**Package `networkTomography`**

February 20, 2015

**Type** Package  
**Title** Tools for network tomography  
**Version** 0.3  
**Author** Alexander W Blocker, Paul Koulick, Edoardo Airoldi  
**Maintainer** Alexander W Blocker <ablocker@gmail.com>  

**Description** networkTomography implements the methods developed and evaluated in Blocker and Airoldi (2011) and Airoldi and Blocker (2012). These include the authors’ own dynamic multilevel model with calibration based upon a Gaussian state-space model in addition to implementations of the methods of Tebaldi & West (1998; Poisson-Gamma model with MCMC sampling), Zhang et al. (2002; tomogravity), Cao et al. (2000; Gaussian model with mean-variance relation), and Vardi (1996; method of moments). Data from the 1router network of Cao et al. (2000), the Abilene network of Fang et al. (2007), and the CMU network of Blocker and Airoldi (2011) are included for testing and reproducibility.

**License** LGPL-2  
**LazyLoad** yes  
**URL** [https://github.com/awblocker/networkTomography](https://github.com/awblocker/networkTomography)  

**Depends** R (>= 2.10.0),  
**Imports** coda (>= 0.11-3), igraph (>= 0.5), KFAS (>= 1.0), limSolve (>= 1.4), plyr, Rglpk (>= 0.3),  

**NeedsCompilation** yes  
**Repository** CRAN  
**Date/Publication** 2014-01-10 07:28:23

**R topics documented:**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>abilene</td>
<td>2</td>
</tr>
<tr>
<td>agg</td>
<td>3</td>
</tr>
<tr>
<td>bayesianDynamicFilter</td>
<td>4</td>
</tr>
<tr>
<td>bell.labs</td>
<td>6</td>
</tr>
<tr>
<td>buildPrior</td>
<td>7</td>
</tr>
</tbody>
</table>
Abilene data from Fang et al. (2007)

Data from the 12 node Abilene network from Fang et al. (2007). Both the OD flows and the topology correspond to the actual network. This is the X1 dataset from the given paper.
Usage

abilene

Objects

The list abilene, which contains several objects:

- A, the routing matrix for this network (truncated for full row rank)
- X, a matrix of origin-destination flows formatted for analysis
- Y, a matrix of link loads formatted for analysis
- A.full, the routing matrix for this network without truncation for full row rank
- Y.full, a matrix of link loads corresponding to codeA.full

In this data, we have A %*% t(X) == t(Y) and A.full %*% t(X) == t(Y.full)

Variables

The list abilene contains the following:

- The routing matrix A. The columns of this matrix correspond to individual OD flows (the columns of X), and its rows correspond to individual link loads (the columns of Y).
- The OD matrix X. Columns correspond to individual OD flows, and the rows correspond to observations.
- The link load matrix Y. Columns of the Y matrix correspond to individual link loads, and the rows correspond to observations.
- The routing matrix A.full. This is the complete routing matrix before reduction for full row-rank.
- The link load matrix Y.full, corresponding to A.full.

References


---

agg

Function to aggregate results from matrix to matrix

Description

Defaults to mean, SD, limits, and given quantiles. Used to limit memory consumption from MCMC runs.
Usage

agg(mat, q = c(0.05, 0.16, 0.5, 0.84, 0.95))

Arguments

mat: input numeric matrix to summarize
q: quantiles of mat’s columns to provide in summary matrix

Value

matrix with each row corresponding to a summary measure and each column corresponding to a column of mat

Examples

mat <- matrix(rnorm(5e3), ncol=5)
agg(mat)

bayesianDynamicFilter
Function for inference with multilevel state-space model

Description

Particle filtering with sample-resample-move algorithm for multilevel state-space model of Blocker & Airoldi (2011). This has log-normal autoregressive dynamics on OD intensities, log-normal emission distributions, and truncated normal observation densities. This can return full (all particles) output, but it is typically better to aggregate results as you go to reduce memory consumption. It can also run forward or backward filtering for smoothing. These results are combined via a separate function for smoothing; however, this procedure typically performs poorly due to differences between the distributions of particles from forward and reverse filtering.

Usage

bayesianDynamicFilter(Y, A, prior, lambda0, sigma0, phi0, rho = 0.1, tau = 2, m = 1000, verbose = FALSE, Xdraws = 5 * m, Xburnin = m, Movedraws = 10, nThresh = 10, aggregate = FALSE, backward = FALSE, tStart = 1)

Arguments

Y: matrix (n x l) of observed link loads over time, one observation per row
A: routing matrix (l x k) for network; must be of full row rank
prior: list containing priors for lambda and phi; must have
  • mu, a matrix (n x k) containing the prior means for the log-change in each lambda at each time
• sigma, a matrix (n x k) containing the prior standard deviations for the log-change in each lambda at each time
• a list phi, containing the numeric prior df and a numeric vector scale of length n
lambda0 numeric vector (length k) of time 0 prior means for OD flows
sigma0 numeric vector (length k) of time 0 prior standard deviations for OD flows
phi0 numeric starting value for phi at time 0
rho numeric fixed autoregressive parameter for dynamics on lambda; see reference for details
tau numeric fixed power parameter for variance structure on truncated normal noise; see reference for details
m integer number of particles to use
verbose logical activates verbose diagnostic output
xdraws integer number of draws to perform for xsample RDA
xburnin integer number of burnin draws to discard for xsample proposals RDA in addition to baseline number of draws
Movedraws integer number of iterations to run for each move step
nthresh numeric effective number of independent particles below which redraw will be performed
aggregate logical to activate aggregation of MCMC results; highly
backward logical to activate reverse filtering (for smoothing
tStart integer time index to begin iterations from

Value

list containing:
• xList
• lambdaList
• phiList
• y
• rho
• prior
• n
• l
• k
• A
• A_qr
• A1
• A1_inv
• A2
bell.labs

- nEff
- tStart
- backward
- aggregate

References


See Also

Other bayesianDynamicModel: buildPrior; move_step

bell.labs  Bell Labs 1router data from Cao et al. (2000)

Description

Data from 4-node network with star topology collected from Bell Labs; used in Cao et al. (2000).

Usage

bell.labs

Objects

The list bell.labs, which contains several objects:

- A, the routing matrix for this network (truncated for full row rank)
- df, a data.frame with all data
- X, a matrix of origin-destination flows formatted for analysis
- Y, a matrix of link loads formatted for analysis
- tvec, a vector of times

In this data, we have A %*% t(X) == t(Y).
Variables

The list bell.labs contains the following:

- The routing matrix A. The columns of this matrix correspond to individual OD flows (the columns of X), and its rows correspond to individual link loads (the columns of Y).
- The data.frame df, containing
  - value, level of traffic recorded
  - nme, name of flow or load
  - method, whether flow was directly observed or inferred (all observed)
  - time, time of observation
  - od, flag for origin-destination vs. link loads
  - orig, origin of flow or load
  - dest, destination of flow or load
  - node, node involved in flow or load
- The OD matrix X. Columns correspond to individual OD flows, and the rows correspond to observations.
- The link load matrix Y. Columns of the Y matrix correspond to individual link loads, and the rows correspond to observations.
- The vector tvec, containing the time in decimal hours since midnight for each observation.

References


buildPrior

Construct prior from calibration model estimates

Description

Builds prior from appropriately structured output of the calibration model from Blocker & Airoldi (2011). Handles all formatting so result can be fed directly to bayesianDynamicFilter.

Usage

buildPrior(xHat, varHat, phiHat, Y, A, rho = 0.9, phiPriorDF = ncol(A)/2, backward = FALSE, lambdaMin = 1, ipfp.maxit = 1e+06, ipfp.tol = 1e-06)
Arguments

- **xHat**: matrix \((n \times k)\) of estimates for OD flows from calibration model, one time point per row
- **varHat**: matrix \((n \times k)\) of estimated variances for OD flows from calibration, one time point per row
- **phiHat**: numeric vector (length \(n\)) of estimates for \(\phi\) from calibration model
- **Y**: matrix \((n \times l)\) of observed link loads, one time point per row
- **A**: routing matrix \((l \times k)\) for network; must be of full row rank
- **phiPriorDf**: numeric prior convolution parameter for independent inverse-gamma priors on \(\phi_t\)
- **rho**: numeric fixed autoregressive parameter for dynamics on \(\lambda\); see reference for details
- **backward**: logical to activate construction of reversed prior (for smoothing applications)
- **lambdaMin**: numeric value at which to floor estimated OD flows for prior construction
- **ipfp.maxit**: integer maximum number of iterations for IPFP
- **ipfp.tol**: numeric tolerance for convergence of IPFP iterations

Value

- list containing priors for \(\lambda\) and \(\phi\), consisting of:
  - **mu**: a matrix \((n \times k)\) containing the prior means for the log-change in each \(\lambda\) at each time
  - **sigma**: a matrix \((n \times k)\) containing the prior standard deviations for the log-change in each \(\lambda\) at each time
  - **a list phi**, containing the numeric prior df and a numeric vector scale of length \(n\)

References


See Also

Other bayesianDynamicModel: `bayesianDynamicFilter`; `move_step`
buildRoutingMat

Build routing matrices for linked star topologies; that is, a set of star-topology networks with links between a subset of routers

Description
Build routing matrices for linked star topologies; that is, a set of star-topology networks with links between a subset of routers

Usage
buildRoutingMat(nVec, Cmat)

Arguments
- **nVec**: integer vector containing number of nodes in each sub-network (length m)
- **Cmat**: matrix (m x m) containing a one for each linked sub-network; only upper triangular part is used

Value
routing matrix of dimension at least 2*sum(nVec) x sum(nVec^2)

See Also
- buildStarMat, which this function depends upon

Examples
```r
nVec <- c(3, 3, 3)
Cmat <- diag(3)
Cmat[1,2] <- Cmat[2,3] <- 1
buildRoutingMat(nVec, Cmat)
```

---

buildRoutingMatrix

Build routing matrix from table of link relationships

Description
Constructs routing matrix from link relationships. Determines routes using (weighted) shortest-path calculation (mirroring OSPF). Currently handles tied paths arbitrarily; will incorporate fractions for tie resolution in next version. Can optionally include aggregate source and destination flows for each node; this can make a major difference for some topologies. Tomogravity methods typically make use of such information, which most routers collect. Note that resulting routing matrix need not be of full row rank.
Usage

buildRoutingMatrix(nodes, src, dest, weights = NULL, agg = FALSE, 
sep = " ", aggChar = ",", verbose = 0)

Arguments

nodes vector (length n) of node identifiers
src vector (length m) of sources, one per link, matched with dest
dest vector (length n) of destination identifiers, one per link, matched with src
weights numeric vector (length m) of weights for each link; used in shortest-path routing 
calculations (roughly OSPF)
agg logical for whether to include aggregate source and destination flows for each 
node
sep character separator between node id’s for link and OD names
aggChar character to indicate aggregate flows; should be distinct from sep
verbose integer level of verbosity; 0 is silent, >=1 are increasing levels of reporting

Value

List consisting of routing matrix A (dense) of dimensions m x n and iGraph object for network topo

Description

Build routing matrix for star network topology

Usage

buildStarMat(n)

Arguments

n integer number of nodes in the network

Value

matrix of dimension 2n x n^2 that transforms OD flows to link loads

Examples

buildStarMat(3)
**calcN**

Compute total traffic from a particular time.

**Description**

Compute total traffic from a particular time.

**Usage**

calcN(yt, A1)

**Arguments**

- **yt**: length-m numeric vectors of observed aggregate flows at a particular time
- **A1**: m x m matrix containing the full-rank portion of the network’s routing matrix, as supplied by `decomposeA`

**Examples**

data(bell.labs)
A.decomp <- decomposeA(bell.labs$A)
total.traffic <- calcN(yt=bell.labs$Y[1,], A=A.decomp$A1)
total.traffic == sum(bell.labs$X[1,])

**calibration_ssm**

Estimation for the linear SSM calibration model of Blocker & Airoldi (2011)

**Description**

Maximum likelihood estimation of the parameters of the calibration model from Blocker & Airoldi (2011) via direct numerical maximization of the marginal log-likelihood. This relies upon efficient Kalman smoothing to evaluate the marginal likelihood, which is provided here by the KFAS package.

**Usage**

calibration_ssm(tme, y, A, Ft, Rt, lambda0, phihat0, tau = 2, w = 11, initScale = 1/(1 - diag(Ft)^2), nugget = sqrt(.Machine$double.eps), verbose = FALSE, logTrans = TRUE, method = "L-BFGS-B", optimArgs = list())
Arguments

time integer time at which to center moving window for estimation

y matrix (n x m) of observed link loads from all times (not just the window used for estimation; one observation per row

A routing matrix (m x k) for network; should be full row rank

Ft matrix (k x k) containing fixed autoregressive parameters for state evolution equation; upper-left block of overall matrix for expanded state

Rt covariance matrix for observation equation; typically small and fixed

lambda0 matrix (n x k) of initial estimates for lambda (e.g. obtained via IPFP)

phihat0 numeric vector (length n) of initial estimates for phi

tau numeric power parameter for mean-variance relationship

w number of observations to use for rolling-window estimation; handles boundary cases cleanly

initScale numeric inflation factor for time-zero state covariance; defaults to steady-state variance setting

nugget small positive value to add to diagonal of state evolution covariance matrix to ensure numerical stability

verbose logical to select verbose output from algorithm

logTrans logical whether to log-transform parameters for optimization. If FALSE, sets method to "L-BFGS-B".

method optimization method to use (in optim calls)

optimArgs list of arguments to append to control argument for optim. Can include all arguments except for fnscale, which is automatically set

Value

list containing lambdahat, a numeric vector (length k) containing the MLE for lambda; phihat, the MLE for phi; xhat, the smoothed estimates of the OD flows for the window used as a k x w matrix; and varhat, a k x w matrix containing the diagonal of the estimated covariance for each OD flow in the window

References


See Also

Other calibrationModel: l1Calibration; mle_filter
Examples

data(bell.labs)

lambda0 <- matrix(1, nrow(bell.labs$Y), ncol(bell.labs$A))
lambda0[100,] <- ipfp(y=bell.labs$Y[100,], A=bell.labs$A,
x0=rep(1, ncol(bell.labs$A)))
phihat0 <- rep(1, nrow(bell.labs$Y))
Ft <- 0.5 * diag_mat(rep(1, ncol(bell.labs$A)))
Rt <- 0.01 * diag_mat(rep(1, nrow(bell.labs$A)))

# Not run
#fit.calibration <- calibration_ssm(tme=100, y=bell.labs$Y, A=bell.labs$A,
#    Ft=Ft, Rt=Rt, lambda0=lambda0,
#    phihat0=phihat0, w=23)

---

cmu

CMU data from Blocker & Airoldi (2011)

Description

Data from the 12 node CMU network used in Blocker & Airoldi (2011). The OD flows are actual, observed traffic from a CMU network. The topology does not, however, correspond to the original network due to security considerations.

Usage
cmu

Objects

The list cmu, which contains several objects:

- A, the routing matrix for this network (truncated for full row rank)
- X, a matrix of origin-destination flows formatted for analysis
- Y, a matrix of link loads formatted for analysis
- A.full, the routing matrix for this network without truncation for full row rank
- Y.full, a matrix of link loads corresponding to codeA.full

In this data, we have A %*% t(X) == t(Y) and A.full %*% t(X) == t(Y.full)

Variables

The list cmu contains the following:

- The routing matrix A. The columns of this matrix correspond to individual OD flows (the columns of X), and its rows correspond to individual link loads (the columns of Y).
- The OD matrix X. Columns correspond to individual OD flows, and the rows correspond to observations.
• The link load matrix $Y$. Columns of the $Y$ matrix correspond to individual link loads, and the rows correspond to observations.
• The routing matrix $A_{\text{full}}$. This is the complete routing matrix before reduction for full row-rank.
• The link load matrix $Y_{\text{full}}$, corresponding to $A_{\text{full}}$.

References


```
decomposeA
```

**Description**

Compute pivoted decomposition of routing matrix $A$ into full-rank and remainder, as in Cao et al. 2000, via the QR decomposition.

**Usage**

```r
decomposeA(A)
```

**Arguments**

- **A**: routing matrix of dimension $m \times k$

**Value**

list containing two matrices: $A_1$ ($m \times m$), a full-rank subset of the columns of $A$, and $A_2$ ($m \times k - m$), the remaining columns

```
diag_ind
```

**Description**

Make vector of 1-dimensional diagonal indices for square matrix

**Usage**

```r
diag_ind(n)
```
**diag_mat**

**Arguments**

- `n` integer dimension of (square) matrix

**Value**

integer vector of length `n` with indices (unidimensional) of square matrix

**See Also**

`diag_mat`

**Examples**

```r
ind <- diag_ind(5)
diag_mat(seq(5))[ind]
```

---

**Description**

Build matrix with supplied vector on diagonal; this is much faster than `diag` due to the use of matrix instead of array

**Usage**

`diag_mat(x)`

**Arguments**

- `x` numeric vector for diagonal

**Value**

matrix of size `length(x) x length(x)` with `x` along diagonal

**See Also**

`diag_ind`

**Examples**

`diag_mat(seq(5))`
**dobj.dxt.tomogravity**  
*Analytic gradient of objective function of Zhang et al. 2003*

**Description**

Requires bounded optimization to maintain positive OD flows, and only those flows that are not deterministically zero should be included in the estimation.

**Usage**

```r
dobj.dxt.tomogravity(xt, yt, A, srcDstInd, lambda)
```

**Arguments**

- `xt`: length-k numeric vector of point-to-point flows
- `yt`: length-m numeric vector of observed aggregate flows
- `A`: m x k routing matrix, yt = A xt
- `srcDstInd`: list of source and destination flow indices corresponding to each point-to-point flow, as produced by `getSrcDstIndices`
- `lambda`: regularization parameter for mutual information prior. Note that this is scaled by the squared total traffic in the objective function before scaling the mutual information prior.

**Value**

numeric vector of length k containing gradient of objective function with respect to xt

---

**getActive**  
*Check for deterministically-known OD flows at single time*

**Description**

Uses xranges from limSolve to find deterministically-known OD flows

**Usage**

```r
getActive(y, A)
```

**Arguments**

- `y`: numeric vector of link loads, dimension m
- `A`: routing matrix of dimension m x k
Value

logical vector of length \( k \); TRUE for unknown OD flows, FALSE for known

Examples

data(bell.labs)
getActive(bell.labs$Y[1,], bell.labs$A)

data(cmu)
src.dst.ind <- getSrcDstIndices(cmu$A.full)
grad_iid

*Compute analytic gradient of Q-function for locally IID EM algorithm of Cao et al. (2000)*

**Description**
Computation of gradient of Q-function with respect to log(c(lambda,phi)) for EM algorithm from Cao et al. (2000) for their locally IID model.

**Usage**

```r
grad_iid(logtheta, c, M, rdiag, epsilon)
```

**Arguments**
- *logtheta*: numeric vector (length k+1) of log(lambda) (1:k) and log(phi) (last entry)
- *c*: power parameter in model of Cao et al. (2000)
- *M*: matrix (n x k) of conditional expectations for OD flows, one time per row
- *rdiag*: numeric vector (length k) containing diagonal of conditional covariance matrix
- *epsilon*: numeric nugget to add to diagonal of covariance for numerical stability

**Value**
numeric vector of same length as logtheta containing calculated gradient

**References**

**See Also**
Other CaoEtAl: Q_iid; Q_smoothed; R_estep; grad_smoothed; locally_iid_EM; m_estep; phi_init; smoothed_EM
`grad_smoothed`  
Compute analytic gradient of $Q$-function for smoothed EM algorithm of Cao et al. (2000)

**Description**
Computes gradient of Q-function with respect to $\log(c(\lambda,\phi))$ for EM algorithm from Cao et al. (2000) for their smoothed model.

**Usage**

```
grad_smoothed(logtheta, c, M, rdiag, eta0, sigma0, V, eps.lambda, eps.phi)
```

**Arguments**
- `logtheta`: numeric vector (length k+1) of $\log(\lambda)$ (1:k) and $\log(\phi)$ (last entry)
- `c`: power parameter in model of Cao et al. (2000)
- `M`: matrix (n x k) of conditional expectations for OD flows, one time per row
- `rdiag`: numeric vector (length k) containing diagonal of conditional covariance matrix
- `eta0`: numeric vector (length k+1) containing value for $\log(c(\lambda, \phi))$ from previous time (or initial value)
- `sigma0`: covariance matrix (k+1 x k+1) of $\log(c(\lambda, \phi))$ from previous time (or initial value)
- `V`: evolution covariance matrix (k+1 x k+1) for $\log(c(\lambda, \phi))$ (random walk)
- `eps.lambda`: numeric small positive value to add to $\lambda$ for numerical stability; typically 0
- `eps.phi`: numeric small positive value to add to $\phi$ for numerical stability; typically 0

**Value**
numeric vector of same length as `logtheta` containing calculated gradient

**References**

**See Also**
Other CaoEtAl: `Q_iid`; `Q_smoothed`; `R_estep`; `grad_iid`; `locally_iid_EM`; `m_estep`; `phi_init`; `smoothed_EM`
gravity  

*Run tomo.gravity estimation on complete time series of aggregate flows*

**Description**

Run tomo.gravity estimation on complete time series of aggregate flows

**Usage**

```
gravity(Y, srcDstInd)
```

**Arguments**

- `Y`  
  
  `n x m` matrix contain one vector of observed aggregate flows per row

- `srcDstInd`  
  
  list of source and destination flow indices corresponding to each point-to-point flow, as produced by `getSrcDstIndices`

**Value**

`Xhat`, a `n x k` matrix containing a vector of estimated point-to-point flows (for each time point) per row

**See Also**

Other gravity: `gravity.fit`

**Examples**

```r
data(cmu)
srcDstInd <- getSrcDstIndices(cmu$A.full)
estimate <- gravity(Y=cmu$Y[1:3,], srcDstInd=srcDstInd)
```

---

**gravity.fit**  

*Gravity estimation for a single time point*

**Description**

Gravity estimation for a single time point

**Usage**

```
gravity.fit(yt, srcDstInd)
```
Arguments

- `yt`: length-m numeric vector of observed aggregate flows at time t
- `srcDstInd`: list of source and destination flow indices corresponding to each point-to-point flow, as produced by `getSrcDstIndices`.

Value

- `xhat`, a numeric vector of length k providing gravity estimates of the point-to-point flows of interest.

See Also

- Other gravity: `gravity`

Examples

```r
data(cmu)
srcDstInd <- getSrcDstIndices(cmu$A.full)
estimate <- gravity.fit(yt=cmu$Y.full[,], srcDstInd=srcDstInd)
```

---

**Description**

Use IPFP starting from `x0` to produce vector `x` s.t. `Ax = y` within tolerance. Need to ensure that `x0` >= 0.

**Usage**

```r
ipfp(y, A, x0, tol = .Machine$double.eps, maxit = 1000, verbose = FALSE, full = FALSE)
```

**Arguments**

- `y`: numeric constraint vector (length `nrow`)
- `A`: constraint matrix (nrow x ncol)
- `x0`: numeric initial vector (length `ncol`)
- `tol`: numeric tolerance for IPFP; defaults to `Machine$double.eps`
- `maxit`: integer maximum number of iterations for IPFP; defaults to 1e3
- `verbose`: logical parameter to select verbose output from C function
- `full`: logical parameter to select full return (with diagnostic info)

**Value**

- If not full, vector of length `ncol` containing solution obtained by IPFP. If full, list containing solution (as `x`), number of iterations (as `iter`), and norm of `Ax - y` (as `errNorm`).
Examples

A <- buildStarMat(3)
x <- rgamma(ncol(A), 10, 1/100)
y <- A %*% x
x0 <- x * rgamma(length(x), 10, 10)
ans <- ipfp(y, A, x0, full=TRUE)
print(ans)
print(x)

llCalibration

Evaluate marginal log-likelihood for calibration SSM

Description

Evaluates marginal log-likelihood for calibration SSM of Blocker & Airoldi (2011) using Kalman filtering. This is very fast and numerically stable, using the univariate Kalman filtering and smoothing functions of kfas with Fortran implementations.

Usage

llCalibration(theta, Ft, yt, Zt, Rt, k = ncol(Ft), tau = 2,
              initScale = 1/(1 - diag(Ft)^2), nugget = sqrt(.Machine$double.eps))

Arguments

theta numeric vector (length k+1) of parameters. theta[-1] = log(lambda), and theta[1] = log(phi)
Ft evolution matrix (k x k) for OD flows; include fixed
yt matrix (k x n) of observed link loads, one observation per column
Zt observation matrix for system; should be routing matrix A
Rt covariance matrix for observation equation; typically small and fixed
k integer number of OD flows to infer
tau numeric power parameter for mean-variance relationship
initScale numeric inflation factor for time-zero state covariance; defaults to steady-state variance setting
nugget small positive value to add to diagonal of state evolution covariance matrix to ensure numerical stability

Value

t numeric marginal log-likelihood obtained via Kalman smoothing

References

**locally_iid_EM**

**See Also**

Other calibrationModel: `calibration_ssm`; `mle_filter`

| locally_iid_EM | Run EM algorithm to obtain MLE for locally IID model of Cao et al. (2000) |

**Description**

Runs EM algorithm to compute MLE for the locally IID model of Cao et al. (2000). Uses numerical optimization of Q-function for each M-step with analytic computation of its gradient.

**Usage**

```r
deploy_iid_EM(Y, A, lambda0, phi0 = NULL, c = 2, maxiter = 1000, 
tol = 1e-06, epsilon = 0.01, method = "L-BFGS-B", checkActive = FALSE)
```

**Arguments**

- `Y`: matrix (h x k) of observations in local window; columns correspond to OD flows, and rows are individual observations
- `A`: routing matrix (m x k) for network being analyzed
- `lambda0`: initial vector of values (length k) for lambda; `ipfp` is a good way to obtain this
- `phi0`: initial value for covariance scale phi; initializes automatically using `phi_init` if `NULL`, but you can likely do better
- `c`: power parameter in model of Cao et al. (2000)
- `maxiter`: maximum number of EM iterations to run
- `tol`: tolerance (in relative change in Q function value) for stopping EM iterations
- `epsilon`: numeric nugget to add to diagonal of covariance for numerical stability
- `method`: optimization method to use (in optim calls)
- `checkActive`: logical check for deterministically known OD flows

**Value**

list with 3 elements: lambda, the estimated value of lambda; phi, the estimated value of phi; and `iter`, the number of iterations run

**References**


**See Also**

Other CaoEtAl: `Q_iid`; `Q_smoothed`; `R_estep`; `grad_iid`; `grad_smoothed`; `m_estep`; `phi_init`; `smoothed_EM`
mle_filter      Filtering & smoothing at MLE for calibration SSM

Description

Run Kalman filtering and smoothing at calculated MLE for parameters of calibration SSM. This is used to obtain point and covariance estimates for the actual OD flows \( X \) following estimation of other parameters.

Usage

\[
mle\_filter(\text{mle}, \text{Ft}, \text{yt}, \text{Zt}, \text{Rt}, k = \text{ncol(Ft)}, \text{tau} = 2, \text{initScale} = 1/(1 - \text{diag(Ft)}^2), \text{nugget} = \text{sqrt(.Machine}$\text{double}\text{.eps}))
\]

Arguments

- **mle**: numeric vector (length \( k+1 \)) of parameters. \( \text{theta[-1]} = \text{log(lambda)} \), and \( \text{theta[1]} = \text{log(phi)} \)
- **Ft**: evolution matrix \( (k \times k) \) for OD flows; include fixed
- **yt**: matrix \( (k \times n) \) of observed link loads, one observation per column
- **Zt**: observation matrix for system; should be routing matrix \( A \)
- **Rt**: covariance matrix for observation equation; typically small and fixed
- **k**: integer number of OD flows to infer
- **tau**: numeric power parameter for mean-variance relationship
- **initScale**: numeric inflation factor for time-zero state covariance; defaults to steady-state variance setting
- **nugget**: small positive value to add to diagonal of state evolution covariance matrix to ensure numerical stability

Value

- numeric marginal log-likelihood obtained via Kalman smoothing
- list containing result of Kalman smoothing; see \text{SSModel} and \text{KFS} for details

References


See Also

Other calibrationModel: \text{calibration\_ssm}; \text{llCalibration}
**move_step**

*Move step of sample-resample-move algorithm for multilevel state-space model*

**Description**

Function to execute single MCMC-based move step for `bayesianDynamicFilter`. This can use two types of stopping rules: number of iterations or number of accepted moves for the X particles. The former is used by default, but the latter adapts better to low acceptance rates (sometimes with substantial computational cost). Most updates in this algorithm are Metropolis-Hastings with customized proposals.

**Usage**

```r
code

move_step(y, X, tme, lambda, phi, lambdatm1, phitm1, prior, A, A1_inv, A2, rho, tau, m = ncol(X), l = nrow(A1_inv), k = length(lambda), ndraws = 10, minAccepts = 0, verbose = FALSE)
```

**Arguments**

- **y**: numeric vector (length l) of observed link loads
- **X**: matrix (m x k) of particles for OD flows, one particle per row, in pivoted order
- **tme**: integer time index currently used in estimation
- **lambda**: matrix (m x k) of particles for OD intensities, one particle per row, in pivoted order
- **phi**: numeric vector (length m) of particles for phi
- **lambdatm1**: lambda matrix (m x k) of particles for OD intensities from previous time, one particle per row, in pivoted order
- **phitm1**: numeric vector (length m) of particles for phi from previous time
- **prior**: list containing priors for hyperparameters; see `bayesianDynamicFilter` for details
- **A**: routing matrix (l x k) for network
- **A1_inv**: inverse of full-rank portion of routing matrix (l x l)
- **A2**: remainder of routing matrix (l x k-l)
- **rho**: numeric fixed autoregressive parameter for dynamics on lambda; see reference for details
- **tau**: numeric fixed power parameter for variance structure on truncated normal noise; see reference for details
- **m**: integer number of particles
- **l**: integer number of observed link loads
- **k**: integer number of OD flows to infer
- **ndraws**: integer number of draws to perform (can be overridden by minAccepts)
m_estep

minAccepts  integer minimum number of acceptances before results are returned; activates alternative stopping rule if >= 1
verbose  logical activates verbose diagnostic output

Value

list containing updated values of X, lambda, and phi

References


See Also

Other bayesianDynamicModel: bayesianDynamicFilter; buildPrior

---

m_estep

*Compute conditional expectations for EM algorithms of Cao et al. (2000)*

Description

Computes conditional expectation of OD flows for E-step of EM algorithm from Cao et al. (2000) for their locally IID model.

Usage

m_estep(yt, lambda, phi, A, c, epsilon)

Arguments

- **yt**  numeric vector (length m) of link loads from single time
- **lambda**  numeric vector (length k) of mean OD flows from last M-step
- **phi**  numeric scalar scale for covariance matrix of xt
- **A**  routing matrix (m x k) for network being analyzed
- **c**  power parameter in model of Cao et al. (2000)
- **epsilon**  numeric nugget to add to diagonal of covariance for numerical stability

Value

numeric vector of same size as lambda with conditional expectations of x
**References**


**See Also**

Other CaoEtAl: Q_iid; Q_smoothed; R_estep; grad_iid; grad_smoothed; locally_iid_EM; phi_init; smoothed_EM

---

**obj.tomography**

*Objective function of Zhang et al. 2003*

**Description**

Requires bounded optimization to maintain positive OD flows, and only those flows that are not deterministically zero should be included in the estimation.

**Usage**

```r
obj.tomography(xt, yt, A, srcDstInd, lambda)
```

**Arguments**

- `xt` length-k numeric vector of point-to-point flows
- `yt` length-m numeric vector of observed aggregate flows
- `A` m x k routing matrix, yt = A xt
- `srcDstInd` list of source and destination flow indices corresponding to each point-to-point flow, as produced by `getSrcDstIndices`
- `lambda` regularization parameter for mutual information prior. Note that this is scaled by the squared total traffic in the objective function before scaling the mututal information prior.

**Value**

numeric value of objective function to minimize in tomogravity estimation
**phi_init**

*Simple initialization for phi in model of Cao et al. (2000)*

**Description**

Uses a crude estimator to get a starting point for phi in the model of Cao et al. (2000).

**Usage**

```r
phi_init(Y, A, lambda0, c)
```

**Arguments**

- **Y**: matrix (n x k) of observed link loads over time
- **A**: routing matrix (m x k)
- **lambda0**: numeric vector (length k) of initial guesses for lambda
- **c**: power parameter in model of Cao et al. (2000)

**Value**

numeric starting value for phi

**References**


**See Also**

Other CaoEtAl: `q_iid`; `q_smoothed`; `R_estep`; `grad_iid`; `grad_smoothed`; `locally_iid_EM`; `m_estep`; `smoothed_EM`

---

**Q_iid**

*Q function for locally IID EM algorithm of Cao et al. (2000)*

**Description**

Computes the Q function (expected log-likelihood) for the EM algorithm of Cao et al. (2000) for their locally IID model.

**Usage**

```r
Q_iid(logtheta, c, M, rdiag, epsilon)
```
**Q_smoothed**

**Arguments**

- **logtheta**: numeric vector (length k+1) of log(lambda) (1:k) and log(phi) (last entry)
- **c**: power parameter in model of Cao et al. (2000)
- **M**: matrix (n x k) of conditional expectations for OD flows, one time per row
- **rdiag**: numeric vector (length k) containing diagonal of conditional covariance matrix
- **epsilon**: numeric nugget to add to diagonal of covariance for numerical stability

**Value**

numeric value of Q function; not vectorized in any way

**References**


**See Also**

Other CaoEtAl: Q_smoothed; R_estep; grad_iid; grad_smoothed; locally_iid_EM; m_estep; phi_init; smoothed_EM

---

**Q_smoothed**

*Q function for smoothed EM algorithm of Cao et al. (2000)*

**Description**

Computes the Q function (expected log-likelihood) for the EM algorithm of Cao et al. (2000) for their smoothed model.

**Usage**

`Q_smoothed(logtheta, c, M, rdiag, eta0, sigma0, V, eps.lambda, eps.phi)`

**Arguments**

- **logtheta**: numeric vector (length k+1) of log(lambda) (1:k) and log(phi) (last entry)
- **c**: power parameter in model of Cao et al. (2000)
- **M**: matrix (n x k) of conditional expectations for OD flows, one time per row
- **rdiag**: numeric vector (length k) containing diagonal of conditional covariance matrix
- **eta0**: numeric vector (length k+1) containing value for log(c(lambda, phi)) from previous time (or initial value)
- **sigma0**: covariance matrix (k+1 x k+1) of log(c(lambda, phi)) from previous time (or initial value)
### R_estep

V

evolution covariance matrix (k+1 x k+1) for log(c(lambda, phi)) (random walk)

eps.lambda

numeric small positive value to add to lambda for numerical stability; typically 0

eps.phi

numeric small positive value to add to phi for numerical stability; typically 0

**Value**

numeric value of Q function; not vectorized in any way

**References**


**See Also**

Other CaoEtAl: q_iid; R_estep; grad_iid; grad_smoothed; locally_iid_EM; m_estep; phi_init; smoothed_EM

---

| **R_estep** | Compute conditional covariance matrix for EM algorithms of Cao et al. (2000) |

**Description**

Computes conditional covariance of OD flows for E-step of EM algorithm from Cao et al. (2000) for their locally IID model.

**Usage**

R_estep(lambda, phi, A, c, epsilon)

**Arguments**

lambda

numeric vector (length k) of mean OD flows from last M-step

phi

numeric scalar scale for covariance matrix of xt

A

routing matrix (m x k) for network being analyzed

c

power parameter in model of Cao et al. (2000)

epsilon

numeric nugget to add to diagonal of covariance for numerical stability

**Value**

conditional covariance matrix (k x k) of OD flows given parameters
smoothed_EM

References


See Also

Other CaoEtAl: Q_iid; Q_smoothed; grad_iid; grad_smoothed; locally_iid_EM; m_estep; phi_init; smoothed_EM

smoothed_EM

Run EM algorithm to obtain MLE (single time) for smoothed model of Cao et al. (2000)

Description

Runs EM algorithm to compute MLE for the smoothed model of Cao et al. (2000). Uses numerical optimization of Q-function for each M-step with analytic computation of its gradient. This performs estimation for a single time point using output from the previous one.

Usage

smoothed_EM(Y, A, eta0, sigma0, V, c = 2, maxiter = 1000, tol = 1e-06, eps.lambda = 0, eps.phi = 0, method = "L-BFGS-B")

Arguments

Y  matrix (h x k) of observations in local window; columns correspond to OD flows, and rows are individual observations
A  routing matrix (m x k) for network being analyzed
eta0 numeric vector (length k+1) containing value for log(c(lambda, phi)) from previous time (or initial value)
sigma0 covariance matrix (k+1 x k+1) of log(c(lambda, phi)) from previous time (or initial value)
V  evolution covariance matrix (k+1 x k+1) for log(c(lambda, phi)) (random walk)
c  power parameter in model of Cao et al. (2000)
maxiter maximum number of EM iterations to run
tol tolerance (in relative change in Q function value) for stopping EM iterations
eps.lambda numeric small positive value to add to lambda for numerical stability; typically 0
eps.phi numeric small positive value to add to phi for numerical stability; typically 0
method optimization method to use (in optim calls)
Value

list with 5 elements: lambda, the estimated value of lambda; phi, the estimated value of phi; iter, the number of iterations run; etat, log(c(lambda, phi)); and sigmat, the inverse of the Q functions Hessian at its mode

References


See Also

Other CaoEtAl: Q_iid; Q_smoothed; R_estep; grad_iid; grad_smoothed; locally_iid_EM; m_estep; phi_init

strphour

Convert time string to decimal hour

Description

Convert time string to decimal hour

Usage

strphour(x, fmt = "(\%m/\%d/\%y \%H:\%M:\%S)"")

Arguments

  x  input character vector of times
  fmt input character format for times

Value

numeric vector of decimal times in hours

Examples

strphour("31/08/87 12:53:29")
thin

**Thinning vector of indices for MCMC**

**Description**
Returns a vector of indices with a given spacing for thinning MCMC results.

**Usage**
```
thin(m, interval = 10)
```

**Arguments**
- `m` integer length of results
- `interval` thinning interval

**Value**
integer vector of indices for thinning

tomogravity

**Run tomogravity estimation on complete time series of aggregate flows**

**Description**
The aggregate flows Y and their corresponding routing matrix A must include all aggregate source and destination flows.

**Usage**
```
tomogravity(Y, A, lambda, lower = 0, normalize = FALSE, 
.progress = "none", control = list())
```

**Arguments**
- `Y` n x m matrix contain one vector of observed aggregate flows per row. This should include all observed aggregate flows with none removed due to redundancy.
- `A` m x k routing matrix. This need not be of full row rank and must include all source and destination flows.
- `lambda` Regularization parameter for mutual information prior. Note that this is scaled by the squared total traffic in the objective function before scaling the mututal information prior.
- `lower` Component-wise lower bound for xt in L-BFGS-B optimization.
normalize If TRUE, xt and yt are scaled by N. Typically used in conjunction with calcN to normalize traffic to proportions, easing the tuning of lambda.

.progress name of the progress bar to use, see create_progress_bar in plyr documentation

control List of control information for optim.

Value

A list containing three elements:

- resultList, a list containing the output from running tomogravity.fit on each timepoint
- changeFromInit, a vector of length n containing the relative L_1 change between the initial (IPFP) point-to-point flow estimates and the final tomogravity estimates
- Xhat, a n x k matrix containing a vector of estimated point-to-point flows (for each time point) per row

See Also

Other tomogravity: tomogravity.fit

Examples

data(cmu)
estimate <- tomogravity(Y=cmu$Y.full[1, , drop=FALSE], A=cmu$A.full,
lambda=0.01, .progress='text')

Value

A list containing three elements:

- resultList, a list containing the output from running tomogravity.fit on each timepoint
- changeFromInit, a vector of length n containing the relative L_1 change between the initial (IPFP) point-to-point flow estimates and the final tomogravity estimates
- Xhat, a n x k matrix containing a vector of estimated point-to-point flows (for each time point) per row

See Also

Other tomogravity: tomogravity.fit

Examples

data(cmu)
estimate <- tomogravity(Y=cmu$Y.full[1, , drop=FALSE], A=cmu$A.full,
lambda=0.01, .progress='text')
N total traffic for normalization. Unused if normalized is FALSE.

normalize If TRUE, xt and yt are scaled by N. Typically used in conjunction with calcN to normalize traffic to proportions, easing the tuning of lambda.

lower Component-wise lower bound for xt in L-BFGS-B optimization.

control List of control information for optim.

Value
A list as returned by optim, with element par containing the estimated point-to-point flows and elementer gr containing the analytic gradient evaluated at the estimate.

See Also
Other tomogravity: tomogravity

Examples

data(cmu)
srcDstInd <- getSrcDstIndices(cmu$A.full)
estimate <- tomogravity.fit(yt=cmu$Y.full[1, ], A=cmu$A.full, srcDstInd=srcDstInd, lambda=0.01)

Description
Runs MCMC sampling for the gamma-Poisson model presented in Tebaldi & West (1998). The algorithm used is a modification of that presented in the original paper. It uses a joint proposal for (x_k, lambda_k) to greatly accelerate convergence.

Usage
twMCMC(Y, A, prior, ndraws = 120000, burnin = 20000, verbose = 0)

Arguments
Y numeric vector of observed link loads at a single time (length k)
A routing matrix of dimension (k x n); needs to be full row rank
prior parameters for conjugate gamma prior (convolution and rate)
ndraws integer number of draws for sampler to produce (excluding burn-in)
burnin integer number of additional draws to discard as burnin
verbose integer level of verbosity; levels > 1 have no effect currently
**Value**

list consisting of matrix of draws for X draws, matrix of draws for lambda draws, and vector of acceptances per OD flow accepts

**References**


**Examples**

data(bell.labs)
# Quick, simple run to test the function
prior <- list(a=rep(1, ncol(bell.labs$A)), b=rep(0, ncol(bell.labs$A)))
mcmcOut <- twMCMC(Y=bell.labs$Y[1,], A=bell.labs$A, prior=prior,
                  ndraws=1000, burnin=100, verbose=0)
print(summary(mcmcOut$Xdraws))
print(mcmcOut$accepts)

---

**vardi.algorithm**  
Run algorithm of Vardi (1996) given B and S matrices

**Description**

Runs moment-matching algorithm of Vardi (1996) until convergence

**Usage**

vardi.algorithm(A, Y, lambda, tol = 0.001)

**Arguments**

- **A**: routing matrix (m x k)
- **Y**: matrix of link loads over time (m x n, one column per time)
- **lambda**: numeric vector of starting values for OD flows (length k)
- **tol**: numeric tolerance for halting iterations

**Value**

numeric vector of length k with estimated OD flows

**References**

See Also

Other vardi: vardi.compute.BS; vardi.iteration

---

**vardi.compute.BS**

Compute B and S matrices in algorithm of Vardi (1996)

**Description**

Function to compute B and S matrices for moment equations of Vardi’s method (1996). It’s not particularly efficient, but it works.

**Usage**

vardi.compute.BS(A, Y)

**Arguments**

- **A**
  - routing matrix (m x k)
- **Y**
  - matrix of link loads over time (m x n, one column per time)

**Value**

list containing two entries for the B and S matrices, respectively

**References**


See Also

Other vardi: vardi.algorithm; vardi.iteration

---

**vardi.iteration**

Execute single iteration for algorithm of Vardi (1996)

**Description**

Function to compute B and S matrices for moment equations of Vardi’s method (1996). It’s not particularly efficient, but it works.

**Usage**

vardi.iteration(A, yBar, lambda, B, S)
Arguments

A routing matrix (m x k)
yBar numeric vector of mean link loads (length m)
lambda value of lambda from last iteration
B B matrix computed by vardi.compute.BS
S S matrix computed by vardi.compute.BS

Value

numeric vector of length k with updated lambda

References


See Also

Other vardi: vardi.algorithm; vardi.compute.BS
Index

*Topic **algebra**
calcN, 11
decomposeA, 14
getActive, 16
getSrcDstIndices, 17

*Topic **arith**
agg, 3

*Topic **array**
builtRoutingMatrix, 9
buildStarMat, 10
diag_ind, 14
diag_mat, 15
ipfp, 21

*Topic **character**
strphour, 32

*Topic **datasets**
abilene, 2
bell.labs, 6
cmu, 13

*Topic **iteration**
ipfp, 21

*Topic **manip**
agg, 3
thin, 33

*Topic **models**
builtPrior, 7
calibration_ssm, 11
gravity, 20
gravity.fit, 20
l1Calibration, 22
mle_filter, 24
move_step, 25
tomogravity, 33
tomogravity.fit, 34
twMCMC, 35
vardi.algorithm, 36
vardi.compute.BS, 37
vardi.iteration, 37

*Topic **multivariate**
builtPrior, 7
calibration_ssm, 11
gravity, 20
gravity.fit, 20
l1Calibration, 22
mle_filter, 24
move_step, 25
tomogravity, 33
tomogravity.fit, 34

*Topic **ts**
builtPrior, 7
calibration_ssm, 11
gravity, 20
gravity.fit, 20
l1Calibration, 22
mle_filter, 24
move_step, 25
thin, 33
tomogravity, 33
tomogravity.fit, 34

abilene, 2
agg, 3
bayesianDynamicFilter, 4, 7, 8, 25, 26
bell.labs, 6
builtPrior, 6, 7, 26
builtRoutingMat, 9
builtRoutingMatrix, 9
buildStarMat, 9, 10
calcN, 11
calibration_ssm, 11, 23, 24
cmu, 13
create_progress_bar, 34
decomposeA, 11, 14
diag_ind, 14, 15
diag_mat, 15, 15
dobj.dxt.tomogravity, 16

getActive, 16
getSrcDstIndices, 16, 17, 20, 21, 27, 34
grad_iid, 18, 19, 23, 27–32
grad_smoothed, 18, 19, 23, 27–32
gravity, 20, 21
gravity.fit, 20, 20

ipfp, 21

KFS, 24

llCalibration, 12, 22, 24
locally_iid_EM, 18, 19, 23, 27–32

m_estep, 18, 19, 23, 26, 28–32
mle_filter, 12, 23, 24
move_step, 6, 8, 25

obj.tomogravity, 27

phi_init, 18, 19, 23, 27, 28, 29–32

Q_iid, 18, 19, 23, 27, 28, 29, 30–32
Q_smoothed, 18, 19, 23, 27–29, 29, 31, 32

R_estep, 18, 19, 23, 27–30, 30, 32

smoothed_EM, 18, 19, 23, 27–31, 31

SSModel, 24
strphour, 32

thin, 33
tomogravity, 33, 35
tomogravity.fit, 34, 34
twMCMC, 35

vardi.algorithm, 36, 37, 38
vardi.compute.BS, 37, 37, 38
vardi.iteration, 37, 37