Appendices

StateSpaces

R code	Description
`turnOnBurnIn = function(...)` `turnOffBurnIn = function(...)`	These methods pass the information to the State Space on whether it should avoid remembering sample points because of the burn-in period, or not.
`proposeLogsOfUMeasures = function(...)`	Calls the Target Measure to evaluate the logarithms of unnormalalised density or probability function.
`randomWalkProposal = function(...)`	Generates the proposal in the Random Walk phase and returns the results to the Algorithm.
`updateStatesAfterRandomWalk = function(...)` `updateStatesAfterSwap = function(...)`	Realises the call from the Algorithm to update the required parameters.
`calculateBetweenSteps = function(...)`	General wrapper for updates to be executed between the Random Walk phase and the Random Swap phase.

Real (State Space)

R code	Description
`insertInitialStates = function( initialStates = matrix(ncol=0, nrow=0), spaceDim = 0L, chainsNo = 0L, ... )`	Checks whether the user inserted correct initial states. If their dimension is different then the dimension supplied by the user, or they were not enough of them, i.e. less than the number of chains, or if initial states were not supplied, it generates new ones uniformly from a hypersquare $[0, 10]^{\text{Problem Dimenstion}}$.
`createDataStorage = function(...)`	Creates an appropriately big matrix for the storage of sample points.
`insertProposal−−Covariances = function( proposalCovariances = matrix( ncol=0, nrow=0 ), ... )`	Checks whether the user provided correct proposal covariances. The user can provide one matrix if he wants the covariances to be the same for all chains. If that is not the case, the user is obliged to provide a list of matrices that contains as many matrices as there are temperature levels. If the user provides wrong data, unit variances are to be chosen. If the user provides an object whose type is neither matrix, nor list, then the algorithm stops.
`checkCovariance = function( covarianceMatrix, ... )`	Returns true of the given object is a matrix and its dimensions match the dimension of the State Space provided by the user beforehand.

Real Tempered (State Space)

R code	Description
`insertTemperatures = function( temperatures, ... )`	Checks whether the user inserted correct temperature values and stores them.
`getIteration = function( iteration = 1L, type = 'initial states', ... )`	For a given iteration extracts results of a given step type, to choose among 'initial states', 'random walk', and 'swap'.
`prepareDataForPlot = function(...)`	Reshuffles the entire history of states so that the entire result conforms to the data frame templates of **ggplot2**
`plotAllTemperatures = function(...)`	Performs a plot of all simulated chains with an overlayed map of the real density from the Liang example.
`plotBasics = function( algorithmName, ... )`	Performs a plot of the base level temperature chain of main interest with an overlayed map of the real density from the Liang example.

R code	Description
`measureQuasiDistance = function( iState, jState, ... )`	Measures the quasi distance between states needed for.

Target Measure

R code	Description
`measure = function(...)`	Evaluates the density or the probability function at a given point.
`establishTrueValues = function(...)`	Evaluates the values of the provided density on a grid $[-2, 12]^2$ with the mesh set at 0.1. These are to be plotted so that the user may compare the results of the simulation with how it really looks.

Target Unnormalised Densities

R code	Description
`initialize = function( targetDensity = function(){}, ... )`	Stores the unnormalised density function provided by the user.

Target Liang Densities and Matteo Densities

R code	Description
`initialize = function( iterationsNo = NULL, quantile- -SimulationsNo=, mixturesNo =, mixturesWeight = , mixturesMeans = , sigma = , weightConstant = , algorithmName = 'Liang', ... )`	Initialises the reference example of the Liang density described in detail in chapter ....
`plotDistribuant = function(...)`	Plots the distibuant of Liang density based on the grid points from $[-2, 12]^2$ with the mesh set at 0.1.
`distribuant = function( x, ... )`	Calculates the true value of Liang's measure distribuant at point x. It serves as a reference value when evaluating the Колмогоров-Смирнов distance.
`marginalDistribuant = function( proposedState, coordinateNo, ... )`	Calculates the true value of Liang's measure marginal distribuants at point x. One can insert the coodinate number, i.e. 1 or 2, which does not assume the infinite value. It serves as a reference value when evaluating the Колмогоров-Смирнов distance.

R code	Description
`getSquareGrid = function( minimum , maximum , mesh , ... )`	Prepares the matrix with values from $[-2, 12]^2$ with the mesh set at 0.1.
`getQuantiles = function( simulationsNo , ... )`	Simulates the values of Liang Measure's quantiles using Monte Carlo simulation scheme.
`simulateQuantiles = function( simulationsNo , ... )`	Writes down the results of the quantile simulation for further calculations. This function is handy when trying to evaluate visually the correctness of different simulation strategies.
`measureDistance− −FromMeans = function( point , ... )`	Calculates the distance of a given point from all of the means in the mixture of the 20 gaussian densities.
`classify = function( point , ... )`	Start the classification procedures. Their aim is to prescribe the sample points to particular densities that are components of the Liang mixture of measures. The aim of this procedure is to check whether the presence of the sample points in the approximity of modes approximates well their probability input to the whole distribution, which is set to $1/20^{\text{th}}$.

R CODE	DESCRIPTION
`classifyByLength = function( point, ... )`	Performs the classification of sample points to the modes using the distance-from-mean criterion.
`classifyByChiSquare = function( point, ... )`	Performs the classification of sample points to the modes using the minimal-chi-square criterion.
`getFirstAndSecondMoments = function(...)`	Calculates all first and second true moments of the Liang distribution.

Simulations

R code	Description
`initialize = function(space = , algo = , target = , example = , iterationsNo = , burnIn = , chainsNo = , spaceDim = , initialStates = , targetDensity = , temperatures = , strategyNo = , quasiMetric = , covariances = , detailedOutput = , save = , trialNo = , evaluateKS = , integratedFunction=, rememberStates = , evaluateSojourn = , ...)`	Initialises the simulation - prepares the appropriate Target Measure, links it to appropriate State Space, chooses the right Algorithm.
`checkTemperatures = function( temperatures , ... )`	Checks whether the user provided correct temperatures.
`setIterationsNo = function( iterationsNo , ... )`	Checks whether the user provided correct maximal number of iterations that the algorithm is to execute and stores it.

R CODE	DESCRIPTION
`write = function(...)`	Creates the file where the results of the simulations will get stored, in the `./data` catalogue, under a naming including the number of iterations, number of simulations, and the number of used strategy, in case the user has chosen the PARALLEL TEMPERING algorithm. Also, it calls the particular saving procedure of different objects.
`furnishResults = function(...)`	Prepares a vector containing the statistics on a particular run of the simulation. Among them: the strategy number, the rejections of the Random Walk, the rejections of the Random Swap, the Колмогоров-Смирнов statistic, the sojourn estimates, and the integral estimates.
`simulate = function(...)`	Performs the simulation.