

# Package ‘BeastJar’

January 20, 2025

**Type** Package

**Title** JAR Dependency for MCMC Using 'BEAST'

**Version** 1.10.6

**Description** Provides JAR to perform Markov chain Monte Carlo (MCMC) inference using the popular Bayesian Evolutionary Analysis by Sampling Trees 'BEAST' software library of Suchard et al (2018) <[doi:10.1093/ve/vey016](https://doi.org/10.1093/ve/vey016)>. 'BEAST' 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' JAR in this package, we offer an efficient distribution system for 'BEAST' use by other R packages using CRAN.

**License** Apache License 2.0

**Encoding** UTF-8

**LazyData** true

**Imports** rJava

**URL** <https://github.com/beast-dev/BeastJar>

**Copyright** See file COPYRIGHTS

**RoxygenNote** 7.1.1

**SystemRequirements** Java (>= 1.8)

**Suggests** testthat

**NeedsCompilation** no

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

**Maintainer** Marc A. Suchard <[msuchard@ucla.edu](mailto:msuchard@ucla.edu)>

**Repository** CRAN

**Date/Publication** 2020-10-26 14:20:06 UTC

## Contents

BeastJar . . . . .	2
supportsJava8 . . . . .	4
<b>Index</b>	<b>5</b>

---

BeastJar	<i>BeastJar</i>
----------	-----------------

---

## 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"), 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$getValues(as.integer(0),
    as.integer(trace$valueCount()))

```

```
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$string("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  $\geq 8$ .

### **Usage**

```
supportsJava8()
```

### **Value**

Returns TRUE if JVM supports Java  $\geq 8$ .

### **Examples**

```
supportsJava8()
```

# Index

BeastJar, [2](#)

supportsJava8, [4](#)