R documentation

of ‘EM.run.Rd’ etc.

August 28, 2014

EM.run	*Inference via EM algorithm*

Description

This function runs the EM algorithm from an initial guess. Infers the coefficient vector in setting where rates depend on patient-specific covariates. The EM algorithm alternates between calling ESTEP and MSTEP until the change in observed log-likelihood changes less than a specified relative tolerance between iterations

Usage

```
EM.run(betaInit, t.pat, num.patients, PATIENTDATA,
  patients.design, s1.seq, s2.seq, relTol)
```

Arguments

betaInit	A vector, the initial guess for coefficients beta
t.pat	A number, the observation interval length
num.patients	An integer, number of unique patients
PATIENTDATA	A matrix in the form returned by MakePatientData containing the set of observation intervals
patients.design	A design matrix in the same form as returned by PatientDesignExample
s1.seq	A vector of complex arguments evenly spaced along the unit circle
s2.seq	A vector of complex arguments evenly spaced along the unit circle
relTol	A number, the relative convergence criterion

Value

A list containing the log-likelihood value at convergence, the final beta estimate, and the number of iterations

ESTEP *Perform one E-step of the EM algorithm*

Description

ESTEP performs one E-step of the EM algorithm, computing expected sufficient statistics given current settings of the parameters. This function is the "accelerated" version, meaning that intervals with no observed changes are computed more efficiently using closed form expressions, bypassing generating function and FFT calculations on these intervals.

Usage

```
ESTEP(betaVec, t.pat, num.patients, PATIENTDATA,
  patients.design, s1.seq, s2.seq)
```

Arguments

betaVec	A vector, the setting of beta coefficients
t.pat	A number, the observation interval length
num.patients	An integer, number of unique patients
PATIENTDATA	A matrix in the form returned by MakePatientData containing the set of observation intervals
patients.design	A design matrix in the same form as returned by PatientDesignExample
s1.seq	A vector of complex arguments evenly spaced along the unit circle
s2.seq	A vector of complex arguments evenly spaced along the unit circle

Value

A list containing a matrix of the expected sufficient statistics as well as the observed log likelihood value

ESTEP.slow *Perform one E-step of the EM algorithm*

Description

ESTEP.slow performs one E-step of the EM algorithm, computing expected sufficient statistics given current settings of the parameters. This function is the un-accelerated version, simply using the generating function approach to compute necessary quantities for all observation intervals.

Usage

```
ESTEP.slow(betaVec, t.pat, num.patients, PATIENTDATA,
  patients.design, s1.seq, s2.seq)
```

Arguments

- `betaVec` A vector, the setting of beta coefficients
- `t.pat` A number, the observation interval length
- `num.patients` An integer, number of unique patients
- `PATIENTDATA` A matrix in the form returned by `MakePatientData` containing the set of observation intervals
- `patients.design` A design matrix in the same form as returned by `PatientDesignExample`
- `s1.seq` A vector of complex arguments evenly spaced along the unit circle
- `s2.seq` A vector of complex arguments evenly spaced along the unit circle

Value

A list containing a matrix of the expected sufficient statistics as well as the observed log likelihood value

---

`FFT.optim` *Maximize the log-likelihood in* `logFFT`

Description

`FFT.optim` maximizes the log-likelihood using `optim`, with hessian = TRUE.

Usage

```
FFT.optim(simDataList, u, initGuess, initList, s1.seq,
  s2.seq)
```

Arguments

- `initGuess` A vector containing the initial guess of birth, shift, and death rates
- `u` A number, the observation interval length
- `initList` A vector of possible initial populations
- `s1.seq` A vector of complex arguments evenly spaced along the unit circle
- `s2.seq` A vector of complex arguments evenly spaced along the unit circle

Value

An optim type object

FFT.pat.optim *Optimizes the function* `logFFT.patients` *using* `optim` *package*

Description

Function uses Nelder-Mead optimization as implemented in `optim` to maximize the log likelihood function `logFFT.patients`.

Usage

```
FFT.pat.optim(betaInit, t.pat, num.patients, PATIENTDATA,
  patients.design, s1.seq, s2.seq, tol, max)
```

Arguments

`betaInit`	A vector, the initial guess for the algorithm
`t.pat`	A number, the observation interval length
`num.patients`	An integer, number of unique patients
`PATIENTDATA`	A matrix in the form returned by `MakePatientData` containing the set of observation intervals
`patients.design`	A design matrix in the same form as returned by `PatientDesignExample`
`s1.seq`	A vector of complex arguments evenly spaced along the unit circle
`s2.seq`	A vector of complex arguments evenly spaced along the unit circle
`tol`	A number for setting the relative tolerance for the algorithm (the reltol argument in `optim`)
`max`	An integer, the max number of iterations before termination

Value

An `optim` type object

FFT.replicate *Replicate the function* `FFT.run`

Description

Replicate the function `FFT.run`

Usage

```
FFT.replicate(numReps, N, tList, lam, v, mu, initList,
  initGuess, s1.seq, s2.seq)
```

Arguments

`numReps`	The number of replications
`N`	An integer specifying the number of observation intervals/realization from the simple BSD process.
`tList`	A list of observation interval lengths. The number of datasets returned is equal to the length of tList
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initList`	A vector containing possible initial population sizes
`initGuess`	A vector containing the initial guess of birth, shift, and death rates
`s1.seq`	A vector of complex arguments evenly spaced along the unit circle
`s2.seq`	A vector of complex arguments evenly spaced along the unit circle

Value

An array of optim objects in the same layout as `FM.replicate`

`FFT.run` *Generate synthetic data from simple BSD process and infer rates using generating function method*

Description

The main function for simulation studies assessing our generating function approach in the the discretely observed simple birth-shift-death-process without covariates. Generates synthetic datasets using `makedata.simple`, and infers the MLE rates for each using `FFT.optim`.

Usage

```
FFT.run(N, tList, lam, v, mu, initList, initGuess,
  s1.seq, s2.seq)
```

Arguments

`N`	An integer specifying the number of observation intervals/realization from the simple BSD process.
`tList`	A list of observation interval lengths. The number of datasets returned is equal to the length of tList
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initList`	A vector containing possible initial population sizes
`initGuess`	A vector containing the initial guess of birth, shift, and death rates
`s1.seq`	A vector of complex arguments evenly spaced along the unit circle
`s2.seq`	A vector of complex arguments evenly spaced along the unit circle

Value

A list of optim objects

`FM.data` *Finds all intervals compatible under frequent monitoring assumption in a synthetic dataset*

Description

`FM.data` takes a simulated dataset in the format generated by `makedata.simple` and finds the subset of observation intervals where at most one event occurred. This is necessary for computation of the log likelihood in `logFM`

Usage

```
FM.data(simDataList, u)
```

Arguments

`simDataList`	A list of synthetic observed datasets, returned by `makedata.simple`
`u`	The index of the desired entry of simDataList

Value

A list containing information about each type of FM event

`FM.optim` *Optimizes the frequent monitoring log-likelihood*

Description

Optimizes the frequent monitoring log-likelihood

Usage

```
FM.optim(simDataList, u, initGuess)
```

Arguments

`simDataList`	A list of synthetic observed datasets, returned by `makedata.simple`
`u`	The index of the desired entry of simDataList
`initGuess`	A vector, initial guess for beta

Value

An optim object corresponding to maximized frequent monitoring log-likelihood

FM.replicate — *Replicate* `FM.run`

Description

Replicate `FM.run`

Usage

```
FM.replicate(numReps, N, tList, lam, v, mu, initList,
  initGuess)
```

Arguments

`numReps`	The number of desired replications
`N`	An integer specifying the number of observation intervals/realization from the simple BSD process.
`tList`	A list of observation interval lengths. The number of datasets returned is equal to the length of tList
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initList`	A vector containing possible initial population sizes
`initGuess`	Vector of numbers, initial guess for `optim`

Value

An array with entries of type returned by `FM.run`. Rows correspond to a dt value in tList, and columns correspond to replications

Examples

```
tList <- c(.2,.4,.6); initList <- c(1:15)
lam = .06; v = .02; mu = .11
trueParam <- c(lam,v,mu)
example <- FM.replicate(numReps,N,tList,lam,v,mu,initList, trueParam)
```

FM.run — *Main function for generating observations from birth-shift-death process and inferring parameters under frequent monitoring*

Description

This function generates observation intervals from the simple birth-shift-death process without covariates, and generates a dataset for each observation time length in tList. Next, the frequent monitoring log-likelihood is maximized using `FM.optim` for each dataset.

Usage

```
FM.run(N, tList, lam, v, mu, initList, initGuess)
```

Arguments

`initGuess`	Vector of numbers, initial guess for `optim`
`N`	An integer specifying the number of observation intervals/realization from the simple BSD process.
`tList`	A list of observation interval lengths. The number of datasets returned is equal to the length of tList
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initList`	A vector containing possible initial population sizes

Value

A list of optim objects

MSTEP *Execute a Newton-Raphson step in M-step of the EM algorithm*

Description

`MSTEP` executes one iteration of a Newton-Raphson algorithm as part of the maximization (M-step) of the EM algorithm. Given the matrix of expected sufficient statistics returned by `ESTEP`, this function uses closed form gradient and hessian expressions to efficiently optimize the current settings of the coefficients beta. This is called up to 10 times per M-step within `EM.run`

Usage

```
MSTEP(matrix, betaVec, num.patients, patients.design)
```

Arguments

`matrix`	A matrix in the format returned by `ESTEP` or `ESTEP.slow`
`betaVec`	A vector of regression coefficients
`num.patients`	An integer, the number of unique patients
`patients.design`	A design matrix in the format generated by `PatientDesignExample`

Value

An updated coefficient vector after one Newton-Raphson step

MakePatientData *Generates synthetic patient data for inference in discretely observed BSD process with covariates*

Description

MakePatientData is the main function for generating a synthetic dataset with covariates. It simulates observations from a birth-shift-death process for a number of synthetic "patients", using ransim.N.true. This provides data for simulation studies toward assessing inference in a panel data setting, with rates depending on multiple covariates.

Usage

```
MakePatientData(t, num.patients, patients.rates)
```

Arguments

t	A number giving the observation interval length
num.patients	An integer, the number of synthetic patients
patients.rates	A matrix in the format returned by MakePatientRates

Details

Each observation interval has a common fixed length given by the argument t. For each patient, between 2 and 6 observation intervals are generated, and each begins with between 2 and 12 initial particles. These numbers are initialized uniformly at random, and passed in as arguments to ransim.N.true, with the true rates set to the corresponding column of patients.rates.

Value

A $5 \times m$ matrix where m is the number of total observation intervals generated. The first row corresponds to the patient ID, the second row gives the initial number of particles for that observation interval, the third through fifth column are in the format returned by ransim.N.true

Examples

```
num.patients = 10; t = .4
patients.design <- PatientDesignExample(num.patients)
beta.lam <- c(log(8), log(.6)); beta.v <- c( log(.5), log(.7)); beta.mu <- c(log(.8), log(.8))
betas <- c(beta.lam,beta.v,beta.mu)
patients.rates <- MakePatientRates(patients.design, betas)
MakePatientData(t, num.patients, patients.rates)
```

MakePatientRates — *Makes matrix of patient-specific birth, shift, and death rates*

Description

This function returns a matrix containing the birth, shift, and death rates of each patient. The rates are calculated based on the log-linear relationship between regression coefficients $\beta = (\beta^\lambda, \beta^\nu, \beta^\mu)$ and covariates $\mathbf{z_p}$ in the *p*th column of the design matrix given by $\log(\lambda_p) = \beta^\lambda \cdot \mathbf{z_p}, \log(\nu_p) = \beta^\nu \cdot \mathbf{z_p}, \log(\mu_p) = \beta^\mu \cdot \mathbf{z_p}$

Usage

```
MakePatientRates(patients.design, betaVec)
```

Arguments

`patients.design`
: A $n \times m$ matrix, where n is number of covariates (including intercept) in the model and m is number of patients

`betaVec`
: The vector β of regression coefficients. Note this function assumes that $\beta^\lambda, \beta^\nu, \beta^\mu$ are equal in length. That is, each rate depends on the same number of covariates

Value

A $3 \times m$ matrix where m is the number of patients, and rows correspond to birth, shift, and death rates respectively.

Examples

```
num.patients = 100
patients.design <- PatientDesignExample(num.patients)
beta.lam <- c(log(8), log(.6)); beta.v <- c( log(.5), log(.7)); beta.mu <- c(log(.8), log(.8))
betas <- c(beta.lam,beta.v,beta.mu)
MakePatientRates(patients.design, betas)
```

PatientDesignExample — *Create example design matrix in the format required for* MakePatientRates

Description

This function creates one possible design matrix as an input for the function MakePatientRates This particular design matrix features one covariate uniformly sampled between 6 and 10, as well as a row of 1's corresponding to an intercept term. The design matrix is thus 2 by number of patients

Usage

```
PatientDesignExample(num.patients)
```

Arguments

`num.patients`	An integer giving the number of patients in the design matrix

Value

A 2 by num.patients matrix: the first row is constant 1's corresponding to intercepts, and second row is a patient-specific covariate sampled uniformly between 6 and 10

`de.birth` *Expected births ODE*

Description

Evaluates the ODE for generating function corresponding to transition probabilities. This is in a format to be solved using zvode from package deSolve; see deSolve documentation/vignette for further details

Usage

```
de.birth(t, state, param)
```

Arguments

`t`	A number or vector of numbers: evaluation times of ODE
`state`	A named vector containing initial value of the state variable G, complex valued
`param`	A named vector of numbers containing the other arguments lam, v, mu, r, and complex number s2

Value

The rate of change of G, the generating function for expected births, in list form

Examples

```
t = .5; state = c(G=exp(2*1i)); param = c(lam = .2, v = .05, mu = .1, r = 3, s2 = exp(3*1i))
de.birth(t,state,param)
```

`de.death` *Expected deaths ODE*

Description

Evaluates the ODE for generating function corresponding to transition probabilities. This is in a format to be solved using zvode from package deSolve; see deSolve documentation/vignette for further details

Usage

```
de.death(t, state, param)
```

Arguments

`t`	A number or vector of numbers: evaluation times of ODE
`state`	A named vector containing initial value of the state variable G, complex valued
`param`	A named vector of numbers containing the other arguments lam, v, mu, r, and complex number s2

Value

The rate of change of G, the generating function for expected deaths, in list form

Examples

```
t = .5; state = c(G=exp(2*1i)); param = c(lam = .2, v = .05, mu = .1, r = 3, s2 = exp(3*1i))
de.death(t,state,param)
```

---

de.particleT *Expected particle time ODE*

Description

Evaluates the ODE for generating function corresponding to transition probabilities. This is in a format to be solved using zvode from package deSolve; see deSolve documentation/vignette for further details

Usage

```
de.particleT(t, state, param)
```

Arguments

`t`	A number or vector of numbers: evaluation times of ODE
`state`	A named vector containing initial value of the state variable G, complex valued
`param`	A named vector of numbers containing the other arguments lam, v, mu, r, and complex number s2

Value

The rate of change of G, the generating function for expected particle time, in list form

Examples

```
t = .5; state = c(G=exp(2*1i)); param = c(lam = .2, v = .05, mu = .1, r = 3, s2 = exp(3*1i))
de.particleT(t,state,param)
```

de.shift — *Expected shifts ODE*

Description

Evaluates the ODE for generating function corresponding to transition probabilities. This is in a format to be solved using zvode from package deSolve; see deSolve documentation/vignette for further details

Usage

```
de.shift(t, state, param)
```

Arguments

`t`	A number or vector of numbers: evaluation times of ODE
`state`	A named vector containing initial value of the state variable G, complex valued
`param`	A named vector of numbers containing the other arguments lam, v, mu, r, and complex number s2

Value

The rate of change of G, the generating function for expected shifts, in list form

Examples

```
t = .5; state = c(G=exp(2*1i)); param = c(lam = .2, v = .05, mu = .1, r = 3, s2 = exp(3*1i))
de.shift(t,state,param)
```

de.trans — *Transition probability ODE*

Description

Evaluates the ODE for generating function corresponding to transition probabilities. This is in a format to be solved using zvode from package deSolve; see deSolve documentation/vignette for further details

Usage

```
de.trans(t, state, param)
```

Arguments

`t`	A number or vector of numbers: evaluation times of ODE
`state`	A named vector containing initial value of the state variable G, complex valued
`param`	A named vector of numbers containing the other arguments lam, v, mu, and complex number s2

Value

The rate of change of G the generating function in list form

Examples

```
t = .5; state = c(G=exp(2*1i)); param = c(lam = .2, v = .05, mu = .1, s2 = exp(3*1i))
de.trans(t,state,param)
```

---

getBirthMeans.initList
Compute expected births over a grid at a list of initial particle counts

---

Description

Wrapper that transforms the output matrices from `makeGrid.birth.r1` and `makeGrid.birth.partial` to a list of expected sufficient statistics matrices using `fft`. Does this at several initial particle counts from initList Returns a list of matrices, where each list entry corresponds to a number of initial particles The i,j entry of each matrix corresponds to the expected births when the process has i type 1 particles and j type 2 particles, beginning with initNum type 1 particles, by the end of corresponding time interval.

Usage

```
getBirthMeans.initList(u, initList, lam, v, mu, s1.seq,
  s2.seq, dt)
```

Arguments

`initList`	A vector of integers corresponding to the desired initial particle counts
`dt`	A number giving the increment length used in solving the ODE
`s1.seq`	A vector of complex numbers; initial values of the ODE G
`s2.seq`	A vector of complex numbers as inputs of s2.seq
`u`	A number giving the observation interval length, equivalently the time to evaluate the ODEs
`lam`	Birth rate, must be positive
`v`	Shift rate, must be positive
`mu`	Death rate

Value

A list of matrices of dimension length(s1.seq) by length(s2.seq); each list entry corresponds to an initial number of particles from initList

Examples

```
initList = c(10,11) #gives matrices of expected sufficient statistics corresponding to 10 initial particles
u = 1; dt = 1; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
getBirthMeans.initList(u, initList, lam, v, mu, s1.seq, s2.seq, dt)
```

`getBirthMeans.timeList`
Compute expected birth over a grid at a list of evaluation times

---

Description

Wrapper that transforms the output matrices from `makeGrid.birth.r1` and `makeGrid.birth.partial` to a list of expected sufficient statistics matrices using `fft`. Does this at several input times given in tList. Returns a list of matrices, where each list entry corresponds to a time in tList. The i,j entry of each matrix corresponds to the number of expected births when process has i type 1 particles and j type 2 particles, beginning with initNum type 1 particles, at the end of corresponding time interval.

Usage

```
getBirthMeans.timeList(tList, lam, v, mu, initNum,
  s1.seq, s2.seq, dt)
```

Arguments

`tList`	A vector of numbers corresponding to the desired evaluation times
`dt`	A number giving the increment length used in solving the ODE
`s1.seq`	A vector of complex numbers; initial values of the ODE G
`s2.seq`	A vector of complex numbers as inputs of s2.seq
`initNum`	An integer giving the number of initial particles
`lam`	Birth rate, must be positive
`v`	Shift rate, must be positive
`mu`	Death rate

Value

A list of matrices of dimension length(s1.seq) by length(s2.seq); each list entry corresponds to an evaluation time from tList

Examples

```
tList = c(1,2);  dt = 1; lam = .5; v = .2; mu = .4; initNum = 10
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
getBirthMeans.timeList(tList, lam, v, mu, initNum, s1.seq, s2.seq, dt)
```

getCover — *Calculates coverage indicators for estimated parameters*

Description

This function takes in an object returned by `optim` and returns logicals on whether the corresponding 95 confidence interval of each estimate contains the true value.

Usage

```
getCover(opt, trueParam)
```

Arguments

`opt`	An optim object
`trueParam`	A vector containing the true parameters

Value

A vector of indicators, 1 indicating that the true parameter was included in the corresponding confidence interval

getDeathMeans.initList — *Compute expected deaths over a grid at a list of initial particle counts*

Description

Wrapper that transforms the output matrices from `makeGrid.death.r1` and `makeGrid.death.partial` to a list of expected sufficient statistics matrices using `fft`. Does this at several initial particle counts from initList Returns a list of matrices, where each list entry corresponds to a number of initial particles The i,j entry of each matrix corresponds to the expected deaths when the process has i type 1 particles and j type 2 particles, beginning with initNum type 1 particles, by the end of corresponding time interval.

Usage

```
getDeathMeans.initList(u, initList, lam, v, mu, s1.seq,
  s2.seq, dt)
```

Arguments

`initList`	A vector of integers corresponding to the desired initial particle counts
`dt`	A number giving the increment length used in solving the ODE
`s1.seq`	A vector of complex numbers; initial values of the ODE G
`s2.seq`	A vector of complex numbers as inputs of s2.seq
`u`	A number giving the observation interval length, equivalently the time to evaluate the ODEs
`lam`	Birth rate, must be positive
`v`	Shift rate, must be positive
`mu`	Death rate

Value

A list of matrices of dimension length(s1.seq) by length(s2.seq); each list entry corresponds to an initial number of particles from initList

Examples

```
initList = c(10,11) #gives matrices of expected sufficient statistics corresponding to 10 initial particles
u = 1; dt = 1; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
getDeathMeans.initList(u, initList, lam, v, mu, s1.seq, s2.seq, dt)
```

`getDeathMeans.timeList` *Compute expected deaths over a grid at a list of evaluation times*

Description

Wrapper that transforms the output matrices from `makeGrid.death.r1` and `makeGrid.death.partial` to a list of expected sufficient statistics matrices using `fft`. Does this at several input times given in tList. Returns a list of matrices, where each list entry corresponds to a time in tList. The i,j entry of each matrix corresponds to the number of expected deaths when process has i type 1 particles and j type 2 particles, beginning with initNum type 1 particles, at the end of corresponding time interval.

Usage

```
getDeathMeans.timeList(tList, lam, v, mu, initNum,
  s1.seq, s2.seq, dt)
```

Arguments

`tList`	A vector of numbers corresponding to the desired evaluation times
`dt`	A number giving the increment length used in solving the ODE
`s1.seq`	A vector of complex numbers; initial values of the ODE G
`s2.seq`	A vector of complex numbers as inputs of s2.seq
`initNum`	An integer giving the number of initial particles
`lam`	Birth rate, must be positive
`v`	Shift rate, must be positive
`mu`	Death rate

Value

A list of matrices of dimension length(s1.seq) by length(s2.seq); each list entry corresponds to an evaluation time from tList

Examples

```
tList = c(1,2);  dt = 1; lam = .5; v = .2; mu = .4; initNum = 10
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
getDeathMeans.timeList(tList, lam, v, mu, initNum, s1.seq, s2.seq, dt)
```

---

getEst — *Get coefficient estimate from optim object*

Description

Get coefficient estimate from optim object

Usage

```
getEst(opt)
```

Arguments

opt	An optim object

Value

The estimates contained in the optim object

---

getParticleT.initList — *Compute expected particle time over a grid at a list of initial particle counts*

Description

Wrapper that transforms the output matrices from `makeGrid.particleT.r0` and `makeGrid.particleT.partial` to a list of expected sufficient statistics matrices using `fft`. Does this at several initial particle counts from initList Returns a list of matrices, where each list entry corresponds to a number of initial particles The i,j entry of each matrix corresponds to the expected particle time spent with i type 1 particles and j type 2 particles, beginning with initNum type 1 particles, by the end of corresponding time interval.

Usage

```
getParticleT.initList(u, initList, lam, v, mu, s1.seq,
  s2.seq, dt)
```

Arguments

`initList`	A vector of integers corresponding to the desired initial particle counts
`dt`	A number giving the increment length used in solving the ODE
`s1.seq`	A vector of complex numbers; initial values of the ODE G
`s2.seq`	A vector of complex numbers as inputs of s2.seq
`u`	A number giving the observation interval length, equivalently the time to evaluate the ODEs
`lam`	Birth rate, must be positive
`v`	Shift rate, must be positive
`mu`	Death rate

Value

A list of matrices of dimension length(s1.seq) by length(s2.seq); each list entry corresponds to an initial number of particles from initList

Examples

```
initList = c(10,11) #gives matrices of expected sufficient statistics corresponding to 10 initial particles
u = 1; dt = 1; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
getParticleT.initList(u, initList, lam, v, mu, s1.seq, s2.seq, dt)
```

`getParticleT.timeList` *Compute expected particle time over a grid at a list of evaluation times*

Description

Wrapper that transforms the output matrices from `makeGrid.particleT.r0` and `makeGrid.particleT.partial` to a list of expected sufficient statistics matrices using `fft`. Does this at several input times given in tList. Returns a list of matrices, where each list entry corresponds to a time in tList. The i,j entry of each matrix corresponds to the expected particle time spent with i type 1 particles and j type 2 particles, beginning with initNum type 1 particles, over the corresponding time length.

Usage

```
getParticleT.timeList(tList, lam, v, mu, initNum, s1.seq,
  s2.seq, dt)
```

Arguments

`tList`	A vector of numbers corresponding to the desired evaluation times
`dt`	A number giving the increment length used in solving the ODE
`s1.seq`	A vector of complex numbers; initial values of the ODE G
`s2.seq`	A vector of complex numbers as inputs of s2.seq
`initNum`	An integer giving the number of initial particles
`lam`	Birth rate, must be positive
`v`	Shift rate, must be positive
`mu`	Death rate

Value

A list of matrices of dimension length(s1.seq) by length(s2.seq); each list entry corresponds to an evaluation time from tList

Examples

```
tList = c(1,2);  dt = 1; lam = .5; v = .2; mu = .4; initNum = 10
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
getParticleT.timeList(tList, lam, v, mu, initNum, s1.seq, s2.seq, dt)
```

---

getShiftMeans.initList *Compute expected shifts over a grid at a list of initial particle counts*

Description

Wrapper that transforms the output matrices from `makeGrid.shift.r1` and `makeGrid.shift.partial` to a list of expected sufficient statistics matrices using `fft`. Does this at several initial particle counts from initList Returns a list of matrices, where each list entry corresponds to a number of initial particles The i,j entry of each matrix corresponds to the expected shifts when the process has i type 1 particles and j type 2 particles, beginning with initNum type 1 particles, by the end of corresponding time interval.

Usage

```
getShiftMeans.initList(u, initList, lam, v, mu, s1.seq,
  s2.seq, dt)
```

Arguments

`initList`	A vector of integers corresponding to the desired initial particle counts
`dt`	A number giving the increment length used in solving the ODE
`s1.seq`	A vector of complex numbers; initial values of the ODE G
`s2.seq`	A vector of complex numbers as inputs of s2.seq
`u`	A number giving the observation interval length, equivalently the time to evaluate the ODEs
`lam`	Birth rate, must be positive
`v`	Shift rate, must be positive
`mu`	Death rate

Value

A list of matrices of dimension length(s1.seq) by length(s2.seq); each list entry corresponds to an initial number of particles from initList

Examples

```
initList = c(10,11) #gives matrices of expected sufficient statistics corresponding to 10 initial particles
u = 1; dt = 1; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
getShiftMeans.initList(u, initList, lam, v, mu, s1.seq, s2.seq, dt)
```

`getShiftMeans.timeList` *Compute expected shifts over a grid at a list of evaluation times*

Description

Wrapper that transforms the output matrices from `makeGrid.shift.r1` and `makeGrid.shift.partial` to a list of expected sufficient statistics matrices using `fft`. Does this at several input times given in tList. Returns a list of matrices, where each list entry corresponds to a time in tList. The i,j entry of each matrix corresponds to the number of expected shifts when process has i type 1 particles and j type 2 particles, beginning with initNum type 1 particles, at the end of corresponding time interval.

Usage

```
getShiftMeans.timeList(tList, lam, v, mu, initNum,
  s1.seq, s2.seq, dt)
```

Arguments

`tList`	A vector of numbers corresponding to the desired evaluation times
`dt`	A number giving the increment length used in solving the ODE
`s1.seq`	A vector of complex numbers; initial values of the ODE G
`s2.seq`	A vector of complex numbers as inputs of s2.seq
`initNum`	An integer giving the number of initial particles
`lam`	Birth rate, must be positive
`v`	Shift rate, must be positive
`mu`	Death rate

Value

A list of matrices of dimension length(s1.seq) by length(s2.seq); each list entry corresponds to an evaluation time from tList

Examples

```
tList = c(1,2);  dt = 1; lam = .5; v = .2; mu = .4; initNum = 10
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
getShiftMeans.timeList(tList, lam, v, mu, initNum, s1.seq, s2.seq, dt)
```

getStandardErrors — *Calculate numerical standard errors of estimated coefficients*

Description

getStandardErrors uses optimHess to numerically compute the hessian at the value of the estimated MLE, which can be obtained using EM.run. The function inverts the hessian and takes the square root of diagonal entries to yield standard errors for each coefficient estimate.

Usage

```
getStandardErrors(betas, t.pat, num.patients,
  PATIENTDATA, patients.design, s1.seq, s2.seq)
```

Arguments

betas	A vector of numbers $\beta = (\beta^{\lambda}, \beta^{\nu}, \beta^{\mu})$
t.pat	A number, the observation interval length
num.patients	An integer, number of unique patients
PATIENTDATA	A matrix in the form returned by MakePatientData containing the set of observation intervals
patients.design	A design matrix in the same form as returned by PatientDesignExample
s1.seq	A vector of complex arguments evenly spaced along the unit circle
s2.seq	A vector of complex arguments evenly spaced along the unit circle

Value

A vector the length of the coefficient vector, giving standard errors for each estimated coefficient

getTrans.FreqMon — *Calculates the four transition probabilities defined under Frequent Monitoring*

Description

getTrans.FreqMon uses simple closed form expressions to compute transition probabilities available under the Frequent Monitoring assumption, which allows at most one event to occur per observation interval. Computes the four possible transition possibilities for intervals with lengths corresponding to entries in tList

Usage

```
getTrans.FreqMon(tList, lam, v, mu, initNum)
```

Arguments

`tList`	A vector of observation interval lengths
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initNum`	Integer, the number of initial particles

Value

A list of 2×2 matrices, each containing the probabilitiy of one birth, one shift, one death, or no event occurring over a corresponding observation interval length from tList

Examples

```
getTrans.FreqMon(c(.5,1,10),.3,.1,.2,10)
```

`getTrans.MC` *Calculate empirical transition probabilities*

Description

Calculates transition probabilities empirically using Monte Carlo simulation

Usage

```
getTrans.MC(N, tList, lam, v, mu, initNum)
```

Arguments

`N`	An integer, the number of MC simulations per element of tList
`tList`	A list of observation interval lengths to simulate
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initNum`	Integer giving the number of initial particles

Details

Note: function can be modified to initialize matrices to be larger sized in the case where rates are large

Value

A list of matrices, where each entry of the list corresponds to an element of tList. The i,j entry of each matrix in the list give the probability of the process ending with i type 1 particles and j type 2 particles, beginning with initNum type 1 particles, by the end of the corresponding observation length.

Examples

```
N = 500; tList = c(.5,1); lam = .2; v = .1; mu = .15; initNum = 10
getTrans.MC(N,tList,lam,v,mu,initNum)
```

getTrans.initList *Compute transition probabilites over a grid at a list of initial particle counts*

Description

Wrapper that transforms the output matrices from `makeGrid.trans` to a list of expected sufficient statistics matrices using `fft`. Does this at several initial particle counts from initList Returns a list of matrices, where each list entry corresponds to a number of initial particles The i,j entry of each matrix corresponds to the probability of transitioning to i type 1 particles and j type 2 particles, beginning with initNum type 1 particles, by the end of corresponding time interval.

Usage

```
getTrans.initList(u, initList, lam, v, mu, s1.seq,
  s2.seq, dt)
```

Arguments

`initList`	A vector of integers corresponding to the desired initial particle counts
`dt`	A number giving the increment length used in solving the ODE
`s1.seq`	A vector of complex numbers; initial values of the ODE G
`s2.seq`	A vector of complex numbers as inputs of s2.seq
`u`	A number giving the observation interval length, equivalently the time to evaluate the ODEs
`lam`	Birth rate, must be positive
`v`	Shift rate, must be positive
`mu`	Death rate

Value

A list of matrices of dimension length(s1.seq) by length(s2.seq); each list entry corresponds to an initial number of particles from initList

Examples

```
initList = c(10,11) #gives matrices of transition probabilities corresponding to 10 initial particles and 1
u = 1; dt = 1; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
getTrans.initList(u, initList, lam, v, mu, s1.seq, s2.seq, dt)
```

`getTrans.timeList` *Compute transition probabilities over a grid at a list of evaluation times*

Description

Wrapper that transforms the output matrix from `makeGrid.trans` to interpretable transition probabilities using `fft`. Does this at several input times given in tList. Returns a list of matrices, where each list entry corresponds to a time in tList. The i,j entry of each matrix corresponds to the probability of transitioning from initNum type 1 particles and 0 type 2 particles to i type 1 particles, j type 2 particles, over the corresponding time length.

Usage

```
getTrans.timeList(tList, lam, v, mu, initNum, s1.seq,
  s2.seq, dt)
```

Arguments

`tList`	A vector of numbers corresponding to the desired evaluation times
`dt`	A number giving the increment length used in solving the ODE
`s1.seq`	A vector of complex numbers; initial values of the ODE G
`s2.seq`	A vector of complex numbers as inputs of s2.seq
`initNum`	An integer giving the number of initial particles
`lam`	Birth rate, must be positive
`v`	Shift rate, must be positive
`mu`	Death rate

Value

A list of matrices of dimension length(s1.seq) by length(s2.seq), each list entry corresponds to an evaluation time from tList, each matrix is a matrix of transition probabilities

Examples

```
tList = c(1,2);  dt = 1; lam = .5; v = .2; mu = .4; initNum = 10
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
getTrans.timeList(tList, lam, v, mu, initNum, s1.seq, s2.seq, dt)
```

logFFT *Calculate log-likelihood using generating function technique*

Description

`logFFT` calculates the log-likelihood where transition probabilities are computed using our FFT generating function method

Usage

```
logFFT(param, u, simData, initList, s1.seq, s2.seq)
```

Arguments

`param`	A vector of three numbers containing the birth, shift, and death rate respectively
`u`	A number, the observation interval length
`simData`	An matrix in the format of a list entry returned by `makedata.simple`
`initList`	A vector of possible initial populations
`s1.seq`	A vector of complex arguments evenly spaced along the unit circle
`s2.seq`	A vector of complex arguments evenly spaced along the unit circle

Value

The negative log-likelihood value

logFFT.patients *Log likelihood for BSD process with covariates*

Description

`logFFT.patients` evaluates the log likelihood of a dataset with observations corresponding to "patients" in the setting where rates of the process depend on patient-specific covariates. The transition probabilities given the states of the process at endpoints of each observation interval are computed using the FFT/generating function method, relying on `getTrans.initList`. The log likelihood is then the sum of these log transition probabilities.

Usage

```
logFFT.patients(betas, t.pat, num.patients, PATIENTDATA,
  patients.design, s1.seq, s2.seq)
```

Arguments

`betas`	A vector of numbers $\beta = (\beta^{\lambda}, \beta^{\nu}, \beta^{\mu})$
`t.pat`	A number, the observation interval length
`num.patients`	An integer, number of unique patients
`PATIENTDATA`	A matrix in the form returned by `MakePatientData` containing the set of observation intervals
`patients.design`	A design matrix in the same form as returned by `PatientDesignExample`
`s1.seq`	A vector of complex arguments evenly spaced along the unit circle
`s2.seq`	A vector of complex arguments evenly spaced along the unit circle

Details

Note: this function is used so that MLE estimation of the coefficient vector can be accomplished, i.e. using `optim`, and is also used in numerically computing standard errors at the MLE.

Vectors s1.seq and s2.seq should be of length greater than the total number of particles of either type at any observation interval

Value

The negative log likelihood of the observations in PATIENTDATA, given rates determined by coefficient vector betas and covariate values in patients.design

---

`logFM` *Calculate approximate log-likelihood based on Frequent Monitoring computations*

Description

`logFM` calculates the log-likelihood where transition probabilities are computed under the frequent monitoring assumption. Used for simulation studies comparing results using frequent monitoring with our methods.

Usage

```
logFM(param, u, FM)
```

Arguments

`param`	A vector of three numbers containing the birth, shift, and death rate respectively
`u`	A number, the observation interval length
`FM`	An object returned by `FM.data` containing the number of each event under FM

Value

The negative frequent monitoring log-likelihood

makeGrid.birth.partial

Evaluates partial derivative of expected births ODE over 2D grid

Description

Wrapper that numerically partially differentiates the solution given in solve.birth over a grid of input values s1, s2 at one fixed time, and r=1

Usage

```
makeGrid.birth.partial(time, dt, s1.seq, s2.seq, lam, v,
  mu)
```

Arguments

time	A number corresponding to the desired evaluation time of ODEs
dt	A number giving the increment length used in solving the ODE
s1.seq	A vector of complex numbers; initial values of the ODE G
s2.seq	A vector of complex numbers as inputs of s2.seq
lam	Birth rate
v	Shift rate
mu	Death rate

Value

A matrix of dimension length(s1.seq) by length(s2.seq) of the function values

Examples

```
time = 5;  dt = 5; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
makeGrid.birth.partial(time,dt,s1.seq,s2.seq,lam,v,mu)
```

makeGrid.birth.r1

Evaluates expected births ODE over 2D grid of arguments

Description

Wrapper applying solve.birth to a grid of inputs s1, s2 at one fixed time and r=1

Usage

```
makeGrid.birth.r1(time, dt, s1.seq, s2.seq, lam, v, mu)
```

Arguments

time	A number corresponding to the desired evaluation time of ODEs
dt	A number giving the increment length used in solving the ODE
s1.seq	A vector of complex numbers; initial values of the ODE G
s2.seq	A vector of complex numbers as inputs of s2.seq
lam	Birth rate
v	Shift rate
mu	Death rate

Value

A matrix of dimension length(s1.seq) by length(s2.seq) of the function values

Examples

```
time = 5;  dt = 5; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
makeGrid.birth.r1(time,dt,s1.seq,s2.seq,lam,v,mu)
```

---

makeGrid.death.partial

Evaluates partial derivative of expected deaths ODE over 2D grid

---

Description

Wrapper that numerically partially differentiates the solution given in solve.death over a grid of input values s1, s2 at one fixed time, and r=1

Usage

```
makeGrid.death.partial(time, dt, s1.seq, s2.seq, lam, v,
  mu)
```

Arguments

time	A number corresponding to the desired evaluation time of ODEs
dt	A number giving the increment length used in solving the ODE
s1.seq	A vector of complex numbers; initial values of the ODE G
s2.seq	A vector of complex numbers as inputs of s2.seq
lam	Birth rate
v	Shift rate
mu	Death rate

Value

A matrix of dimension length(s1.seq) by length(s2.seq) of the function values

Examples

```
time = 5;  dt = 5; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
makeGrid.death.partial(time,dt,s1.seq,s2.seq,lam,v,mu)
```

---

makeGrid.death.r1 *Evaluates expected deaths ODE over 2D grid of arguments*

Description

Wrapper applying solve.death to a grid of inputs s1, s2 at one fixed time and r=1

Usage

```
makeGrid.death.r1(time, dt, s1.seq, s2.seq, lam, v, mu)
```

Arguments

time	A number corresponding to the desired evaluation time of ODEs
dt	A number giving the increment length used in solving the ODE
s1.seq	A vector of complex numbers; initial values of the ODE G
s2.seq	A vector of complex numbers as inputs of s2.seq
lam	Birth rate
v	Shift rate
mu	Death rate

Value

A matrix of dimension length(s1.seq) by length(s2.seq) of the function values

Examples

```
time = 5;  dt = 5; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
makeGrid.death.r1(time,dt,s1.seq,s2.seq,lam,v,mu)
```

makeGrid.particleT.partial *Evaluates partial derivative of expected particle time ODE over 2D grid*

Description

Wrapper that numerically partially differentiates the solution given in solve.particleT over a grid of input values s1, s2 at one fixed time, and r=0

Usage

```
makeGrid.particleT.partial(time, dt, s1.seq, s2.seq, lam,
  v, mu)
```

Arguments

time	A number corresponding to the desired evaluation time of ODEs
dt	A number giving the increment length used in solving the ODE
s1.seq	A vector of complex numbers; initial values of the ODE G
s2.seq	A vector of complex numbers as inputs of s2.seq
lam	Birth rate
v	Shift rate
mu	Death rate

Value

A matrix of dimension length(s1.seq) by length(s2.seq) of the function values

Examples

```
time = 5;  dt = 5; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
makeGrid.particleT.partial(time,dt,s1.seq,s2.seq,lam,v,mu)
```

makeGrid.particleT.r0 *Evaluates expected particle time ODE over 2D grid of arguments*

Description

Wrapper applying solve.particleT to a grid of inputs s1, s2 at one fixed time and r=0

Usage

```
makeGrid.particleT.r0(time, dt, s1.seq, s2.seq, lam, v,
  mu)
```

Arguments

time	A number corresponding to the desired evaluation time of ODEs
dt	A number giving the increment length used in solving the ODE
s1.seq	A vector of complex numbers; initial values of the ODE G
s2.seq	A vector of complex numbers as inputs of s2.seq
lam	Birth rate
v	Shift rate
mu	Death rate

Value

A matrix of dimension length(s1.seq) by length(s2.seq) of the function values

Examples

```
time = 5;  dt = 5; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
makeGrid.particleT.r0(time,dt,s1.seq,s2.seq,lam,v,mu)
```

---

makeGrid.shift.partial *Evaluates partial derivative of expected shifts ODE over 2D grid*

Description

Wrapper that numerically partially differentiates the solution given in solve.shift over a grid of input values s1, s2 at one fixed time, and r=1

Usage

```
makeGrid.shift.partial(time, dt, s1.seq, s2.seq, lam, v,
  mu)
```

Arguments

time	A number corresponding to the desired evaluation time of ODEs
dt	A number giving the increment length used in solving the ODE
s1.seq	A vector of complex numbers; initial values of the ODE G
s2.seq	A vector of complex numbers as inputs of s2.seq
lam	Birth rate
v	Shift rate
mu	Death rate

Value

A matrix of dimension length(s1.seq) by length(s2.seq) of the function values

Examples

```
time = 5;  dt = 5; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
makeGrid.shift.partial(time,dt,s1.seq,s2.seq,lam,v,mu)
```

---

makeGrid.shift.r1 *Evaluates expected shifts ODE over 2D grid of arguments*

Description

Wrapper applying solve.shift to a grid of inputs s1, s2 at one fixed time and r=1

Usage

```
makeGrid.shift.r1(time, dt, s1.seq, s2.seq, lam, v, mu)
```

Arguments

time	A number corresponding to the desired evaluation time of ODEs
dt	A number giving the increment length used in solving the ODE
s1.seq	A vector of complex numbers; initial values of the ODE G
s2.seq	A vector of complex numbers as inputs of s2.seq
lam	Birth rate
v	Shift rate
mu	Death rate

Value

A matrix of dimension length(s1.seq) by length(s2.seq) of the function values

Examples

```
time = 5;  dt = 5; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
makeGrid.shift.r1(time,dt,s1.seq,s2.seq,lam,v,mu)
```

makeGrid.trans — *Evaluates transition probability ODE over 2D grid of arguments*

Description

Wrapper applying `solve.trans` to a grid of inputs s1, s2 for a fixed time.

Usage

```
makeGrid.trans(time, dt, s1.seq, s2.seq, lam, v, mu)
```

Arguments

`time`	A number corresponding to the desired evaluation time of ODE
`dt`	A number giving the increment length used in solving the ODE
`s1.seq`	A vector of complex numbers; initial values of the ODE G
`s2.seq`	A vector of complex numbers as inputs of s2.seq
`lam`	Birth rate
`v`	Shift rate
`mu`	Death rate

Value

A matrix of dimension length(s1.seq) by length(s2.seq) of the function values

Examples

```
time = 5;  dt = 5; lam = .5; v = .2; mu = .4
gridLength = 32
s1.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
s2.seq <- exp(2*pi*1i*seq(from = 0, to = (gridLength-1))/gridLength)
makeGrid.trans(time,dt,s1.seq,s2.seq,lam,v,mu)
```

makedata.simple — *Generate synthetic dataset for BSD process with simple rates*

Description

`makedata.simple` creates a list of synthetic observed datasets, each with N observation intervals from the BSD process. Observation interval lengths are entries in tList. This is used in simulation studies checking inferential procedures. Simulates observations using `ransim.N.true`.

Usage

```
makedata.simple(N, tList, lam, v, mu, initList)
```

Arguments

`N`	An integer specifying the number of observation intervals/realization from the simple BSD process.
`tList`	A list of observation interval lengths. The number of datasets returned is equal to the length of tList
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initList`	A vector containing possible initial population sizes

Value

A list of matrices. Each entry in the list is in the format returned by `ransim.N.true`, and corresponds to equal observation times specified in the corresponding entry in tList

Examples

```
makedata.simple(30,c(.2,.4,.6),.2,.12,.15,seq(8,12))
```

`ransim.N.true` *Simulate N realizations from the BSD process, each with random initial number each time*

Description

Wrapper for `simulate.one.ran` that simply replicates the function N times. Each replication begins with a random number of initial particles uniformly generated from a specified range.

Usage

```
ransim.N.true(N, t.end, lam, v, mu, range)
```

Arguments

`N`	An integer, the number of realizations to simulate
`range`	A vector containing possible initial populations, typically a sequence of integers
`t.end`	A number giving the length of time for the simulation
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate

Value

A 3 by N matrix; each ith column corresponds to the ith realization. The first row contains initial numbers, the second row contains the number of original indices still present in the population by t.end, and the third row contains the number of new indices present.

Examples

```
ransim.N.true(10,2,.2,.12,.15, seq(7,15))
```

sim.N.eventcount *Simulate N BSD processes and return sufficient statistics*

Description

Wrapper that replicates `simulate.one.eventcount` N times. Each replication begins with initNum particles.

Usage

```
sim.N.eventcount(N, t.end, lam, v, mu, initNum)
```

Arguments

`N`	The number of replications, an integer
`t.end`	A number giving the length of time for the simulation
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initNum`	An integer, initial total number of particles

Value

A 4 by N matrix, where rows correspond to total copies, shifts, deaths, and particle time per realization, respectively. Each column corresponds to one realization of the process.

Examples

```
sim.N.eventcount(10,2,.2,.12,.15,10)
```

sim.N.true *Simulate N realizations from the BSD process*

Description

Wrapper for `simulate.one.true` that simply replicates the function N times

Usage

```
sim.N.true(N, t.end, lam, v, mu, initNum)
```

Arguments

`N`	An integer, the number of realizations to simulate
`t.end`	A number giving the length of time for the simulation
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initNum`	An integer, initial total number of particles

Value

A 2 by N matrix; each ith column is a pair of integers giving the number of initial indices still present followed by the number of new indices present in the population in the ith realization

Examples

```
sim.N.true(10,2,.2,.12,.15,10)
```

---

sim.eventcount — *Simulate a BSD process and return sufficient statistics*

Description

Simulates a birth-shift-death process and records the number of births, deaths, and shifts, as well as total particle time, over the observation interval until time t.end. Used toward simulation functions that check accuracy of expected restricted moment calculations

Usage

```
sim.eventcount(t.end, lam, v, mu, initNum)
```

Arguments

`t.end`	A number giving the length of time for the simulation
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initNum`	An integer, initial total number of particles

Value

A vector of four numbers giving total copies, shifts, deaths, and particle time, respectively. Returns the integer '999' as an error code if error occurs.

Examples

```
sim.eventcount(2,.2,.12,.15,10)
```

`sim.one.eventcount` *Simulate a BSD process and return sufficient statistics with error catch*

Description

Simulates a birth-shift-death process and records the number of births, deaths, and shifts, as well as total particle time, over the observation interval until time t.end. Used toward simulation functions that check accuracy of expected restricted moment calculations

Usage

```
sim.one.eventcount(t.end, lam, v, mu, initNum)
```

Arguments

`t.end`	A number giving the length of time for the simulation
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initNum`	An integer, initial total number of particles

Value

A vector of four numbers giving total copies, shifts, deaths, and particle time, respectively.

Examples

```
sim.one.eventcount(2,.2,.12,.15,10)
```

`sim.one.ran` *Simulate a BSD process, returning initial number as well as end state*

Description

Wrapper for `simulate.true` that catches any possible errors and returns the initial number.

Usage

```
sim.one.ran(t.end, lam, v, mu, initNum)
```

Arguments

`t.end`	A number giving the length of time for the simulation
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initNum`	An integer, initial total number of particles

Value

A vector containing the initial number of particles, followed by a pair of integers giving the number of initial indices still present followed by the number of new indices present in the population

Examples

```
sim.one.ran(2,.2,.12,.15,10)
```

---

sim.one.true   *Simulate a BSD process with error catch*

Description

Wrapper for `simulate.true` that catches any possible errors

Usage

```
sim.one.true(t.end, lam, v, mu, initNum)
```

Arguments

`t.end`	A number giving the length of time for the simulation
`lam`	Per-particle birth rate
`v`	Per-particle shift rate
`mu`	Per-particle death rate
`initNum`	An integer, initial total number of particles

Value

A pair of integers giving the number of initial indices still present followed by the number of new indices present in the population

Examples

```
sim.one.true(2,.2,.12,.15,10)
```

simulate.true *Simulate a birth-shift-death-process*

Description

This function simulates one realization of a birth-shift-death CTMC with per-particle birth, death, and shift rates lambda, nu, and mu.

Usage

```
simulate.true(t.end, lam, v, mu, initNum)
```

Arguments

t.end	A number giving the length of time for the simulation
lam	Per-particle birth rate
v	Per-particle shift rate
mu	Per-particle death rate
initNum	An integer, initial total number of particles

Details

The process begins with initNum particles, each assigned a random location indexed by a number between 1 and 100,000. Simulation occurs until time t.end. The process returns an ordered pair giving the number of initial locations (indices) still occupied, followed by the number of new indices present, by t.end.

Value

A pair of integers giving the number of initial indices still present followed by the number of new indices present in the population. Otherwise, returns '999' as an error code

Examples

```
simulate.true(2,.2,.12,.15,10)
```

solve.birth *Expected births ODE solver*

Description

Wrapper that numerically solves the ODE defined in `de.birth` using `zvode` from package `deSolve` given evaluation time, initial state values s1 and s2, and birth/shift/death rates

Usage

```
solve.birth(time, dt, s1, s2, r, lam, v, mu)
```

Arguments

`time`	A number corresponding to the desired evaluation time of ODE
`dt`	A number giving the increment length used in solving the ODE
`s1`	A complex number giving the initial value of the ODE G
`s2`	A complex number
`lam`	Birth rate
`v`	Shift rate
`mu`	Death rate
`r`	a real number

Value

The function value of the births generating function

Examples

```
time = 5;  dt = 1; s1 = exp(2*1i); s2 = exp(3*1i); lam = .5; v = .2; mu = .4; r = 3
solve.birth(time,dt,s1,s2,r,lam,v,mu)
```

`solve.death` *Expected deaths ODE solver*

Description

Wrapper that numerically solves the ODE defined in `de.death` using `zvode` from package `deSolve` given evaluation time, initial state values s1 and s2, and birth/shift/death rates

Usage

```
solve.death(time, dt, s1, s2, r, lam, v, mu)
```

Arguments

`time`	A number corresponding to the desired evaluation time of ODE
`dt`	A number giving the increment length used in solving the ODE
`s1`	A complex number giving the initial value of the ODE G
`s2`	A complex number
`lam`	Birth rate
`v`	Shift rate
`mu`	Death rate
`r`	a real number

Value

The function value of the deaths generating function

Examples

```
time = 5;  dt = 1; s1 = exp(2*1i); s2 = exp(3*1i); lam = .5; v = .2; mu = .4; r = 3
solve.death(time,dt,s1,s2,r,lam,v,mu)
```

solve.particleT *Expected births ODE solver*

Description

Wrapper that numerically solves the ODE defined in `de.particleT` using `zvode` from package `deSolve` given evaluation time, initial state values s1 and s2, and birth/shift/death rates

Usage

```
solve.particleT(time, dt, s1, s2, r, lam, v, mu)
```

Arguments

`time`	A number corresponding to the desired evaluation time of ODE
`dt`	A number giving the increment length used in solving the ODE
`s1`	A complex number giving the initial value of the ODE G
`s2`	A complex number
`lam`	Birth rate
`v`	Shift rate
`mu`	Death rate
`r`	a real number

Value

The function value of the particle time generating function

Examples

```
time = 5;  dt = 1; s1 = exp(2*1i); s2 = exp(3*1i); lam = .5; v = .2; mu = .4; r = 3
solve.particleT(time,dt,s1,s2,r,lam,v,mu)
```

solve.shift *Expected shifts ODE solver*

Description

Wrapper that numerically solves the ODE defined in `de.shift` using `zvode` from package `deSolve` given evaluation time, initial state values s1 and s2, and birth/shift/death rates

Usage

```
solve.shift(time, dt, s1, s2, r, lam, v, mu)
```

Arguments

`time`	A number corresponding to the desired evaluation time of ODE
`dt`	A number giving the increment length used in solving the ODE
`s1`	A complex number giving the initial value of the ODE G
`s2`	A complex number
`lam`	Birth rate
`v`	Shift rate
`mu`	Death rate
`r`	A real number

Value

The function value of the shifts generating function

Examples

```
time = 5;  dt = 1; s1 = exp(2*1i); s2 = exp(3*1i); lam = .5; v = .2; mu = .4; r = 2
solve.shift(time,dt,s1,s2, r, lam,v,mu)
```

`solve.trans` *Transition ODE solver*

Description

Wrapper that numerically solves the ODE defined in `de.trans` using `zvode` from package `deSolve` given evaluation time, initial state values s1 and s2, and birth/shift/death rates

Usage

```
solve.trans(time, dt, s1, s2, lam, v, mu)
```

Arguments

`time`	A number corresponding to the desired evaluation time of ODE
`dt`	A number giving the increment length used in solving the ODE
`s1`	A complex number giving the initial value of the ODE G
`s2`	A complex number
`lam`	Birth rate
`v`	Shift rate
`mu`	Death rate

Value

The function value of the transition probability generating function

Examples

```
time = 5;  dt = 1; s1 = exp(2*1i); s2 = exp(3*1i); lam = .5; v = .2; mu = .4
solve.trans(time,dt,s1,s2,lam,v,mu)
```

Index