=99, boot.l=2, seed=1442556)
effmass <- bootstrap.effectivemass(cf=samplecf)
summary(effmass)
plot(effmass, ylim=c(0.14,0.15))
```

`bootstrap.gevp` *perform a bootstrap analysis of a GEVP*

Description

perform a bootstrap analysis of a GEVP for a real, symmetric correlator matrix

Usage

```
bootstrap.gevp(cf, t0 = 1, element.order = 1:cf$nrObs,
  sort.type = "vectors", sort.t0 = TRUE)
```

Arguments

`cf`	correlation matrix obtained with a call to `extrac.obs`.
`t0`	initial time value of the GEVP, must be in between 0 and `Time/2-2`. Default is 1.
`element.order`	specifies how to fit the `n` linearly ordered single correlators into the correlator matrix. `element.order=c(1,2,3,4)` leads to a matrix `matrix(cf[element.order],nrow=2)`. Double indexing is allowed.

`sort.type`	Sort the eigenvalues either in descending order, or by using the scalar product of the eigenvectors with the eigenvectors at $t = t_0 + 1$. Possible values are "values", "vectors" and "det". The last one represents a time consuming, but in principle better version of sorting by vectors.
`sort.t0`	for `sort.type` "vectors" use t_0 as reference or $t - 1$.

Details

Say something on "det" sorting method.

Value

Returns an object of class `gevp` with member objects:

`cf`:
The input data, if needed bootstrapped with `bootstrap.cf`.

`res.gevp`:
The object returned from the call to `gevp`. For the format see `gevp`.

`gevp.tsboot`:
The bootstrap samples of the GEVP. For the format see `gevp`.

Author(s)

Carsten Urbach, <`curbach@gmx.de`>

References

Michael, Christopher and Teasdale, I., Nucl.Phys.B215 (1983) 433, DOI: 10.1016/0550-3213(83)90674-0
Blossier, B. et al., JHEP 0904 (2009) 094, DOI: 10.1088/1126-6708/2009/04/094, arXiv:0902.1265

See Also

`gevp`, `extract.obs`, `bootstrap.cf`

Examples

```
data(correlatormatrix)
## bootstrap the correlator matrix
correlatormatrix <- bootstrap.cf(correlatormatrix, boot.R=99, boot.l=1, seed=132435)
## solve the GEVP
t0 <- 4
correlatormatrix.gevp <- bootstrap.gevp(cf=correlatormatrix, t0=t0, element.order=c(1,2,3,4))
## extract the ground state and plot
pc1 <- gevp2cf(gevp=correlatormatrix.gevp, id=1)
plot(pc1, log="y")
## determine the corresponding effective masses
pc1.effectivemass <- bootstrap.effectivemass(cf=pc1)
pc1.effectivemass <- fit.effectivemass(cf=pc1.effectivemass, t1=5, t2=20)
## summary and plot
```

```
summary(pc1.effectivemass)
plot(pc1.effectivemass)

## we can also use matrixfit with a special model for a principal
## correlators
pc1.matrixfit <- matrixfit(pc1, t1=2, t2=24, fit.method="lm", model="pc", useCov=FALSE,
                     parlist=array(c(1,1), dim=c(2,1)), sym.vec=c("cosh"), neg.vec=c(1))
summary(pc1.matrixfit)
plot(pc1.matrixfit)

## the same can be achieved using bootstrap.nlsfit
model <- function(par, x, t0, ...) {
  return(exp(-par[1]*(x-t0))*(par[3]+(1-par[3])*exp(-par[2]*(x-t0))))
}
ii <- c(2:4, 6:25)
fitres <- parametric.nlsfit(fn=model, par.guess=c(0.5, 1, .9),
                            y=pc1$cf0[ii], dy=pc1$tsboot.se[ii],
                            x=ii-1, boot.R=pc1$boot.R, t0=t0)
summary(fitres)
plot(fitres, log="y")
```

`bootstrap.hankel` *GEVP method based on Hankel matrices.*

Description

Alternative method to determine energy levels from correlation matrices. A so-called Hankel matrix is generated from an input cf object and a generalised eigenvalue problem is solved then. This is the function to call. It will perform a bootstrap analysis.

Usage

```
bootstrap.hankel(cf, t0 = 1, n = 2, N = (cf$Time/2 + 1),
  t0fixed = TRUE, deltat = 1, Delta = 1, submatrix.size = 1,
  element.order = 1)
```

Arguments

`cf`	object of type cf
`t0`	Integer. Initial time value of the GEVP, must be in between 0 and `Time/2-n`. Default is 1. Used when `t0fixed=TRUE`.
`n`	Integer. Size of the Hankel matrices to generate
`N`	Integer. Maximal time index in correlation function to be used in Hankel matrix
`t0fixed`	Integer. If set to `TRUE`, keep t0 fixed and vary deltat, otherwise keep deltat fixed and vary t0.
`deltat`	Integer. value of deltat used in the hankel GEVP. Default is 1. Used `t0fixed=FALSE`

`Delta`	integer. Delta is the time shift used in the Hankel matrix.
`submatrix.size`	Integer. Submatrix size to be used in build of Hankel matrices. Submatrix.size > 1 is experimental.
`element.order`	Integer vector. specifies how to fit the `n` linearly ordered single correlators into the correlator matrix for submatrix.size > 1. `element.order=c(1,2,3,4)` leads to a matrix `matrix(cf[element.order],nrow=2)`. Matrix elements can occur multiple times, such as `c(1,2,2,3)` for the symmetric case, for example.

Details

See `vignette(name="hankel",package="hadron")`

Value

List object of class "hankel". The eigenvalues are stored in a numeric vector `t0`, the corresonding samples in `t`. The reference input time `t0` is stored as `reference_time` in the returned list.

See Also

Other hankel: `bootstrap.hankel_summed()`, `gevp.hankel_summed()`, `gevp.hankel()`, `hankel2cf()`, `hankel2effectivemass()`, `plot_hankel_spectrum()`

Examples

```
data(correlatormatrix)
correlatormatrix <- bootstrap.cf(correlatormatrix, boot.R=99, boot.l=1, seed=132435)
t0 <- 4
correlatormatrix.gevp <- bootstrap.gevp(cf=correlatormatrix, t0=t0, element.order=c(1,2,3,4))
pc1 <- gevp2cf(gevp=correlatormatrix.gevp, id=1)
pc1.hankel <- bootstrap.hankel(cf=pc1, t0=1, n=2)
hpc1 <- hankel2cf(hankel=pc1.hankel, id=1)
plot(hpc1, log="y")
heffectivemass1 <- hankel2effectivemass(hankel=pc1.hankel, id=1)
```

`bootstrap.hankel_summed`
GEVP method based on Hankel matrices.

Description

Alternative method to determine energy levels from correlation matrices. A so-called Hankel matrix is generated from an input cf object and a generalised eigenvalue problem is solved then. This is the function to call. It will perform a bootstrap analysis.

Usage

```
bootstrap.hankel_summed(cf, t0values = c(1:(N - 2 * n - deltat)),
  deltat = 1, n = 2, N = cf$Time/2 + 1)
```

Arguments

`cf`	object of type cf
`t0values`	Integer vector. The t0 values to sum over. Default is `c(1:max)`. All elements must be larger or equal to zero and smaller or equal than `max=N-2*n-deltat`
`deltat`	Integer. value of deltat used in the hankel GEVP. Default is 1.
`n`	Integer. Size of the Hankel matrices to generate, default is 2.
`N`	Integer. Maximal time index in correlation function to be used in Hankel matrix

Details

See `vignette(name="hankel",package="hadron")`

Value

List object of class "hankel.summed". The eigenvalues are stored in a numeric vector `t0`, the corresonding samples in `t`. The reference input times `t0values` is stored as `t0values` in the returned list. In addition, `deltat` is stored in the returned list.

See Also

Other hankel: `bootstrap.hankel()`, `gevp.hankel_summed()`, `gevp.hankel()`, `hankel2cf()`, `hankel2effectivemass()`, `plot_hankel_spectrum()`

Examples

```
data(correlatormatrix)
correlatormatrix <- bootstrap.cf(correlatormatrix, boot.R=99, boot.l=1, seed=132435)
t0 <- 4
correlatormatrix.gevp <- bootstrap.gevp(cf=correlatormatrix, t0=t0, element.order=c(1,2,3,4))
pc1 <- gevp2cf(gevp=correlatormatrix.gevp, id=1)
pc1.hankel <- bootstrap.hankel_summed(cf=pc1, t0=c(1:15), n=2)
```

`bootstrap.meanerror` *Compute the bootstrap error of the mean*

Description

Compute the bootstrap error of the mean

Usage

```
bootstrap.meanerror(data, R = 400, l = 20)
```

Arguments

`data`	Original data to bootstrap
`R`	Number of bootstrap replicates.
`l`	Block length.

Value

Returns a numeric vector with the estimated standard error of the mean.

`bootstrap.nlsfit` *Bootstrap a non-linear least-squares fit*

Description

Performs and bootstraps a non-linear least-squares fit to data with y and x errors.

Usage

```
bootstrap.nlsfit(fn, par.guess, y, x, bsamples, priors = list(param = c(), p
  = c(), psamples = c()), ..., lower = rep(x = -Inf, times =
  length(par.guess)), upper = rep(x = +Inf, times = length(par.guess)), dy,
  dx, CovMatrix, gr, dfn, mask, use.minpack.lm = TRUE, parallel = FALSE,
  error = sd, cov_fn = cov, maxiter = 500, success.infos = 1:3,
  relative.weights = FALSE, na.rm = FALSE)
```

Arguments

`fn`	`fn(par,x,...)`. The (non-linear) function to be fitted to the data. Its first argument must be the fit parameters named `par`. The second must be `x`, the explaining variable. Additional parameters might be passed to the function. Currently we pass `boot.r` which is `0` for the original data and the ID (1, ...) of the bootstrap sample otherwise. As more parameters might be added in the future it is recommended that the fit function accepts `...` as the last parameter to be forward compatible.
`par.guess`	initial guess values for the fit parameters.
`y`	the data as a one-dimensional numerical vector to be described by the fit function.
`x`	values of the explaining variable in form of a one-dimensional numerical vector.
`bsamples`	bootstrap samples of `y` (and `x`, if applicable). Must be provided as array of dimensions `c(boot.R,n)` with `n` equals to `length(y)` in case of 'yerrors' and For 'xyerrors' to `length(y) + length(x)`.

`priors`	List possessing the elements `param`, `p` and `psamples`. The vector `param` includes the indices of all fit parameters that are to be constrained and the vector `p` the corresponding paramater values (e.g. known from a previous fit). The list element `psamples` is a matrix of dimensions `(boot.R,length(param))` and contains the corresponding bootstrap samples. If this list is not specified priors are omitted within the fit.
`...`	Additional parameters passed to `fn`, `gr` and `dfn`.
`lower`	Numeric vector of length `length(par.guess)` of lower bounds on the fit parameters. If missing, `-Inf` will be set for all.
`upper`	Numeric vector of length `length(par.guess)` of upper bounds on the fit parameters. If missing, `+Inf` will be set for all.
`dy, dx`	Numeric vector. Errors of the dependent and independent variable, respectively. These do not need to be specified as they can be computed from the bootstrap samples. In the case of parametric bootstrap it might would lead to a loss of information if they were computed from the pseudo-bootstrap samples. They must not be specified if a covariance matrix is given.
`CovMatrix`	complete variance-covariance matrix of dimensions `c(length(y),length(y))` or `c(length(y)+length(x),length(y)+length(x))` depending on the error-model. Pass `NULL` if the matrix has to be calculated from the `bsamples`. In that case, if the number of boostrap samples is small compared to the number of variables, singular value decomposition with small eigenvalue replacement will be used (see invertCovMatrix) to attempt a clean inversion. In case a variance-covariance matrix is passed, the inversion will simply be attempted using `solve` on the Cholesky decomposition. Finally, if `CovMatrix` is missing, an uncorrelated fit will be performed.
`gr`	`gr(par,x,...)`. `gr=d(fn) / d(par)` is a function to return the gradient of `fn`. It must return an array with `length(x)` rows and `length(par)` columns.
`dfn`	`dfn(par,x,...)`. `dfn=d(fn) / dx` is the canonical derivative of `fn` by `x` and only relevant if x-errors are provided.
`mask`	logical or integer index vector. The mask is applied to select the observations from the data that are to be used in the fit. It is applied to `x`, `y`, `dx`, `dy`, `bsamples` and `CovMatrix` as applicable.
`use.minpack.lm`	use the `minpack.lm` library if available. This is usually faster than the default `optim` but somtimes also less stable.
`parallel`	parallelise over bootstrap samples. The package `parallel` is required.
`error`	Function that takes a sample vector and returns the error estimate. This is a parameter in order to support different resampling methods like jackknife.
`cov_fn`	function. Function to compute the covariance (matrix). Default is cov.
`maxiter`	integer. Maximum number of iterations that can be used in the optimization process.
`success.infos`	integer vector. When using `minpack.lm` there is the `info` in the return value. Values of 1, 2 or 3 are certain success. A value of 4 could either be a success or a saddle point. If you want to interpret this as a success as well just pass `1:4` instead of the default `1:3`.

`relative.weights`	are the errors on y (and x) to be interpreted as relative weights instead of absolute ones? If TRUE, the covariance martix of the fit parameter results is multiplied by chi^2/dof. This is the default in many fit programs, e.g. gnuplot.
`na.rm`	logical. If set to `true`, NAs in `y` and `dy` will be ignored. If x-errors are taken into account, NAs in `x` and `dx` will be ignored, too.

Value

returns a list of class 'bootstrapfit'. It returns all input parameters and adds in addition the following:

`t0`	the one dimensional numerical vector of length `npar+1`. `npar` is the number of fit parameters. In case of 'yerrors' this equals `length(par.guess)`. For 'xy-errors' this equals `length(par.guess) + length(x)`. `t0` contains the best fit parameters obtained on the original data. The last element in `t0` is the chisquare value.
`t`	an array of dimensions `(npar+1,boot.R)` with `npar` as in `t0`. The rows contain the individual bootstrap observations.
`bsamples`	the bootstrap samples used as an array of dimensions `(length(y),boot.R)` or `(length(y)+length(x),boot.R)` depending on the error model with `npar` as in `t0`.
`Qval`	the p-value of the fit on the original data
`chisqr`	the residual chisqr value.
`dof`	the residual degrees of freedom of the fit.
`nx`	the number of x-values.
`tofn`	the original `...` list of parameters to be passed on to the fit function
`mask`	original `mask` value

See Also

Other NLS fit functions: `parametric.bootstrap.cov()`, `parametric.bootstrap()`, `parametric.nlsfit.cov()`, `parametric.nlsfit()`, `plot.bootstrapfit()`, `predict.bootstrapfit()`, `print.bootstrapfit()`, `simple.nlsfit()`, `summary.bootstrapfit()`

Examples

```
## Declare some data.
value <- c(0.1, 0.2, 0.31)
dvalue <- c(0.01, 0.01, 0.015)
x <- c(1, 2, 3)
dx <- c(0.1, 0.1, 0.1)
boot.R <- 1500

fn <- function (par, x, boot.r, ...) par[1] + par[2] * x

## Before we can use the fit with this data, we need to create bootstrap
## samples. We do not want to use the correlation matrix here. Note that you
## can simply use the parametric.nlsfit function as a convenient wrapper of
## the two steps.
bsamples <- parametric.bootstrap(boot.R, c(value, x), c(dvalue, dx))
head(bsamples)

fit.result <- bootstrap.nlsfit(fn, c(1, 1), value, x, bsamples)
summary(fit.result)
plot(fit.result, main = 'Ribbon on top')
plot(fit.result, ribbon.on.top = FALSE, main = 'Ribbon below')
residual_plot(fit.result, main = 'Residual Plot')
```

`c.cf` *Concatenate correlation function objects*

Description

Concatenate correlation function objects

Usage

```
## S3 method for class 'cf'
c(...)
```

Arguments

`...`	Zero or multiple objects of type `cf`.

Value

Returns an object of class `cf` representing the concatenation of all the input objects of class `cf`.

`c.raw_cf` *Concatenate* `raw_cf` *correlation function objects*

Description

Concatenate `raw_cf` correlation function objects

Usage

```
## S3 method for class 'raw_cf'
c(...)
```

Arguments

`...`	Zero or multiple objects of type `raw_cf`.

Value

Returns an object of S3 class `raw_cf`, the concatenation of the input objects.

`cA2.09.48_3pi_I3_0_A1u_1_pc` *A three pion correlator with significant thermal states.*

Description

A three pion correlator with significant thermal states.

Usage

```
cA2.09.48_3pi_I3_0_A1u_1_pc
```

Format

An object of class `list` (inherits from `cf`, `cf_meta`, `cf_boot`, `cf_principal_correlator`) of length 19.

`cdh` *finite size corrections a la Colangelo, Duerr, Haefeli*

Description

finite size corrections a la Colangelo, Duerr, Haefeli

Usage

```
cdh(parm = rep(0, times = 6), rev = -1, aLamb1 = 0.055, aLamb2 = 0.58,
  aLamb3, aLamb4, ampiV, afpiV, aF0, a_fm, L, printit = FALSE,
  incim6 = FALSE, rtilde = c(-1.5, 3.2, -4.2, -2.5, 3.8, 1),
  use.cimpl = TRUE)
```

Arguments

`parm`	parameters
`rev`	$rev = -1$ corrects from L to $L = \infty$, $rev = +1$ the other way around
`aLamb1`	The four low energy Λ_{1-4}constants in lattice units.
`aLamb2`	see `aLamb1`.
`aLamb3`	see `aLamb1`.
`aLamb4`	see `aLamb1`.
`ampiV`	pseudo scalar mass values to be corrected

`afpiV`	pseudo scalar decay constant values to be corrected
`aF0`	af_0 in lattice units
`a_fm`	the value of the lattice spacing in fermi
`L`	the lattice spatial extent
`printit`	if set to TRUE the corrections are printed
`incim6`	in- or exclude the NNNLO correction for the mass
`rtilde`	the low energy constants $\tilde{r}$, needed only if `incim6=TRUE`
`use.cimpl`	use the four times faster direct c Implementation of the correction routine

Details

see reference for details. We use the simplyfied formulae for the S quantities, see eq. (59) in the reference.

Value

a list with the corrected values for mpi and fpi

Author(s)

Carsten Urbach curbach@gmx.de

References

Gilberto Colangelo, Stephan Durr, Christoph Haefeli, Nucl.Phys.B721:136-174,2005. hep-lat/0503014

Examples

```
L <- c(24, 24, 24, 24, 32)
mps <- c(0.14448, 0.17261, 0.19858, 0.22276, 0.14320)
fps <- c(0.06577, 0.07169, 0.07623, 0.07924, 0.06730)
aLamb1 <- 0.05
aLamb2 <- 0.5
aLamb3 <- 0.38
aLamb4 <- 0.66
cdhres <- cdh(rev=+1, aLamb1=aLamb1, aLamb2=aLamb2, aLamb3=aLamb3, aLamb4=aLamb4,
              ampiV=mps, afpiV=fps, aF0=fps, a_fm=0.08, L=L, printit=TRUE,
              incim6=FALSE)
cdhres$mpiFV
cdhres$fpiFV
```

`cdhnew` *finite size corrections a la Colangelo, Duerr, Haefeli, but re-expanded as series in the quark mass*

Description

finite size corrections a la Colangelo, Duerr, Haefeli, but re-expanded as series in the quark mass

Usage

```
cdhnew(parm = rep(0, times = 6), rev = -1, aLamb1 = 0.055,
  aLamb2 = 0.58, aLamb3, aLamb4, ampiV, afpiV, aF0, a2B0mu, L,
  printit = FALSE, use.cimpl = TRUE)
```

Arguments

`parm`	m parameters
`rev`	$rev = -1$ corrects from L to $L = \infty$, $rev = +1$ the other way around
`aLamb1`	The four low energy Λ_{1-4}constants in lattice units.
`aLamb2`	see `aLamb1`.
`aLamb3`	see `aLamb1`.
`aLamb4`	see `aLamb1`.
`ampiV`	pseudo scalar mass values to be corrected
`afpiV`	pseudo scalar decay constant values to be corrected
`aF0`	af_0 in lattice units
`a2B0mu`	$2B_0\mu$ in lattice units, where μ is the quark mass and B_0 a low energy constant
`L`	the lattice spatial extent
`printit`	if set to TRUE the corrections are printed
`use.cimpl`	use the four times faster direct c Implementation of the correction routine

Details

see reference for details. We use the simplyfied formulae for the S quantities, see eq. (59) in first reference.

Value

a list with the corrected values for mpi and fpi

Author(s)

Carsten Urbach curbach@gmx.de

References

Gilberto Colangelo, Stephan Durr, Christoph Haefeli, Nucl.Phys.B721:136-174,2005. hep-lat/0503014

and

R. Frezzotti, V. Lubicz, S. Simula, arXiv:0812.4042 hep-lat

Examples

```
mu <- c(0.004, 0.006, 0.008, 0.010, 0.004)
L <- c(24, 24, 24, 24, 32)
mps <- c(0.14448, 0.17261, 0.19858, 0.22276, 0.14320)
fps <- c(0.06577, 0.07169, 0.07623, 0.07924, 0.06730)
aLamb1 <- 0.05
aLamb2 <- 0.5
aLamb3 <- 0.38
aLamb4 <- 0.66
aF0    <- 0.051
a2B    <- 5.64
cdhres <- cdhnew(rev=+1, aLamb1=aLamb1, aLamb2=aLamb2, aLamb3=aLamb3,
                 aLamb4=aLamb4, ampiV=mps, afpiV=fps, aF0=aF0,
                 a2B0mu=a2B*mu, L=L, printit=TRUE)
cdhres$mpiFV
cdhres$fpiFV
```

`CExp` *Cosh Or Sinh Build Out Of Two Exps*

Description

Evaluates

$$f(x) = \frac{1}{2}(\exp(-m(T-x)) \pm \exp(-mx))$$

for given mass m, vector x and time extent T. This form is better usable in χ^2 fitting than cosh or sinh.

Usage

```
CExp(m, Time, x, sign = 1)
```

Arguments

`m`	mass value
`Time`	Time extent
`x`	vector of values on which to evaluate the function
`sign`	with sign=1 cosh is evaluated, with sign=-1 sinh

Value

vector $f(x)$

Author(s)

Carsten Urbach <carsten.urbach@liverpool.ac.uk>

Examples

```
m <- 0.1
Time <- 48
x <- seq(0, 48, 1)
CExp(m=m, Time=Time, x=x)
```

---

cf *Correlation function container*

Description

This function `cf()` creates containers for correlation functions of class `cf`. This class is particularly designed to deal with correlation functions emerging in statistical and quantum field theory simulations. Arithmetic operations are defined for this class in several ways, as well as concatenation and is.cf.

Usage

```
cf()
```

Details

And last but not least, these are the fields that are used somewhere in the library but we have not figured out which mixin these should belong to:

- `conf.index`: TODO
- `N`: Integer, number of measurements.
- `blockind`: TODO
- `jack.boot.se`: TODO

Value

returns an object of S3 class `cf` derived from a `list`

See Also

Other cf constructors: `cf_boot()`, `cf_meta()`, `cf_orig()`, `cf_principal_correlator()`, `cf_shifted()`, `cf_smeared()`, `cf_subtracted()`, `cf_weighted()`

Examples

```
newcf <- cf()
```

`cf_boot` *Bootstrapped CF mixin constructor*

Description

Bootstrapped CF mixin constructor

Usage

```
cf_boot(.cf = cf(), boot.R, boot.l, seed, sim, endcorr, cf.tsboot,
  icf.tsboot = NULL, resampling_method)
```

Arguments

`.cf`	`cf` object to extend.
`boot.R`	Integer, number of bootstrap samples used.
`boot.l`	Integer, block length in the time-series bootstrap process.
`seed`	Integer, random number generator seed used in bootstrap.
`sim`	Character, `sim` argument of tsboot.
`endcorr`	Boolean, `endcorr` argumetn of tsboot.
`cf.tsboot`	List, result from the tsboot function for the real part.
`icf.tsboot`	List, result from the tsboot function for the imaginay part.
`resampling_method`	Character, either 'bootstrap' or 'jackknife'

Details

The following fields will also be made available:

- `cf0`: Numeric vector, mean value of original measurements, convenience copy of `cf.tsboot$t0`.
- `tsboot.se`: Numeric vector, standard deviation over bootstrap samples.
- `boot.samples`: Logical, indicating whether there are bootstrap samples available. This is deprecated and instead the presence of bootstrap samples should be queried with `inherits(cf,'cf_boot')`.
- `error_fn`: Function, takes a vector of samples and computes the error. In the bootstrap case this is just the `sd` function. Use this function instead of a `sd` in order to make the code compatible with jackknife samples.

Value

returns the input object of class `cf` with the bootstrap mixin added

See Also

Other cf constructors: `cf_meta()`, `cf_orig()`, `cf_principal_correlator()`, `cf_shifted()`, `cf_smeared()`, `cf_subtracted()`, `cf_weighted()`, `cf()`

`cf_key_meson_2pt` *Generate key string to identify a meson 2pt function*

Description

Generate key string to identify a meson 2pt function

Usage

```
cf_key_meson_2pt(fwd_flav, bwd_flav, snk_gamma, src_gamma, src_p, snk_p)
```

Arguments

`fwd_flav`	String, "forward" quark flavour identifier.
`bwd_flav`	String, "backward" quark flavour identifier.
`snk_gamma`	Integer, CVC convention gamma matrix identifier at the source.
`src_gamma`	Integer, CVC convention gamma matrix identified at the sink.
`src_p`	Integer vector of length 3. (x,y,z) components of the source momentum vector in lattice units.
`snk_p`	Integer vector of length 3. (x,y,z) components of the sink momentum vector in lattice units.

Value

A character vector with the HDF5 key.

`cf_key_meson_3pt` *Generate HDF5 key for CVC 'correlators' meson 3pt function with a local or derivative insertion The key for a meson three-point function has the form: /sud+-g-u-g/t10/dt12/gf5/pfx0pfy0pfz0/gc0/Ddim0_dir0/Ddim1_dir1/D.../gi5/pix0piy0piz0 where, from left to right:*

- *'u' is the flavour of the "backward" propagator*
- *'d' is the flavour of the "sequential" propagator*
- *'+' indicates that 'sud' is daggered*
- *'g' indicates a gamma insertion*
- *'u' is the flavour of the foward propagator*
- *'g' indicates a Dirac structure at the source*
- *'tXX' is the source time slice*
- *'dtYY' is the source-sink separation*
- *'gfN' gamma structure at the sink in CVC indexing*
- *'pfxXpfyYpfzZ' is the sink momentum in CVC convention (sink and source phases are both e^ipx)*
- *'gcN' gamma structure at the current insertion point in CVC indexing*
- *'DdimJ_dirK' covariant displacement applied in dimension 'J', direction 'K' where it should be noted that this is. in operator notation, i.e., the right-most displacement is the one applied first.*
- *...*
- *'giN' gamma structure at the souce in CVC indexing*
- *'pixXpiyYpizZ' at the source in CVC convention*

Description

Generate HDF5 key for CVC 'correlators' meson 3pt function with a local or derivative insertion

The key for a meson three-point function has the form:

/sud+-g-u-g/t10/dt12/gf5/pfx0pfy0pfz0/gc0/Ddim0_dir0/Ddim1_dir1/D.../gi5/pix0piy0piz0

where, from left to right:

- 'u' is the flavour of the "backward" propagator
- 'd' is the flavour of the "sequential" propagator
- '+' indicates that 'sud' is daggered
- 'g' indicates a gamma insertion
- 'u' is the flavour of the foward propagator
- 'g' indicates a Dirac structure at the source

- 'tXX' is the source time slice
- 'dtYY' is the source-sink separation
- 'gfN' gamma structure at the sink in CVC indexing
- 'pfxXpfyYpfzZ' is the sink momentum in CVC convention (sink and source phases are both e^ipx)
- 'gcN' gamma structure at the current insertion point in CVC indexing
- 'DdimJ_dirK' covariant displacement applied in dimension 'J', direction 'K' where it should be noted that this is. in operator notation, i.e., the right-most displacement is the one applied first.
- ...
- 'giN' gamma structure at the souce in CVC indexing
- 'pixXpiyYpizZ' at the source in CVC convention

Usage

```
cf_key_meson_3pt(fwd_flav, bwd_flav, seq_flav, dt, snk_gamma, cur_gamma,
  cur_displ_dim = NA, cur_displ_dir = NA, src_gamma, src_p, snk_p)
```

Arguments

`fwd_flav`	String, "forward" quark flavour identifier.
`bwd_flav`	String, "backward" quark flavour identifier.
`seq_flav`	String, "sequential" quark flavour identifier.
`dt`	Integer, source-sink separation.
`snk_gamma`	Integer, CVC convention gamma matrix identifier at the source.
`cur_gamma`	Integer, CVC convention gamma matrix identified at the insertion.
`cur_displ_dim`	Integer vector of dimensions (0,1,2,3 <-> t,x,y,z) in which covariant displacements have been applied. This vector will be parsed in reverse order, such that the first element here is the first displacement applied to the spinor in the calculation and the right-most element in the key. Length must be matched to 'cur_displ_dir'. Defaults to 'NA' for no displacements.
`cur_displ_dir`	Integer vector of directions (forward, backward) <-> (0,1) in which the covariant displacements have been applied. Parsing as for 'cur_displ_dim'. Length must be matched to 'cur_displ_dim'. Defaults to 'NA' for no displacements.
`src_gamma`	Integer, CVC convention gamma matrix identified at the sink.
`src_p`	Integer vector of length 3. (x,y,z) components of the source momentum vector in lattice units.
`snk_p`	Integer vector of length 3. (x,y,z) components of the sink momentum vector in lattice units.

Value

A character vector with the HDF5 key.

cf_meta *CF metadata mixin constructor*

Description

CF metadata mixin constructor

Usage

```
cf_meta(.cf = cf(), nrObs = 1, Time = NA, nrStypes = 1, symmetrised = FALSE)
```

Arguments

`.cf`	`cf` object to extend.
`nrObs`	Integer, number of different measurements contained in this correlation function. One can use c.cf to add multiple observables into one container. This is for instance needed when passing to the gevp function.
`Time`	Integer, full time extent.
`nrStypes`	Integer, number of smearing types.
`symmetrised`	Logical, indicating whether the correlation function has been symmetrized.

Value

returns the input object of class `cf` with the metadata mixin added

See Also

Other cf constructors: `cf_boot()`, `cf_orig()`, `cf_principal_correlator()`, `cf_shifted()`, `cf_smeared()`, `cf_subtracted()`, `cf_weighted()`, `cf()`

Examples

```
newcf <- cf_orig(cf=array(rnorm(25*100), dim=c(100, 25)))
newcf <- cf_meta(newcf, nrObs=1, Time=48, symmetrised=TRUE)
```

`cf_orig` *Original data CF mixin constructor*

Description

Original data CF mixin constructor

Usage

```
cf_orig(.cf = cf(), cf, icf = NULL)
```

Arguments

`.cf`	`cf` object to extend. Named with a leading period just to distinguish it from the member also named `cf`.
`cf`	Numeric matrix, original data for all observables and measurements.
`icf`	Numeric matrix, imaginary part of original data. Be very careful with this as quite a few functions just ignore the imaginary part and drop it in operations.

Value

returns the input object of class `cf` with the original data mixin added

See Also

Other cf constructors: `cf_boot()`, `cf_meta()`, `cf_principal_correlator()`, `cf_shifted()`, `cf_smeared()`, `cf_subtracted()`, `cf_weighted()`, `cf()`

Examples

```
newcf <- cf_orig(cf=array(rnorm(25*100), dim=c(100, 25)))
newcf <- cf_meta(newcf, nrObs=1, Time=48, symmetrised=TRUE)
newcf <- bootstrap.cf(newcf)
plot(newcf)
```

cf_principal_correlator *Principal correlator CF mixin constructor*

Description

Principal correlator CF mixin constructor

Usage

```
cf_principal_correlator(.cf = cf(), id, gevp_reference_time)
```

Arguments

`.cf`	`cf` object to extend.
`id`	Integer, number of the principal correlator from the GEVP. Ascending with eigenvalue, so `id` = 1 is the lowest state.
`gevp_reference_time`	Integer, reference time t_0 that has been used in the GEVP.

Value

returns the input object of class `cf` with the principal correlator mixin added

See Also

Other cf constructors: `cf_boot()`, `cf_meta()`, `cf_orig()`, `cf_shifted()`, `cf_smeared()`, `cf_subtracted()`, `cf_weighted()`, `cf()`

cf_shifted *Shifted CF mixin constructor*

Description

Shifted CF mixin constructor

Usage

```
cf_shifted(.cf = cf(), deltat, forwardshift)
```

Arguments

`.cf`	`cf` object to extend.
`deltat`	TODO
`forwardshift`	Logical, TODO

Details

The following fields will also be made available:

- `shifted`: Logical, whether the correlation function has been shifted This is deprecated and instead the presence of a shift should be queried with `inherits(cf,'cf_shifted')`.

Value

returns the input object of class `cf` with the shifted mixin added

See Also

Other cf constructors: `cf_boot()`, `cf_meta()`, `cf_orig()`, `cf_principal_correlator()`, `cf_smeared()`, `cf_subtracted()`, `cf_weighted()`, `cf()`

`cf_smeared` *Smeared CF mixin constructor*

Description

Smeared CF mixin constructor

Usage

```
cf_smeared(.cf = cf(), scf, iscf = NULL, nrSamples, obs)
```

Arguments

`.cf`	`cf` object to extend.
`scf`	Like `cf`, but with the smeared data.
`iscf`	Like `icf`, but with the smeared data.
`nrSamples`	TODO
`obs`	TODO

Details

The following fields will also be made available:

- `smeared`: Logical, whether the correlation function has smeared data. This is deprecated and instead the presence of bootstrap samples should be queried with `inherits(cf,'cf_smeared')`.

Value

returns the input object of class `cf` with the smeared mixin added

See Also

Other cf constructors: `cf_boot()`, `cf_meta()`, `cf_orig()`, `cf_principal_correlator()`, `cf_shifted()`, `cf_subtracted()`, `cf_weighted()`, `cf()`

`cf_subtracted` *Subtracted CF mixin constructor*

Description

Subtracted CF mixin constructor

Usage

```
cf_subtracted(.cf = cf(), subtracted.values, subtracted.ii)
```

Arguments

`.cf`	`cf` object to extend.
`subtracted.values`	Numeric matrix, TODO
`subtracted.ii`	Integer vector, TODO

Value

returns the input object of class `cf` with the subtracted mixin added

See Also

Other cf constructors: `cf_boot()`, `cf_meta()`, `cf_orig()`, `cf_principal_correlator()`, `cf_shifted()`, `cf_smeared()`, `cf_weighted()`, `cf()`

`cf_weighted` *Weighted CF mixin constructor*

Description

Weighted CF mixin constructor

Usage

```
cf_weighted(.cf = cf(), weight.factor, weight.cosh)
```

Arguments

`.cf`	`cf` object to extend.
`weight.factor`	TODO
`weight.cosh`	TODO

Details

The following fields will also be made available:

- `weighted`: Logical, indicating whether the correlation function has been weighted. This is deprecated and instead the presence of this should be queried with `inherits(cf,'cf_weighted')`.

Value

returns the input object of class `cf` with the weighted mixin added

See Also

Other cf constructors: `cf_boot()`, `cf_meta()`, `cf_orig()`, `cf_principal_correlator()`, `cf_shifted()`, `cf_smeared()`, `cf_subtracted()`, `cf()`

`compute.plotlims` *compute.plotlims*

Description

Computes limits for plots

Usage

```
compute.plotlims(val, logscale, cumul.dval, cumul.mdval)
```

Arguments

`val`	Numeric. Value.
`logscale`	Boolean.
`cumul.dval`	Numeric. Cumulative error.
`cumul.mdval`	Numeric. Cumulative error.

Value

The computed plot limits are returned as a two component numeric vector.

`computeacf` *Computes The ACF and Integrated AC Time*

Description

Computes the ACF and integrated autocorrelation time of a time series. It also estimates the corresponding standard errors.

Usage

```
computeacf(tseries, W.max, Lambda = 100)
```

Arguments

`tseries`	the time series.
`W.max`	maximal time lag to be used.
`Lambda`	cut-off needed to estimate the standard error of the ACF.

Details

The standard error of the ACF is computed using equation (E.11) of M. Luescher, hep-lat/0409106. The error of the integrated autocorrelation time using the Madras Sokal formula, see also hep-lat/0409106.

Value

It returns a list of class `hadronacf` with members

`lags`	time lags of the integrated autocorrelation function
`Gamma`	normalised autocorrelation function
`dGamma`	error of normalised autocorrelation function
`W.max`	max time lag used for the call of `acf`
`W`	the cut-off up to which the ACF is integrated for the integrated autocorrelation time
`tdata`	the original time series
`tau`	the estimated integrated autocorrelation time
`dtau`	the estimated error of the integrated autocorrelation time

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

References

'Monte Carlo errors with less errors', Ulli Wolff, http://arxiv.org/abs/hep-lat/0306017hep-lat/0306017

'Schwarz-preconditioned HMC algorithm for two-flavour lattice QCD', Martin Luescher, http://arxiv.org/abs/hep-lat/0409106hep-lat/0409106

N. Madras, A. D. Sokal, J. Stat. Phys. 50 (1988) 109

See Also

`uwerr`, `acf` `bootstrap.analysis`

Examples

```
data(plaq.sample)
myacf <- computeacf(plaq.sample, 300)
plot(myacf)
summary(myacf)
```

`computeDisc` *computes a disconnected correlation function from loops*

Description

The dimension of `cf$cf` and `cf$icf` must be `dim(Time,S,N)`, where `Time` is the time extent, `S` is the number of samples and `N` the number of measurements (gauges). `cf2` is the same, but needed only for cross-correlators.

Usage

```
computeDisc(cf, cf2, real = TRUE, real2 = TRUE, smeared = FALSE,
  smeared2 = FALSE, subtract.vev = TRUE, subtract.vev2 = TRUE,
  subtract.equal = TRUE, use.samples, use.samples2, type = "cosh",
  verbose = FALSE)
```

Arguments

`cf`	loop data as produced by `readcmidisc` or `readbinarydisc`.
`cf2`	second set of loop data as produced by `readcmidisc` or `readbinarydisc`. This is needed for cross-correlators
`real`	use the real part `cf$cf`, if set to `TRUE`, otherwise the imaginary part `cf$icf`.
`real2`	use the real part `cf2$cf`, if set to `TRUE`, otherwise the imaginary part `cf2$icf`.
`smeared`	use the loops instead of the local ones for `cf`.
`smeared2`	use the loops instead of the local ones for `cf2`.

`subtract.vev`	subtract a vacuum expectation value. It will be estimated as mean over all samples, gauges and times available.
`subtract.vev2`	subtract a vacuum expectation value for the second set of loops. It will be estimated as mean over all samples, gauges and times available.
`subtract.equal`	subtract contributions of products computed on identical samples. This will introduce a bias, if set to FALSE for missing cf2 or if cf and cf2 are computed on the same set of random sources.
`use.samples`	If set to an integer, only the specified number of samples will be used for `cf`, instead of all samples.
`use.samples2`	Same like `use.samples`, but for `cf2`.
`type`	The correlation function can either be symmetric or anti-symmetric in time. Anti-symmetric is of course only possible for cross-correlators. In this case with `type="cosh"` it is assumed to be symmetric, anti-symmetric otherwise.
`verbose`	Print some debug output, like the VEVs of the loops.

Details

If `subtract.vev=TRUE` the vev is estimated as the mean over all gauges, samples and times available and subtracted from the original loop data. (Same for `subtrac.vev2`.

The correlation is computed such as to avoid correlation between equal samples, unless `nrSamples` is equal to 1.

`cf` and `cf2` must agree in `Time`, number of gauges and number of samples. Matching of gauges is assumed. If this is not the case results are wrong.

Value

Returns an object of type `cf` derived from a `list` with elements `cf`, an array of dimension `dim(N,Time)`, where `N` is the number of samples and `Time` the time extent, integers `Time` for the time extent, `nrStypes` and `nrObs` for the available smearing types and operators, and finally `nrSamples`, the number of samples used to generate the correlation function `cf`.

Author(s)

Carsten Urbach, <curbach@gmx.de>

See Also

`readcmidisc`, `readbinarydisc`, `bootstrap.cf`, `add.cf`, `c.cf`

Examples

```
data(loopdata)
Cpi0v4 <- computeDisc(cf=loopdata, real=TRUE, subtract.vev=TRUE)
Cpi0v4 <- bootstrap.cf(Cpi0v4, boot.R=99, boot.l=1, seed=14556)
```

computefps	*Computes the pseudoscalar decay constant for the twisted mass case from the pseudoscalar amplitude and mass*

Description

From a mass and amplitude determination (using `matrixfit` or `fit.effectivemass`, `bootstrap.gevp` and `gevp2amplitude` the pseudoscalar decay constant is determined for the case of Wilson twisted mass fermions from the pseudoscalar amplitude and mass

Usage

```
computefps(mfit, PP, mass, mu1, mu2, Kappa, normalisation = "cmi",
  disprel = "continuum", boot.fit = TRUE)
```

Arguments

`mfit`	An object of type `matrixfit` or `gevp.amplitude` generated with `matrixfit` or `gevp2amplitude`, respectively.
`PP`	If `mfit` is missing this must contain the value for the pseudoscalar amplitude.
`mass`	If `mfit` is missing this must contain the value for the pseudoscalar mass.
`mu1, mu2`	The values for the twisted quark masses involved in the pseudoscalar meson. If `mu2` is missing it will be assumed to be equal to `mu1`.
`Kappa`	The κ-value of the run, needed only if `normalisation="cmi"`.
`normalisation`	normalisation of the correlators. If set to "cmi" the κ value must be specified.
`disprel`	One of "continuum" or "lattice". Indicates whether the formula for the decay constant should take into account the lattice dispersion relation for the meson. Theoretically this can reduce lattice artefacts for heavy mesons.
`boot.fit`	If set to `FALSE`, the computation is not bootstrapped, even if the `matrixfit` or `gevp.amplitude` contain bootstrap samples. This is a useful time-saver if error information is not strictly necessary. Of course, this affects the return values related to the bootstrap, which are set to `NA`.

Details

The pseudoscalar decay constant is computed from

$$f_{\mathrm{PS}} = 2\kappa(\mu_1+\mu_2)\frac{PP}{\sqrt{2}\sqrt{m_{\mathrm{PS}}}^3}$$

for `normalisation="cmi"` or

$$f_{\mathrm{PS}} = (\mu_1+\mu_2)\frac{PP}{\sqrt{2}\sqrt{m_{\mathrm{PS}}}^3}$$

expecting physical normalisation of the amplitudes.
When `disprel="lattice"`,

$$\sqrt{m_{\mathrm{PS}}^3}$$

is replaced with

$$\sqrt{m_{\mathrm{PS}}}\sinh m_{\mathrm{PS}}$$

which can reduce lattice artefacts for heavy meson masses.

Value

If `mfit` ist missing the value of fps will printed to stdout and returned as a simple numerical value.

If `mfit` is available, this object will be returned but with additional objects added: `fps`, `fps.tsboot`, `mu1,mu2`, `normalistaion` and `Kappa` if applicable.

Author(s)

Carsten Urbach, <curbach@gmx.de>

See Also

`matrixfit`, `gevp2amplitude`,

Examples

```
cfnew <- extractSingleCor.cf(correlatormatrix, id=1)
cfnew <- bootstrap.cf(cfnew, boot.R=99, boot.l=1)
cfnew.fit <- matrixfit(cf=cfnew, t1=12, t2=20, parlist=array(c(1,1),
                       dim=c(2,1)), sym.vec=c("cosh"), neg.vec=c(1))
cfnew.fps <- computefps(mfit=cfnew.fit, mu1=0.004, normalisation="new")
summary(cfnew.fps)
```

`computefpsOS` *Computes the pseudoscalar decay constant for the Osterwalder Seiler case from the pseudoscalar amplitude and mass*

Description

From a mass and amplitude determination (using `matrixfit`) the pseudoscalar decay constant is determined for the case of Osterwalder Seiler (OS) fermions from the AS and SS amplitude (in the twisted basis), ZA and the OS pion mass.

Usage

```
computefpsOS(mfit, Kappa = sqrt(0.5), normalisation = "cmi",
  boot.fit = TRUE, ZA = 1, ZAboot, dZA)
```

Arguments

`mfit`	An object of type `matrixfit` generated with `matrixfit`. The correlation matrix (SS, SA, AS, AA) must have been analysed, where the correlators are in the twisted basis.
`Kappa`	The κ-value of the run, needed only if `normalisation="cmi"`.
`normalisation`	normalisation of the correlators. If set to "cmi" the κ value must be specified.
`boot.fit`	If set to `FALSE`, the computation is not bootstrapped, even if the `matrixfit` or `gevp.amplitude` contain bootstrap samples. This is a useful time-saver if error information is not strictly necessary. Of course, this affects the return values related to the bootstrap, which are set to `NA`.
`ZA`	The value of the renormalisation constant Z_A.
`ZAboot`	Bootstrap samples for Z_A. If they are provided, they are used for computing fps, if not, bootstrap samples are generated from `dZA`. If both are missing, the error of Z_A is not taken into account.
`dZA`	The value of the (normally distributed) error of the renormalisation constant Z_A.

Details

The pseudoscalar decay constant is computed from

$$f_\mathrm{PS}^\mathrm{OS} = Z_A\sqrt{2}\kappa\frac{\langle 0|A|\pi\rangle}{m_\mathrm{PS}}$$

for `normalisation="cmi"` or

$$f_\mathrm{PS}^\mathrm{OS} = Z_A\frac{\langle 0|A|\pi\rangle}{m_\mathrm{PS}}$$

expecting physical normalisation of the amplitudes.

Value

If `mfit` is available, this object will be returned but with additional objects added: `fpsOS`, `fpsOS.tsboot`, `normalistaion`, `ZA`, `ZAboot` and `kappa` if applicable.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

See Also

`matrixfit`

`concat.cf` *Concatenate two correlation function objects*

Description

Concatenate two correlation function objects

Usage

```
concat.cf(left, right)
```

Arguments

`left, right`	`cf` objects to concatenate.

Value

Returns an object of class `cf` representing the concatenation of the two input objects of class `cf`.

`concat.raw_cf` *Concatenate two* `raw_cf` *correlation function objects*

Description

The data of the `left` and `right` objects is concatenated along the second array dimension such that the output contains the tensor slices of `right` after the slices of `left`

Usage

```
concat.raw_cf(left, right)
```

Arguments

`left`	`raw_cf` object to be concatenated with `right`
`right`	`raw_cf` object to be concatenated with `left`

Value

Returns an object of S3 class `raw_cf`, the concatenation of the two input objects.

conj_raw_cf — *Take the complex conjugate of a `raw_cf` object*

Description

Take the complex conjugate of a `raw_cf` object

Usage

```
conj_raw_cf(cf)
```

Arguments

`cf`	`raw_cf` cotnainer with data

Value

`raw_cf`

construct_onlinemeas_rundir — *Construct a run directory string for analysis_online*

Description

Construct a run directory string for analysis_online

Usage

```
construct_onlinemeas_rundir(type, beta, L, Time, kappa = 0, mul = 0,
  csw = 0, musigma = 0, mudelta = 0, muh = 0, addon = "",
  debug = FALSE)
```

Arguments

`type`	String. Short identifier for gauge action type. For example, `iwa` for Iwasaki gauge action.
`beta`	Numeric. Inverse gauge coupling.
`L`	Integer. Spatial lattice extent.
`Time`	Integer. Temporal lattice extent.
`kappa`	Numeric. Sea quark action hopping parameter.
`mul`	Numeric. Sea light quark twisted mass.
`csw`	Numeric. Sea quark action clover parameter.
`musigma`	Numeric. Sea 1+1 "heavy" average twisted quark mass.

`mudelta`	Numeric. Sea 1+1 "heavy" splitting twisted quark mass.
`muh`	Numeric. In case of `n_f=2+2` run, "heavy" twisted quark mass.
`addon`	String. Arbitratry string which will be suffixed to the constructed run directory.
`debug`	Boolean. If `TRUE`, the constructed directory name is printed to screen.

Value

String. Directory name constructed out of the various function parameters. See source code for details.

`correlatormatrix` *Sample correlator matrix*

Description

Sample data for a correlation function for a 24 cube times 48 lattice QCD simulation representing a pion propagation. It is stored in form of an object of class `cf`, which is derived from `list`.

Format

list of 7 elements: "nrObs" "Time" "nrStypes" "symmetrised" "cf" "icf" "cf0"

Examples

```
data("correlatormatrix")
```

`correlators_key_meson_2pt` *Generate HDF5 key for CVC 'correlators' meson 2pt function*

Description

Generate HDF5 key for CVC 'correlators' meson 2pt function

Usage

```
correlators_key_meson_2pt(fwd_flav, bwd_flav, src_ts, snk_gamma, src_gamma,
  src_p, snk_p)
```

Arguments

`fwd_flav`	String, "forward" quark flavour identifier.
`bwd_flav`	String, "backward" quark flavour identifier.
`src_ts`	Integer, source time slice.
`snk_gamma`	Integer, CVC convention gamma matrix identifier at the source.
`src_gamma`	Integer, CVC convention gamma matrix identified at the sink.
`src_p`	Integer vector of length 3. (x,y,z) components of the source momentum vector in lattice units.
`snk_p`	Integer vector of length 3. (x,y,z) components of the sink momentum vector in lattice units.

Value

A character vector with the HDF5 key.

`correlators_key_meson_3pt`

Generate HDF5 key for CVC 'correlators' meson 3pt function with a local or derivative insertion The key for a meson three-point function has the form: /sud+-g-u-g/t10/dt12/gf5/pfx0pfy0pfz0/gc0/Ddim0_dir0/Ddim1_dir1/D.../gi5/pix0piy0piz0 where, from left to right:

- *'u' is the flavour of the "backward" propagator*
- *'d' is the flavour of the "sequential" propagator*
- *'+' indicates that 'sud' is daggered*
- *'g' indicates a gamma insertion*
- *'u' is the flavour of the foward propagator*
- *'g' indicates a Dirac structure at the source*
- *'tXX' is the source time slice*
- *'dtYY' is the source-sink separation*
- *'gfN' gamma structure at the sink in CVC indexing*
- *'pfxXpfyYpfzZ' is the sink momentum in CVC convention (sink and source phases are both e^ipx)*
- *'gcN' gamma structure at the current insertion point in CVC indexing*
- *'DdimJ_dirK' covariant displacement applied in dimension 'J', direction 'K' where it should be noted that this is. in operator notation, i.e., the right-most displacement is the one applied first.*
- ...
- *'giN' gamma structure at the souce in CVC indexing*
- *'pixXpiyYpizZ' at the source in CVC convention*

Description

Generate HDF5 key for CVC 'correlators' meson 3pt function with a local or derivative insertion

The key for a meson three-point function has the form:

/sud+-g-u-g/t10/dt12/gf5/pfx0pfy0pfz0/gc0/Ddim0_dir0/Ddim1_dir1/D.../gi5/pix0piy0piz0

where, from left to right:

- 'u' is the flavour of the "backward" propagator
- 'd' is the flavour of the "sequential" propagator
- '+' indicates that 'sud' is daggered
- 'g' indicates a gamma insertion
- 'u' is the flavour of the foward propagator
- 'g' indicates a Dirac structure at the source
- 'tXX' is the source time slice
- 'dtYY' is the source-sink separation
- 'gfN' gamma structure at the sink in CVC indexing
- 'pfxXpfyYpfzZ' is the sink momentum in CVC convention (sink and source phases are both e^ipx)
- 'gcN' gamma structure at the current insertion point in CVC indexing
- 'DdimJ_dirK' covariant displacement applied in dimension 'J', direction 'K' where it should be noted that this is. in operator notation, i.e., the right-most displacement is the one applied first.
- ...
- 'giN' gamma structure at the souce in CVC indexing
- 'pixXpiyYpizZ' at the source in CVC convention

Usage

```
correlators_key_meson_3pt(fwd_flav, bwd_flav, seq_flav, src_ts, dt, snk_gamma,
  cur_gamma, cur_displ_dim = NA, cur_displ_dir = NA, src_gamma, src_p,
  snk_p)
```

Arguments

`fwd_flav`	String, "forward" quark flavour identifier.
`bwd_flav`	String, "backward" quark flavour identifier.
`seq_flav`	String, "sequential" quark flavour identifier.
`src_ts`	Integer, source time slice.
`dt`	Integer, source-sink separation.
`snk_gamma`	Integer, CVC convention gamma matrix identifier at the source.
`cur_gamma`	Integer, CVC convention gamma matrix identified at the insertion.

`cur_displ_dim`	Integer vector of dimensions (0,1,2,3 <-> t,x,y,z) in which covariant displacements have been applied. This vector will be parsed in reverse order, such that the first element here is the first displacement applied to the spinor in the calculation and the right-most element in the key. Length must be matched to 'cur_displ_dir'. Defaults to 'NA' for no displacements.
`cur_displ_dir`	Integer vector of directions (forward, backward) <-> (0,1) in which the covariant displacements have been applied. Parsing as for 'cur_displ_dim'. Length must be matched to 'cur_displ_dim'. Defaults to 'NA' for no displacements.
`src_gamma`	Integer, CVC convention gamma matrix identified at the sink.
`src_p`	Integer vector of length 3. (x,y,z) components of the source momentum vector in lattice units.
`snk_p`	Integer vector of length 3. (x,y,z) components of the sink momentum vector in lattice units.

Value

A character vector with the HDF5 key.

`create_displ_chains` *create list of chains of displacements Multilpe covariant displacements, when applied in order, form a list of displacments. Each consists of a direction and a dimension.*

Description

create list of chains of displacements Multilpe covariant displacements, when applied in order, form a list of displacments. Each consists of a direction and a dimension.

Usage

```
create_displ_chains(max_depth, dims = c(0:3), dirs = c(0, 1))
```

Arguments

`max_depth`	Positive integer, number of displacement combinations to construct.
`dims`	Integer vector, which lattice dimensions to consider. Default 0:3
`dirs`	Integer vector, which displacement directions to consider. Default forward and backward <-> c(0,1)

Value

List of data frames, each with columns 'dim' and 'dir' of 'max_depth' rows.

`cvc_local_loop_key` *Generate HDF5 key for a momentum and spin-projected CVC loop*

Description

Generate HDF5 key for a momentum and spin-projected CVC loop

Generate key to identify a momentum and spin-projected loop

Usage

```
cvc_local_loop_key(loop_type, istoch, gamma, p)
```

```
cvc_local_loop_key(loop_type, istoch, gamma, p)
```

Arguments

`loop_type`	String, loop type.
`istoch`	Integer, index of the stochastic sample.
`gamma`	Integer, CVC convention gamma matrix identifier.
`p`	Integer vector of length 3, (x,y,z) components of the momentum vector in lattice units.

Value

A character vector with the HDF5 key.

A character vector with the HDF5 key.

`cvc_read_loops` *read HDF5 loop files in the CVC loop format*

Description

The CVC naive_loops code produces HDF5 files which contain a matrix of momenta and the data for the loops (without spin projection) organised by stochastic sample. Currently, the reading code assumes that there is a single configuration stored per file and the "trajectory" parameter in CalcLoops is assumed to take its default value of ’4’.

Usage

```
cvc_read_loops(selections, files, Time, nstoch, verbose = FALSE,
  check_group_names = FALSE)
```

Arguments

`selections`	Named list with names from the list 'Naive', 'Scalar', 'dOp', 'Loops' 'LpsDw', 'LpsDwCv', 'LoopsCv' specifying the requesetd loop types. The elements of this list are in turn expected be data frames of the form

qx	qy	qz
0	0	1
-2	1	-3
...	...	...

specifying the momentum combinations to be extracted for each loop type.

`files`	Vector of strings, list of HDF5 files to be processed.
`Time`	Integer, time extent of the lattice.
`nstoch`	Integer, number of stochastic samples to be expected in file.
`verbose`	Boolean, output I/O time per file. Requires 'tictoc' package. Default FALSE.
`check_group_names`	Boolean, check if the group names that we're about to read actually exist in the file. This is quite slow because it uses `rhdf5::h5ls`. Default FALSE.

Value

Named nested list of the same length as `selections` containg the loop data in the raw_cf format. Each named element corresponds to one loop type and each element of the underlying numbered list corresponds to one momentum combination as specified via `selections` for this loop type in the same order.

`cvc_to_raw_cf` *Convert correlation function read from CVC HDF5 or AFF format to 'raw_cf'*

Description

Given a numeric vector of alternating real and imaginary parts of a correlation function, creates an object of class 'raw_cf' with a single measurement, inferring `Time` from the passed numeric vector while the shape of the internal dimensions has to be specified explicitly if larger than one by one (`c(1,1)`).

Usage

```
cvc_to_raw_cf(cf_dat, dims = c(1, 1))
```

Arguments

`cf_dat`	Numeric vector of alternating real and imaginary parts of a correlation function. Ordering of the input should be complex, internal dimensions, time (left to right, fastest to slowest).
`dims`	Integer vector with the sizes of the internal dimensions. For example, `c(4,4)` for spin correlators.

Value

`raw_cf` object with a `data` member which contains the data (as complex numbers) in the shape `c(1,nts,dims)`, where `nts` is the number of time slices inferred from the length of `cfdat` and the product of the internal dimensions `dims`.

`cyprus_make_key_scalar` *HDF5 key for Cyprus CalcLoops scalar-type loops*

Description

Generates an HDF5 key (full path) for the scalar type loops from the Cyprus CalcLoops application.

Usage

```
cyprus_make_key_scalar(istoch, loop_type, cid = 4, accumulated = FALSE)
```

Arguments

`istoch`	Integer, index of the stochastic sample that the key should be generated for.
`loop_type`	String, name of loop type. Allowed values: 'Scalar', 'dOp'
`cid`	Integer, configuration number, internally produced by the CalcLoops tool via the "trajectory" input flag. The default is '4' as this is often not used in practice.
`accumulated`	Boolean, depending on whether the loop data was accumulated over the stochastic source d.o.f. or not, the keys are different. Default: FALSE

Value

A character vector with the HDF5 key.

`cyprus_make_key_vector` *HDF5 key for Cyprus CalcLoops derivative-type loops*

Description

Generates an HDF5 key (full path) for the derivative type loops from the Cyprus CalcLoops application.

Usage

```
cyprus_make_key_vector(istoch, loop_type, dir, cid = 4, accumulated = FALSE)
```

Arguments

`istoch`	Integer, index of the stochastic sample that the key should be generated for.
`loop_type`	String, name of loop type. Allowed values: 'Loops', 'LpsDw', 'LpsDwCv', 'LoopsCv'
`dir`	Integer, lattice direction of the derivative. Allowed values: `0 == x`, `1 == y`, `2 == z`, `3 == t`.
`cid`	Integer, configuration number, internally produced by the CalcLoops tool via the "trajectory" input flag. The default is '4' as this is often not used in practice.
`accumulated`	Boolean, depending on whether the loop data was accumulated over the stochastic source d.o.f. or not, the keys are different. Default: FALSE

Value

A character vector with the HDF5 key.

`cyprus_read_loops` *read HDF5 loop files in the Cyprus CalcLoops format*

Description

The CalcLoops code produces HDF5 files which contain a matrix of momenta and the data for the loops (without spin projection) organised by stochastic sample. Currently, the reading code assumes that there is a single configuration stored per file.

Usage

```
cyprus_read_loops(selections, files, Time, nstoch, accumulated = TRUE,
  legacy_traj = TRUE, verbose = FALSE, check_group_names = FALSE,
  spin_project = FALSE, project_gamma = NULL, use_parallel = TRUE)
```

Arguments

`selections` Named list with names from the list 'Naive', 'Scalar', 'dOp', 'Loops' 'LpsDw', 'LpsDwCv', 'LoopsCv' specifying the requesetd loop types. The elements of this list are in turn expected be data frames of the form

px	**py**	**pz**
0	0	1
-2	1	-3
...	...	...

specifying the momentum combinations to be extracted for each loop type.

`files`	Vector of strings, list of HDF5 files to be processed.
`Time`	Integer, time extent of the lattice.
`nstoch`	Integer, number of stochastic samples to be expected in file.

`accumulated`	Boolean or vector of boolean, specifies whether the loops, as organised by stochastic sample, are accumulated, such that, say, element `n` corresponds to the sum over the first `n` stochastic samples. If specified as `TRUE`, the data is post-processed to recover the measurements for the particular samples. In case this is specified as a vector, it must be of the same length as `files`. Default: TRUE.
`legacy_traj`	Boolean. The root group for the loop data is 'conf_xxxx', where 'xxxx' corresponds to what is passed via the 'traj' flag to CalcLoops. When left empty, this defaults to '0004'. If this was left emtpy when the loop files were generated, set this to `TRUE` and the paths will be constructed with 'conf_0004' as their root group. When specified as a vector, it must be of length `length(files)` giving the integer configuration indices, such as `c(0,2,4,6)` Default: TRUE.
`verbose`	Boolean, output I/O time per file. Requires 'tictoc' package. Default FALSE.
`check_group_names`	Boolean, employ `rhdf5::h5ls` to check if all the group names that we want to read are actually in the file. This can be slow for large files. Default: FALSE.
`spin_project`	Boolean, whether the loops should be spin projected after being read. Must be provided to together with `project_gamma`! Default: FALSE
`project_gamma`	Named list of the same length as `selections` containing, for each selected loop type a 4x4 complex-valued projection matrix. For vector loop types, one matrix must be provided per direction (so `project_gamma$loop_type` is a numbered list with indices `c(1,2,3,4)`. Default: NULL
`use_parallel`	Boolean, whether to parallelise over the files using the `mclapply` from the `parallel` package.

Value

Named nested list of the same length as `selections` containg the loop data in the raw_cf format. Each named element corresponds to one loop type. For scalar-valued loop types, each element of the underlying numbered list corresponds to one momentum combination as specified via `selections` for this loop type in the same order. For the vector-valued loop types, the first level of the underlying numbered list has four elements corresponding to the four derivative directions in the order t,x,y,z. At the next level, the underlying numbered list corresponds to the momentum combinations for this loop type and derivative direction, just as for the scalar type.

`disc_3pt` *disconnected contribution to current insertion three-point function*

Description

Computes the quark-line disconnected contribution to a three-point function of the form

$$C_3(t, \Delta t = t_{snk} - t_{src}) = C_2(t_{snk}, t_{src}) * L(t)$$

$\forall t$ considering only the case t_snk > t_src.

Usage

```
disc_3pt(cf_2pt, loop, src_ts, dt, reim_loop = "both", reim_2pt = "both",
  vev_subtract = FALSE)
```

Arguments

`cf_2pt`	'raw_cf' container holding two-point part of three-point function in lattice-absolute coordinates (not relative to source!)
`loop`	'raw_cf' container holding loop contribution, suitably spin-projected and averaged over stochastic samples.
`src_ts`	Integer vector, the source time slices that were used for the computation of the two-point function in lattice-absolute coordinates. Must be of the same length as the number of measurements in `cf_2pt`.
`dt`	Integer, the source-sink separation that should be computed.
`reim_loop`	String, one of 'real', 'imag' or 'both'. Specifies whether just the real or imaginary part should be considered when constructing the correlation with the two-point function.
`reim_2pt`	String, same as `reim_loop` but for the two-point contribution to the three-point function.
`vev_subtract`	Boolean, whether the loop contains a vev which should be subtracted.

Value

`raw_cf` container with the product of loop and 2pt function, shifted in time to be relative to source using the info from `src_ts`

`dispersion_relation` *Continuum dispersion relation for CM to lattice frame*

Description

Converts a center of mass (CM) frame energy to the lattice frame using the continuum dispersion relation.

Usage

```
dispersion_relation(energy, momentum_d, extent_space, plus = TRUE,
  lattice_disp = FALSE)
```

Arguments

`energy`	`double`. CM energy in lattice units, aE.
`momentum_d`	`integer`. Total momentum squared of the moving frame in lattice units, d^2.
`extent_space`	`integer`. Spatial extent of the lattice as a dimensionless quantity, L/a.
`plus`	Boolean. Sign of a^2 artefacts.
`lattice_disp`	Boolean. Use the lattice dispersion relation instead of the continuum one

Value

`double`. Energy in the lattice frame, aW.

`effectivemass` *effectivemass*

Description

computes the effective mass with error analysis using UWerr

Usage

```
effectivemass(from, to, Time, Z, pl = TRUE, S, ...)
```

Arguments

`from`	integer. Fit in fitrange (from, to)
`to`	integer. see from.
`Time`	integer. time extent of the lattice
`Z`	data
`pl`	boolean. plot
`S`	numeric. see uwerr
`...`	additional parameters passed to uwerr

Value

Returns a data.frame with named columns `t`, `mass`, `dmass`, `ddmass`, `tauint` and `dtauint`.

See Also

uwerr

effectivemass.cf *Computes effective mass values for a correlation function*

Description

Computes effective mass values for a correlation function using different type of definitions for the effective mass. This function is mainly indented for internal usage by `bootstrap.effectivemass`.

Usage

```
effectivemass.cf(cf, Thalf, type = "solve", nrObs = 1,
  replace.inf = TRUE, interval = c(1e-06, 2), weight.factor = NULL,
  deltat = 1, tmax = Thalf - 1)
```

Arguments

`cf`	The correlation function either as a vector of length `nrObs*(Thalf+1)` or as an array of dimension Nx`nrObs*(Thalf+1)`, where N is the number of observations. N will be averaged over.
`Thalf`	Half of the time extent of the lattice
`type`	The function to be used to compute the effective mass values. Possibilities are "acosh", "solve", "log", "temporal", "shifted" and "weighted". While the first three assume normal cosh behaviour of the correlation function, "temporal" is desigend to remove an additional constant stemming from temporal states in two particle correlation functions. The same for "subtracted" and "weighted", the latter for the case of two particle energies with the two particle having different energies. In the latter case only the leading polution is removed by `removeTemporal.cf` and taken into account here.
`nrObs`	The number of "observables" included in the correlator
`replace.inf`	If set to `TRUE`, all `Inf` values will be replaced by `NA`. This is needed for instance for `bootstrap.effectivemass`.
`interval`	initial interval for the `uniroot` function when numerically solving for the effective mass.
`weight.factor`	relative weight for type "weighted" only, see details
`deltat`	time shift for shifted correlation functions
`tmax`	t-value up to which the effectivemass is to be computed

Details

A number of types is implemented to compute effective mass values from the correlation function:

"solve": the ratio
$C(t+1)/C(t) = \cosh(-m*(t+1))/\cosh(-m*t)$
is numerically solved for $m(t)$.

"acosh": the effective mass is computed from
$m(t) = acosh((C(t-1) + C(t+1))/(2C(t)))$
Note that this definition is less tolerant against noise.

"log": the effective mass is defined via
$m(t) = \log(C(t)/C(t+1))$
which has artifacts of the periodicity at large t-values.

"temporal": the ratio
$[C(t) - C(t+1)]/[C(t-1) - C(t)] = [\cosh(-m*(t)) - \cosh(-m*(t+1))]/[\cosh(-m*(t-1)) - \cosh(-m(t))]$
is numerically solved for $m(t)$.

"subtracted": like "temporal", but the differences $C(t) - C(t+1)$ are assumed to be taken already at the correlator matrix level using `removeTemporal.cf` and hence the ratio
$[C(t+1)]/[C(t)] = [\cosh(-m*(t)) - \cosh(-m*(t+1))]/[\cosh(-m*(t-1)) - \cosh(-m(t))]$
is numerically solved for $m(t)$.

"weighted": like "subtracted", but now there is an additional weight factor w from `removeTemporal.cf` to be taken into account, such that the ratio
$[C(t+1)]/[C(t)] = [\cosh(-m*(t)) - w*\cosh(-m*(t+1))]/[\cosh(-m*(t-1)) - w*\cosh(-m(t))]$
is numerically solved for $m(t)$ with w as input.

Value

Returns a vector of length `Thalf` with the effective mass values for t-values running from 0 to `Thalf-1`

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

References

arXiv:1203.6041

See Also

`bootstrap.effectivemass`

Examples

```
data(correlatormatrix)
cfnew <- extractSingleCor.cf(correlatormatrix, id=1)
cfnew <- bootstrap.cf(cfnew, boot.R=99, boot.l=1)
X <- effectivemass.cf(cfnew$cf, Thalf=25, tmax=24)
```

effmass *effmass*

Description

computes the effective mass via the inverse cosh

Usage

```
effmass(data, timeextent, t)
```

Arguments

`data`	numeric vector. data vector of length 4
`timeextent`	integer. time extent of the lattice
`t`	integer. physical time at which to evaluate the cosh

Value

Returns the effective mass as a single numeric value.

effmass2 *effmass2*

Description

computes the effective mass via the inverse cosh

Usage

```
effmass2(data, timeextent, t)
```

Arguments

`data`	numeric vector. data vector of length 4
`timeextent`	integer. time extent of the lattice
`t`	integer. physical time at which to evaluate the cosh

Value

Returns the effective mass as a single numeric value.

`escapeLatexSpecials` *Escape special LaTeX characters for use in LaTeX labels*

Description

Escape special LaTeX characters for use in LaTeX labels

Usage

```
escapeLatexSpecials(x)
```

Arguments

`x`	String or vector of strings.

Value

String or vector of strings with all occurences of "#", "$", "%", "&", "~", "_", "^", ">", "<" replaced by escaped counterparts which should render fine when used in a tikz plot, for example.

References

from https://stackoverflow.com/questions/36338629/escaping-special-latex-characters-in-r

`extract.loop` *Extract a single loop from an object of class* `cmiloop`

Description

Extracts all loop values from an object of class `cmiloop` for all available times, samples and configurations.

Usage

```
extract.loop(cmiloop, obs = 9, ind.vec = c(2, 3, 4, 5, 6, 7, 8, 1), L)
```

Arguments

`cmiloop`	input object of class `cmiloop` generated for instance with `readcmiloopfiles`.
`obs`	the observable to extract
`ind.vec`	index vector to be used during extraction with `ind.vec[1]` the column with the observable number, `ind.vec[2]` the time values, `ind.vec[3]` the sample numbers, `ind.vec[4]` the real part of the local loop, `ind.vec[5]` the imaginary part of the local loop, `ind.vec[6]` and `ind.vec[7]` the same for fuzzed (or smeared) loops and `ind.vec[8]` for the configuraton number.
`L`	The spatial lattice extent needed for normalisation. If not given set to `Time/2`.

Value

a list with elements as follows:

`cf`: real part of the local loop

`icf`: imaginary part of the local loop

`scf`: real part of the smeared loop

`iscf`: imaginary part of the smeared loop

`Time=Time`, `nrSamples`, `nrObs=1`, `nrStypes=2`, `obs=obs` and `conf.index`. The last is the list of configurations corresponding to the loops.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

See Also

`readcmiloopfiles`

`extract.obs` *Extract One or More Gamma Combinations from am CMI Correlator*

Description

Extracts one or more gamma matrix combinations (observables) from a correlator stored in cmi format

Usage

```
extract.obs(cmicor, vec.obs = c(1), ind.vec = c(1, 2, 3, 4, 5), sym.vec,
  sign.vec, verbose = FALSE, symmetrise = TRUE)
```

Arguments

`cmicor`	an correlator object in cmi format
`vec.obs`	vector containing the numbers of observables to be extracted.
`ind.vec`	Index vector indexing the column numbers in cmicor to be used. The first must be the observable index, the second the smearing type index, the third the time, the fourth C(+t) and the fifth C(-t). Index vector indexing the column numbers in cmiloop to be used. The first must be the observable index, the second the smearing type index, the third the time, the fourth ReTL, the fifth ImTL, the sixth ReTF and the seventh ImTF.
`sym.vec`	a vector of bools of length equal to the number of observables indicating whether C(t) is symmetric in t, i.e. whether C(+t) and C(-t) should be added or subtracted. If not given C(+t) and C(-t) will be assumed to be symmetric.

`sign.vec`	a sign vector of length equal to the number of observables indicating whether the corresponding correlation function should be multiplied by +-1.
`verbose`	Increases verbosity of the function.
`symmetrise`	if set to `TRUE`, the correlation function will be averaged for `t` and `Time-t`, with the sign depending on the value of `sym`. Note that currently the correlator with t-values larger than `Time/2` will be discarded.

Details

C(t) and C(-t) are averaged as indicated by `sym.vec`.

Value

returns a list containing

`cf`	for `extract.obs`: array containing the correlation function with dimension number of files times (nrObs*nrStypes*(Time/2+1)). C(t) and C(-t) are averaged according to `sym.vec`. for `extract.loop`: ReTL
`icf`	for `extract.loop` only: ImTL
`scf`	for `extract.loop` only: ReTF
`sicf`	for `extract.loop` only: ImTF
`Time`	The time extent of the correlation functions.
`nrStypes`	The number of smearing combinations.
`nrObs`	The number of observables.
`nrSamples`	for `extrac.loop` only: the number of samples found in the files.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

See Also

`readcmicor`, `readcmidatafiles`,

Examples

```
files <- paste0(system.file(package="hadron"), "/extdata/outprcvn.dddd.00.0000")
X <- readcmifiles(files, skip=0,
                  colClasses=c("integer", "integer","integer","numeric","numeric"))
Y <- extract.obs(X)
Y
```

extractSingleCor.cf *extract one single correlator object as cf object from a large cf object.*

Description

cf objects are capable of storing several correlation functions in form of a correlation matrix. extractSingleCor.cf lets one extract one of them.

Usage

```
extractSingleCor.cf(cf, id = c(1))
```

Arguments

cf	input object of class cf
id	id of the correlators in cf to be extracted

Value

A cf object containing only the single correlator

Returns an object of class cf corresponding ot the ids element in the input object cf

Author(s)

Carsten Urbach <curbach@gmx.de>

See Also

cf

extract_mass *generic function to extract a fitted mass*

Description

One of the main analysis tasks in hadron is the estimation of energy levels or masses from correlation functions. The corresponding analysis functions return objects, typically lists, containing the masses or energy levels. extract_mass is a generic function to extrac such fitted mass values.

Usage

```
extract_mass(object)
```

Arguments

object	Object to extract the mass from.

Value

Numeric. The mass value.

---

extract_mass.effectivemassfit

specialisation of extract_mass to objects of type `effectivemassfit`

Description

specialisation of extract_mass to objects of type `effectivemassfit`

Usage

```
## S3 method for class 'effectivemassfit'
extract_mass(object)
```

Arguments

`object`	Object of type `effectivemassfit` to extract the mass from.

Value

Numeric. The mass value.

---

extract_mass.matrixfit

specialisation of extract_mass to objects of type `matrixfit`

Description

specialisation of extract_mass to objects of type `matrixfit`

Usage

```
## S3 method for class 'matrixfit'
extract_mass(object)
```

Arguments

`object`	Object of type `matrixfit` to extract the mass from.

Value

Numeric. The mass value.

fit.cosh — *Fits a sum of several* cosh*-functions*

Description

Performs a correlated fit of a sum of several cosh-functions $\sum_i a_i \cosh(m_i t)$ to data generated with `bootstrap.effectivemass`. Requires the same input and produces analogous output as fit.effectivemass. The fit itself is performed by bootstrap.nlsfit.

Usage

```
fit.cosh(effMass, cf, t1, t2, useCov = FALSE, m.init, par, n.cosh = 2,
  adjust.n.cosh = FALSE, every, ...)
```

Arguments

`effMass`	An object of class `effectivemass` generated by a call to `bootstrap.effectivemass`. Either `effMass` or `cf` has to be provided, but not both!
`cf`	An object of class `cf_boot` generated by a call to `bootstrap.cf`. Either `cf` or `effMass` has to be provided, but not both!
`t1`	The fit range. If several correlators are fitted, this is automatically replicated accordingly. The fit range is adjusted such that `NA`s are removed from the fit. They must fulfill $t_1 < t_2$. For symmetric correlators, they must both run from 0 to `T/2-1`, otherwise from 0 to `T-1`.
`t2`	see t1
`useCov`	Use the correlated chisquare. This works only for not too noisy data.
`m.init`	Initial guess of the effective mass, i.e. the smallest m_i.
`par`	Array of length `2*n.cosh` with initial guesses for the effective masses in the first `n.cosh` entries and initial guesses for the amplitudes in the last `n.cosh` entries.
`n.cosh`	Number of cosh-functions summed over.
`adjust.n.cosh`	Only relevant, if `n.cosh=2`. If set to `TRUE`, `n.cosh` can be adjusted to `n.cosh=1` automatically in case the excited state cannot be resolved.
`every`	Fit only a part of the data points. Indices that are not multiples of `every` are skipped. If no value is provided, all points are taken into account.
`...`	Additional parameters passed to the fit function. But the fit function is fixed and does not accept any arguments, so it will just crash. Therefore, don't use this!

Value

An object with class `coshfit` is returned. It contains all the data of the input object `effMass` or the `cf` object as a member. The following member objects are added:

`t0`: the object returned by the `optim` on the original data. The format is as in `par`.

`t`: the bootstrap values of the results.

se: errors calculated via bootstrap on the results.

ii: the index array of data used in the fit.

invCovMatrix: the inverse covariance matrix.

dof: the degrees of freedom of the fit.

chisqr: Chi squared value of the fit.

Qval: p-value of the fit.

Author(s)

Johann Ostmeyer, <ostmeyer@hiskp.uni-bonn.de>

See Also

bootstrap.effectivemass, bootstrap.gevp, invertCovMatrix, bootstrap.nlsfit, fit.effectivemass

Examples

```
data(samplecf)
samplecf <- bootstrap.cf(cf=samplecf, boot.R=99, boot.l=2, seed=1442556)
effmass <- fit.cosh(bootstrap.effectivemass(cf=samplecf), t1=15, t2=23)
summary(effmass)
plot(effmass, ylim=c(0.14,0.15))
```

---

fit.effectivemass *Fits a constant to effective mass data*

Description

Performs a correlated fit of a constant to data generated with bootstrap.effectivemass.

Usage

```
fit.effectivemass(cf, t1, t2, useCov = FALSE, replace.na = TRUE,
  boot.fit = TRUE, autoproceed = FALSE, every)
```

Arguments

cf	An object of class effectivemass generated by a call to bootstrap.effectivemass.
t1, t2	The fit range. If several correlators are fitted, this is automatically replicated accordingly. The fit range is adjusted such that NAs are removed from the fit. They must fulfill $t_1 < t_2$. For symmetric correlators, they must both run from 0 to Time/2-1, otherwise from 0 to Time-1.
useCov	Use the correlated chisquare. This works only for not too noisy data.

`replace.na`	The functions inverted to determine the effective mass values might, due to fluctuations, return `NA`. If `replace.na=TRUE`, these are reaplaced in the bootstrap samples by randomly chosen values from the distribution that are not `NA`. Otherwise the fits in which the `NA` values occur will fail.
`boot.fit`	If set to `FALSE`, the effective mass fit is not bootstrapped, even though bootstrap samples are still used to estimate the variance-covariance matrix for the correlated fit. This is a useful time-saver if error information is not strictly necessary. Of course, this affects the return values related to the bootstrap, which are set to `NA`.
`autoproceed`	When the inversion of the variance-covariance matrix fails, the default behaviour is to abort the fit. Setting this to `TRUE` means that the fit is instead continued with a diagonal inverse of the variance-covariance matrix.
`every`	Fit only a part of the data points. Indices that are not multiples of `every` are skipped. If no value is provided, all points are taken into account.

Details

A correlated chisquare minimisation is performed on the original data as well as on all bootstrap samples generated by `bootstrap.effectivemass`. The inverse covariance matrix is generated as described in hep-lat/9412087 in case of too little data to relibably estimate it.

Value

An object with class `effectivemassfit` is returned. It contains all the data of the input object `effMass` with the following additional member objects:

`opt.res`: the object returned by the `optim` on the original data.

`massfit.tsboot`: the bootstrap values of the mass and the chisquare function.

`ii`: the index array of data used in the fit.

`invCovMatrix`: the inverse covariance matrix.

`dof`: the degrees of freedom of the fit.

`t1,t2`: the fit range.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

References

C.Michael, A.McKerrell, Phys.Rev. D51 (1995) 3745-3750, hep-lat/9412087

See Also

`bootstrap.effectivemass`, `bootstrap.gevp`, `gevp2cf`, `invertCovMatrix`

Examples

```
data(samplecf)
samplecf <- bootstrap.cf(cf=samplecf, boot.R=99, boot.l=2, seed=1442556)
effmass <- fit.effectivemass(bootstrap.effectivemass(cf=samplecf), t1=15, t2=23)
summary(effmass)
plot(effmass, ylim=c(0.14,0.15))
```

`fit.plateau2cf` *fits a plateau to an object of class* `cf`

Description

where applicable, a plateau is fitted to the averaged data in `cf` using a (correlated) chisquare fit.

Usage

```
fit.plateau2cf(cf, t1, t2, useCov = FALSE)
```

Arguments

`cf`	input object of class `cf`
`t1`	starting t-value for the fit
`t2`	final t-value for the fit.
`useCov`	perform a correlated chisquare fit or not.

Value

Returns a list with elements

`plateau`	the fitted plateau value
`dplateau`	its error

Author(s)

Carsten Urbach <`curbach@gmx.de`>

See Also

`cf`

Examples

```
data(correlatormatrix)
cfnew <- extractSingleCor.cf(correlatormatrix, id=1)
cfnew <- bootstrap.cf(cfnew, boot.R=99, boot.l=1)
X <- fit.plateau2cf(cfnew, t1=13, t2=20)
```

`foldr1` *Folds the non-empty list with the binary function*

Description

A right fold without the need for a neutral element. Does not work with empty lists.

Usage

```
foldr1(f, xs)
```

Arguments

f	`function`. A binary function that takes two elements of the type contained in `xs` and returns another such element.
xs	`list` or vector. Homogenious list or vector of elements. There is a `Reduce` function in base R that does left and right folds. It always needs a starting element, which usually is the neutral element with respect to the binary operation. We do not want to specify such a neutral element for certain operations, like `+.cf`. Still a functional programming style should be supported such that one can use maps and folds.

Examples

```
# We generate some random numbers.
numbers <- rnorm(10)

# The sum is easiest computed with the `sum` function:
sum(numbers)

# If we wanted to implement `sum` ourselves, we can use a right fold to do
# so:
Reduce(`+`, numbers, 0.0)

# With this new function we do not need a neutral element any more, but give
# up the possibility to fold empty lists.
foldr1(`+`, numbers)
```

`fs.a0` *Finite Size Corrections to $q\cot\delta$ for I=2 $\pi\pi$ near threshold*

Description

`fs.qcotdelta` computes the finite size corrections to $q\cot\delta$ while `fs.mpia0` computes the corresponding finite size corrections to $M_\pi a_0$ directly using the Gasser Leutwyler result from M_π.

Usage

```
fs.a0(a0, mps, L)
```

Arguments

`a0`	scattering length at finite L
`mps`	pion mass as a scalar variable (must not be a vector)
`L`	spatial lattice extent as a scalar variable (must not be a vector)

Value

returns a numeric value representing the finite size correction or in case of `fs.a0` the corrected value for a0.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

References

For the original formula see Eq. (31) from hep-lat/0601033

Examples

```
fs.a0(a0=1., mps=0.123, L=24)
```

`fs.mpia0` *Finite Size Corrections to $q\cot\delta$ for I=2 $\pi\pi$ near threshold*

Description

`fs.qcotdelta` computes the finite size corrections to $q\cot\delta$ while `fs.mpia0` computes the corresponding finite size corrections to $M_\pi a_0$ directly using the Gasser Leutwyler result from M_π.

Usage

```
fs.mpia0(mps, fps, L)
```

Arguments

`mps`	pion mass as a scalar variable (must not be a vector)
`fps`	pion decay constant as a scalar variable (must not be a vector)
`L`	spatial lattice extent as a scalar variable (must not be a vector)

Value

returns a numeric value representing the finite size correction or in case of `fs.a0` the corrected value for a0.

Author(s)

Carsten Urbach, <curbach@gmx.de>

References

For the original formula see Eq. (31) from hep-lat/0601033

Examples

```
fs.mpia0(mps=0.123, fps=0.2, L=24)
```

`fs.qcotdelta` *Finite Size Corrections to $q\cot\delta$ for I=2 $\pi\pi$ near threshold*

Description

`fs.qcotdelta` computes the finite size corrections to $q\cot\delta$ while `fs.mpia0` computes the corresponding finite size corrections to $M_\pi a_0$ directly using the Gasser Leutwyler result from M_π.

Usage

```
fs.qcotdelta(mps, L)
```

Arguments

`mps`	pion mass as a scalar variable (must not be a vector)
`L`	spatial lattice extent as a scalar variable (must not be a vector)

Value

returns a numeric value representing the finite size correction or in case of `fs.a0` the corrected value for a0.

Author(s)

Carsten Urbach, <curbach@gmx.de>

References

For the original formula see Eq. (31) from hep-lat/0601033

Examples

```
  fs.qcotdelta(mps=0.123, L=24)
```

g1 *g1*

Description

Implementation of the Gasser-Leutwyler function g_1 for computing finite volume effects.

Usage

```
g1(x)
```

Arguments

`x`	Numeric. x-value

getorderedconfignumbers *Creates an ordered vector of gauge config file numbers*

Description

These functions generate an ordered list of config numbers by using a path and a basename and '*'.

Usage

```
getorderedconfignumbers(path = "./", basename = "onlinemeas",
  last.digits = 4, ending = "")
```

Arguments

`path`	the path to be searched
`basename`	the basename of the files
`last.digits`	the number of last characters in each filename to be used for ordering the list.
`ending`	the file extension after the digits.

Details

All filenames are assumend to have equal length.

Value

returns the ordered list of gauge config numbers as a numeric vector.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

See Also

readcmidatafiles, extract.obs

Examples

```
confignumbers <- getorderedconfignumbers(path=paste0(system.file(package="hadron"), "/extdata/"),
                                         basename="testfile", last.digits=3, ending=".dat")
confignumbers
```

---

getorderedfilelist *Creates an ordered filelist from a basename and a path*

Description

These functions generate an ordered filelist and an order list of config numbers by using a path and a basename and '*'.

Usage

```
getorderedfilelist(path = "./", basename = "onlinemeas", last.digits = 4,
  ending = "")
```

Arguments

path	the path to be searched
basename	the basename of the files
last.digits	the number of last characters in each filename to be used for ordering the list.
ending	the file extension after the digits.

Details

All filenames are assumend to have equal length.

Value

returns the ordered list of strings.

Author(s)

Carsten Urbach, <curbach@gmx.de>

See Also

readcmidatafiles, extract.obs

Examples

```
filelist <- getorderedfilelist(path=paste0(system.file(package="hadron"), "/extdata/"),
                               basename="testfile", last.digits=3, ending=".dat")
filelist
```

`get_plotdata_raw_cf` *extract data from 'raw_cf' in format convenient to plot*

Description

When dealing with with tensorial `raw_cf` objects pre-processing and reshaping is always required to prepare the data for plotting (or similar). This function conveniently prepares a named list of prepared data. The list elements are themselves lists which contain `val` and `dval` members with the central value and error of the element in question. These are in turn arrays of dimension `c( cf$nts,cf$dim )` and thus lack the first index compared to `cf$data`.

Usage

```
get_plotdata_raw_cf(cf, reim, tauint, relerr)
```

Arguments

`cf`	`raw_cf` object with meta-data and data.
`reim`	String, one of 'real', 'imag' or 'both'. Specifies whether the real and/or imaginary parts should be extracted.
`tauint`	Boolean, specifies if the tensor of auto-correlation times and corresponding errors should be extracted.
`relerr`	Boolean, specifies if the return value should also include estimates of the relative error and its error.

Value

List of up to six named elements (depending on what was passed for `reim`, `tauint`, `relerr`) containing the central values and errors of the real and/or imaginary part of `cf$data` as well as the corresponding arrays of auto-correlation times and relative errors. The list elements come in the order `real`, `imag`, `relerr_real`, `relerr_imag`, `tauint_real`, `tauint_imag` if `reim` is `both` and `tauint` and `relerr` are `TRUE`. The `val` and `dval` members of these list elements are arrays of dimension `c( cf$nts,cf$dim )` and thus lack the first index compared to `cf$data`.

gevp *solve GEVP for correlator matrix*

Description

solve GEVP for a real, symmetric correlator matrix

Usage

```
gevp(cf, Time, t0 = 1, element.order = 1:cf$nrObs, for.tsboot = TRUE,
  sort.type = "vectors", sort.t0 = TRUE)
```

Arguments

`cf`	correlation matrix preferably obtained with a call to `extrac.obs` (or at leas with the same structure) or an already averaged one. cf is supposed to be an array of `dim=c(N,n*(Time/2+1))`, where `N` is the number of observations and `n` is the number of single correlators in the matrix. E.g. for a 2x2 matrix `n` would be 4.
`Time`	time extent of the lattice.
`t0`	initial time value of the GEVP, must be in between 0 and `Time/2-2`. Default is 1.
`element.order`	specifies how to fit the `n` linearly ordered single correlators into the correlator matrix. `element.order=c(1,2,3,4)` leads to a matrix `matrix(cf[element.order],nrow=2)`.
`for.tsboot`	for internal use of `bootstrap.gevp`. Alters the returned values, see details.
`sort.type`	Sort the eigenvalues either in descending order, or by using the scalar product of the eigenvectors with the eigenvectors at $t = t_0 + 1$. Possible values are "values", "vectors" or "det".
`sort.t0`	if true (default), sort with respect to data at t0, otherwise with respect to t-1.

Details

The generalised eigenvalue problem
$C(t)v(t,t_0) = C(t_0)\lambda(t,t_0)v(t,t_0)$
is solved by performing a Cholesky decomposition of $C(t_0) = L^tL$ and transforming the GEVP into a standard eigenvalue problem for all values of t. The matrices C are symmetrised for all t. So we solve for λ
$(L^t)^{-1}C(t)L^{-1}w = \lambda w$
with
$w = Lv$ or the wanted $v = L^{-1}w$.

The amplitudes can be computed from
$A_i^{(n)}(t) = \sum_j C_{ij}(t)v_j^{(n)}(t,t_0)/(\sqrt{(v^{(n)}, Cv^{(n)})(\exp(-mt) \pm \exp(-m(t-t)))})$ and this is what the code returns up to the factor
$1/\sqrt{\exp(-mt) \pm \exp(-m(t-t))}$ The states are sorted by their eigenvalues when "values" is chosen. If "vectors" is chosen, we take $\max(\sum_i \langle v(t_0,i), v(t,j)\rangle)$ with v the eigenvectors. For sort type "det" we compute $\max(...)$

Value

Returns a list with the sorted eigenvalues, sorted eigenvectors and sorted (reduced) amplitudes for all t > t0.

In case `for.tsboot=TRUE` the same is returned as one long vector with first all eigenvalues concatenated, then all eigenvectors and then all (reduced) amplitudes concatenated.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

References

Michael, Christopher and Teasdale, I., Nucl.Phys.B215 (1983) 433, DOI: 10.1016/0550-3213(83)90674-0
Blossier, B. et al., JHEP 0904 (2009) 094, DOI: 10.1088/1126-6708/2009/04/094, arXiv:0902.1265

See Also

`boostrap.gevp`, `extract.obs`

`gevp.hankel` *GEVP method based on Hankel matrices.*

Description

Alternative method to determine energy levels from correlation matrices. A so-called Hankel matrix is generated from an input real numeric vector and a generalised eigenvalue problem is solved then.

Usage

```
gevp.hankel(cf, t0 = 1, deltat = 1, n, N, submatrix.size = 1,
  element.order = c(1, 2, 3, 4), Delta = 1)
```

Arguments

`cf`	Numeric vector (this will generally be the time slices of a correlation function).
`t0`	Integer. Initial time value of the GEVP, must be in between 0 and `Time/2-2`. Default is 1.
`deltat`	Integer. Time shift to be used to build the Hankel matrix
`n`	Integer. Size of the Hankel matrices to generate
`N`	Integer. Maximal time index in correlation function to be used in Hankel matrix
`submatrix.size`	Integer. Submatrix size to be used in build of Hankel matrices. Submatrix.size > 1 is experimental.

`element.order`	Integer vector. specifies how to fit the `n` linearly ordered single correlators into the correlator matrix for submatrix.size > 1. `element.order=c(1,2,3,4)` leads to a matrix `matrix(cf[element.order],nrow=2)`. Matrix elements can occur multiple times, such as `c(1,2,2,3)` for the symmetric case, for example.
`Delta`	integer. Delta is the time shift used in the Hankel matrix.

Value

A complex vector of length `n + n^2` which contains the eigenvalues in the first `n` elements and the eigenvectors in the remaining `n^2` elements.

A vector of NAs of `n + n^2` is returend in case the QR decomposition fails.

See Also

Other hankel: `bootstrap.hankel_summed()`, `bootstrap.hankel()`, `gevp.hankel_summed()`, `hankel2cf()`, `hankel2effectivemass()`, `plot_hankel_spectrum()`

`gevp.hankel_summed` *GEVP method based on Hankel matrices.*

Description

Alternative method to determine energy levels from correlation matrices. A so-called Hankel matrix is generated from an input real numeric vector and a generalised eigenvalue problem is solved then.

Usage

```
gevp.hankel_summed(cf, t0values = c(1), deltat = 1, n, N)
```

Arguments

`cf`	Numeric vector (this will generally be the time slices of a correlation function).
`t0values`	Integer vector. The t0 values to sum over.
`deltat`	Integer. The value of the time shift to use to build the Hankel matrices.
`n`	Integer. Size of the Hankel matrices to generate
`N`	Integer. Maximal time index in correlation function to be used in Hankel matrix

Value

A complex vector of length `n + n^2` which contains the eigenvalues in the first `n` elements and the eigenvectors in the remaining `n^2` elements.

A vector of NAs of `n + n^2` is returend in case the QR decomposition fails.

See Also

Other hankel: `bootstrap.hankel_summed()`, `bootstrap.hankel()`, `gevp.hankel()`, `hankel2cf()`, `hankel2effectivemass()`, `plot_hankel_spectrum()`

`gevp2amplitude` *Extracts physical amplitudes from a GEVP*

Description

Given a GEVP generated with `bootstrap.gevp` and masses determined from the principle correlator with given `id`, the physical amplitudes are extracted and bootstraped. The man amplitude is determined from a constant fit to the data in the specified time range.

Usage

```
gevp2amplitude(gevp, mass, id = 1, op.id = 1, type = "cosh", t1, t2,
  useCov = TRUE, fit = TRUE)
```

Arguments

`gevp`	An object of class `gevp` as generated with a call to `bootstrap.gevp`.
`mass`	Optimally, this is an object either of class `effectivemassfit` generated using `fit.effectivemass` or of class `matrixfit` generated with `matrixfit` to the principal correlator extracted using `gevp2cf` applied to `gevp` using the same value of `id`. It can also be given as a numerical vector with the bootstrap samples as entries. The mean will then be computed as the bootstrap mean over this vector. The number of samples must agree with the number of bootstrap samples in `gevp`.
`id`	The index of the principal correlator to extract, i.e. the physical state to extract.
`op.id`	The index of the operator for which to extract the amplitude.
`type`	The symmetry of the pricipal correlator in time, can be either "cosh" or "sinh".
`t1, t2`	The time range in which to fit the amplitude starting with 0. If not given it will be tried to infer these from the `mass` object.
`useCov`	Use the covariance matrix for fitting the constant to the amplitude data.
`fit`	perform a fit to the data.

Value

Returns an object of S3 class `gevp.amplitude`, generated as a list with named elements `amplitude` the numeric vector of amplitudes, `amplitude.tsboot` the corresponding bootstrap samples, `damplitude` the estimates for the standard errors, `fit` the object returned by the fit routine, `meanAmplitude` and `meanAmplitude.tsboot` mean amplitude and its bootstrap samples, `chisqr` the residual sum of squares, `dof` the numberi of degrees of freedom, `t1` and `t2` the fit range, and then all the input objects.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

See Also

matrixfit, fit.effectivemass, gevp, gevp2cf, computefps

Examples

```
data(correlatormatrix)
## bootstrap the correlator matrix
correlatormatrix <- bootstrap.cf(correlatormatrix, boot.R=99, boot.l=1, seed=132435)
## solve the GEVP
t0 <- 4
correlatormatrix.gevp <- bootstrap.gevp(cf=correlatormatrix, t0=t0, element.order=c(1,2,3,4))
## extract the ground state and plot
pion.pc1 <- gevp2cf(gevp=correlatormatrix.gevp, id=1)
pion.pc1.effectivemass <- bootstrap.effectivemass(cf=pion.pc1, type="solve")
pion.pc1.effectivemass <- fit.effectivemass(pion.pc1.effectivemass, t1=8, t2=23,
                                            useCov=FALSE)
## now determine the amplitude
pion.pc1.amplitude <- gevp2amplitude(correlatormatrix.gevp, pion.pc1.effectivemass,
                                     useCov=FALSE, t1=8, t2=14)
plot(pion.pc1.amplitude)
summary(pion.pc1.amplitude)
```

---

gevp2cf *Extracts a principle correlator from a GEVEP*

Description

Extracts a principle correlator from a GEVP and converts it into an object of class `cf`

Usage

```
gevp2cf(gevp, id = 1)
```

Arguments

`gevp`	An object returned by `bootstrap.gevp`.
`id`	The index of the principal correlator to extract.

Value

An object of class `cf`, which contains bootstrap samples already. So a call to `bootstrap.cf` is neither needed nor possible. It can be treated further by `bootstrap.effectivemass` or `matrixfit` to extract a mass value.

Author(s)

Carsten Urbach, <curbach@gmx.de>

See Also

gevp, matrixfit, bootstrap.effectivemass

Examples

```
data(correlatormatrix)
## bootstrap the correlator matrix
correlatormatrix <- bootstrap.cf(correlatormatrix, boot.R=99, boot.l=1, seed=132435)
## solve the GEVP
t0 <- 4
correlatormatrix.gevp <- bootstrap.gevp(cf=correlatormatrix, t0=t0, element.order=c(1,2,3,4))
## extract the ground state and plot
pc1 <- gevp2cf(gevp=correlatormatrix.gevp, id=1)
plot(pc1, log="y")
```

---

gm *List of arrays of gamma structures*

Description

List of arrays of 4x4 complex gamma matrices in the tmLQCD chiral gamma basis, where $\gamma^5 = \gamma^0\gamma^1\gamma^2\gamma^3 =$ `diag(c(1,1,-1,-1))` and the UKQCD gamma basis, where $\gamma^5 = \gamma^0\gamma^1\gamma^2\gamma^3$.

The index mappings are as follows

- `gm[['chiral_tmlqcd']][1,,]` γ^0
- `gm[['chiral_tmlqcd']][2,,]` γ^1
- `gm[['chiral_tmlqcd']][3,,]` γ^2
- `gm[['chiral_tmlqcd']][4,,]` γ^3
- `gm[['chiral_tmlqcd']][5,,]` γ^5
- `gm[['chiral_tmlqcd']][6,,]` positive parity projector $\frac{1}{2}(1+\gamma^0)$
- `gm[['chiral_tmlqcd']][7,,]` negative parity projector $\frac{1}{2}(1-\gamma^0)$

- `gm[['ukqcd']][1,,]` γ^1
- `gm[['ukqcd']][2,,]` γ^2
- `gm[['ukqcd']][3,,]` γ^3
- `gm[['ukqcd']][4,,]` γ^4
- `gm[['ukqcd']][5,,]` γ^5
- `gm[['ukqcd']][6,,]` positive parity projector $\frac{1}{2}(1+\gamma^4)$
- `gm[['ukqcd']][7,,]` negative parity projector $\frac{1}{2}(1-\gamma^4)$

The function gm_mu can be used to access its elements using a more "natural" indexing.

`gm_mu` *Accessor function for* `gm`

Description

Retrieve the entries of the gm list of three-index arrays containing various gamma structures in a natural indexing.

Usage

```
gm_mu(mu, basis = "chiral_tmlqcd")
```

Arguments

`mu`	Number or string denoting • Lorentz index (0,1,2,3,4) for γ^μ • 5 for γ^5 • "Pp" or "Pm" for the positive and negative parity projectors respectively
`basis`	String, gamma basis to use. Possible values **'ukqcd':** UKQCD gamma basis with $\gamma^i, i \in [1,2,3,4]$ and $\gamma^5 = \gamma^1\gamma^2\gamma^3\gamma^4$, such that $1 = x$, $4 = t$. **'chiral_tmlqcd':** Chiral gamma basis used by tmLQCD with $\gamma^\mu, \mu \in [0,1,2,3]$ and $\gamma^5 = \gamma^0\gamma^1\gamma^2\gamma^3$, such that $0 = t$, $3 = z$.

Value

Returns the requested γ matrix as a 4x4 complex valued array, see gm.

`h5_get_dataset` *get dataset from HDF5 file*

Description

get dataset from HDF5 file

Usage

```
h5_get_dataset(h5f, key, check_exists = TRUE)
```

Arguments

`h5f`	HDF5 file opened with `rhdf5::H5Fopen`
`key`	String, full path to dataset.
`check_exists`	Boolean, check if key actually exists (keep in mind overhead).

Value

Returns the requested dataset, if successfully read from file.

`h5_names_exist` *check if group names exist in HDF5 file*

Description

The group names in an HDF5 file are stored as full paths as well as a flat vector. It is thus possible to check if a particular set of group names exist in the file by parsing the `name` member of the output of `rhdf5::h5ls`. This function does just that.

Usage

```
h5_names_exist(h5f, nms_to_find)
```

Arguments

`h5f`	HDF5 file handle openend with `rhdf5::H5Fopen`
`nms_to_find`	Vector of strings, group names (not full paths) which are to be located in the file.

Value

Vector of booleans of the same length as `nms_to_find` indicating whether the name at the same index position was located in the file.

`hadron` *The Hadron Package*

Description

An R implementation of fitting routines used in lattice QCD. It provides useful functions for extraction hadronic quantities and such like.

Details

Toolkit to perform statistical analyses of correlation functions generated from Lattice Monte Carlo simulations. In particular, a class `cf` for correlation functions and methods to analyse those are defined. This includes (blocked) bootstrap and jackknife, but also an automatic determination of integrated autocorrelation times. `hadron` also provides a very general function `bootstrap.nlsfit` to bootstrap a non-linear least squares fit. More specific functions are provided to extract hadronic quantities from Lattice Quantum Chromodynamics simulations, a particular Monte Carlo simulation,(see e.g. European Twisted Mass Collaboration, P. Boucaud et al. (2008) doi: 10.1016/j.cpc.2008.06.013). Here, to determine energy eigenvalues of hadronic states, specific fitting routines and in particular the generalised eigenvalue method (see e.g. B. Blossier et al. (2009) doi: 10.1088/11266708/2009/04/094 and M. Fischer et al. (2020) `https://inspirehep.net/literature/1792113`) are implemented. In addition, input/output and plotting routines are available.

Author(s)

Carsten Urbach, <urbach@hiskp.uni-bonn.de>

`hankel2cf` *hankel2cf*

Description

hankel2cf

Usage

```
hankel2cf(hankel, id = c(1), range = c(0, 1), eps = 1e-16,
  sort.type = "values", sort.t0 = TRUE)
```

Arguments

`hankel`	object as returned from bootstrap.hankel
`id`	Integer. Index of eigenvalue to consider, $1 \leq id \leq n$.
`range`	Numeric vector. Value-range for the real part of the eigenvalues (not the energies). If outside this range, the eigenvalue will be discarded
`eps`	Numeric. Cut-off: if the imaginary part of the generalised eigenvalues is larger than eps, the eigenvalue is discarded.
`sort.type`	the sort algorithm to be used to sort the eigenvalues. This can be either simply "values", or the eigenvector information is used in addition with "vectors"
`sort.t0`	Boolean. Whether to use the eigenvector at t0 or the one at deltat-1 for sorting

Value

Returns an object of S3 class `cf`.

See Also

input is generated via bootstrap.hankel alternatively use hankel2effectivemass. For the `cf` class see cf

Other hankel: `bootstrap.hankel_summed()`, `bootstrap.hankel()`, `gevp.hankel_summed()`, `gevp.hankel()`, `hankel2effectivemass()`, `plot_hankel_spectrum()`

hankel2effectivemass *hankel2effectivemass*

Description

hankel2effectivemass

Usage

```
hankel2effectivemass(hankel, id = c(1), type = "log", range = c(0, 1),
  eps = 1e-16, sort.type = "values", sort.t0 = TRUE)
```

Arguments

`hankel`	object as returned from bootstrap.hankel
`id`	Integer. Index of eigenvalue to consider, $1 \le id \le n$.
`type`	Character vector. Type of effective mass to use. Must be in `c("log","acosh")`
`range`	Numeric vector. Value-range for the real part of the eigenvalues (not the energies). If outside this range, the eigenvalue will be discarded
`eps`	Numeric. Cut-off: if the imaginary part of the generalised eigenvalues is larger than eps, the eigenvalue is discarded.
`sort.type`	the sort algorithm to be used to sort the eigenvalues. This can be either simply "values", or the eigenvector information is used in addition with "vectors"
`sort.t0`	Boolean. Whether to use the eigenvector at t0 or the one at deltat-1 for sorting

Value

Returns an object of S3 class `effectivemass`.

See Also

input is generated via bootstrap.hankel alternatively use hankel2cf. See also bootstrap.effectivemass

Other hankel: `bootstrap.hankel_summed()`, `bootstrap.hankel()`, `gevp.hankel_summed()`, `gevp.hankel()`, `hankel2cf()`, `plot_hankel_spectrum()`

hankeldensity2effectivemass *hankeldensity2effectivemass*

Description

computes the density of all bootstrap replicates of effective masses

Usage

```
hankeldensity2effectivemass(hankel, range = c(0, 1), method = "median")
```

Arguments

`hankel`	object as returned from bootstrap.hankel
`range`	Numeric vector. Value-range for the real part of the eigenvalues. If outside this range, the eigenvalue will be discarded
`method`	Character vector. Method to be used to determine the central value of the effective mass. Must be "median" (default) or "density"

Value

Returns an object of S3 class `effectivemass`. #'

See Also

bootstrap.effectivemass, hankel2effectivemass

has_icf *Checks whether the cf object contains an imaginary part*

Description

Checks whether the cf object contains an imaginary part

Usage

```
has_icf(.cf)
```

Arguments

`.cf`	`cf` object

Value

Returns `TRUE` if the `.cf` object has an element `icf`, which is the imaginary component of the correlation function.

`idx_matrix.raw_cf` *Construct the tensor index set for the entire raw correlator*

Description

Construct the tensor index set for the entire raw correlator

Usage

```
idx_matrix.raw_cf(cf, component)
```

Arguments

`cf`	'raw_cf' container with data and meta-data
`component`	Integer vector. Optional argument to obtain a subset of the index matrix to access a particular element of the interior dimensions. Must of the the same length as cf$dim.

Value

An object of type matrix is returned containing the tensor index set.

`int_idx_matrix.raw_cf` *Construct tensor index set for the internal degrees of freedom*

Description

Construct tensor index set for the internal degrees of freedom

Usage

```
int_idx_matrix.raw_cf(cf)
```

Arguments

`cf`	`raw_cf` container

Value

Returns a matrix containing the above mentioned index set.

invalidate.samples.cf *Invalidate samples*

Description

When a correlation function is modified, any resampling should be invalidated. We could instead also choose to properly work with the samples, but most computations are done with the original data anyway.

Usage

```
invalidate.samples.cf(cf)
```

Arguments

`cf`	`cf` object.

Value

Returns an object of class `cf` with all resampling removed.

invcosh *numerically invert the cosh function for the mass*

Description

numerically invert the cosh function for the mass

Usage

```
invcosh(ratio, timeextent, t, eps = 1e-09, maxiterations = 1000)
```

Arguments

`ratio`	Numeric. The value of the ratio.
`timeextent`	Integer. Time extent of the lattice.
`t`	Integer. The t-value where the ratio was taken.
`eps`	Numeric. Precision of the numerical solution
`maxiterations`	Integer. Maximal number of iterations to be used in the iterative solver.

Value

A single numeric value is returned corresponding to the mass.

Examples

```
invcosh(1.2, timeextent=24, t=12)
```

`invertCovMatrix` *Inverts the covariance matrix for noisy data*

Description

The covariance matrix of noisy data is inverted. Special care is taken in treating spurious small modes of the matrix, which are likely to arise from too much noise in the data.

Usage

```
invertCovMatrix(cf, boot.l = 1, boot.samples = FALSE, cov_fn = cov)
```

Arguments

`cf`	The data for which the covariance matrix is to be computed. It is expected to be an array or matrix with dimension RxN, where R is the number of observations and N the number of observables. `cf` can be either real data or bootstrap data. In the latter case `boot.samples=TRUE` must be set for proper normalisation of the inverse matrix.
`boot.l`	If set to a value larger than 1 the data will be blocked with blocklength `boot.l` before the covariance matrix is computed.
`boot.samples`	If set to `TRUE` the data is treated a pseudo data from a bootstrap procedure.
`cov_fn`	Function that computes the covariance matrix from the given samples.

Details

The inverse covariance matrix is estimated. If the number of observations is too small the procedure described in the reference is used to remove spuriously small eigenvalues of the covariance matrix. We always keep the $\sqrt{R}$ largest eigenvalues exactly and replace the remaining smallest ones by their mean.

Value

Returns the inverse covariance matrix as an object of class `matrix`.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

References

C.Michael, A.McKerrell, Phys.Rev. D51 (1995) 3745-3750, hep-lat/9412087

See Also

`cov`, `matrix`

Examples

```
X <- array(rnorm(4000), dim=c(1000, 4))
invertCovMatrix(cf=X, boot.samples=TRUE)
M <- invertCovMatrix(cf=X, boot.samples=TRUE)
M
```

`is.cf` *Checks whether an object is a cf*

Description

Checks whether an object is a cf

Usage

```
is.cf(x)
```

Arguments

`x`	Object, possibly of class `cf`.

Value

Returns TRUE if the input object is of class `cf`, FALSE otherwise.

`is.raw_cf` *check if an object is of class* `raw_cf`

Description

check if an object is of class `raw_cf`

Usage

```
is.raw_cf(x)
```

Arguments

`x`	object to be checked

Value

Returns TRUE if `x` is an object of class `raw_cf`, FALSE otherwise.

`is_empty.cf` *Checks whether the cf object contains no data*

Description

Checks whether the cf object contains no data

Usage

```
is_empty.cf(.cf)
```

Arguments

`.cf`	`cf` object.

Value

returns `FALSE` if `.cf` contains no data, `TRUE` otherwise

Examples

```
# The empty cf object must be empty:
is_empty.cf(cf())

# The sample cf must not be empty:
is_empty.cf(samplecf)
```

`is_empty.raw_cf` *check if an obect is of class* `raw_cf` *and empty otherwise*

Description

check if an obect is of class `raw_cf` and empty otherwise

Usage

```
is_empty.raw_cf(x)
```

Arguments

`x`	object to be checked

Value

Returns TRUE if `x` is an empty object of class `raw_cf`, FALSE otherwise.

jackknife.cf *jackknife a set of correlation functions*

Description

jackknife a set of correlation functions

Usage

```
jackknife.cf(cf, boot.l = 1)
```

Arguments

`cf`	correlation matrix of class `cf` e.g. obtained with a call to `extrac.obs`.
`boot.l`	block size for autocorrelation analysis

Value

returns an object of class `cf` with blocked jackknife samples added for the correlation function called `cf.jackknife`. Currently, only the moving block jackknife approach is implemented. Moreover, the original average of `cf` is returned as `cf0` and the bootstrap errors as `jackknife.se`. We also copy the input parameters over and set `jackknife.samples` to `TRUE`.

Author(s)

Carsten Urbach, <curbach@gmx.de>

References

H.R. Künsch, "The jackknife and the bootstrap for general stationary observations", The Annals of Statistics, 1989, Vol. 17, No. 3, 1217-1241

S.N. Lahiri, "On the jackknife-after-bootstrap method for dependent data and its consistency properties", Econometric Theory, 2002, Vol. 18, 79-98

See Also

`boot::tsboot`, `bootstrap.cf`

Examples

```
data(samplecf)
samplecf <- jackknife.cf(samplecf, boot.l=1)
plot(samplecf, log="y")
```

`jackknife_cov` *jackknife_cov*

Description

Computes covariance matrix for jackknife samples.

Usage

```
jackknife_cov(x, y = NULL, na.rm = FALSE, ...)
```

Arguments

`x`	a numeric vector, matrix or data frame.
`y`	'NULL' (default) or a vector, matrix or data frame with compatible dimensions to 'x'. The default is equivalent to 'y = x' (but more efficient).
`na.rm`	logical. The rows containing any `NA` will be deleted if this option is set.
`...`	parameters to be forwarded to cov.

Value

returns a matrix corresponding to the jackknife estimate of the covariance matrix

`jackknife_error` *Estimates error from jackknife samples*

Description

Computes the jackknife error which is just

$$\sum_{i=0}^{N} (x_i - \bar{x})^2 \,.$$

Internally we use

$$\frac{(N-1)^2}{N} \, \mathrm{sd}(X)$$

in order to benefit from the optimized standard deviation function.

The width of the bootstrap distribution does not change with the number of elements. The jackknife distribution crucially depends on the number of measurements that one started with. Therefore we cannot just drop the NA values and are done with it. Instead we need to rescale with the $\sqrt{N/m}$ where N is the number of original measurements and m is the number of non-NA values. With NA values removed we would otherwise underestimate the uncertainty.

Usage

```
jackknife_error(samples, boot.l = 1, na.rm = FALSE)
```

Arguments

`samples`	Numeric vector.
`boot.l`	Block length for bootstrapping.
`na.rm`	Logical. Determines whether `NA` values shall be removed, see Description for details.

Details

Currently this uses the mean over the jackknife samples in order to compute the error. It would be better in the case of a bias to use the mean over the original data instead. This would require a second parameter and therefore is incompatible with the previously used `sd` everywhere for the bootstrap samples. As the `sd` for the bootstrap samples also does not include the original data, this likely is similar in terms of bias.

Value

returns a single numeric representing the jackknife estimate of error

---

`loopdata` *Sample loop data*

Description

Sample data for fermion loops for a 24 cube times 48 lattice QCD simulation. It is stored in form of a list.

Format

list of 10 elements: "nrObs" "Time" "nrStypes" "symmetrised" "cf" "icf" "scf" "iscf" "nrSamples" "obs"

Examples

```
data("loopdata")
```

`loop_2pt` *compute two-point correlation function between quark loops*

Description

compute two-point correlation function between quark loops

Usage

```
loop_2pt(loop_snk, loop_src, random_vectors_outer_product = FALSE,
  nstoch_to_avg = "all")
```

Arguments

`loop_snk`	'raw_cf' container with spin-projected quark loop at sink (see loop_spin_project)
`loop_src`	'raw_cf' container with spin-projected quark loop at source (see loop_spin_project)
`random_vectors_outer_product`	Boolean. For testing purposes, the average over all random sample combinations can be carried out explicitly as $\sum_{i \neq j} Tr[\Gamma_s nk M_i] Tr[\Gamma_s rc M_j]$ instead of the (much faster) equivalent $(\sum_i Tr[\Gamma_s nk M_i]) * (\sum_j Tr[\Gamma_s rc M_j]) - \sum_i (Tr[\Gamma_s nk M_i] Tr[\Gamma_s rc M_i])$.
`nstoch_to_avg`	String or integer, how many of the available stochastic samples should be averaged over. See loop_stochav for details.

Value

'raw_cf' container with two-point function of these two quark loops. In the calculation, both averaging over all source locations and the average over all stochastic sample combinations are performed.

`loop_spin_project` *spin projection of quark loop data*

Description

Implements the operation

$$L = a * (\Gamma_{ik} M_{ki})$$

to give the trace of a quark loop M multiplied by a gamma structure Γ and scaled by a complex factor a.

Usage

```
loop_spin_project(loop, gamma, reim = "both", stochav = FALSE,
  scale_factor = as.complex(1), herm_conj = FALSE)
```

Arguments

`loop`	'raw_cf' container with loop data
`gamma`	4x4 complex matrix
`reim`	String, one of 'real', 'imag' or 'both'. After the spin projection and trace, the result can be restricted to just the real or imaginary part, if desired. Useful for the cases in which it is clear that only one or the other contains any signal.
`stochav`	Boolean, specifies whether the average over stochastic samples should be performed. This makes the projection much faster but of course prevents the projected loop data to be used for the construction of diagrams with multiple quark loops.
`scale_factor`	Complex scaling factor to be applied.
`herm_conj`	Boolean, optionally the loop matrix M can be hermitian conjugated before the spin projection is performed.

Value

Returns an object of class raw_cf.

`loop_stochav` *average over stochastic samples of loop*

Description

Perform mean over the third dimension of the loop data.

Usage

```
loop_stochav(loop, nstoch_to_avg = "all")
```

Arguments

`loop`	'raw_cf' container with loop data
`nstoch_to_avg`	String or integer, number of stochastic samples to average over. Only possible string is 'all'. If an integer is supplied, it must be at least '1' and at most consistent with the number of stochastic samples in `loop`.

Value

Returns the input `loop` object with named elemens `data` and `dim` added.

`loop_vev_subtract` *subtract vev from loop data*

Description

Convenience function to subtract any possible vacuum-expectation value from a loop matrix. The expectation value of each component of the internal dimensions is subtracted individually. Averaging over stochstic samples can be restricted to a subset, see `nstoch_to_avg` input parameter.

Usage

```
loop_vev_subtract(loop, nstoch_to_avg = "all")
```

Arguments

`loop`	'raw_cf' container with loop data
`nstoch_to_avg`	String or integer, number of stochastic samples to average over. Only possible string is 'all'. If an integer is provided it must be at least '1' and at most consistent with the number of stochastic samples in `loop`.

Value

Returns the input `loop` object with added data.

`make_parind` *Create a parameter index matrix for* `matrixfit`

Description

Create a parameter index matrix for `matrixfit`

Usage

```
make_parind(parlist, length_time, summands = 1)
```

Arguments

`parlist`	integer array. Parameter list generated with `make_parlist`.
`length_time`	integer. Number of time slices per correlator.
`summands`	integer. Number of summands in the fit model that shall be fitted. The signal counts as one summand, each explicit pollution term with independent amplitudes counts as its own summand.

Value

Returns an array with the parameter indices.

make_parlist *Create a parameter list for* `matrixfit`

Description

Create a parameter list for `matrixfit`

Usage

```
make_parlist(corr_matrix_size)
```

Arguments

`corr_matrix_size`	integer. Number of correlators in the matrix. This must be a the square of an integer.

Value

Returns a square, integer-valued matrix.

matrixfit *Routine For A Factorising Matrix Fit*

Description

Performs a factorising fit on a correlation matrix

Usage

```
matrixfit(cf, t1, t2, parlist, sym.vec, neg.vec, useCov = FALSE,
  model = "single", boot.fit = TRUE, fit.method = "optim",
  autoproceed = FALSE, every)
```

Arguments

`cf`	correlation matrix obtained with a call to `extrac.obs`.
`t1`	lower bound for the fitrange in time (t1,t2). Counting starts with 0.
`t2`	upper bound for the fitrange in time (t1,t2). Counting starts with 0.
`parlist`	a two dimensional array of dimension 2 times number of correlators in cf. Every column assigns a pair of fit parameters to the corresponding correlator in cf. In case this is missing there are defaults provided for certain matrix sizes.

`sym.vec`	a vector of length number of correlators in cf indicating whether the correlation function is a cosh, a sinh or an exponential. Possible values are `"cosh"`, `"sinh"` and `"exp"`. In case this is missing there are defaults provided for certain matrix sizes.
`neg.vec`	a vector of length number of correlators in cf indicating whether the correlation function is to be multiplied globally with a minus sign. In case this is missing there are defaults provided for certain matrix sizes.
`useCov`	use correlated or uncorrelated chisquare. Default is `useCov=FALSE`.
`model`	Sets the fit model to be used in the fit. The default model is $0.5 p_i p_j (\exp(-Et) \pm c * \exp(-E(Time - t)))$ with sign depending on `"cosh"` or `"sinh"`. `c` equals one except for the `"exp"` functional dependence. When model is set to `"shifted"`, the fit uses the function $p_i p_j (\exp(-E(t + 1/2)) \mp c * \exp(-E(Time - (t + 1/2))))$ which is useful when the original correlation function or matrix is shifted, see e.g. bootstrap.gevp. In case only a single principal correlator from a GEVP is to be fitted the additional model `"pc"` is available. It implements $\exp(-E(t - t_0))(A + (1 - A) \exp(-DeltaE(t - t_0))$ with t_0 the reference timesclice of the GEVP. See bootstrap.gevp for details.
`boot.fit`	If set to `FALSE`, the fit is not bootstrapped, even if the bootstrapping parameters have been set and the correlation function has been bootstrapped. This is a useful time-saver if error information is not strictly necessary. Of course, this affects the return values related to the bootstrap, which are set to `NA`.
`fit.method`	Can be either `"optim"` or `"lm"`. The latter works only if the library `"minpack.lm"` can be loaded. Default and fallback is `"optim"`.
`autoproceed`	When the inversion of the variance-covariance matrix fails, the default behaviour is to abort the fit. Setting this to `TRUE` means that the fit is instead continued with a diagonal inverse of the variance-covariance matrix.
`every`	Fit only a part of the data points. Indices that are not multiples of `every` are skipped. If no value is provided, all points are taken into account.

Details

The routine expects in `cf$cf` a set of correlation functions. The mapping of this linear construct to a matrix or a part of a matrix is achieved via `parlist`. The symmetry properties of the individual correlation functions must be encoded in `sym.vec`.

`matrixfit` will fit to every correlator in `cf$cf` a function $p_i p_j f(t)$. The indices i, j are determined from `parlist` and f is either $cosh$ or $sinh$, depending on `sym.vec`.

The inverse covariance matrix is computed using a singular value decomposition. If the sample size N is too small, only sqrt(N) eigenvalues of the matrix are kept exactly, while all others are replaced by the mean of the rest. This helps to reduce instabilities induced by too small eigenvalues of the covariance matrix.

Value

returns an object of class `matrixfit` with entries:

`CF`	object of class cf which contains the mean correlation functions
`M`	inverse variance-covariance matrix for weighted Chi squared minimization
`L`	squre root of `M`.
`parind`	indices in the parameter vector used for the different matrix combinations
`sign.vec`	vector of signs
`ii`	vector of vector indices giving the columns of the correlation function arrays (CF above, say), which are contained in the fit range
`opt.res`	return value of the minimization (see ?optim) on the original data.
`t0`	Result of the chisqr fit on the original data. `t0` is a vector of length npar+1, where `npar` the number of fit parameters. The last value is the chisqr value.
`t`	Bootstrap samples of the `R` Chi squared minimizations of length(par)+1. `t` has dimension $Rx(npar+1)$, where `R` is the number of bootstrap samples and `npar` the number of fit parameters. The last column corresponds to the chisquare values.
`se`	Bootstrap estimate of standard error for all parameters. `se` is a vector of length `npar`, where `npar` the number of fit parameters.
`useCov`	whether covariances in the data were taken into account
`invCovMatrix`	inverse of covariance matrix or inverse variance weighted if useCov=FALSE
`Qval`	real number between 0 and 1 giving the "quality" of the fit
`chisqr`	total Chi squared of the fit
`dof`	fit degrees of freedom
`mSize`	integer size of the matrix which was fitted
`cf`	object of type cf which contains, amongst other objects, cf$cf which is a concatenated array of raw correlation functions where each row is one of N observations and there are mSize*Time columns (see ?extract.obs)
`boot.R`	number of bootstrap samples
`boot.l`	block size for blocked bootstrap
`t1`	beginning of fit range
`t2`	end of fit range
`parlist`	array of parameter combinations for the matrix fit
`sym.vec`	vector of strings indicating the functional form of correlation functions which were fitted
`seed`	RNG seed for bootstrap procedure
`model`	see input.
`fit.method`	see input.
`reference_time`	The GEVP reference time for the principal correlator model

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

References

C. Michael, hep-lat/9412087hep-lat/9412087

See Also

`cf`, `bootstrap.cf`

Examples

```
data(samplecf)
samplecf <- bootstrap.cf(cf=samplecf, boot.R=99, boot.l=2, seed=1442556)
fitres <- matrixfit(cf=samplecf, t1=16, t2=24, useCov=FALSE,
                    parlist=array(c(1,1), dim=c(2,1)),
                    sym.vec=c("cosh"), fit.method="lm")
summary(fitres)
plot(fitres)
```

`matrixModel` *Correlator matrix model.*

Description

Correlator matrix model.

Usage

```
matrixModel(par, t, Time, parind, sign.vec, ov.sign.vec, deltat = 0)
```

Arguments

`par`	Numeric vector: Fit parameters of the model. In an object of type `matrixfit`, this should be located at `$opt.res$par`.
`t`	integer vector: Time of interest.
`Time`	integer: Time extent of the lattice.
`parind`	See `matrixfit`.
`sign.vec`	Numeric vector: Relative sign between forward and backwards propagating part. A plus makes it cosh, a minus makes it sinh.
`ov.sign.vec`	Numeric vector: Overal sign.
`deltat`	Numeric: time shift.

Value

Returns a numeric vector with the same length as the input vector `t` containing the model evaluation for these t-values.

See Also

`matrixfit`

`mom_combinations` *Generate table of momentum component combinations*

Description

Generate table of momentum component combinations

Usage

```
mom_combinations(psqmax)
```

Arguments

`psqmax`	Integer, maximum p^2 = px^2 + py^2 + pz^2 to be included in momentum list

Value

Returns a data.frame with all possible momentum combinations.

`mul.cf` *Arithmetically scale a correlator by a scalar a*

Description

Note that this function is fundamentally different from `*.cf`.

Usage

```
mul.cf(cf, a = 1)
```

Arguments

`cf`	`cf_orig` objects.
`a`	Numeric, scaling factor.

Value

Returns an object of class `cf`.

`mul.raw_cf` *scale* `raw_cf` *data*

Description

scale `raw_cf` data

Usage

```
mul.raw_cf(cf, a = 1)
```

Arguments

`cf`	'raw_cf' container with data to be scaled by the factor `a`
`a`	Numeric or complex scaling factor, although it could also be an array of dimensions compatible with `cf$data`

Value

`raw_cf` object with `res$data` == `a*cf$data`

`new_matrixfit` *perform a factorising fit of a matrix of correlation functions*

Description

Modernised and extended implementation of matrixfit

Usage

```
new_matrixfit(cf, t1, t2, parlist, sym.vec = rep(1, cf$nrObs),
  neg.vec = rep("cosh", cf$nrObs), useCov = FALSE, model = "single",
  boot.fit = TRUE, fit.method = "optim", autoproceed = FALSE, par.guess,
  every, higher_states = list(val = numeric(0), boot = matrix(nrow = 0, ncol
  = 0), ampl = numeric(0)), ...)
```

Arguments

`cf`	Object of class `cf` with `cf_meta` and `cf_boot`.
`t1`	Integer, start time slice of fit range (inclusive).
`t2`	Integer, end time slie of fit range (inclusive).
`parlist`	Numeric vector, list of parameters for the model function.
`sym.vec`	Integer, numeric or vectors thereof specifying the symmetry properties of the correlation functions stored in `cf`. See matrixfit for details.

`neg.vec`	Integer or integer vector of global signs, see matrixfit for details.
`useCov`	Boolean, specifies whether a correlated chi^2 fit should be performed.
`model`	String, specifies the type of model to be assumed for the correlator. See below for details.
`boot.fit`	Boolean, specifies if the fit should be bootstrapped.
`fit.method`	String, specifies which minimizer should be used. See matrixfit for details.
`autoproceed`	Boolean, if TRUE, specifies that if inversion of the covariance matrix fails, the function should proceed anyway assuming no correlation (diagonal covariance matrix).
`par.guess`	Numeric vector, initial values for the paramters, should be of the same length as `parlist`.
`every`	Integer, specifies a stride length by which the fit range should be sparsened, using just `every`th time slice in the fit.
`higher_states`	List with elements `val` and `boot`. Only used in the `n_particles` fit model. The member `val` must have the central energy values for all the states that are to be fitted. The `boot` member will be a matrix that has the various states as columns and the corresponding bootstrap samples as rows. The length of `val` must be the column number of `boot`. The row number of `boot` must be the number of samples.
`...`	Further parameters.

Details

There are different fit models available. The models generally depend on one or multiple energies E and amplitudes p_i which for a general matrix are row- and column-amplitudes. The relative sign factor $c \in \{-1, 0, +1\}$ depends on the chosen symmetry of the correlator. It is a plus for a "cosh" symmetry and a minus for a "sinh" symmetry. If the back propagating part is to be neglected (just "exp" model), it will be zero.

When the back propagating part is not taken into account, then the `single`, `shifted` and `weighted` model become the same except for changes in the amplitude.

- `single`: The default model for a single state correlator is
$$\frac{1}{2} p_i p_j (\exp(-p_1 t) \pm c \exp(-p_2(T-t)))\,.$$
- `shifted`: If the correlator has been shifted (using takeTimeDiff.cf, then the following model is applicable:
$$p_i p_j (\exp(-p_1(t+1/2)) \mp c \exp(-p_1(T-(t+1/2))))\,.$$
- `weighted`: Works similarly to the `shifted` model but includes the effect of the weight factor from removeTemporal.cf.
- `pc`: In case only a single principal correlator from a GEVP is to be fitted this model can be used. It implements
$$\exp(-p_1(t-t_0))(p_2 + (1-p_2)\exp(-p_3(t-t_0)))$$
with t_0 the reference timesclice of the GEVP. See bootstrap.gevp for details.

- `two_amplitudes`: Should there be a single state but different amplitudes in the forward and backwards part, the following method is applicable.

  $$\frac{1}{2}(p_2 \exp(-p_1 t) \pm c p_3 \exp(p_1 t))$$

  This only works with a single correlator at the moment.
- `single_constant`: Uses the `single` model and simply adds $+p_3$ to the model such that a constant offset can be fitted. In total the model is

  $$\mathrm{single}(p_1, p_2) + p_3\,.$$

- `n_particles`: A sum of `single` models with independent energies and amplitudes:

  $$\sum_{i=1}^{n} \mathrm{single}(p_{2n-1}, p_{2n})\,.$$

  Use the `higher_states` parameter to restrict the thermal states with priors to stabilize the fit.

Value

See bootstrap.nlsfit.

`old_removeTemporal.cf` *Remove temporal states*

Description

Performs weighting and shifting in the rest and moving frames.

Usage

```
old_removeTemporal.cf(cf, single.cf1, single.cf2, p1 = c(0, 0, 0),
  p2 = c(0, 0, 0), L, lat.disp = TRUE, weight.cosh = FALSE, deltat = 1)
```

Arguments

`cf`	Object of type `cf`, two-to-two particle correlation function which shall be weighted and shifted. It must be a correlation function in the frame $p_1 + p_2$.
`single.cf1, single.cf2`	Object of type `effectivemassfit` or `matrixfit` which contains the one particle mass in the rest frame. If `single.cf2` is missing, then the mass given as `single.cf1` is used as well. This is sensibly done when one scatters identical particles. But be careful: Even when `single.cf2` is missing, the `p2` is *not* automatically copied from `p1`. In case `single.cf1` is missing, no weighting is performed. Instead it is assumed that the user only wants to have a simple shifting. Then this function just calls `takeTimeDiff.cf`.

`p1, p2`	Integer vector with three elements, containing the momenta that the one particle mass should be boosted to.
`L`	Integer, spatial extent of the lattice.
`lat.disp`	Logical, true when the lattice dispersion relation shall be used, otherwise continuum dispersion relation.
`weight.cosh`	Logical, If single.cf1 is a pure cosh, the leading two thermal states also may be expressed as a cosh. If `weight.cosh` is set, they are removed simultaneously.
`deltat`	Integer. Time shift value.

Value

Returns an object of class `cf`, see cf.

`onlinemeas` *determines pion mass and pcac mass from online measured correlator of the HMC code*

Description

determines pion mass and pcac mass from online measured correlator of the HMC code

Usage

```
onlinemeas(data, t1, t2, stat_range, S = 1.5, pl = FALSE, skip = 0,
  iobs = 1, ind.vec = c(1, 3, 4, 5), mu = 0.1, kappa = 0.125,
  boot.R = 99, boot.l = 10, tsboot.sim = "geom", method = "uwerr",
  fit.routine = "optim", nrep, oldnorm = FALSE)
```

Arguments

`data`	data to be fitted to as e.g. the output of `readcmicor`. Currently only `cmicor` format is supported.
`t1`	lower bound for the fitrange in time (t1,t2). Counting starts with 0.
`t2`	upper bound for the fitrange in time (t1,t2). Counting starts with 0.
`stat_range`	range of data to be included in the analysis.
`S`	passed to `uwerr`, see documentation of `uwerr`.
`pl`	logical: if set to TRUE the function produces plots
`skip`	number of measurements to be discarded at the beginning of the time series. `skip` has no effect if two or more replica are used, see argument `nrep`.
`iobs`	if there are several operators available (local-local, local-smeared, etc.), then this labels these (for cmi format)
`ind.vec`	index vector indexing the column numbers in cmicor to be used
`mu`	twisted mass parameter.

`kappa`	hopping parameter.
`boot.R`	number of bootstrap samples for bootstrap analysis
`boot.l`	average block size for blocking analysis with tsboot
`tsboot.sim`	The type of simulation required to generate the replicate time series. See `tsboot` for details.
`method`	the type of error analysis to be used. Can be either "uwerr", "boot", "all" or "no". For "no" (or any other string) no error analysis is performed. This might be helpful for a first impression and also to test different initial values for the fitting parameters. The latter is in particular needed for more than one state in the fit.
`fit.routine`	The fit routine to be used. Default is "gsl", which uses the gnu scientific library "gsl_multifit_fdfsolver" solver to minimise the chisquare. All other values lead to the usage of R's optim function. The latter choice might be significantly slower.
`nrep`	vector (N1, N2, ...) of replica length N1, N2. If missing it is assumed that there is only one ensemble. If there are two or more replica the parameter `skip` has no effect.
`oldnorm`	If set to "TRUE", the old online measurement normalisation of "tmLQCD" prior to version 5.2.0 is used in order to get correct values for the pion decay constant.

Details

The online measurements in the HMC code compute the PP and PA correlation functions summed over spatial x for all t. We analyse these correlators in different ways:

First, only the PP correlator is analysed and fitted by $p_1^2\cosh(-m(t-T/2))$ for m and p_1.

Second, PP and PA correlators are fitted together with three parameters as $C_{\mathrm{PP}} = p_1^2\cosh(-m(t-T/2))$ and $C_{\mathrm{PA}} = p_1 p_2\cosh(-m(t-T/2))$ in a simultaneous fit. m is then the pseudo scalar mass and the pcac mass is determined from

$$m_{\mathrm{PCAC}} = m_{\mathrm{PS}}\frac{p_2}{2p_1}$$

Finally, the PCAC mass can also be determined computing

$$m_{\mathrm{PCAC}}(t) = \frac{C_{\mathrm{PA}}(t+1) - C_{\mathrm{PA}}(t-1)}{4C_{\mathrm{PP}}(t)}$$

using the symmetric finite difference operator.

Value

returns an object of `class ofit` with the following items

`fitresult`	result from the fit as returned by `optim`
`fitresultpp`	Fit result of the PP correlator only
`t1`	lower bound for the fitrange in time (t1,t2). Counting starts with 0.
`t2`	upper bound for the fitrange in time (t1,t2). Counting starts with 0.

`N`	number of measurements found in the data
`Time`	Time extent found in the data
`fitdata`	`data.frame` containing the time values used in the fit, the averaged correlator and its error and the value of Chi for each time value
`uwerrresultmps`	the result of the time series analysis for the lowest mass as carried out by `uwerr`
`uwerrresultmpcac`	the result of the time series analysis for the PCAC mass carried out by `uwerr`, see details
`effmass`	effective masses in the pion channel
`matrix.size`	size of the data matrix, copied from input
`boot`	object returned by the call to `boot` if `method` was set correspodingly. Otherwise `NULL`.
`tsboot`	object returned by the call to `tsboot` if `method` was set correspodingly. Otherwise `NULL`.
`method`	error analysis method as copied from input
`fit.routine`	`fit.routine` as copied from input
`nrep`	`nrep` as copied from input
`dpaopp`	`data.frame` containing the pcac masses computed not with a fit, but with the derivative method for all time values in between `t1` and `t2`

Author(s)

Carsten Urbach, <curbach@gmx.de>

See Also

`readcmicor`, `uwerr`,

overview_plot_raw_cf *create convenient overview plots for a* raw_cf *object*

Description

create convenient overview plots for a `raw_cf` object

Usage

```
overview_plot_raw_cf(cf, grid, reim = "real", reim_same = FALSE,
  relerr = FALSE, tauint = FALSE, value_logplot = TRUE,
  value_factor = c(1), title = "")
```

Arguments

`cf`	'raw_cf' container with data and meta-data
`grid`	Optional, integer vector which satisfies `prod(grid) == prod(cf$dim)`. This is passed to `par` via `par(mfrow=grid)` to produce a grid of plots as defined by the components of `grid`.
`reim`	Vector of strings, one of 'real', 'imag' or 'both'. Specified whether the real or imaginary parts (or both) should be plotted.
`reim_same`	Boolean, whether real and imaginary parts should be plotted on the same plot. If `TRUE`, then `reim` must be 'both'. If this is given, the imaginary part as well as its relative error and per-time-slice integrated autocorreation times
`relerr`	Boolean, whether a plot of the relative error per time slice should be added.
`tauint`	Boolean, whether a plot of the integrated auto-correlation time on each time slice should be added.
`value_logplot`	Boolean, whether the plot of the correlator should be on a logarithmic vertical axis. (does not affect `tauint` and `relerr`).
`value_factor`	Numeric, either of length '1' or as long as the number of correlation functions in `cf`. The data will be scaled by this factor before plotting.
`title`	Character vector, will be passed as the `main` argument to plotwitherror which in turn passes it to plot. Can be either of length '1' or `prod(cf$dim)`

Value

No return value, only plots are generated.

`parametric.bootstrap` *Parametric bootstrap*

Description

Parametric bootstrap

Usage

```
parametric.bootstrap(boot.R, x, dx, seed)
```

Arguments

`boot.R`	numeric. Number of bootstrap samples to generate.
`x`	numeric vector. Actual values for the data.
`dx`	numeric vector of the same length as `x` or missing. Errors of the values.
`seed`	integer. Seed to use for the random number generation. If it is missing, the seed will not be set to any particular value. If there was a default value, all results would be exactly correlated. So if you want reproducibility by fixing the seeds, make sure you choose different seeds for independent variables.

Value

A matrix with as many columns as there are variables in `x` and as many rows as `boot.R`.

See Also

Other NLS fit functions: `bootstrap.nlsfit()`, `parametric.bootstrap.cov()`, `parametric.nlsfit.cov()`, `parametric.nlsfit()`, `plot.bootstrapfit()`, `predict.bootstrapfit()`, `print.bootstrapfit()`, `simple.nlsfit()`, `summary.bootstrapfit()`

Examples

```
x <- 1:3
dx <- 1:3 * 0.1
parametric.bootstrap(5, x, dx)
```

`parametric.bootstrap.cov`   *Parametric bootstrap with covariance*

Description

Parametric bootstrap with covariance

Usage

```
parametric.bootstrap.cov(boot.R, x, cov, seed)
```

Arguments

`boot.R`	numeric. Number of bootstrap samples to generate.
`x`	numeric vector. Actual values for the data.
`cov`	numeric matrix, square, length of `x` or missing. Covariance between the various variables in the vector `x`.
`seed`	integer. Seed to use for the random number generation. If it is missing, the seed will not be set to any particular value. If there was a default value, all results would be exactly correlated. So if you want reproducability by fixing the seeds, make sure you choose different seeds for independent variables.

Value

A matrix with as many columns as there are variables in `x` and as many rows as `boot.R`.

See Also

Other NLS fit functions: `bootstrap.nlsfit()`, `parametric.bootstrap()`, `parametric.nlsfit.cov()`, `parametric.nlsfit()`, `plot.bootstrapfit()`, `predict.bootstrapfit()`, `print.bootstrapfit()`, `simple.nlsfit()`, `summary.bootstrapfit()`

Examples

```
x <- 1:3
cov <- matrix(c(0.1, 0, 0.01,
              0, 0.15, 0.02,
              0.01, 0.02, 0.2), nrow = 3)
parametric.bootstrap.cov(5, x, cov)
```

`parametric.nlsfit` *NLS fit with parametric bootstrap*

Description

NLS fit with parametric bootstrap

Usage

```
parametric.nlsfit(fn, par.guess, boot.R, y, dy, x, dx, lower = rep(x = -Inf,
  times = length(par.guess)), upper = rep(x = +Inf, times =
  length(par.guess)), ..., bootstrap = TRUE)
```

Arguments

`fn`	`fn(par,x,...)`. The (non-linear) function to be fitted to the data. Its first argument must be the fit parameters named `par`. The second must be `x`, the explaining variable. Additional parameters might be passed to the function. Currently we pass `boot.r` which is `0` for the original data and the ID (1, ...) of the bootstrap sample otherwise. As more parameters might be added in the future it is recommended that the fit function accepts `...` as the last parameter to be forward compatible.
`par.guess`	initial guess values for the fit parameters.
`boot.R`	numeric. Number of bootstrap samples to generate.
`y`	the data as a one-dimensional numerical vector to be described by the fit function.
`dy`	Numeric vector. Errors of the dependent and independent variable, respectively. These do not need to be specified as they can be computed from the bootstrap samples. In the case of parametric bootstrap it might would lead to a loss of information if they were computed from the pseudo-bootstrap samples. They must not be specified if a covariance matrix is given.
`x`	values of the explaining variable in form of a one-dimensional numerical vector.
`dx`	Numeric vector. Errors of the dependent and independent variable, respectively. These do not need to be specified as they can be computed from the bootstrap samples. In the case of parametric bootstrap it might would lead to a loss of information if they were computed from the pseudo-bootstrap samples. They must not be specified if a covariance matrix is given.

`lower`	Numeric vector of length `length(par.guess)` of lower bounds on the fit parameters. If missing, `-Inf` will be set for all.
`upper`	Numeric vector of length `length(par.guess)` of upper bounds on the fit parameters. If missing, `+Inf` will be set for all.
`...`	Additional parameters passed to `fn`, `gr` and `dfn`.
`bootstrap`	Shall the error calculation be performed using boostrap? If not, the errors are estimated with help of the jacobian (either provided in `gr` or calculated using the `numDeriv`-package).

Value

See simple.nlsfit.

See Also

Other NLS fit functions: `bootstrap.nlsfit()`, `parametric.bootstrap.cov()`, `parametric.bootstrap()`, `parametric.nlsfit.cov()`, `plot.bootstrapfit()`, `predict.bootstrapfit()`, `print.bootstrapfit()`, `simple.nlsfit()`, `summary.bootstrapfit()`

Examples

```
## Declare some data.
value <- c(0.1, 0.2, 0.3)
dvalue <- c(0.01, 0.01, 0.015)
x <- c(1, 2, 3)
dx <- c(0.1, 0.1, 0.1)
boot.R <- 1500

fn <- function (par, x, ...) par[1] + par[2] * x

fit.result <- parametric.nlsfit(fn, c(1, 1), boot.R, value, dvalue, x, dx)
summary(fit.result)
```

`parametric.nlsfit.cov` *parametric.nlsfit.cov*

Description

NLS fit with parametric bootstrap and covariance

Usage

```
parametric.nlsfit.cov(fn, par.guess, boot.R, y, x, cov, lower = rep(x = -Inf,
  times = length(par.guess)), upper = rep(x = +Inf, times =
  length(par.guess)), ..., bootstrap = TRUE, na.rm = FALSE)
```

Arguments

`fn`	`fn(par,x,...)`. The (non-linear) function to be fitted to the data. Its first argument must be the fit parameters named `par`. The second must be `x`, the explaining variable. Additional parameters might be passed to the function. Currently we pass `boot.r` which is `0` for the original data and the ID (1, ...) of the bootstrap sample otherwise. As more parameters might be added in the future it is recommended that the fit function accepts `...` as the last parameter to be forward compatible.
`par.guess`	initial guess values for the fit parameters.
`boot.R`	numeric. Number of bootstrap samples to generate.
`y`	the data as a one-dimensional numerical vector to be described by the fit function.
`x`	values of the explaining variable in form of a one-dimensional numerical vector.
`cov`	numeric matrix, square, length of `x` or missing. Covariance between the various variables in the vector `x`.
`lower`	Numeric vector of length `length(par.guess)` of lower bounds on the fit parameters. If missing, `-Inf` will be set for all.
`upper`	Numeric vector of length `length(par.guess)` of upper bounds on the fit parameters. If missing, `+Inf` will be set for all.
`...`	Additional parameters passed to `fn`, `gr` and `dfn`.
`bootstrap`	boolean. If `TRUE`, bootstrap is used.
`na.rm`	logical. If set to `true`, NAs in `y` and `dy` will be ignored. If `x`-errors are taken into account, NAs in `x` and `dx` will be ignored, too.

Value

See simple.nlsfit.

See Also

Other NLS fit functions: `bootstrap.nlsfit()`, `parametric.bootstrap.cov()`, `parametric.bootstrap()`, `parametric.nlsfit()`, `plot.bootstrapfit()`, `predict.bootstrapfit()`, `print.bootstrapfit()`, `simple.nlsfit()`, `summary.bootstrapfit()`

`pcac` *Computes the pcac mass*

Description

Computes the pcac mass from the PP and the AP (PA) correlators and estimates the errors using the gamma method

Usage

```
pcac(psfilename, apfilename, pafilename, from = 3, to = 3, fit = F,
  skip = 0, plotit = F, S = 1.5)
```

Arguments

`psfilename`	filename of the file from which to read the PP correlator. It is supposed to be in GWC code format. mandatory.
`apfilename`	filename of the file from which to read the AP correlator. It is supposed to be in GWC code format. Either PA or AP correlator (or both) must be given. If both are given, both are used by averaging.
`pafilename`	filename of the file from which to read the PA correlator. It is supposed to be in GWC code format. Either PA or AP correlator (or both) must be given. If both are given, both are used by averaging.
`from`	the effective mass is computed starting with t=from
`to`	the effective mass is computed ending with t=to
`fit`	logical. if TRUE a fit is performed to all t-values to determine the pcac mass.
`skip`	no of measurements to skip at the beginning of the file
`plotit`	logical. if TRUE a plot is drawn.
`S`	passed to `uwerr`, see documentation of `uwerr`.

Details

the symmetric difference operator is used.

Value

returns a data.frame with the results. The object is also of class `massfit` which can be plotted using the generic function `plot`.

Author(s)

Carsten Urbach, <carsten.urbach@liverpool.ac.uk>

`pcacfit` *pcacfit*

Description

Computes the average PCAC mass

Usage

```
pcacfit(data, from, to, T2, pa = FALSE)
```

Arguments

`data`	'Effectivemasses' from correlators
`from`	initial value of fit range
`to`	final value of fit range
`T2`	Time extent
`pa`	Boolean.

Value

Single numeric value, the mass.

`pcModel` *Principal correlator two state model.*

Description

Principal correlator two state model.

Usage

```
pcModel(par, t, Time, delta1 = 1, reference_time)
```

Arguments

`par`	Numeric vector: Fit parameters of the model. In an object of type `matrixfit`, this should be located at `$opt.res$par`.
`t`	Numeric vector: Time of interest.
`Time`	Numeric: Time extent of the lattice.
`delta1`	dummy parameter for compatibility
`reference_time`	Numeric: GEVP reference time value in physical time convention

Value

Returns a numeric vector with the same length as the input vector `t` containing the model evaluation for these t-values.

See Also

`matrixfit`

`plaq.sample` *Sample plaquette time series*

Description

A time series of so-called plaquette values generated by a Markov Chain MC process using the Hybrid Monte-Carlo algorithm. Plaquettes are the smallest possible closed loops which can be build in lattice QCD in discretised Euclidean space time.

Format

The format is: num 0.583 0.582 0.582 0.582 0.582 ...

Examples

```
data(plaq.sample)
plot(x=c(1:length(plaq.sample)), y=plaq.sample, type="l", xlab="t", ylab="<P>")
```

`plot.averx` *Plots averx data*

Description

Plots averx data

Usage

```
## S3 method for class 'averx'
plot(x, ...)
```

Arguments

`x`	`averx` object
`...`	ignored

Value

Returns the plotted data in from of a data.frame with named columns `t` (the time index), `averx` the values of average x and `daverx` the statistical error estimate.

plot.bootstrapfit *Plot a bootstrap NLS fit*

Description

Plot a bootstrap NLS fit

Usage

```
## S3 method for class 'bootstrapfit'
plot(x, ..., col.line = "black", col.band = "gray",
  opacity.band = 0.65, lty = c(1), lwd = c(1), supports = 1000,
  plot.range, error = x$error.function, ribbon.on.top = TRUE)
```

Arguments

x	object returned by bootstrap.nlsfit
...	Additional parameters passed to the plotwitherror function.
col.line	line colour.
col.band	error band colour.
opacity.band	error band opacity.
lty	line type of fitted curve.
lwd	line width for fitted curve.
supports	number of supporting points for plotting the function.
plot.range	vector with two elements c(min,max) defining the range in which fitline and errorband are plotted. Default is the range of the data.
error	Function to compute the standard error in resampling schemes. Default is sd for bootstrap. For other resampling schemes this might need to be changed.
ribbon.on.top	Logical, controls whether the ribbon should be in front of the data points. This is recommended when there are very many data points and a highly constrained model.

Value

No return value.

See Also

Other NLS fit functions: bootstrap.nlsfit(), parametric.bootstrap.cov(), parametric.bootstrap(), parametric.nlsfit.cov(), parametric.nlsfit(), predict.bootstrapfit(), print.bootstrapfit(), simple.nlsfit(), summary.bootstrapfit()

plot.cf — *Plot a correlation function*

Description

Plot a correlation function

Usage

```
## S3 method for class 'cf'
plot(x, neg.vec = rep(1, times = length(cf$cf0)), rep = FALSE, ...)
```

Arguments

`x`	`cf_boot` object
`neg.vec`	Numeric vector of length `cf$cf0`. This allows switching the sign for certain time slices or observables such that displaying in log-scale is sensible.
`rep`	See `plotwitherror`.
`...`	Graphical parameter to be passed on to plotwitherror

Value

Invisibly returns a data.frame with named columns `t` containing the (physical) t-values, `CF` the mean values of the correlation function and `Err` its standard error.

plot.cfit — *plot.c1fit*

Description

Generic function to plot an object of type `c1fit`

Usage

```
## S3 method for class 'cfit'
plot(x, ...)
```

Arguments

`x`	Object of type `c1fit`
`...`	Generic graphical parameter to be passed on to plotwitherror

Value

No return value, only plots are generated.

plot.coshfit *Plot a cosh-fit*

Description

Plot a cosh-fit

Usage

```
## S3 method for class 'coshfit'
plot(x, col.fitline = "black", plot.mass = TRUE, plot.corr = FALSE, ...)
```

Arguments

`x`	An object generated by `fit.cosh`.
`col.fitline`	Color in which the fit is visualized.
`plot.mass, plot.corr`	The plot can show the fitted correlator (`plot.corr`) as well as the corresponding effective mass (`plot.mass`, if fitted with effMass).
`...`	graphical parameters to be passed on to plotwitherror

Value

No return value.

plot.effectivemass *plot.effectivemass*

Description

plot.effectivemass

Usage

```
## S3 method for class 'effectivemass'
plot(x, ..., ref.value, col, col.fitline)
```

Arguments

`x`	Object of class `effectivemass`
`...`	Graphical parameters to be passed on to plotwitherror
`ref.value`	Numeric. A reference value to be plotted as a horizontal line
`col`	String. Colour of the data points.
`col.fitline`	String. Colour of the fitted line.

Value

No return value.

---

plot.effmass *plot.effmass*

Description

plot.effmass

Usage

```
## S3 method for class 'effmass'
plot(x, ..., ll, lf, ff)
```

Arguments

x	Object of class `effmass`
...	Graphical parameters to be passed on.
ll	local-local effective mass object
lf	local-fuzzed effective mass object
ff	fuzzed-fuzzed effective mass object

Value

No value returned, only plots are generated.

---

plot.gevp.amplitude *plot.gevp.amplitude*

Description

plot.gevp.amplitude

Usage

```
## S3 method for class 'gevp.amplitude'
plot(x, xlab = "t", ylab = paste0("P[,", x$id, ",", x$op.id, "]"), ...)
```

Arguments

x	Object of type `gevp.amplitude`.
xlab	x axis label
ylab	y axis label
...	Graphical parameters to be passed on.

Value

No return value.

`plot.hadronacf` *plot.hadronacf*

Description

generic function to plot an object of class "myGamma"

Usage

```
## S3 method for class 'hadronacf'
plot(x, ..., col = "black")
```

Arguments

`x`	Object of type `hadronacf` generated by computeacf
`...`	Generic graphical parameters to be passed on
`col`	String. Color to be used for the data points.

Value

No return value.

`plot.massfit` *plot.massfit*

Description

Generic function to plot an object of type `massfit`

Usage

```
## S3 method for class 'massfit'
plot(x, ..., xlab = "t", ylab = "m")
```

Arguments

`x`	Object of type `massfit`
`...`	Generic graphical parameter to be passed on to plotwitherror
`xlab`	String. Label for x-axis
`ylab`	String. Lable for y-axis

Value

See plotwitherror.

`plot.matrixfit` *Plot a matrixfit*

Description

Plot a matrixfit

Usage

```
## S3 method for class 'matrixfit'
plot(x, plot.errorband = FALSE, ylim, xlab = "t/a",
  ylab = "y", do.qqplot = TRUE, plot.raw = TRUE, rep = FALSE, col,
  every, ...)
```

Arguments

`x`	an object of class matrixfit
`plot.errorband`	Boolean: whether or not to plot an errorband
`ylim`	Numeric vector: y-limit of the plot
`xlab`	String: label of x-axis
`ylab`	String: label of y-axis
`do.qqplot`	Boolean: whether or not to plot an QQ-plot
`plot.raw`	Boolean: plot the raw data or multiply out the leading exponetial behaviour
`rep`	Boolean: whether or not to add to existing plot
`col`	String vector: vector of colours for the different correlation functions
`every`	Fit only a part of the data points. Indices that are not multiples of `every` are skipped. If no value is provided, all points are taken into account.
`...`	Graphical parameters to be passed on to plot or plotwitherror.

Value

Returns no value, generated only plots.

See Also

`matrixfit`

`plot.ofit` *plot.ofit*

Description

Generic function to plot an object of type `ofit`

Usage

```
## S3 method for class 'ofit'
plot(x, ...)
```

Arguments

`x`	Object of type `ofit`
`...`	Generic graphical parameter to be passed on to plotwitherror

Value

See plot.cfit

`plot.outputdata` *Plot Command For Class Ouputdata*

Description

Generic plot routine for class "ouputdata". Currently it plots the plaquette history and the history of ΔH

Usage

```
## S3 method for class 'outputdata'
plot(x, skip = 0, ...)
```

Arguments

`x`	object of class "outputdata" obtained from a read with `readoutputdata`
`skip`	number of trajectories to be skipped in analysis for plaquette and $\exp(-\Delta H)$.
`...`	additional arguments passed to the generic plot function.

Value

list containing the "data", an object of class "uwerr" called "plaq.res" containing the statisical analysis for the plaquette and a second object of type "uwerr" called "dH.res" with the statisical analysis for $\exp(-\Delta H)$.

The plotted data is return in form of a list with named elements `data` containing the input data, plaq.res an object returned by uwerrprimary for the plaquette data dn `dH.res` an object returned by uwerrprimary for ΔH.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

See Also

`readoutputdata`, `uwerr`

Examples

```
plaq <- readoutputdata(paste0(system.file(package="hadron"), "/extdata/output.data"))
plaq.plot <- plot(plaq, skip=100)
summary(plaq.plot$plaq.res)
```

`plot.pionff` *plot.pionff*

Description

Generic function to plot an object of type `pionff`

Usage

```
## S3 method for class 'pionff'
plot(x, ...)
```

Arguments

`x`	Object of type `pionff`
`...`	Generic graphical parameter to be passed on to plotwitherror

Value

No return value, only plots are generated.

plot.raw_cf — *plot all correlators in* raw_cf *object*

Description

plot all correlators in raw_cf object

Usage

```
## S3 method for class 'raw_cf'
plot(x, ..., reim = "real", reim_same = FALSE)
```

Arguments

x	Object of class raw_cf with data and meta-data.
...	Further parameters passed to plotwitherror.
reim	Character vector, may contain 'real', 'imag' or 'both'. Determines whether the real and/or imaginary parts of the correlation funtions should be plotted.
reim_same	Boolean, determines whether the real and imaginary parts, if both are to be plotted, will be plotted in the same plot.

Value

Invisibly returns the plotdata, see get_plotdata_raw_cf.

plot.uwerr — *Plot Command For Class UWerr*

Description

Plot Command For Class UWerr

Usage

```
## S3 method for class 'uwerr'
plot(x, ..., main = "x", plot.hist = TRUE, index = 1, Lambda = 100)
```

Arguments

x	object of class uwerr
...	generic parameters, not used here.
main	main title of the plots.
plot.hist	whether or not to generate a histogram
index	index of the observable to plot.
Lambda	Cutoff to be used in the error computation for the ACF.

Value

produces various plots, including a histogram, the autocorrelationfunction and the integrated autocorrelation time, all with error bars.

No return value.

Author(s)

Carsten Urbach, `<carsten.urbach@liverpool.ac.uk>`

See Also

`uwerr`

Examples

```
data(plaq.sample)
plaq.res <- uwerrprimary(plaq.sample)
plot(plaq.res)
```

`plothlinewitherror` *plothlinewitherror*

Description

plot a horizontal line with error band

Usage

```
plothlinewitherror(m, dp, dm, col = c("red"), x0, x1)
```

Arguments

`m`	Numeric. Mean value of the line to plot.
`dp`	Numeric. Error up.
`dm`	Numeric. Error down.
`col`	String. Colour.
`x0`	Numeric. Left value of the range of the horizontal line.
`x1`	Numeric. Right value of the range of the horizontal line.

Value

No return value, only graphics is generated.

`plotwitherror` *Plot Command For XY Plots With Error Bars*

Description

Plot command for XY scatterplots based on plot and points which provides support for multiple, non-symmetric error bars. Error bars are drawn as vertical or horizontal lines originating from the point with narrow, perpendicular lines at the end of the error bar (end caps). When multiple errors are drawn, the width of the perpendicular line increases from the innermost error bar to the outermost one. Different summation methods for the individual errors are supported.

Usage

```
plotwitherror(x, y, dy, ylim = NULL, dx, xlim = NULL, mdx, mdy,
  errsum.method = "linear.quadrature", rep = FALSE, col = "black", ...)
```

Arguments

`x`	vector of x coordinates
`y`	vector of y coordinates
`dy`	one of: • Vector of errors on y coordinates. • Array, matrix or data frame if multiple error bars are to be drawn, such that each column refers to one error. The individual errors should be provided as is, because they are summed internally to draw the final error bars. A given column can also be provided with 0 entries, in which case the error bar will be drawn, but it will have zero length, such that only the end caps for this error will be visible.
`ylim`	limits for y-axis
`dx`	Same as `dy`, but for the x coordinate.
`xlim`	limits for x-axis
`mdx`	Support for non-symmetric error bars. Same as `dx`, but for errors in the negative x-direction. Errors should be provided as positive numbers, the correct sign will be added internally. If not provided, `dx` is used as a symmetric error.
`mdy`	Same as `mdx` but for the y coordinate.
`errsum.method`	Determines how the invidual errors should be summed for display purposes. Valid argument values are: • "linear" – Individual errors are summed linearly, such that the distance from the point to the i'th error bar, l_i, is

$$l_i = \sum_{j=1}^{i} e_j$$

Hence, the third error bar, for example, would be located at

$$l_3 = e_1 + e_2 + e_3$$

while the second error bar is at

$$l_2 = e_1 + e_2$$

- "quadrature"
  - Individual errors are summed in quadrature and error bars are drawn at the fractional position according to the following formula:

$$l_{max} = \sqrt{\sum_{j=1}^{max} e_j^2}$$

$$l_i = \sum_{j=1}^{i} e_j^2 / l_{max}$$

- "linear.quadrature"
  - Errors are summed as for "linear", but the total error summed in quadrature is also indicated as an end cap of triple line width

`rep`	If set to `TRUE`, operate like "replot" in gnuplot. Allows adding points with error bars to the current plot. Switches the underlying plotting routine from `plot` to `points`.
`col`	colour of plotted data
`...`	any graphic options passed over to `plot`

Value

a plot with error bars is drawn on the current device

Returns for convenience a list with elements `xlim` and `ylim` representing the x- and y-limits chosen by the routine.

Author(s)

Carsten Urbach, `<urbach@hiskp.uni-bonn.de>`
Bartosz Kostrzewa, `<bartosz.kostrzewa@desy.de>`

See Also

`plot`, `points`

Examples

```
# Create some random data, set one error to zero.
x <- 1:50
y <- runif(50, 0, 1)
```

```
dy <- runif(50, 0.1, 0.2)
dy[4] <- 0

plotwitherror(x, y, dy)
```

`plot_eigenvalue_timeseries` *plot_eigenvalue_timeseries*

Description

function to plot timeseries of eigenvlues, including minimum and maximum eigenvalue bands as found in the monomial_0x.data files produced by tmLQCD

Usage

```
plot_eigenvalue_timeseries(dat, stat_range, ylab, plotsize, filelabel,
  titletext, pdf.filename, errorband_color = rgb(0.6, 0, 0, 0.6),
  debug = FALSE)
```

Arguments

`dat`	Timeseries to analyse.
`stat_range`	range of statistics to use.
`ylab`	Y-axis label.
`plotsize`	Width and Height of plot.
`filelabel`	String. Label of the file.
`titletext`	Text in the plot title.
`pdf.filename`	String. PDF filename.
`errorband_color`	String. Colour of the error band.
`debug`	Boolean. Generate debug output.

Value

Returns a list with two named elements `mineval` and `maxeval` for the minimal and the maximal eigenvalue, see plot_timeseries.

plot_hankel_spectrum *plot_hankel_spectrum*

Description

produces a scatter plot of the complex $-\log$ of the eigenvalues produced by the bootstrap.hankel method. In addition, produces a histogramm of all real and positive eigenvalues after computing $-\log(ev)/\delta t$ in the range (0,1) and determines its mode.

Usage

```
plot_hankel_spectrum(hankel, deltat = 1, id = c(1:hankel$n))
```

Arguments

`hankel`	object as returned from bootstrap.hankel
`deltat`	Integer. Time shift at which to plot
`id`	Integer vector. Indices of eigenvalues to be plotted. Must be part of `c(1:hankel$n)`.

Value

No return value.

See Also

Other hankel: `bootstrap.hankel_summed()`, `bootstrap.hankel()`, `gevp.hankel_summed()`, `gevp.hankel()`, `hankel2cf()`, `hankel2effectivemass()`

plot_timeseries *plot_timeseries*

Description

function to plot timeseries data, a corresponding histogram and an error shading for an error analysis via uwerr

Usage

```
plot_timeseries(dat, ylab, plotsize, titletext, hist.by, stat_range = c(1,
  length(dat$y)), pdf.filename, name = "", xlab = "$t_\\mathrm{MD}$",
  hist.probs = c(0, 1), errorband_color = rgb(0.6, 0, 0, 0.6),
  type = "l", uwerr.S = 2, smooth_density = FALSE, periodogram = FALSE,
  debug = FALSE, uw.summary = TRUE, ...)
```

Arguments

dat	Timeseries to analyse.
ylab	Y-axis label.
plotsize	Width and Height of plot.
titletext	Text in the plot title.
hist.by	Numeric. Stepping to compute the histogram breaks.
stat_range	Optional integer vector of length 2. Start and end indices of the subset of `dat` to be plotted. If left empty, all of `dat` will be plotted.
pdf.filename	String. PDF filename.
name	String. Timeseries name.
xlab	X-axis label.
hist.probs	Optional numeric vector of length 2. Probability extrema to limit the width of the histogram or smoothed density plots. By default all data is used. Note: this has not effect on the analysis as a whole or other plots.
errorband_color	String. Colour of the error band.
type	String. Plot type, see plot for details.
uwerr.S	Numeric. `S` of the uwerr method to be used.
smooth_density	Boolean. Instead of plotting a histogram, use a smoothed density.
periodogram	Boolean. Whether to show a periodogram.
debug	Boolean. Generate debug output.
uw.summary	Boolean. Generate an uwerr summary plot.
...	Generic graphical parameters to be passed on.

Value

Returns a data.frame with named columns `val`, `dval`, `tauint`, `dtauint`, `Wopt` and `stringsAsFactors`, see uwerr.

---

pointswithslantederror *pointswithslantederror*

Description

This function plots points with x- and y-errors visualised as a slanted errorbar. The length of the error bar represents x- and y-errors added in quadrature. The slope of the error bar is positive of negative depending on whether the correlation betwenn x and y is positive or negative, respectively.

Usage

```
pointswithslantederror(x, y, dx, dy, cor, col = "black", bcol = "black", ...)
```

Arguments

`x`	numeric vector. x-values
`y`	numeric vector. y-values
`dx`	numeric vector. x-standard errors
`dy`	numeric vector. y-standard errors
`cor`	numeric vector. Correlation coefficients between x- and y- errors.
`col`	the color of the points
`bcol`	the color of the slanted error bars
`...`	further graphical parameters to be passed on to `points`

Details

plots data points with slanted error bars

Examples

```
x <- c(1:5)
y <- x^2
dx <- c(0.1, 0.2, 0.2, 0.1, 0.05)
dy <- c(0.05, 0.2, 0.1, 0.2, 0.1)
cor <- c(1, -1, -1, 1, 1)
plot(NA, xlim=range(x), ylim=range(y), xlab="y", ylab="y")
pointswithslantederror(x=x, y=y, dx=dx, dy=dy, cor=cor)
```

`predict.bootstrapfit` *Predict values for bootstrapfit*

Description

Predict values for bootstrapfit

Usage

```
## S3 method for class 'bootstrapfit'
predict(object, x, error = object$error.function, ...)
```

Arguments

`object`	Object of type bootstrapfit.
`x`	Numeric vector with independent variable.
`error`	Function to compute error from samples.
`...`	additional parameters to be passed on to the prediction function.

Value

List with independent variable `x`, predicted central value `val`, error estimate `err` and sample matrix `boot`.

See Also

Other NLS fit functions: `bootstrap.nlsfit()`, `parametric.bootstrap.cov()`, `parametric.bootstrap()`, `parametric.nlsfit.cov()`, `parametric.nlsfit()`, `plot.bootstrapfit()`, `print.bootstrapfit()`, `simple.nlsfit()`, `summary.bootstrapfit()`

`print.bootstrapfit` *Print a bootstrap NLS fit*

Description

Print a bootstrap NLS fit

Usage

```
## S3 method for class 'bootstrapfit'
print(x, ..., digits = 2)
```

Arguments

`x`	object returned by `bootstrap.nlsfit`
`...`	Additional parameters passed to the `summary.bootstrapfit` function.
`digits`	number of significant digits to print in summary or print.

Value

No return value.

See Also

Other NLS fit functions: `bootstrap.nlsfit()`, `parametric.bootstrap.cov()`, `parametric.bootstrap()`, `parametric.nlsfit.cov()`, `parametric.nlsfit()`, `plot.bootstrapfit()`, `predict.bootstrapfit()`, `simple.nlsfit()`, `summary.bootstrapfit()`

`print.cf` *print.cf*

Description

print.cf

Usage

```
## S3 method for class 'cf'
print(x, ...)
```

Arguments

`x`	Object of type cf
`...`	Generic parameters to pass on.

Value

No return value, only output is produced.

`print.effectivemassfit` *print.effectivemassfit*

Description

print.effectivemassfit

Usage

```
## S3 method for class 'effectivemassfit'
print(x, ..., verbose = FALSE)
```

Arguments

`x`	Object of class `effectivemass`
`...`	Additional parameters to be passed on.
`verbose`	Boolean. More verbose output.

Value

No return value.

`print.ofit` *print.ofit*

Description

print.ofit

Usage

```
## S3 method for class 'ofit'
print(x, ...)
```

Arguments

`x`	Object of type `ofit`
`...`	Generic parameters to pass on.

Value

No return value.

`print.raw_cf` *Print summary of data contained in* `raw_cf` *container*

Description

Print summary of data contained in `raw_cf` container

Usage

```
## S3 method for class 'raw_cf'
print(x, ...)
```

Arguments

`x`	`raw_cf` container with data and meta-data
`...`	ignored

Value

See summary.raw_cf.

pscor.sample *Sample pseudoscalar correlator*

Description

Sample data for a pseudoscalar correlator for time extent Time=48.

Format

list of 2 elements: "t" "ps"

Examples

```
data("pscor.sample")
```

raw_cf *Container for raw correlation functions*

Description

This function `raw_cf()` creates containers for raw correlation functions of class `raw_cf`. This class is particularly designed to deal with complex and matrix-valued correlation functions emerging in statistical mechanics and quantum field theory simulations. Arithmetic operations are defined for this class and utility functions such as `is.raw_cf` and `is_empty.raw_cf`.

Usage

```
raw_cf()
```

Value

An object of S3 class `raw_cf`.

See Also

Other raw_cf constructors: `raw_cf_data()`, `raw_cf_meta()`

`raw_cf_data` *Original data mixin constructor for* `raw_cf`

Description

Original data mixin constructor for `raw_cf`

Usage

```
raw_cf_data(cf, data)
```

Arguments

`cf`	`raw_cf` object to extend.
`data`	Numeric or complex array, original data for all observables and measurements. This should have dimensions c(Nmeas,cf$Time*cf$nrObs*cf$nrStypes,cf$dim). Having the internal dimensions innermost is not as efficient, but it allows different transformations to be applied to different observables in the same container more easily.

Value

An object of S3 class `raw_cf` with original data mixin added.

See Also

Other raw_cf constructors: `raw_cf_meta()`, `raw_cf()`

`raw_cf_meta` `raw_cf` *metadata mixin constructor*

Description

`raw_cf` metadata mixin constructor

Usage

```
raw_cf_meta(cf = raw_cf(), nrObs = 1, Time = NA, nrStypes = 1,
  dim = c(1, 1), nts = Time)
```

Arguments

`cf`	initial raw_cf object
`nrObs`	Integer, number of different observables assembled in the data field of this container.
`Time`	Integer, full time extent.
`nrStypes`	Integer, number of smearing types.
`dim`	Integer vector of "internal" dimensions for matrix-valued correlation functions. For a scalar correlation, this should be specified as `c(1,1)`.
`nts`	Integer, number of time separations actually stored in the data field.

Value

An object of S3 class `raw_cf` with metadat mixing added.

See Also

Other raw_cf constructors: `raw_cf_data()`, `raw_cf()`

raw_cf_to_cf — *Extract a particular internal component of a 'raw_cf' into a 'cf'*

Description

Extract a particular internal component of a 'raw_cf' into a 'cf'

Usage

```
raw_cf_to_cf(x, component)
```

Arguments

`x`	'raw_cf' container with 'raw_cf_data' and 'raw_cf_meta'
`component`	Integer vector of the same length as the internal dimension of the 'raw_cf' specifying which component should be extracted.

Value

'cf' object

`readbinarycf` *read correlation function from binary files*

Description

Reads a correlation function from binary files, including hdf5 formatted files.

Usage

```
readbinarycf(files, Time, obs = 5, Nop = 1, symmetrise = TRUE,
  endian = "little", op = "aver", excludelist = c(""), sym = TRUE,
  path = "", hdf5format = FALSE, hdf5name, hdf5index = c(1, 2))
```

Arguments

`files`	list of filenames to be read. Can be created using `getorderedfilelist`. The filelist is assumed to be order according to ascending gauge fields.
`Time`	time extent of correlation functions.
`obs`	each file may contain many correlation functions. With 'obs' one choses which observable to read in. To be precise, in each file the reading will start at point Time*obs*sizeof(complex<`double`>) and read Nop*Time*sizeof(complex<`double`>).
`Nop`	number of replicas for the correlator to read in.
`symmetrise`	symmetrise the correlation function or not
`endian`	the endianess of the binary file.
`op`	the N replicas can be either averaged (`op="aver"`) or summed (`op="sum"`).
`excludelist`	files to exclude from reading.
`sym`	if `TRUE` average C(+t) and C(-t), otherwise C(+t) and -C(-t).
`path`	path to be prepended to every filename.
`hdf5format`	if `TRUE`, try to read from an hdf5 file.
`hdf5name`	Name of the data set as a string.
`hdf5index`	The data might be an array of size n x Time. `hdf5index` is used to convert two columns of the data to a complex valued vector using the first and second index for real and imaginary part, respectively. If `hdf5index` has length smaller than 2 the first index is reused.

Details

It is assumend that each file contains at least `(obs+N)*Time` complex doubles, where `Time` is the time extent, `obs` is the number of the observable to read in and `Nop` the number of replicas for this observable. It is assumed that complex is the fastest running index, next time and then obs. The filelist is assumed to be ordered according to the gauge configuration MC history.

Value

returns a list with two arrays `cf` and `icf` with real and imaginary parts of the correlator, and integers `Time`, `nrStypes=1` and `nrObs=1`. Both of the arrays have dimension `c(N,(Time/2+1))`, where `N` is the number of measurements (gauges). `Time` is the time extent, `nrStypes` the number of smearing levels and `nrObs` the number of operators, both of which are currently fixed to 1.

Author(s)

Carsten Urbach, <curbach@gmx.de>

See Also

`readcmidatafiles`, `readbinarydisc`, `readcmidisc`, `readcmicor`

Examples

```
X <- readbinarycf(path=paste0(system.file(package="hadron"), "/extdata/"),
                  files="C2_bin.dat", Time=64, obs=0)
X
X$cf
```

---

readbinarydisc   *read disconnected loops from binary files*

Description

Reads disconnected loops from binary files.

Usage

```
readbinarydisc(files, Time = 48, obs = 5, endian = "little",
  excludelist = c(""), nrSamples = 1, path = "")
```

Arguments

`files`	list of filenames to be read. Can be created for instance using `getorderedfilelist`. The filelist is assumed to be ordered with number of samples running fastest, and the next to fastest nubmer of gauges.
`Time`	time extent of correlation functions.
`obs`	each file may contain Time*obs correlation functions. With `obs` one choses which observable to read in.
`endian`	the endianess of the binary file.
`excludelist`	files to exclude from reading.
`nrSamples`	the number of samples
`path`	path to be prepended to every filename.

Details

It is assumend that each file contains O*Time complex doubles, where Time is the time extent and O the number of observables in the file. It is assumed that complex is the fastest running index, next time and then observables. The different samples are assumend to be in different files. The file list is assumed to be ordered with number of samples running fastest, and then number of gauges.

Value

returns a list with two arrays `cf` and `icf` with real and imaginary parts of the loops, and integers `Time`, `nrStypes=1`, `nrSamples` and `nrObs=1`. Both of the arrays have dimension `c(Time,N)`, where `N` is the number of measurements (gauges) and `Time` the time extent, `nrStypes` the number of smearing levels and `nrObs` the number of operators, both of which are currently fixed to 1.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

See Also

`readcmidatafiles`, `readbinarycf`, `readcmidisc`, `readcmicor`

Examples

```
## running toy example
file <- paste0(system.file("extdata", package = "hadron"), "/C2_pi0.dat")
X <- readbinarydisc(files=file, Time=64, obs=0)
X$cf

## more realistic example
## Not run: files <- character()
## Not run: for(i in seq(600,1744,8))
## Not run:   files <- c(files, "C2_dis_u_conf", sprintf("%.04d", i), ".dat", sep="")
## Not run: cf <- readbinarydisc(files, obs=4, excludelist=c("C2_pi0_conf0632.dat"))
```

`readbinarysamples` *Read binary correlation function by sample*

Description

Read binary correlation functions sample by sample, return as a list of length `nosamples` where increasing indices refer to averaging over increasing numbers of samples.

Usage

```
readbinarysamples(files, Time = 48, nosamples = 2, endian = "little",
  excludelist = c(""), sym = TRUE, path = "", ftype = double())
```

Arguments

`files`	character vector. Paths to the file to read. As `path` is prepended to each element, one can also just pass the filenames here.
`Time`	numeric. Time extent.
`nosamples`	number of samples
`endian`	character, either `little` or `big`.
`excludelist`	character vector. Elements in `files` that are specified in `excludelist` are skipped. The caller could also just pass `setdiff(files,excludelist)`.
`sym`	logical. Whether the read data shall be symmetrized in the end.
`path`	character. Path that is prefixed to each of the paths given in `files`.
`ftype`	numeric type. As the data is read in binary this type has to match exactly the one in the file.

Value

Returns a list of `cf` objects.

`readcmidisc` *reads disconnected loops in cmi format*

Description

reads disconnected loops in cmi (Chris Michael) format from a list of files.

Usage

```
readcmidisc(files, obs = 9, ind.vec = c(2, 3, 4, 5, 6, 7, 8),
  excludelist = c(""), skip = 0, L, colClasses = c("integer", "integer",
  "integer", "integer", "numeric", "numeric", "numeric", "numeric"),
  debug = FALSE)
```

Arguments

`files`	list of filenames to be read. Can be created using `getorderedfilelist`.
`obs`	index of operator to parse from files
`ind.vec`	vector containing the index (column in file) of obs, t, samples, Re(local), Im(local, Re(smeared), Im(smeared).
`excludelist`	files to exclude from reading.
`skip`	lines to skip at beginning of each file.
`L`	the spatial lattice extent, set to `Time/2` if missing.
`colClasses`	The column data type classes, the `read.table`.
`debug`	setting debug to TRUE makes the routine more verbose by spilling out separate filenames.

Value

returns a list with four arrays `cf`, `icf` `scf` and `sicf` containing real and imaginary parts of the local and smeared loops, respectively, and integers `Time`, `nrStypes=2`, `nrSamples` and `nrObs=1`. The four arrays have dimension `c(Time,S,N)`, where `S` is the nubmer of samples, `Time` is the time extent and `N` is the number of measurements (gauges). `Time` is the time extent, `nrStypes` the number of smearing levels and `nrObs` the number of operators, which are currently fixed to 1 and 2, respectively. `nrSamples` is the number of samples.

Note that the arrays are normalised by `1/sqrt(L^2)`.

The routine expects that all files have identical content. Otherwise the routine will stop.

Author(s)

Carsten Urbach, <curbach@gmx.de>

See Also

readcmidatafiles, readbinarycf, readbinarydisc, readcmicor

Examples

```
# a running toy example
hpath <- system.file(package="hadron")
files <- paste0(hpath, "/extdata/newdisc.0.1373.0.006.k0v4.10")
X <- readcmidisc(files=files)
X

## a more realistic example
## Not run: v4files <- character()
## Not run: for(i in seq(600,1744,8))
## Not run:   v4files <-
## Not run:   c(v4files, paste("disc.0.163265.0.006.k0v4.", sprintf("%.04d", i), sep=""))
## Not run: v4data <- readcmidisc(v4files)
```

`readcmifiles` *Read Single Data Files in Chris Michael Format*

Description

reads data from single files in Chris Michael format

Usage

```
readcmifiles(files, excludelist = c(""), skip, verbose = FALSE, colClasses,
  obs = NULL, obs.index, avg = 1, stride = 1)
```

Arguments

`files`	list of filenames to be read. Can be created using `getorderedfilelist`.
`excludelist`	files to exclude from reading.
`skip`	Number of lines to be skipped at the beginning of each file
`verbose`	Increases verbosity of the function.
`colClasses`	The column data type classes, the `read.table`.
`obs`	To reduce memory consumption it is possible to extract only one of the obser-vales. The column in which to match `obs` is to be given with `obs.index`. This will only be affective if `obs` is not `NULL`.
`obs.index`	The column in which to match `obs` is to be given with `obs.index`.
`avg`	Integer. Average over successive number samples
`stride`	Integer. Read only subset of files with corresponding stride.

Details

These functions reads data from single data files. It is assumed that every file has the same number of columns.

The cmi (Chris Michael) format for connected correlators comprises 6 colums per file: 1) the observable type number (itype); 2) the operator type number (iobs); 3) the time difference from source going from 0 to $Time/2$ for each operator type; 4) c_1 correlator value at time value forward in time; 5) c_2 correlator value at time value backward in time; 6) number of gauge configuration.

There are scripts shipped with the package converting the output written into seperate files for each gauge configuration into the expected format. They are called `puttogether.sh` and `puttogether_reverse.sh` which will sort with increasing and with decreasing gauge configuration number, respectively.

Note, that the normalisation of correlators needs multiplication by factor of 0.5 (and possible $(2*\kappa)^2$ and L^3 factors dependent on your conventions).

The values of `itype` run from `1` to the total number of gamma matrix combinations available. `iobs` equals `1` for local-local correlators, `3` for local-smeared, `5` for smeared-local and `7` for smeared-smeared

For charged mesons the order of gamma-matrix combinations is as follows:
order PP PA AP AA 44 P4 4P A4 4A for pion like $P = \gamma_5$ $A = \gamma_4\gamma_5$ $4 = \gamma_4$
order 44 VV AA 4V V4 4A A4 VA AV for rho-a1 like $4 = \gamma_i\gamma_4$ $V = \gamma_i$ $A = \gamma_i\gamma_5$
order BB SS - total 20 $\gamma_i\gamma_4\gamma_5$ $S = I$
itype=21 is conserved vector current at sink, γ_5 at source

For neutral mesons the order of gamma-matrix combinations is as follows:
order PP PA AP AA II PI IP AI IA for pion like $P = \gamma_5$ $A = \gamma_4\gamma_5$ $I = 1$
order 44 VV BB 4V V4 4B B4 VB BV for rho-b1 like $4 = \gamma_i\gamma_4$ $V = \gamma_i$ $B = \gamma_i\gamma_4\gamma_5$
order XX AA - total 20 for a0-X like $A = \gamma_i\gamma_5$ $X = \gamma_4$

For loops (disconnected contributions to neutral mesons) the convention is as follows: files are assumed to have eight columns with gauge, gamma, t, sample, ReTL, ImTL, ReTF, ImTF, where gamma is 1 to 16 as list of (hermitian) gamma matrices: order g_5 g_1 g_2 g_3
-ig_4* g_5 g_1 g_2 g_3
-ig_5* ig*_5 g_1 g_2 g_3 ie 1,..*

-ig_5g_4 -ig*_5 g_1 g_2 g_3 ie g_4, g_5*row 2
(so P is 1; A4 is 5; S is 9; A_i is 10,11,12 etc)

t is t-value of trace (here spatial momentum is zero) sample is sample number 1,...24 (or 96) ReTL is real part of trace at time t, with gamma combination given and Local operator (F is Fuzzed == non-local) operator).

Normalisation is trace M^-1 with M=1+...

To make a disconnected correlator, one combines these traces for different t (and different sample number) as a product. Note only Re Gamma=1 and Im Gamma=gamma_5 have VEV's, see `computeDisc`

Value

`readcmicor` returns an object of class `cmicor`, read from a single file.

`readcmidatafiles` returns an object of class `cmicor`, which is an `rbind` of all `data.frames` read from the single files in the filelist.

`readcmiloopfiles` returns an object of class `cmiloop`, which is an `rbind` of all `data.frames` read from the single files in the filelist.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

See Also

`getorderedfilelist`, `extract.obs`, `readcmidisc`

Examples

```
## a running toy example
files <- paste0(system.file(package="hadron"), "/extdata/outprcvn.dddd.00.0000")
X <- readcmifiles(files, skip=0,
                  colClasses=c("integer", "integer","integer","numeric","numeric"))
X

## a more realistic example
## Not run: filelist <- getorderedfilelist("ouptrc", last.digits=3, ending=".dat")
## Not run: cmicor <- readcmidatafiles(filelist, skip=1)
```

`readgradflow` *Read Gradient Flow Output Files in tmLQCD format*

Description

given a pathname, reads all gradient flow output files in that directory

Usage

```
readgradflow(path, skip = 0, basename = "gradflow", col.names)
```

Arguments

`path`	the path into which the function should descend
`skip`	number of measurements to skip.
`basename`	basename of the files to be read.
`col.names`	column names of the columns in the files to be read. If not given it will be infered from the files, if possible.

Details

This function reads all tmLQCD gradient flow files in the given path and returns a data frame which concatenates them all.

The single files are expected to be in the tmLQCD format which consists of a header with the column names "traj t P Eplaq Esym tsqEplaq tsqEsym Wsym" and the measurement for each flow time in rows. The columns can be ordered arbitrarily as long as the header and the data are consistent.

Value

The function returns a data frame ordered first by the flow time and then by the the trajectory number (so the trajectory number is the index which runs fastest). The data frame has column names

- t - flow time
- traj - trajectory number
- P - plaquette expectation value (at flow time t)
- Eplaq - energy density from plaquette definition (at flow time t)
- Esym - energy density from clover definition (at flow time t)
- tsqEplaq - flow time squared multiplied by plaquette energy density
- tsqEsym - flow time squared multiplied by clover energy density
- Wsym - BMW 'w(t)' observable

.

Author(s)

Bartosz Kostrzewa, `<bartosz.kostrzewa@desy.de>`

Examples

```
path <- system.file("extdata/", package="hadron")
raw.gf <- readgradflow(path)
```

readhlcor *readhlcor*

Description

readhlcor

Usage

```
readhlcor(filename)
```

Arguments

`filename`	String. Filename of the heavy light correlator data file. The file is expected to have nine columns, the first four integer, the second four numeric and the last integer valued again.

Value

Invisibly returns a data.frame object containing the file content.

readnissatextcf *reader for Nissa text format correlation functions*

Description

reader for Nissa text format correlation functions

Usage

```
readnissatextcf(file_basenames_to_read, smear_combs_to_read, Time,
  combs_to_read, nts = Time, sym.vec = c(1), symmetrise = FALSE)
```

Arguments

`file_basenames_to_read`	Character vector of file names without the smearing combination suffixes (such as 'll', 'ls', 'sl', 'ss') which will be added in the reading routine accordign to what was passed via `smear_combs_to_read`. An example would be '0001/mes_contr_2pts', not the lack of the smearing suffix.
`smear_combs_to_read`	Character vector containing the smearing cominations that are to be read. These will be attached to the `file_basenames_to_read` in the reading routine.
`Time`	Integer, time extent of the lattice.

`combs_to_read` Data frame containing the indices of the masses and r-paramter combinations to be read as well as the name of the spin combination. For a two-point function using the second and third mass (0-indexed), the (+^dag,+) r-combination and the pseudoscalar-pseudoscalar spin combination would look as follows:

m1_idx	m2_idx	r1_idx	r2_idx	spin_comb
1	2	0	0	"P5P5"

`nts`	Integer, number of time slices to be read from the correlator files.
`sym.vec`	Integer or numeric vector. Specifies whether the correlator at the given position is symmetric (+1.0) or anti-symmetric (-1.0) under time reflection. This is passed to `symmetrise.cf`. This should be of sufficient length to cover all correlators that are going to be read (one number per row of `combs_to_read` and per entry of `smear_combs_to_read`)
`symmetrise`	Boolean, specifies whether averaging over backward and forward correlators should be done after the correlator has been read in.

Value

Returns an object of class `cf`.

`readoutputdata` *Read Data In output.data Format of tmLQCD*

Description

reads data from an output.data file written by tmLQCD

Usage

```
readoutputdata(filename)
```

Arguments

`filename`	filename of the data file

Details

The data can be plotted directly using “plot”.

Value

returns a data frame of class “outputdata” containing the data.

Returns an object of class `outputdata` derived from a data.frame as generated by read.table applied to the input file.

Author(s)

Carsten Urbach <`curbach@gmx.de`>

Examples

```
plaq <- readoutputdata(paste0(system.file(package="hadron"), "/extdata/output.data"))
plot(plaq)
```

---

readtextcf *Read correlator data from single file*

Description

Reads arbitrary number of samples for a complex correlation function from a text file.

Usage

```
readtextcf(file, Time = 48, sym = TRUE, path = "", skip = 1,
  check.t = 0, ind.vector = c(2, 3), symmetrise = TRUE, stride = 1,
  avg = 1, Nmin = 4, autotruncate = TRUE)
```

Arguments

file	filename of file to read from.
Time	time extent of the correlation function
sym	if TRUE average C(+t) and C(-t), otherwise C(+t) and -C(-t). Averaging can be switched off using the symmetrise option.
path	the path to the files.
skip	number of lines to skip at beginning of file
check.t	if set to an integer value larger than zero the function will assume that in the corresponding column of the file the Euclidean time is counted and it will check whether the maximum in this column is identical to Time-1.
ind.vector	index vector of length 2 with the indices of real and imaginary values of correlator, respectivley.
symmetrise	if set to TRUE, the correlation function will be averaged for t and Time-t, with the sign depending on the value of sym. Note that currently the correlator with t-values larger than Time/2 will be discarded.
stride	Integer. Read only subset of files with corresponding stride.
avg	Integer. Average over successive number samples
Nmin	Integer. Minimal number of measurements that must remain after sparsification and averaging. Default equals to 4.
autotruncate	Boolean. Whether to autotruncate or not

Value

returns a list with two arrays `cf` and `icf` with real and imaginary parts of the correlator, and integers `Time`, `nrStypes=1` and `nrObs=1`. Both of the arrays have dimension `c(N,(Time/2+1))`, where `N` is the number of measurements (gauges). `Time` is the time extent, `nrStypes` the number of smearing levels and `nrObs` the number of operators, both of which are currently fixed to 1.

Author(s)

Carsten Urbach, `<curbach@gmx.de>`

See Also

`readcmidatafiles`, `readbinarydisc`, `readcmidisc`, `readcmicor`, `readbinarycf`

`removeTemporal.cf` *Remove Thermal States by Weighting and Shifting*

Description

Remove Thermal States by Weighting and Shifting

Usage

```
removeTemporal.cf(cf, single.cf1, single.cf2, p1 = c(0, 0, 0), p2 = c(0, 0,
  0), L, lat.disp = TRUE, weight.cosh = FALSE)
```

Arguments

`cf`	Object of type cf
`single.cf1`	Object of type cf
`single.cf2`	Object of type cf
`p1`	Numeric vector. Spatial momentum of first state
`p2`	Numeric vector. Spatial momentum of second state
`L`	Integer. Spatial lattice extent.
`lat.disp`	Boolean. Use lattice dispersion relation instead of continuum one
`weight.cosh`	Boolean. Use cosh functional form in the weighting procedure

Value

weighted and shifted correlation function as a cf object.

resample_hankel *Resample bootstrap samples in Hankel effmass*

Description

The bootstrap distribution in the Hankel effective mass can be quite broad due to outliers and long tails. These screw with proper error estimation. Therefore it can be useful to trim these tails. Just trimming a bootstrap distribution would lead to less samples, therefore we do a parametric resampling.

Usage

```
resample_hankel(hankel_effmass, distance = 5)
```

Arguments

`hankel_effmass`	Hankel effective mass from `hankel2effmass`.
`distance`	Numeric, threshold for marking outliers.

Details

The central values are also inferred from the distribution because they often are outliers themselves. The new central value is the middle between the upper and lower quantile, making the resulting distribution symmetric.

Half the distance between the quantiles is taken to be the error, therefore the quantiles are chosen at 16 and 84 percent to match the standard deviation. All points that are more than "distance" errors away from the new central value are taken to be outliers.

Value

The Hankel effmass object is returned with the same fields, the numbers have been changed.

Additionally there are the followi1ng fields:

- `cov_full` contains the full covariance matrix as determined from all the data. This will be skewed by the outliers.
- `finite_count` gives the number of non-outliers per time slice.
- `complete_count` gives the numbers of complete cases if all outliers are taken out. This number is often zero because the late time slices contain lots of outliers due to the noise.
- `cov_3sigma_pairwise` is the covariance matrix using only the non-outliers and removing NAs in a pairwise fashion, using the maximum of the data. This is the covariance matrix that is used for the resampling.

In case that no time slices had a finite error estimate, this function returns just `NA`.

`resampling_is_compatible` *Checks whether the resampling of two cf objects is compatible*

Description

Checks whether the resampling of two cf objects is compatible

Usage

```
resampling_is_compatible(cf1, cf2)
```

Arguments

`cf1`	`cf` object with `cf_boot`
`cf2`	`cf` object with `cf_boot`

Details

Checks whether operations such as addition can be performed on the resampling samples of `cf1` and `cf2`. In addition to all meta parameters, the dimensions of the resampling sample arrays must be identical.

Value

List of named booleans for each of the checked conditions with elements `boot`, `boot.R`, `boot.l`, `sim`, `endcorr`, `resampling_method`, `boot_dim`, `icf` and, optionally `iboot_dim` (if both `cf1` and `cf2` contain imaginary parts).

`resampling_is_concatenable` *Checks whether the resampling of two cf objects is concatenable*

Description

Checks whether the resampling of two cf objects is concatenable

Usage

```
resampling_is_concatenable(cf1, cf2)
```

Arguments

`cf1`	`cf` object with `cf_boot`
`cf2`	`cf` object with `cf_boot`

Details

In contrast to resampling_is_compatible, this function checks if the resampling samples are concatenable on the horizontal axis. In addition to checking all meta parameters, the number of rows in the resampling arrays must be identical but the number of columns may differ.

Value

List of named booleans for each of the checked conditions with elements `boot`, `boot.R`, `boot.l`, `sim`, `endcorr`, `resampling_method`, `boot_nrow`, `icf` and, optionally `iboot_nrow` (if both `cf1` and `cf2` contain imaginary parts).

`residual_plot` *residual_plot*

Description

generic residual_plot method

Usage

```
residual_plot(x, ...)
```

Arguments

`x`	the object to plot
`...`	additional parameters to be passed on to specialised functions

Value

No return value.

`restore_seed` *Restore random number generator state*

Description

Restore random number generator state

Usage

```
restore_seed(old_seed)
```

Arguments

`old_seed`	integer. Previous seed that should be restored globally.

Value

No return value, but the random seed is reset to `old_seed`.

`samplecf` *Sample cf data*

Description

Sample data for a correlation function for a 24 cube times 48 lattice QCD simulation representing a pion propagation. It is stored in form of an object of class `cf`, which is derived from `list`.

Format

The format is: List of 15 $ cf : num 521 533 532 531 561 ... $ icf : num 521 533 532 531 561 ... $ Time : num 48 $ nrStypes : num 1 $ nrObs : num 1 $ boot.samples : logi TRUE $ jackknife.samples: logi FALSE $ symmetrised : logi TRUE $ boot.R : num 1500 $ boot.l : num 2 $ seed : num 1442556 $ sim : chr "geom" $ cf0 : num 519 375 274 221 185 ... $ cf.tsboot :List of 11 ..$ t0 : num 519 375 274 221 185 ... ..$ t : num 521 518 520 519 519 ... ..$ R : num 1500 ..$ data : num 521 533 532 531 561 ... ..$ seed : int 403 624 -867935848 1692432057 -1535150298 -1438296209 912697060 1838233749 1438572626 999279531 ... ..$ statistic:function (x) ..$ sim : chr "geom" ..$ n.sim : int 1018 ..$ call : language tsboot(tseries = cf$cf, statistic = function(x) return(apply(x, MARGIN = 2L, FUN = mean)) ...) ..$ l : num 2 ..$ endcorr : logi TRUE ..- attr(, *"class")= chr "boot" ..- attr(*, "boot_type")= chr "tsboot" $ tsboot.se : num 1.001 0.615 0.572 0.537 0.499 ... - attr(*, "class")= chr "cf" "list"

Examples

```
data(samplecf)
bootstrapped <- bootstrap.cf(samplecf)
plot(bootstrapped)
```

`shift.cf` *shift a correlation function by 'places' time-slices*

Description

C'(t) = C(t+places) where places can be positive or negative as required and periodic boundary conditions in time are assumed

Usage

```
shift.cf(cf, places)
```

Arguments

`cf`	unsymmetrised correlation function (cf_meta and cf_orig mixins required)
`places`	integer number of time-slices for backward (negative) or forward (positive) shifts

Value

Returns an object of class `cf` containing the shifted correlation function.

`shift.raw_cf` *shift a `raw_cf` correlation function by 'places' time-slices*

Description

shift a `raw_cf` correlation function by 'places' time-slices

Usage

```
shift.raw_cf(cf, places)
```

Arguments

`cf`	`raw_cf` container
`places`	Integer (possibly a vector), number of time slices that the correlation function should be shifted by. Can be positive or negative. This can either be a single value such that a shift by this many time slices will be applied to every measurement or it can be a vector of values of the same length as the number of measurements in `cf`. In that case, a different shift will be applied to each measurement. This is useful if it is important to preserve the absolute time coordinates of a correlation function until some time-dependent transformations have been applied.

Details

The correlation funtion $C(t)$ is shifted in time to produce:

$$C'(t) = C(t + places)$$

using periodic boundary conditions in time.

Value

Returns an object of class `raw_cf`, shifted compared to the input object.

simple.nlsfit	*NLS fit with without bootstrap*

Description

NLS fit with without bootstrap

Usage

```
simple.nlsfit(fn, par.guess, y, x, errormodel, priors = list(param = c(), p =
  c(), psamples = c()), ..., lower = rep(x = -Inf, times =
  length(par.guess)), upper = rep(x = +Inf, times = length(par.guess)), dy,
  dx, CovMatrix, boot.R = 0, gr, dfn, mask, use.minpack.lm = TRUE,
  error = sd, maxiter = 500, success.infos = 1:3,
  relative.weights = FALSE, na.rm = FALSE)
```

Arguments

`fn`	`fn(par,x,...)`. The (non-linear) function to be fitted to the data. Its first argument must be the fit parameters named `par`. The second must be `x`, the explaining variable. Additional parameters might be passed to the function. Currently we pass `boot.r` which is `0` for the original data and the ID (1, ...) of the bootstrap sample otherwise. As more parameters might be added in the future it is recommended that the fit function accepts `...` as the last parameter to be forward compatible.
`par.guess`	initial guess values for the fit parameters.
`y`	the data as a one-dimensional numerical vector to be described by the fit function.
`x`	values of the explaining variable in form of a one-dimensional numerical vector.
`errormodel`	Either "yerror" or "xyerror", depending on the x-values having errors or not.
`priors`	List possessing the elements `param`, `p` and `psamples`. The vector `param` includes the indices of all fit parameters that are to be constrained and the vector `p` the corresponding paramater values (e.g. known from a previous fit). The list element `psamples` is a matrix of dimensions `(boot.R,length(param))` and contains the corresponding bootstrap samples. If this list is not specified priors are omitted within the fit.
`...`	Additional parameters passed to `fn`, `gr` and `dfn`.
`lower`	Numeric vector of length `length(par.guess)` of lower bounds on the fit parameters. If missing, `-Inf` will be set for all.
`upper`	Numeric vector of length `length(par.guess)` of upper bounds on the fit parameters. If missing, `+Inf` will be set for all.
`dy`	Numeric vector. Errors of the dependent and independent variable, respectively. These do not need to be specified as they can be computed from the bootstrap samples. In the case of parametric bootstrap it might would lead to a loss of information if they were computed from the pseudo-bootstrap samples. They must not be specified if a covariance matrix is given.

`dx`	Numeric vector. Errors of the dependent and independent variable, respectively. These do not need to be specified as they can be computed from the bootstrap samples. In the case of parametric bootstrap it might would lead to a loss of information if they were computed from the pseudo-bootstrap samples. They must not be specified if a covariance matrix is given.
`CovMatrix`	complete variance-covariance matrix of dimensions `c(length(y),length(y))` or `c(length(y)+length(x),length(y)+length(x))` depending on the error-model. Pass `NULL` if the matrix has to be calculated from the `bsamples`. In that case, if the number of boostrap samples is small compared to the number of variables, singular value decomposition with small eigenvalue replacement will be used (see invertCovMatrix) to attempt a clean inversion. In case a variance-covariance matrix is passed, the inversion will simply be attempted using `solve` on the Cholesky decomposition. Finally, if `CovMatrix` is missing, an uncorrelated fit will be performed.
`boot.R`	If larger than 0, `boot.R` paramtetric bootstrap samples are generated on the fit results after fit and error calculation are finished. The original data is never boostraped in this function.
`gr`	`gr(par,x,...)`. `gr=d(fn) / d(par)` is a function to return the gradient of `fn`. It must return an array with `length(x)` rows and `length(par)` columns.
`dfn`	`dfn(par,x,...)`. `dfn=d(fn) / dx` is the canonical derivative of `fn` by `x` and only relevant if x-errors are provided.
`mask`	logical or integer index vector. The mask is applied to select the observations from the data that are to be used in the fit. It is applied to `x`, `y`, `dx`, `dy`, `bsamples` and `CovMatrix` as applicable.
`use.minpack.lm`	use the `minpack.lm` library if available. This is usually faster than the default `optim` but somtimes also less stable.
`error`	Function that takes a sample vector and returns the error estimate. This is a parameter in order to support different resampling methods like jackknife.
`maxiter`	integer. Maximum number of iterations that can be used in the optimization process.
`success.infos`	integer vector. When using `minpack.lm` there is the `info` in the return value. Values of 1, 2 or 3 are certain success. A value of 4 could either be a success or a saddle point. If you want to interpret this as a success as well just pass `1:4` instead of the default `1:3`.
`relative.weights`	are the errors on y (and x) to be interpreted as relative weights instead of absolute ones? If TRUE, the covariance martix of the fit parameter results is multiplied by chi^2/dof. This is the default in many fit programs, e.g. gnuplot.
`na.rm`	logical. If set to `true`, NAs in `y` and `dy` will be ignored. If x-errors are taken into account, NAs in `x` and `dx` will be ignored, too.

Value

Returns an object of class `bootstrapfit`, see bootstrap.nlsfit.

See Also

Other NLS fit functions: bootstrap.nlsfit(), parametric.bootstrap.cov(), parametric.bootstrap(), parametric.nlsfit.cov(), parametric.nlsfit(), plot.bootstrapfit(), predict.bootstrapfit(), print.bootstrapfit(), summary.bootstrapfit()

Examples

```
## Declare some data.
value <- c(0.1, 0.2, 0.3)
dvalue <- c(0.01, 0.01, 0.015)
x <- c(1, 2, 3)
dx <- c(0.1, 0.1, 0.1)

fn <- function (par, x, ...) par[1] + par[2] * x

fit.result <- simple.nlsfit(fn, c(1, 1), value, x, "xyerrors", dy=dvalue, dx=dx)
summary(fit.result)
```

`store_correl`	*Store a 'raw_cf' correlator in an associative array together with a description The object* `cf` *will be stored as an element of* `cmap` *under key* `out_key` *in the member* `obj` *of* `cmap`*. The data frame passed via* `desc` *will be appended as a row to* `cmap[[out_key]]$map`*. If* `out_key` *does not exist as a key in* `cmap`*, a new element will be created. If it already exists,* `addStat.raw_cf` *is called to add statistics to the existing* `raw_cf`*. Requires the 'hash' package.*

Description

Store a 'raw_cf' correlator in an associative array together with a description The object `cf` will be stored as an element of `cmap` under key `out_key` in the member `obj` of `cmap`. The data frame passed via `desc` will be appended as a row to `cmap[[out_key]]$map`. If `out_key` does not exist as a key in `cmap`, a new element will be created. If it already exists, `addStat.raw_cf` is called to add statistics to the existing `raw_cf`. Requires the 'hash' package.

Usage

```
store_correl(cmap, cf, out_key, desc)
```

Arguments

`cmap`	Object of class `hash` to act as storage for 'raw_cf' correlators.
`cf`	Object of class `raw_cf` to be stored in `cmap`.
`out_key`	String, key associated with `cf` object to be stored in `cmap`.
`desc`	Single row data frame containing some descriptive parameters for `cf`.

Value

Since objects of class `hash` are passed and modified by reference, there is no explicit return value. Instead, the passed `cmap` is modified.

`string2error` *string2error*

Description

takes a string of the form "x(dx)", where dx are the error digits and returns a numeric vector c(x, y), where `y` is dx as a proper numeric value.

Usage

```
string2error(x)
```

Arguments

`x`	Input character string.

Details

can be used in combination with apply

Value

a numeric vector with the first element the value and the second the error

Examples

```
string2error("0.35667(25)")

s <- c("0.35667(25)", "0.667(50)")
apply(array(s, dim=c(1, length(s))), 2, string2error)
```

`subtract.excitedstates` *Substract excited states.*

Description

Excited states are subtracted from the given correlation function and matching matrixfit. The fit is usually done on late time slices when the thermal states have decayed so much that they can be neglected. On the early time slices there are contributions which cannot be explained with a single cosh (or sinh) function. These are exactly the contributions that we do not want.

Usage

```
subtract.excitedstates(cf, mfit, from.samples = FALSE)
```

Arguments

`cf`	Correlation function of class `cf`.
`mfit`	Fit result of class `matrixfit`.
`from.samples`	Whether to use existing bootstrap samples. If set to `TRUE`, the same operation will be applied to the bootstrap samples. Otherwise the result will not contain bootstrap samples, even if the input correlation function did.

Details

The correlation function is altered on the time slices which are earlier than the start of the fit interval. The correlator is replaced by the model function (cosh or sinh or exp) extrapolated until the first time slice. The deviations of the (bootstrap) samples from the mean value are kept.

Value

A correlation function of class `cf` which is computed from the old correlation function $C(t)$ as $M(t) + C(t) - \bar{C}(t)$, where $M(t)$ is the fit model and $\bar{C}(t)$ denotes the average over the (bootstrap) samples. Only time slices earlier than the fit are altered.

`summary.bootstrapfit` *Summarize a bootstrap NLS fit*

Description

Summarize a bootstrap NLS fit

Usage

```
## S3 method for class 'bootstrapfit'
summary(object, ..., digits = 2, print.correlation = TRUE)
```

Arguments

`object`	object returned by `bootstrap.nlsfit`
`...`	ignored
`digits`	number of significant digits to print in summary or print.
`print.correlation`	Logical. Whether to show the correlation between of the fit parameters.

Value

No return value.

See Also

Other NLS fit functions: `bootstrap.nlsfit()`, `parametric.bootstrap.cov()`, `parametric.bootstrap()`, `parametric.nlsfit.cov()`, `parametric.nlsfit()`, `plot.bootstrapfit()`, `predict.bootstrapfit()`, `print.bootstrapfit()`, `simple.nlsfit()`

`summary.cf` *summary.cf*

Description

summary.cf

Usage

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

Arguments

`object`	Object of type cf
`...`	Generic parameters to pass on.

Value

No return value, only output is produced.

summary.coshfit *Summarize a cosh-fit*

Description

Summarize a cosh-fit

Usage

```
## S3 method for class 'coshfit'
summary(object, verbose = FALSE, ...)
```

Arguments

`object`	An object generated by `fit.cosh`.
`verbose`	If set to `TRUE`, all fit results including the correlation matrix of the fit parameters are showed. Otherwise only the effective mass with error is given.
`...`	additional parameters to match generic summary arguments

Value

No return value.

summary.effectivemass *summary.effectivemass*

Description

summary.effectivemass

Usage

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

Arguments

`object`	Object of type `effectivemass` generated by fit.effectivemass
`...`	Generic parameters to pass on.

Value

No return value.

summary.effectivemassfit
summary.effectivemassfit

Description

summary.effectivemassfit

Usage

```
## S3 method for class 'effectivemassfit'
summary(object, ..., verbose = FALSE)
```

Arguments

`object`	Object of type cf
`...`	Generic parameters to pass on.
`verbose`	More verbose output.

Value

No return value.

summary.gevp.amplitude
summary.gevp.amplitude

Description

summary.gevp.amplitude

Usage

```
## S3 method for class 'gevp.amplitude'
summary(object, ...)
```

Arguments

`object`	Object of type `gevp.amplitude`.
`...`	Generic Parameters to be passed on.

Value

No return values.

summary.hadronacf *summary.hadronacf*

Description

generic function to summarise an object of class "myGamma"

Usage

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

Arguments

`object`	Object of type `hadronacf` generated by computeacf
`...`	Generic parameters to be passed on

Value

No return value.

summary.hankel_summed *summary.hankel_summed*

Description

summary.hankel_summed

Usage

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

Arguments

`object`	Object of type "hankel_summed" generated by bootstrap.hankel_summed
`...`	Generic parameters to pass on.

Value

Returns invisibly a data frame with columns `E`, `dE` (energies and their standard errors) `q16` and `q84` the 16 and 84 percent quantiles.

`summary.matrixfit` *summary.matrixfit*

Description

summary.matrixfit

Usage

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

Arguments

`object`	Object of type matrixfit
`...`	Generic parameters to pass on.

Value

No return value.

`summary.ofit` *summary.ofit*

Description

summary.ofit

Usage

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

Arguments

`object`	Object of type `ofit`
`...`	Generic parameters to pass on.

Value

No return value.

summary.raw_cf — *Print summary of data contained in `raw_cf` container*

Description

Print summary of data contained in `raw_cf` container

Usage

```
## S3 method for class 'raw_cf'
summary(object, ..., statistics = FALSE)
```

Arguments

`object`	`raw_cf` container with data and meta-data
`...`	ignored
`statistics`	Boolean, return central value and error for all components of the 'raw_cf'. This can be slow so the default is `FALSE`.

Value

The summary is returned invisibly in form of a data frame.

summary.uwerr — *summary.uwerr*

Description

summary.uwerr

Usage

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

Arguments

`object`	Object of type uwerr
`...`	Generic parameters to pass on.

Value

No return value.

`swap_seed` *Set seed and store a seed which can be used to reset the random number generator*

Description

Set seed and store a seed which can be used to reset the random number generator

Usage

```
swap_seed(new_seed)
```

Arguments

`new_seed`	integer. The new seed that is to be set. In case this is parameter is missing, no changes are made and the function just returns `NULL`. This is useful because a function can just pass on its own `seed` argument and therefore control whether the seed shall be fixed or left as-is.

Value

The generated seed is returned if it exists. Otherwise `NULL`. In case that `new_seed` was missing, `NULL` is returned.

`symmetrise.cf` *Average backward and forward-dominated parts of the correlation function*

Description

When a correlation function is symmetric or anti-symmetric in time, this symmetry can be exploited by averaging the part from source-sink separation 1 to cf$Time/2 with the part from cf$Time/2+1 to cf$Time-1 in order to improve statistical precision. This function reduces the number of time slices in a `cf` object from cf$Time to cf$Time/2+1 by performing this averaging.

Usage

```
symmetrise.cf(cf, sym.vec = c(1))
```

Arguments

`cf`	Object of type `cf`.
`sym.vec`	Integer or integer vector of length cf$nrObs giving the time-reflection symmetry (1 for symmetric, -1 for anti-symmetric) of the observable in question.

Value

Returns an object of class `cf`, which is the symmetrised version of the input `cf` object.

`takeTimeDiff.cf` *Take time difference*

Description

Performs the calculation of the shifted correlator C_shift(t) = C(t) - C(t +/- deltat).

Usage

```
takeTimeDiff.cf(cf, deltat = 1, forwardshift = FALSE)
```

Arguments

`cf`	Object of type `cf`, a particle correlation function which shall be shifted.
`deltat`	integer. the time shift
`forwardshift`	boolean. If set to `TRUE`, the forward finite difference is used instead of the backward one

Value

The shifted correlator as an object of type `cf`, see cf

`tex.catwitherror` *paste a number with error in tex-ready format*

Description

A number with error is converted to a string in tex-ready format like xx(yy) thereby automatically determining the digit at which the error applies.

Usage

```
tex.catwitherror(x, dx, digits = 1, with.dollar = TRUE, with.cdot = TRUE, ...)
```

Arguments

`x`	either a single numeric value, or a numeric vector, where the first element is the value and the second is its error
`dx`	the error. If supplied, it will be printed as the error and the value is the first element of `x`. If `dx` is missing, the second element of `x`, if available, is used as the error. If `dx` is missing and the length of `x` is one, only the value is converted to a string without error.
`digits`	number of error digits
`with.dollar`	include the tex dollar in the return string or not
`with.cdot`	replace the "e" in scientific notation by tex-style "cdot" or not
`...`	arguments to be passed to `formatC` in case of no error, or to `format.errors` otherwise

Details

It is strongly recommended to install the `errors`-package. Otherwise the formatting options are significantly reduced.

The value of the first element of `x` is properly rounded to its significant digits determined by the values of `dx` or the second element of `x` (see above) and `digits`. Then a tex-ready string is returned.

Value

writes a string to standard output

Author(s)

Carsten Urbach, <`curbach@gmx.de`> and Johann Ostmeyer

Examples

```
tex.catwitherror(x=0.375567, dx=0.001)
tex.catwitherror(x=c(0.375567, 0.001))
## it can be used with apply
x = array(c(0.1187, 0.291, 0.388, 0.011, 0.037, 0.021), dim=c(3,2))
apply(x, 1, tex.catwitherror, digits=2)
```

`tikz.finalize` *tikz.finalize*

Description

initialize and finalize a `tikzDevice` and carry out optional post-processing

Usage

```
tikz.finalize(tikzfiles, crop = TRUE, margins = 0, clean = TRUE)
```

Arguments

`tikzfiles`	a list with members $pdf, $tex, $aux and $log, returned by `tikz.init` which must be passed to `tikz.finalize`
`crop`	boolean indicating whether `pdfcrop` should be called on the resulting pdf (existence of `pdfcrop` is checked before the command is called), default TRUE
`margins`	margins argument for pdfcrop command, should be passed as a string consisting of one or multiple numbers (e.g. "10" or "10.5 7.5 6.2 10"), default 0
`clean`	boolean indicating whether temporary files, e.g. "basename.tex", "basename.aux" and "basename.log" should be deleted after the pdf has been generated, default TRUE

Details

Convenience Functions for `tikzDevice`

Value

No return value, but the output PDF will be created and cropped.

Author(s)

Bartosz Kostrzewa, `<bartosz.kostrzewa@desy.de>`

See Also

`tikz.init`

Other tikzutils: `tikz.init()`

`tikz.init` *tikz.init*

Description

initialize and finalize a `tikzDevice` and carry out optional post-processing

Usage

```
tikz.init(basename, standAlone = TRUE, engine, ...)
```

Arguments

`basename`	the base of the files which will be used by `tikzDevice`, e.g. "basename" -> "basename.pdf", etc.
`standAlone`	A logical value indicating whether the output file should be suitable for direct processing by LaTeX. A value of `FALSE` indicates that the file is intended for inclusion in a larger document.
`engine`	used to specify the LaTex engine. If missing, the standard engine of tikz is used.
`...`	optional arguments which are passed to `tikz`, see `tikzDevice::tikz`

Details

Convenience Functions for `tikzDevice`

Value

`tikz.init` returns a list with character vector members, $pdf, $tex, $aux $log containing the corresponding filenames

Author(s)

Bartosz Kostrzewa, <`bartosz.kostrzewa@desy.de`>

See Also

Other tikzutils: `tikz.finalize()`

Examples

```
tikzfiles <- tikz.init("plotname",width=3,height=4)
plot(x=c(1:3), y=c(1:3)^2, xlab="$x$", ylab="$y$")
tikz.finalize(tikzfiles=tikzfiles, clean=TRUE)
file.remove("plotname.pdf")
```

`unsymmetrise.cf` *Unfold a correlation function which has been symmetrised*

Description

After a symmetric correlation function has been averaged across the central time slice, it is sometimes useful to explicitly duplicate the resulting average to span all cf$Time time slices. This function takes a `cf` with cf$Time/2+1 time slices and turns it into one with cf$Time time slices by reflecting the correlation function along the cf$Time/2 axis.

Usage

```
unsymmetrise.cf(cf, sym.vec = c(1))
```

Arguments

`cf`	`cf` object which has been previously symmetrised
`sym.vec`	Integer vector giving the symmetry properties (see symmetrise.cf) of the original unsymmetrised correlation function. This should be of length cf$nrObs

Value

Returns an object of class `cf`, which is the unfolded version of the input `cf` object.

`uwerr` *Time Series Analysis With Gamma Method*

Description

Analyse time series data with the so called gamma method

Usage

```
uwerr(f, data, nrep, S = 1.5, pl = FALSE, ...)
```

Arguments

`f`	function computing the derived quantity. If not given it is assumed that a primary quantity is analysed. f must have the data vector of length Nalpha as the first argument. Further arguments to f can be passed to uwerr via the `...` argument. f may return a vector object of numeric type.
`data`	array of data to be analysed. It must be of dimension (N x Nalpha) (i.e. N rows and Nalpha columns), where N is the total number of measurements and Nalpha is the number of primary observables
`nrep`	the vector (N1, N2, ...) of replica length N1, N2
`S`	initial guess for the ratio tau/tauint, with tau the exponetial autocorrelation length.
`pl`	logical: if TRUE, the autocorrelation function, the integrated autocorrelation time as function of the integration cut-off and (for primary quantities) the time history of the observable are plotted with plot.uwerr
`...`	arguments passed to function `f`.

Value

In case of a primary observable (`uwerrprimary`), an object of class `uwerr` with basis class `list` containing the following objects

`value`	the expectation value of the obsevable
`dvalue`	the error estimate
`ddvalue`	estimate of the error on the error

`tauint`	estimate of the integrated autocorrelation time for that quantity
`dtauint`	error of tauint
`Qval`	the p-value of the weighted average in case of several replicas

In case of a derived observable (`uwerrderived`), i.e. if a function is specified, the above objects are contained in a list called `res`.

`uwerrprimary` returns in addition

`data`	input data

whereas `uwerrderived` returns

`datamean`	(vector of) mean(s) of the (vector of) data

and in addition

`fgrad`	the estimated gradient of `f`

and

`f`	the input statistics

In both cases the return object containes

`Wopt`	value of optimal cut-off for the Gamma function integration
`Wmax`	maximal value of the cut-off for the Gamma function integration
`tauintofW`	integrated autocorrelation time as a function of the cut-off W
`dtauintofW`	error of the integrated autocorrelation time as a function of the cut-off W
`S`	input parameter S
`N`	total number of observations
`R`	number of replicas
`nrep`	vector of observations per replicum
`Gamma`	normalised autocorrelation function
`primary`	set to 1 for `uwerrprimary` and 0 for `uwerrderived`

Author(s)

Carsten Urbach, <`curbach@gmx.de`>

References

"Monte Carlo errors with less errors", Ulli Wolff, Comput.Phys.Commun. 156 (2004) 143-153, Comput.Phys.Commun. 176 (2007) 383 (erratum), hep-lat/0306017

See Also

`plot.uwerr`

Examples

```
data(plaq.sample)
plaq.res <- uwerrprimary(plaq.sample)
summary(plaq.res)
plot(plaq.res)
```

`uwerr.cf` *uwerr.cf*

Description

Gamma method analysis on all time-slices in a 'cf' object

Usage

```
uwerr.cf(cf)
```

Arguments

`cf`	Object of type `cf` containing `cf_orig`

Value

A list with a named element `uwcf` which contains a data frame with six columns, `value`, `dvalue`, `ddvalue`, `tauint`, `dtauint` corresponding to what is returned by uwerrprimary. The sixth column, `t`, is just an index counting the columns in the original `cf$cf`. If `cf` contains an imaginary part, the return value contains another list element, `uwicf` of the same structure as `uwcf`. There are as many rows as there were columns in `cf$cf` and/or `cf$icf`. When the call to uwerrprimary fails for a particular column of `cf$cf` or `cf$icf`, the corresponding row of `uwcf` and/or `uwicf` will contain `NA` for all members.

Examples

```
data(samplecf)
uwerr.cf(samplecf)
```

`uwerr.raw_cf` *Gamma method analysis on all time-slices in a 'raw_cf' object*

Description

Gamma method analysis on all time-slices in a 'raw_cf' object

Usage

```
uwerr.raw_cf(cf)
```

Arguments

`cf`	Correlation function container of class 'raw_cf'

Value

The return value is a list with elements

`value` central value

`dvalue` statistical error

`ddvalue` error of the statistical error

`tauint` auto-correlation time estimate

`dtauint` error of auto-correlation time estimate

Each of these is in turn an array of dimension `c( cf$nts,cf$dim )` and hance lacks the first dimension index compared for `cf$data`.

`weight.cf` *Weight a correlation function*

Description

Weights a correlation function with the given energy difference ΔE such that the function is first multiplied with $\exp(\Delta E t) + c\exp(\Delta E \cdot (Time - t)$.

Usage

```
weight.cf(cf, energy_difference_val, energy_difference_boot, cosh_factor,
  offset = 0, inverse = FALSE)
```

Arguments

`cf`	cf_orig and possibly cf_boot object.
`energy_difference_val`	numeric. A single energy value ΔE for the weighting.
`energy_difference_boot`	numeric vector. Samples for the energy difference value.
`cosh_factor`	integer, either `+1` or `-1`. Determines the sign $c$ in the weight factor.
`offset`	integer. Offset for the time $t$, needed for the reweighting after a shift.
`inverse`	boolean. If `TRUE` apply inverse weight.

Value

Returns an object of class `cf`, see cf.

`weight_shift_reweight.cf` *Weight-shift-reweight a correlation function*

Description

The correlation function is weighted with `weight.cf`, then shifted, and then weighted again with the inverse weighting factor.

Usage

```
weight_shift_reweight.cf(cf, energy_difference_val, energy_difference_boot,
  cosh_factor)
```

Arguments

`cf`	cf_orig and possibly cf_boot object.
`energy_difference_val`	numeric. A single energy value ΔE for the weighting.
`energy_difference_boot`	numeric vector. Samples for the energy difference value.
`cosh_factor`	integer, either `+1` or `-1`. Determines the sign $c$ in the weight factor.

Value

Returns an object of class `cf`, see cf.

zetazp *Computes the running of Z_P from scale mu0 to scale mu2*

Description

Computes the running of the renomalisation constant Z_P from scale μ_0 to scale μ_2 in the renomalisation schema RI' for $N_f = 2$ only. The running is done using perturbation theory up to $\alpha_s **3$ order. The corresponding values of α_s at the scales μ_0 and μ_2 are needed as input, see `alphas`.

Usage

```
zetazp(zp0, alpha0, alpha2, nl = 3)
```

Arguments

`zp0`	initial value of Z_P
`alpha0`	α_s at initial scale
`alpha2`	α_s at final scale
`nl`	order in PT, range 0 to 3

Value

returns the value of Z_P at scale mu2 in the RI' scheme

Author(s)

Carsten Urbach, <curbach@gmx.de>

See Also

`alphas`

Examples

```
al2 <- alphas(mu = 3.0, nl = 3, lam0 = 0.250, Nc = 3, Nf = 2)
al0 <- alphas(mu = 2.0, nl = 3, lam0 = 0.250, Nc = 3, Nf = 2)
zetazp(zp0 = 0.6, alpha0 = al0, alpha2 = al2, nl = 3)
```

Index