Package: BeastJar (via r-universe)

November 2, 2026

Type Package

Title JAR Dependency for MCMC Using 'BEAST'

Version 10.5.1

Description Provides JAR to perform Markov chain Monte Carlo (MCMC) inference using the popular Bayesian Evolutionary Analysis by Sampling Trees 'BEAST X' software library of Baele et al (2025) <doi:10.1038/s41592-025-02751-x>. 'BEAST X' supports auto-tuning Metropolis-Hastings, slice, Hamiltonian Monte Carlo and Sequential Monte Carlo sampling for a large variety of composable standard and phylogenetic statistical models using high performance computing. By placing the 'BEAST X' JAR in this package, we offer an efficient distribution system for 'BEAST X' use by other R packages using CRAN.

License Apache License 2.0

Encoding UTF-8

Depends R (>= 3.5.0)

Imports rJava

URL https://github.com/beast-dev/BeastJar

Copyright See file COPYRIGHTS

RoxygenNote 7.3.2

SystemRequirements Java (>= 1.8)

Suggests testthat

Language en-US

NeedsCompilation no

Author Marc A. Suchard [aut, cre, cph], Andrew Rambaut [cph], Alexei J. Drummond [cph]

Maintainer Marc A. Suchard <msuchard@ucla.edu>

Config/pak/sysreqs default-jdk

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

Date/Publication 2025-08-21 13:20:02 UTC

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

RemoteRef HEAD

RemoteSha f9e9dda6f79b3b96efc24d9c42c815c7428d49b4

Contents

BeastJar 2
supportsJava8 4

Index **5**

BeastJar *BeastJar*

Description

Convenient packaging of the Bayesian Evolutionary Analysis Sampling Trees (BEAST) software package to facilitate Markov chain Monte Carlo sampling techniques including Hamiltonian Monte Carlo, bouncy particle sampling and zig-zag sampling.

Examples

```
# Example MCMC simulation using BEAST
#
# This function generates a Markov chain to sample from a simple normal distribution.
# It uses a random walk Metropolis kernel that is auto-tuning.

if (supportsJava8()) {
  # Set seed
  seed <- 123
  rJava::J("dr.math.MathUtils")$setSeed(rJava::.jlong(seed));

  # Set up simple model - Normal(mean = 1, sd = 2)
  mean <- 1; sd <- 2
  distribution <- rJava::.jnew("dr.math.distributions.NormalDistribution",
                               as.numeric(mean), as.numeric(sd))
  model <- rJava::.jnew("dr.inference.distribution.DistributionLikelihood",
                    rJava::.jcast(distribution, "dr.math.distributions.Distribution"))
  parameter <- rJava::.jnew("dr.inference.model.Parameter$Default", "p", 1.0,
                            as.numeric(-1.0 / 0.0), as.numeric(1.0 / 0.0))
  model$addData(parameter)

  # Construct posterior
  dummy <- rJava::.jnew("dr.inference.model.DefaultModel",
                        rJava::.jcast(parameter, "dr.inference.model.Parameter"))

  joint <- rJava::.jnew("java.util.ArrayList")
  joint$add(rJava::.jcast(model, "dr.inference.model.Likelihood"))
  joint$add(rJava::.jcast(dummy, "dr.inference.model.Likelihood"))
```

```
joint <- rJava::new(rJava::J("dr.inference.model.CompoundLikelihood"), 1L, joint)

# Specify auto-adapting random-walk Metropolis-Hastings transition kernel
operator <- rJava::.jnew("dr.inference.operators.RandomWalkOperator",
                         rJava::.jcast(parameter, "dr.inference.model.Parameter"),
                         0.75,
                         rJava::J(
                           "dr.inference.operators.RandomWalkOperator"
                         )$BoundaryCondition$reflecting,
                         1.0,
                         rJava::J("dr.inference.operators.AdaptationMode")$DEFAULT
)

schedule <- rJava::.jnew("dr.inference.operators.SimpleOperatorSchedule",
                         as.integer(1000), as.numeric(0.0))

schedule$addOperator(operator)

# Set up what features of posterior to log
subSampleFrequency <- 100
memoryFormatter <- rJava::.jnew("dr.inference.loggers.ArrayLogFormatter", FALSE)
memoryLogger <-
  rJava::.jnew("dr.inference.loggers.MCLogger",
               rJava::.jcast(memoryFormatter, "dr.inference.loggers.LogFormatter"),
               rJava::.jlong(subSampleFrequency), FALSE)
memoryLogger$add(parameter)

# Execute MCMC
mcmc <- rJava::.jnew("dr.inference.mcmc.MCMC", "mcmc1")
mcmc$setShowOperatorAnalysis(FALSE)

chainLength <- 100000

mcmcOptions <- rJava::.jnew("dr.inference.mcmc.MCMCOptions",
                            rJava::.jlong(chainLength),
                            rJava::.jlong(10),
                            as.integer(1),
                            as.numeric(0.1),
                            TRUE,
                            rJava::.jlong(chainLength/100),
                            as.numeric(0.234),
                            FALSE,
                            as.numeric(1.0))

mcmc$init(mcmcOptions,
          joint,
          schedule,
         rJava::.jarray(memoryLogger, contents.class = "dr.inference.loggers.Logger"))

mcmc$run()

# Summarize logged posterior quantities
```

```
  traces <- memoryFormatter$getTraces()
  trace <- traces$get(as.integer(1))

  obj <- trace$getValues(as.integer(0),
                         as.integer(trace$getValueCount()))

  sample <- rJava::J("dr.inference.trace.Trace")$toArray(obj)

  outputStream <- rJava::.jnew("java.io.ByteArrayOutputStream")
  printStream <- rJava::.jnew("java.io.PrintStream",
                              rJava::.jcast(outputStream, "java.io.OutputStream"))

  rJava::J("dr.inference.operators.OperatorAnalysisPrinter")$showOperatorAnalysis(
    printStream, schedule, TRUE)

  operatorAnalysisString <- outputStream$toString("UTF8")

  # Report auto-optimization information
  cat(operatorAnalysisString)

  # Report posterior quantities
  c(mean(sample), sd(sample))
}
```

`supportsJava8` *Determine if Java virtual machine supports Java*

Description

Tests Java virtual machine (JVM) java.version system property to check if version >= 8.

Usage

```
supportsJava8()
```

Value

Returns TRUE if JVM supports Java >= 8.

Examples

```
supportsJava8()
```

Index

BeastJar, 2

supportsJava8, 4