Package: sddinf (via r-universe)

November 20, 2026

Type Package

Title Direct Estimation and Inference for Spectral Differential Networks

Version 0.1.0

Description Estimates and performs inference on the difference between two high-dimensional inverse spectral densities, which describes how the frequency-domain partial correlation network of a multivariate time series changes across two conditions. The difference is estimated directly with the lasso penalized D-trace loss of Yuan et al. (2017) <doi:10.1093/biomet/asx049>, and de-biased confidence intervals and p-values for individual edges are computed following Hellstern, Kim and Shojaie (2026) <doi:10.48550/arXiv.2609.13609>. Also provides multivariate spectral density estimates based on Welch's method and simulated example data.

License GPL (>= 3)

Encoding UTF-8

Imports clime, data.table, glassoFast, gsignal, Matrix, matrixcalc, Rcpp (>= 1.0.11), stats

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

LinkingTo Rcpp, RcppEigen

RoxygenNote 7.3.1

NeedsCompilation yes

Author Michael Hellstern [aut, cre, cph], Huili Yuan [ctb, cph] (Author of the D-trace loss C code in src/dtl.c, ported from the archived 'Difdtl' package)

Maintainer Michael Hellstern <mikeh1@uw.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-20 13:14:11 UTC

RemoteUrl https://github.com/cran/sddinf

RemoteRef HEAD

RemoteSha ce2eafba29aa9244bafa920d4ecb469d58a2486a

Contents

sddinf-package . . . 2
contract_to_imaginary . . . 3
expand_to_real . . . 4
gen_nonzero_idxs . . . 5
mvspec_welch . . . 5
sdd . . . 7
sdd_inference . . . 8
sim_example_data . . . 12

Index **14**

`sddinf-package` *sddinf: Direct Estimation and Inference for Spectral Differential Networks*

Description

Direct estimation and inference for the difference between two high-dimensional inverse spectral densities. The inverse spectral density describes the frequency-domain partial correlation network of a multivariate time series, so its difference describes how that network changes between two conditions (e.g. brain connectivity before and after stimulation).

Workflow

1. Start from an `n` x `p` time series for each condition, or simulate example data with `sim_example_data`.
2. Estimate the spectral density matrix of each condition with `mvspec_welch`.
3. Estimate the difference of the inverse spectral densities at a chosen frequency with `sdd`, which uses the lasso penalized D-trace loss and selects the penalty by eBIC.
4. Choose the entries to test, e.g. the non-zero entries of the estimate with `gen_nonzero_idxs`.
5. Compute de-biased estimates, confidence intervals and p-values with `sdd_inference`.

Author(s)

Maintainer: Michael Hellstern <mikeh1@uw.edu> [copyright holder]

Other contributors:

- Huili Yuan (Author of the D-trace loss C code in src/dtl.c, ported from the archived 'Difdtl' package) [contributor, copyright holder]

References

Hellstern, M., Kim, B. and Shojaie, A. (2026). Assumption-lean inference for spectral differential network analysis of high-dimensional time series. arXiv preprint. doi:10.48550/arXiv.2609.13609

Yuan, H., Xi, R., Chen, C. and Deng, M. (2017). Differential network analysis via lasso penalized D-trace loss. *Biometrika*, 104(4), 755–770. doi:10.1093/biomet/asx049

Examples

```
set.seed(1)
n <- 2000
p <- 10
exdata <- sim_example_data(n, p)
mspecsX <- mvspec_welch(exdata$X$data, window = rep(1, floor(sqrt(n))))
mspecsY <- mvspec_welch(exdata$Y$data, window = rep(1, floor(sqrt(n))))

# difference of inverse spectral densities at the second frequency
sdd_est <- sdd(mspecsY$specs[[2]], mspecsX$specs[[2]])

# test the first few non-zero entries of the estimate
idxs <- head(gen_nonzero_idxs(sdd_est$sdd), 5)
inf <- sdd_inference(mspecsY$specs[[2]], mspecsX$specs[[2]], sdd_est, idxs,
  verbose = FALSE)
inf[, c("infI", "infJ", "part", "theta_tilde", "pval")]
```

`contract_to_imaginary` *Contract expanded complex valued matrix back to complex space*

Description

Opposite of expand_to_real.

Usage

```
contract_to_imaginary(spec)
```

Arguments

`spec`	`2i` x `2j` real-valued matrix

Value

`i` x `j` complex-valued matrix

Examples

```
set.seed(1)
n <- 1000
p <- 20
exdata <- sim_example_data(n,p)
Xdata <- ts(exdata$X$data, frequency = 10)
mspecs <- mvspec_welch(Xdata, window = rep(1, floor(sqrt(n))),
overlap = 0, fs = frequency(Xdata), detrend = "none", range = "whole")
realmspecs1 <- expand_to_real(mspecs$specs[[1]]$spec)
immspecs1 <- contract_to_imaginary(realmspecs1)
```

---

expand_to_real *Expand complex valued matrix to real space*

Description

Expansion occurs by computing `rbind(cbind(Re(spec), -Im(spec)), cbind(Im(spec), Re(spec)))`.

Usage

```
expand_to_real(spec)
```

Arguments

spec	`i` x `j` complex-valued matrix

Value

`2i` x `2j` real-valued matrix

Examples

```
set.seed(1)
n <- 1000
p <- 20
exdata <- sim_example_data(n,p)
Xdata <- ts(exdata$X$data, frequency = 10)
mspecs <- mvspec_welch(Xdata, window = rep(1, floor(sqrt(n))),
overlap = 0, fs = frequency(Xdata), detrend = "none", range = "whole")
realmspecs1 <- expand_to_real(mspecs$specs[[1]]$spec)
```

gen_nonzero_idxs *Generate data.frame of non-zero entries for inference*

Description

This function generates a data.frame of the non-zero entries of the input matrix `est_mat` that is suitable to pass to sdd_inference.

Usage

```
gen_nonzero_idxs(est_mat, lower_tri = TRUE)
```

Arguments

`est_mat`	Matrix. Typically output from sdd. See examples.
`lower_tri`	Boolean (default is `TRUE`). If `TRUE`, only return the non-zero entries of the lower triangular portion of `est_mat`. Includes diagonals.

Value

Data.frame where each row represents a non-zero entry of `est_mat` and the columns are the i indices (`infI`), j indices (`infJ`), and `part = "b"`. `Part = "b"` performs inference on both the real and imaginary components of an entry simultaneously. See sdd_inference.

Examples

```
set.seed(1)
n <- 1000
p <- 20
exdata <- sim_example_data(n,p)
mspecsX <- mvspec_welch(exdata$X$data, window = rep(1, floor(sqrt(n))),
overlap = 0, fs = 1, detrend = "none", range = "whole")
mspecsY <- mvspec_welch(exdata$Y$data, window = rep(1, floor(sqrt(n))),
overlap = 0, fs = 1, detrend = "none", range = "whole")
sdd_est <- sdd(mspecsX$specs[[2]], mspecsY$specs[[2]])
idxs = gen_nonzero_idxs(sdd_est$sdd)
```

mvspec_welch *Multivariate spectral estimates based on Welch's method*

Description

This function uses pwelch in the `gsignal` to form the cross-spectral density matrix from the components of pwelch. Some of the default arguments from pwelch are changed to be suitable for SDD inference. It is also important to note that compared to pwelch this function scales the final spectral density by `1/(2pi)`. That is, the multivariate spectral densities returned by this function will be `1/(2pi)` times those in pwelch.

Usage

```
mvspec_welch(
  x,
  window,
  overlap = 0,
  nfft = if (length(window) == 1L) window else length(window),
  fs = 1,
  detrend = "none",
  range = "whole"
)
```

Arguments

x	n x p data matrix.
window	Window function to use. See the same argument in pwelch.
overlap	Numeric scalar (default is 0). Numeric scalar between 0 and 1 specifying segment overlap. See the same argument in pwelch.
nfft	Integer scalar (default is length(window)). Integer scalar representing FFT length. See the same argument in pwelch.
fs	Integer scalar (default is 1). Integer scalar representing sampling frequency in Hertz. See the same argument in pwelch.
detrend	Character string (default is "none"). Character string specifying detrending. See the same argument in pwelch.
range	Character string (default is "whole"). Character string specifying range. See the same argument in pwelch.

Value

List containing:

specs	List. Elements are (1) spec: estimated spectral density matrix. See pwelch. (2) freq: frequency at which spec is estimated (3) orig.n: number of samples used in computing spec, (4) smoothing span of kernel used. Output is structured this way to easily pass to upstream functions like sdd.
window	Value of window argument.
overlap	Value of overlap argument.
nfft	Value of nfft argument.
fs	Value of fs argument.
detrend	Value of detrend argument.
range	Value of range argument.

Examples

```
set.seed(1)
n <- 1000
p <- 20
exdata <- sim_example_data(n,p)
# get data and specify frequency of 10 Hz (i.e. 10 samples per second)
Xdata <- ts(exdata$X$data, frequency = 10)
mspecs <- mvspec_welch(Xdata, window = rep(1, floor(sqrt(n))),
overlap = 0, fs = frequency(Xdata), detrend = "none", range = "whole")
```

---

sdd *Generate SDD estimate of* $spec1\hat{}-1 - spec2\hat{}-1$

Description

Generates SDD estimate of $spec1^{-1} - spec2^{-1}$ for variety of penalty values and chooses the optimal one based on eBIC. Can handle real or complex valued inputs. If inputs are complex valued they will first be expanded to the real space before estimating the difference. After estimation, the difference will be reformed in the complex space. While the intended use is to estimate the difference at the same frequency, estimating differences across frequencies is also supported.

Usage

```
sdd(spec1, spec2, nlambda = 20, lamseq = NULL, return_all = FALSE)
```

Arguments

spec1	List. Typically this is output from mvspec_welch. See examples. However, it can also be any list with entries: 1. spec: real or complex valued matrix. 2. freq: frequency at which spec is computed 3. orig.n: Number of samples used in computing spec.
spec2	List. Same as spec1 but for condition 2.
nlambda	Integer (default is 20). Number of lambda values to consider. Ignored when lamseq is supplied.
lamseq	Numeric vector (default is NULL). User provided lambda sequence. Note that if both nlambda and lamseq are specified the default will be to use lamseq.
return_all	Boolean (default is FALSE). Whether or not to return SDD estimates for each lambda or only the best chosen via eBIC.

Value

List of SDD estimates. When return_all = TRUE, list is of length nlambda or length(lamseq). When return_all=FALSE and there are multiple estimates with the same eBIC value, then all estimates are returned. Otherwise, only the optimal estimate is returned. Each list of SDD estimates contains:

sdd	Matrix representing sdd estimate. Matrix is either real or complex-valued depending on input type spec1$spec, spec2$spec.
eBIC	eBIC value of sdd.
lam	Lambda value at which sdd is estimated.
freq	Frequency at which sdd is estimated. If both spec1$spec and spec2$spec are same frequency this will be that value. If different it will be c(spec1$freq, spec2$freq).
lam_num	Which number out of lamseq or nlambda is lam.

Examples

```
set.seed(1)
n <- 1000
p <- 20
exdata <- sim_example_data(n,p)
mspecsX <- mvspec_welch(exdata$X$data, window = rep(1, floor(sqrt(n))),
overlap = 0, fs = 1, detrend = "none", range = "whole")
mspecsY <- mvspec_welch(exdata$Y$data, window = rep(1, floor(sqrt(n))),
overlap = 0, fs = 1, detrend = "none", range = "whole")
sdd_est <- sdd(mspecsX$specs[[2]], mspecsY$specs[[2]])
```

---

sdd_inference *Perform inference on SDD estimate of* $spec1^{-1} - spec2^{-1}$

Description

Generates confidence intervals and p-values for a difference in inverse spectral densities (i.e. $spec1^{-1} - spec2^{-1}$) using one of several available methods (see ee_type). Computational efficient implementations are available by specifying n_restrict.

Usage

```
sdd_inference(
  spec1,
  spec2,
  sdd_est,
  idxs,
  nlambda = 20,
  ee_type = "s1Left",
  v_hat_method = "approximate_clime",
  alpha = 0.05,
  lambda_min_ratio = NULL,
  n_restrict = 0.99,
  n_max = 500,
  useMtopN = floor((2 * dim(spec1[["spec"]])[1])^(1/2)),
  verbose = TRUE,
  warn = TRUE
)
```

Arguments

spec1	List. Typically this is output from mvspec_welch. However, it can also be any list with entries: 1. spec: real or complex valued matrix. 2. orig.n: integer number of samples used in computing spec. 3. smooth_span: integer smoothing span used to estimate the spectral density.
spec2	List. same as spec1 but for condition 2.
sdd_est	List. Typically this is output from sdd. It can also be any list with entries: 1. sdd: real or complex valued matrix representing an estimate of the difference spec1^(-1) - spec2^(-1). If complex, will be expanded to real space first. 2. freq: frequency at which the difference is estimated. This should also correspond to the same frequency spec1/spec2 are estimated at.
idxs	Data.frame. Should contains 3 columns: 1. infI: i indices of entries in sdd_est$sdd you want to do inference on. 2. infJ: j indices. 3. part: either "i", "r", "b". Represents the part of the (infI,infJ) entry that you wish to perform inference on. "i" corresponds to the imaginary part only, "r" corresponds the real part only, and "b" corresponds to both. For "i" and "r" the null hypothesis tested is that imaginary and real parts are 0 respectively. All confidence intervals and p-values are two-sided. In the case of "b", the null hypothesis tested is that the (infI, infJ) edge is 0. That is that both the real and imaginary components are 0. Specifically an asymptotic chi-square test is used test H0: real^2 + imaginary^2 = 0. This is a one-sided test.
nlambda	Integer (default is 20). Number of tuning parameter values used to estimate the sparse projections $\hat{v}$.
ee_type	Character (default is "s1Left"). Which estimating equation type used to perform inference. Options are c("s1Left", "s1Right", "s1Left_Right", "symm"). If "s1Left_Right", both "s1Left" and "s1Right" will be used. The theta_tilde, the de-biased estimate, is taken as the mean of "s1Left", "s1Right". The p-values are generated as the maximum of the p-values. The confidence lower limits and upper limits are taken as the min and max (of "s1Left" and "s1Right") respectively, and the test statistic is the min.
v_hat_method	Character (default is "approximate_clime"). Which method used to estimate $\hat{v}$. For ee_type %in% c("s1Left", "s1Right", "s1Left_Right") this can be either "approximate_clime" or "exact". For ee_type = "symm" this can only be "exact". For p > 20, "exact" can be extremely slow and can cause memory issues. Therefore, we recommend "approximate_clime".
alpha	Numeric (default is 0.05). Alpha level of hypothesis testing.
lambda_min_ratio	Numeric (default is ifelse(p >= 100, 0.01, 0.0001)). This represents the smallest value of lambda used in estimating $\hat{v}$ as a fraction of the maximum lambda. for large p we recommend larger values of lambda_min_ratio as this enforces more sparsity by default and speeds up computation.

`n_restrict` Numeric (default is `0.99`). Must be between 0 and 1. If NULL, no restrictions made and uses all entries in $\hat{v}$. Used to reduce computation time and works by restricting $\hat{v}$ to only the largest values that explain `n_restrict` proportion of the total sum of squares. Note that computational complexity scales with the number of non-zero components in $\hat{v}^2$ so reducing the number of non-zero components in $\hat{v}$ by a factor of 2 roughly reduces computation time by a factor of 4. Thus, this can be used to make variance estimation much faster. A value of `n_restrict = 0.99` improves computation time massively and has be shown to have little to no impact on performance in terms of Type I error, power, coverage. When using `v_hat_method = "approximate_clime"`, there are many very small entries (e.g. < 1e-6) so it is recommended to use this parameter or `n_max` in those cases.

`n_max` Numeric (default is `500`). Restricts $\hat{v}$ to only the largest `n_max` entries in absolute value. All others are set to 0. If NULL no restrictions made and uses all entries in $\hat{v}$. Similar to `n_restrict`, reduces computation time by reducing the number of non-zero components in $\hat{v}$.

`useMtopN` Numeric (default is `floor((2*p)^(1/2))`). If NULL or negative, no restrictions made and uses all entries in M. Also used to reduce computation time by restricting the M matrices in the asymptotic variance of the de-biased estimate. This parameter restricts M matrix to only use the `useMtopN` largest entries in absolute value. Unlike `n_restrict` and `n_max`, this can heavily impact Type I error, power, and coverage. We recommend to not reduce this further than the default value.

`verbose` Boolean (default is `TRUE`). Whether or not to print the number of components used to estimate the asymptotic variance. As a result when the number of components is large (approx. > 200) computation can be slow.

`warn` Boolean (default is `TRUE`). Whether or not to warn the user if asymptotic variance computation will take a long time.

Value

Data.table. Columns are

`infI` i index of the entry of sdd that we are performing inference on.

`infJ` j index of the entry of sdd that we are performing inference on.

`part` Options are one of `"i"`, `"r"`, `"b"`. Which part of the `(infI, infJ)` entry of sdd are we performing inference on. `"i"` = imaginary, `"r"` = real, `"b"` = both.

`theta_tilde` Estimates of (infI, infJ) entry. These differ from `theta_hat` in that these are the de-biased estimates.

`ci_lower` Lower part of the CI. For `part = "b"` this is NA.

`ci_upper` Upper part of the CI. For `part = "b"` this is NA.

`pval` p-value.

`test_stat` Test statistic

`asympVars` Asymptotic variance estimate. For `part = "b"` this is NA.

`asympNscale` Asymptotic scaling in n. This is `sqrt(n/smooth_span)`.

n_var_est_used	Number of components used to estimate asymptotic variance.
freq	Frequency for which we are performing inference. sdd$freq
p	Number of variables.
n	Number of observations.
ee_type	Same as ee_type input.
v_hat_method	Same as v_hat_method input.
alpha	Same as alpha input.

Examples

```
set.seed(1)
n <- 2000
p <- 10
exdata <- sim_example_data(n,p)
mspecsX <- mvspec_welch(exdata$X$data, window = rep(1, floor(sqrt(n))),
overlap = 0, fs = 1, detrend = "none", range = "whole")
mspecsY <- mvspec_welch(exdata$Y$data, window = rep(1, floor(sqrt(n))),
overlap = 0, fs = 1, detrend = "none", range = "whole")

spec1 <- mspecsY$specs[[2]]
spec2 <- mspecsX$specs[[2]]
sdd_est <- sdd(spec1, spec2)
idxs_use <- data.frame(infI = c(2, 8,6,8,9, 8,8,9), infJ = c(1, 9, 2, 10, 10, 9, 10, 10),
part = c("r","b", "b","r","r", "i","i","i"))
# s1Right - restricted variance est
inf_s1Right <- sdd_inference(spec1, spec2, sdd_est, nlambda=20, idxs = idxs_use,
ee_type = "s1Right")

# s1Left - restricted variance est
inf_s1Left <- sdd_inference(spec1, spec2, sdd_est, nlambda=20, idxs = idxs_use,
ee_type = "s1Left")

# s1Left_Right - restricted variance est
inf_s1Left_Right <- sdd_inference(spec1, spec2, sdd_est, nlambda=20, idxs = idxs_use,
ee_type = "s1Left_Right")


# symm - restricted variance est
inf_symm <- sdd_inference(spec1, spec2, sdd_est, nlambda=20, idxs = idxs_use,
ee_type = "symm", v_hat_method = "exact")

# unrestricted variance est - can be very slow if components used to estimate variances is large
inf_symm_unr <- sdd_inference(spec1, spec2, sdd_est, idxs_use, nlambda=20,
ee_type = "symm", v_hat_method = "exact", n_restrict = NULL, n_max = NULL, useMtopN = NULL)
```

`sim_example_data` *Simulate example data for differential network analysis*

Description

Simulates data for two conditions (X and Y) from a VAR1 process.

Usage

```
sim_example_data(
  n,
  p,
  minA = 0.4,
  maxA = 0.8,
  propNonZero_full = 0.05,
  propNonZero_add = 0.6,
  addMatDim = 3
)
```

Arguments

`n`	Sample size.
`p`	Dimension.
`minA`	Minimum value of transition matrix.
`maxA`	Maximum value of transition matrix.
`propNonZero_full`	Proportion of entries of the `p - addMatDim` block that are non-zero.
`propNonZero_add`	Proportion of entries of the `addMatDim` block that are non-zero.
`addMatDim`	Size of additional block.

Details

Data is simulated from e.g.

$$X_t = AX_{t-1} + e_t$$

where e_t is are independent standard normal errors.

The transition matrices, `A`, in both conditions will be block diagonal with two blocks. The first is size `p-addMatDim` and the second is of size `addMatDim`. The transition matrix in condition Y will be the same as conditions X except the last `addMatDim` block is multiplied by -1. Entries are generated from either a Uniform(`minA`,`maxA`) or Uniform(`-maxA`, `-minA`) each with equal probability. Density of the `p-addMatDim` and `addMatDim` blocks are controlled with `propNonZero_full` and `propNonZero_add` respectively.

Value

List containing:

X	List of transition matrix (A) and data.
Y	List of transition matrix (A) and data.

Examples

```
set.seed(1)
exdata <- sim_example_data(1000,20)
Xdata <- exdata$X$data
Ydata <- exdata$Y$data
```

Index

contract_to_imaginary, 3

expand_to_real, *3*, 4

gen_nonzero_idxs, *2*, 5

mvspec_welch, *2*, 5, *7*, *9*

pwelch, *5, 6*

sdd, *2*, *5, 6*, 7, *9*
sdd_inference, *2*, *5*, 8
sddinf *(*sddinf-package*)*, 2
sddinf-package, 2
sim_example_data, *2*, 12