Package 'sarsop'

October 19, 2022

Type Package

Title Approximate POMDP Planning Software

Version 0.6.13

Description A toolkit for Partially Observed Markov Decision Processes (POMDP). Provides bindings to C++ libraries implementing the algorithm SARSOP (Successive Approximations of the Reachable Space under Optimal Policies) and described in Kurniawati et al (2008), <doi:10.15607/RSS.2008.IV.009>. This package also provides a high-level interface for generating, solving and simulating POMDP problems and their solutions.

License GPL-2

URL https://github.com/boettiger-lab/sarsop

BugReports https://github.com/boettiger-lab/sarsop/issues

RoxygenNote 7.1.1

Imports xml2, parallel, processx, digest, Matrix

Suggests testthat, roxygen2, knitr, covr, spelling

LinkingTo BH

Encoding UTF-8

Language en-US

SystemRequirements mallinfo, hence Linux, MacOS or Windows

NeedsCompilation yes

Author Carl Boettiger [cre, aut, cph]
(<https://orcid.org/0000-0002-1642-628X>),
Jeroen Ooms [aut],
Milad Memarzadeh [aut],
Hanna Kurniawati [ctb, cph],
David Hsu [ctb, cph],
Hanna Kurniawati [ctb, cph],
Wee Sun Lee [ctb, cph],
Yanzhu Du [ctb],
Xan Huang [ctb],
Trey Smith [ctb, cph],
Tony Cassandra [ctb, cph],

Lee Thomason [ctb, cph],
Carl Kindman [ctb, cph],
Le Trong Dao [ctb, cph],
Amit Jain [ctb, cph],
Rong Nan [ctb, cph],
Ulrich Drepper [ctb],
Free Software Foundation [cph],
Tyge Lovset [ctb, cph],
Yves Berquin [ctb, cph],
Benjamin Grüdelbach [ctb],
RSA Data Security, Inc. [cph]

Maintainer Carl Boettiger <cboettig@gmail.com>

Repository CRAN

Date/Publication 2022-10-19 21:42:35 UTC

R topics documented:

`alphas_from_log` *alphas_from_log*

Description

Read alpha vectors from a log file.

Usage

```
alphas_from_log(meta, log_dir = ".")
```

Arguments

meta	a data frame containing the log metadata for each set of alpha vectors desired, see meta_from_log
log_dir	path to log directory

Value

a list with a matrix of alpha vectors for each entry in the provided metadata (as returned by sarsop).

Examples

```
 # takes > 5s

source(system.file("examples/fisheries-ex.R", package = "sarsop"))
log = tempfile()
alpha <- sarsop(transition, observation, reward, discount, precision = 10,
                log_dir = log)
```

---

assert_has_appl *test the APPL binaries*

Description

Asserts that the C++ binaries for appl have been compiled successfully

Usage

```
assert_has_appl()
```

Value

Will return TRUE if binaries are installed and can be located and executed, and FALSE otherwise.

Examples

```
assert_has_appl()
```

compute_policy *compute_policy*

Description

Derive the corresponding policy function from the alpha vectors

Usage

```
compute_policy(
  alpha,
  transition,
  observation,
  reward,
  state_prior = rep(1, dim(observation)[[1]])/dim(observation)[[1]],
  a_0 = 1
)
```

Arguments

alpha	the matrix of alpha vectors returned by sarsop
transition	Transition matrix, dimension n_s x n_s x n_a
observation	Observation matrix, dimension n_s x n_z x n_a
reward	reward matrix, dimension n_s x n_a
state_prior	initial belief state, optional, defaults to uniform over states
a_0	previous action. Belief in state depends not only on observation, but on prior belief of the state and subsequent action that had been taken.

Value

a data frame providing the optimal policy (choice of action) and corresponding value of the action for each possible belief state

Examples

```
m <- fisheries_matrices()
 ## Takes > 5s
if(assert_has_appl()){
alpha <- sarsop(m$transition, m$observation, m$reward, 0.95, precision = 10)
compute_policy(alpha, m$transition, m$observation, m$reward)
}
```

fisheries_matrices *fisheries_matrices*

Description

Initialize the transition, observation, and reward matrices given a transition function, reward function, and state space

Usage

```
fisheries_matrices(
  states = 0:20,
  actions = states,
  observed_states = states,
  reward_fn = function(x, a) pmin(x, a),
  f = ricker(1, 15),
  sigma_g = 0.1,
  sigma_m = 0.1,
  noise = c("rescaled-lognormal", "lognormal", "uniform", "normal")
)
```

Arguments

states	sequence of possible states
actions	sequence of possible actions
observed_states	sequence of possible observations
reward_fn	function of x and a that gives reward for tacking action a when state is x
f	transition function of state x and action a.
sigma_g	half-width of uniform shock or equivalent variance for log-normal
sigma_m	half-width of uniform shock or equivalent variance for log-normal
noise	distribution for noise, "lognormal" or "uniform"

Details

assumes log-normally distributed observation errors and process errors

Value

list of transition matrix, observation matrix, and reward matrix

Examples

```
m <- fisheries_matrices()
```

f_from_log *f from log*

Description

Read transition function from log

Usage

```
f_from_log(meta)
```

Arguments

meta	a data frame containing the log metadata for each set of alpha vectors desired, see meta_from_log

Details

note this function is unique to the fisheries example problem and assumes that sarsop call is run with logging specifying a column "model" that contains either the string "ricker" (corresponding to a Ricker-type growth function) or "allen" (corresponding to an Allen-type.)

Value

the growth function associated with the model indicated.

Examples

```
 # takes > 5s

source(system.file("examples/fisheries-ex.R", package = "sarsop"))
log = tempfile()
alpha <- sarsop(transition, observation, reward, discount, precision = 10,
                log_dir = log)
```

hindcast_pomdp *hindcast_pomdp*

Description

Compare historical actions to what pomdp recommendation would have been.

Usage

```
hindcast_pomdp(
  transition,
  observation,
  reward,
  discount,
  obs,
  action,
  state_prior = rep(1, dim(observation)[[1]])/dim(observation)[[1]],
  alpha = NULL,
  ...
)
```

Arguments

transition	Transition matrix, dimension n_s x n_s x n_a
observation	Observation matrix, dimension n_s x n_z x n_a
reward	reward matrix, dimension n_s x n_a
discount	the discount factor
obs	a given sequence of observations
action	the corresponding sequence of actions
state_prior	initial belief state, optional, defaults to uniform over states
alpha	the matrix of alpha vectors returned by sarsop
...	additional arguments to appl.

Value

a list, containing: a data frame with columns for time, obs, action, and optimal action, and an array containing the posterior belief distribution at each time t

Examples

```
m <- fisheries_matrices()
 ## Takes > 5s
if(assert_has_appl()){
alpha <- sarsop(m$transition, m$observation, m$reward, 0.95, precision = 10)
sim <- hindcast_pomdp(m$transition, m$observation, m$reward, 0.95,
                      obs = rnorm(21, 15, .1), action = rep(1, 21),
                      alpha = alpha)

}
```

meta_from_log *meta from log*

Description

load metadata from a log file

Usage

```
meta_from_log(
  parameters,
  log_dir = ".",
  metafile = paste0(log_dir, "/meta.csv")
)
```

Arguments

parameters	a data.frame with the desired parameter values as given in metafile
log_dir	path to log directory
metafile	path to metafile index, assumed to be meta.csv in log_dir

Value

a data.frame with the rows of the matching metadata.

Examples

```
 # takes > 5s

source(system.file("examples/fisheries-ex.R", package = "sarsop"))
log = tempfile()
alpha <- sarsop(transition, observation, reward, discount, precision = 10,
                log_dir = log)
```

models_from_log *model from log*

Description

Read model details from log file

Usage

```
models_from_log(meta, reward_fn = function(x, h) pmin(x, h))
```

Arguments

meta	a data frame containing the log metadata for each set of alpha vectors desired, see meta_from_log
reward_fn	a function f(x,a) giving the reward for taking action a given a system in state x.

Details

assumes transition can be determined by the f_from_log function, which is specific to the fisheries example

Value

a list with an element for each row in the requested meta data frame, which itself is a list of the three matrices: transition, observation, and reward, defining the pomdp problem.

Examples

```
 # takes > 5s

source(system.file("examples/fisheries-ex.R", package = "sarsop"))
log = tempfile()
alpha <- sarsop(transition, observation, reward, discount, precision = 10,
                log_dir = log)
```

---

pomdpsol *APPL wrappers*

Description

Wrappers for the APPL executables. The pomdpsol function solves a model file and returns the path to the output policy file.

Usage

```
pomdpsol(
  model,
  output = tempfile(),
  precision = 0.001,
  timeout = NULL,
  fast = FALSE,
  randomization = FALSE,
  memory = NULL,
  improvementConstant = NULL,
  timeInterval = NULL,
  stdout = tempfile(),
```

```
  stderr = tempfile(),
  spinner = TRUE
)

polgraph(
  model,
  policy,
  output = tempfile(),
  max_depth = 3,
  max_branches = 10,
  min_prob = 0.001,
  stdout = "",
  spinner = TRUE
)

pomdpsim(
  model,
  policy,
  output = tempfile(),
  steps = 100,
  simulations = 3,
  stdout = "",
  spinner = TRUE
)

pomdpeval(
  model,
  policy,
  output = tempfile(),
  steps = 100,
  simulations = 3,
  stdout = "",
  spinner = TRUE
)

pomdpconvert(model, stdout = "", spinner = TRUE)
```

Arguments

model	file/path to the pomdp model file
output	file/path of the output policy file. This is also returned by the function.
precision	targetPrecision. Set targetPrecision as the target precision in solution quality; run ends when target precision is reached. The target precision is 1e-3 by default.
timeout	Use timeLimit as the timeout in seconds. If running time exceeds the specified value, pomdpsol writes out a policy and terminates. There is no time limit by default.
fast	logical, default FALSE. use fast (but very picky) alternate parser for .pomdp files.

randomization	logical, default FALSE. Turn on randomization for the sampling algorithm.
memory	Use memoryLimit as the memory limit in MB. No memory limit by default. If memory usage exceeds the specified value, pomdpsol writes out a policy and terminates. Set the value to be less than physical memory to avoid swapping.
improvementConstant	Use improvementConstant as the trial improvement factor in the sampling algorithm. At the default of 0.5, a trial terminates at a belief when the gap between its upper and lower bound is 0.5 of the current precision at the initial belief.
timeInterval	Use timeInterval as the time interval between two consecutive write-out of policy files. If this is not specified, pomdpsol only writes out a policy file upon termination.
stdout	a filename where pomdp run statistics will be stored
stderr	currently ignored.
spinner	should we show a spinner while sarsop is running?
policy	file/path to the policy file
max_depth	the maximum horizon of the generated policy graph
max_branches	maximum number of branches to show in the policy graph
min_prob	the minimum probability threshold for a branch to be shown in the policy graph
steps	number of steps for each simulation run
simulations	as the number of simulation runs

Examples

```
if(assert_has_appl()){
  model <- system.file("models", "example.pomdp", package = "sarsop")
  policy <- tempfile(fileext = ".policyx")
  pomdpsol(model, output = policy, timeout = 1)

# Other tools
  evaluation <- pomdpeval(model, policy, stdout = FALSE)
  graph <- polgraph(model, policy, stdout = FALSE)
  simulations <- pomdpsim(model, policy, stdout = FALSE)
}
```

---

read_policyx *read_policyx*

Description

read a .policyx file created by SARSOP and return alpha vectors and associated actions.

Usage

```
read_policyx(file = "output.policyx")
```

Arguments

file	name of the policyx file to be read.

Value

a list, first element "vectors" is an n_states x n_vectors array of alpha vectors, second element is a numeric vector "action" of length n_vectors whose i'th element indicates the action corresponding to the i'th alpha vector (column) in the vectors array.

Examples

```
f <- system.file("extdata", "out.policy", package="sarsop", mustWork = TRUE)
policy <- read_policyx(f)
```

---

sarsop *sarsop*

Description

sarsop wraps the tasks of writing the pomdpx file defining the problem, running the pomdsol (SARSOP) algorithm in C++, and then reading the resulting policy file back into R. The returned alpha vectors and alpha_action information is then transformed into a more generic, user-friendly representation as a matrix whose columns correspond to actions and rows to states. This function can thus be used at the heart of most pomdp applications.

Usage

```
sarsop(
  transition,
  observation,
  reward,
  discount,
  state_prior = rep(1, dim(observation)[[1]])/dim(observation)[[1]],
  verbose = TRUE,
  log_dir = tempdir(),
  log_data = NULL,
  cache = TRUE,
  ...
)
```

Arguments

transition	Transition matrix, dimension n_s x n_s x n_a
observation	Observation matrix, dimension n_s x n_z x n_a
reward	reward matrix, dimension n_s x n_a
discount	the discount factor
state_prior	initial belief state, optional, defaults to uniform over states
verbose	logical, should the function include a message with pomdp diagnostics (timings, final precision, end condition)
log_dir	pomdpx and policyx files will be saved here, along with a metadata file
log_data	a data.frame of additional columns to include in the log, such as model parameters. A unique id value for each run can be provided as one of the columns, otherwise, a globally unique id will be generated.
cache	should results from the log directory be cached? Default TRUE. Identical functional calls will quickly return previously cached alpha vectors from file rather than re-running.
...	additional arguments to appl.

Value

a matrix of alpha vectors. Column index indicates action associated with the alpha vector, (1:n_actions), rows indicate system state, x. Actions for which no alpha vector was found are included as all -Inf, since such actions are not optimal regardless of belief, and thus have no corresponding alpha vectors in alpha_action list.

Examples

```
 ## Takes > 5s
## Use example code to generate matrices for pomdp problem:
source(system.file("examples/fisheries-ex.R", package = "sarsop"))
alpha <- sarsop(transition, observation, reward, discount, precision = 10)
compute_policy(alpha, transition, observation, reward)
```

---

sim_pomdp *simulate a POMDP*

Description

Simulate a POMDP given the appropriate matrices.

Usage

```
sim_pomdp(
  transition,
  observation,
  reward,
  discount,
  state_prior = rep(1, dim(observation)[[1]])/dim(observation)[[1]],
  x0,
  a0 = 1,
  Tmax = 20,
  policy = NULL,
  alpha = NULL,
  reps = 1,
  ...
)
```

Arguments

transition	Transition matrix, dimension n_s x n_s x n_a
observation	Observation matrix, dimension n_s x n_z x n_a
reward	reward matrix, dimension n_s x n_a
discount	the discount factor
state_prior	initial belief state, optional, defaults to uniform over states
x0	initial state
a0	initial action (default is action 1, e.g. can be arbitrary if the observation process is independent of the action taken)
Tmax	duration of simulation
policy	Simulate using a pre-computed policy (e.g. MDP policy) instead of POMDP
alpha	the matrix of alpha vectors returned by sarsop
reps	number of replicate simulations to compute
...	additional arguments to mclapply

Details

simulation assumes the following order of updating: For system in state[t] at time t, an observation of the system obs[t] is made, and then action[t] is based on that observation and the given policy, returning (discounted) reward[t].

Value

a data frame with columns for time, state, obs, action, and (discounted) value.

Examples

```
m <- fisheries_matrices()
discount <- 0.95
 ## Takes > 5s
if(assert_has_appl()){
alpha <- sarsop(m$transition, m$observation, m$reward, discount, precision = 10)
sim <- sim_pomdp(m$transition, m$observation, m$reward, discount,
                 x0 = 5, Tmax = 20, alpha = alpha)

}
```

`write_pomdpx` *write pomdpx files*

Description

A POMDPX file specifies a POMDP problem in terms of the transition, observation, and reward matrices, the discount factor, and the initial belief.

Usage

```
write_pomdpx(
  P,
  O,
  R,
  gamma,
  b = rep(1/dim(O)[1], dim(O)[1]),
  file = "input.pomdpx",
  digits = 12,
  digits2 = 12,
  format = "f"
)
```

Arguments

`P`	transition matrix
`O`	observation matrix
`R`	reward
`gamma`	discount factor
`b`	initial belief
`file`	pomdpx file to create
`digits`	precision to round to before normalizing. Leave at 4 since sarsop seems unable to do more?
`digits2`	precision to write solution to. Leave at 10, since normalizing requires additional precision

format	floating point format, because sarsop parser doesn't seem to know scientific notation

Examples

```
m <- fisheries_matrices()
f <- tempfile()
write_pomdpx(m$transition, m$observation, m$reward, 0.95,
             file = f)
```

Index