Adaptive Component-wise Multiple-Try Metropolis Sampling Function

DESCRIPTION:

The function implements component-wise multiple-try Metropolis sampling for any given target distribution. The values of the Gaussian proposals standard deviations for each coordinate can be adapted during the MCMC or remain fixed. Here we define nprop = number of proposal standard deviations, d = dimension of parameter space.

USAGE:

```
CMTM_sampling(log.dens.target,N.iter=1000,theta.init=c(0,0),sig.mat=c(0.5,2),
              alpha=2.9,adp=TRUE,vec=FALSE,monitor=TRUE,print.out.freq=500,...)
```

ARGUMENTS:

log.dens.target - log density function of the target distribution. First argument is theta, other parameters can be added through ... .
N.iter - number of MCMC samples to generate for each coordinate (including burn-in).
theta.init - initial values for the chain's components.
sig.mat - matrix of proposal standard deviations; the number of rows is equal to the number of proposals (nprop) and the number of columns is equal to the dimension of the parameter space (d). If a vector of dimension nprop is provided, then it will be used initially for all coordinates.
alpha - exponent of the distance function for weight calculation. Default is 2.9.
adp - logical; if TRUE(default) then *sig.mat* is adapted throughout the MCMC.
vec - logical; if TRUE then vectorization technique is implemented to speed up the procedure. log.dens.target can evaluate many thetas at the same time.
monitor - logical; (only relevant when *adp* is TRUE). If TRUE, when adaptation occurs during the MCMC run the function prints number of iterations from last adaptation and selection rates from last adaptation point.
print.out.freq - prints an output as a chain progresses every print.out.freq iterations.
... - additional arguments to log.dens.target.

VALUE:

The function returns the list of the following objects:
samples - matrix of MCMC samples.
sig.mat - the latest adapted matrix of sigma proposals. If *adp*=FALSE, returns original *sig.mat*.
sel.rate.all - array(nprop by d by 2), first (nprop by d) matrix shows selection rates of proposal standard deviations for each component. Second matrix represents the post-selection acceptance rate for each proposal and coordinate. Calculated based on all samples.
sel.rate.adp - array(nprop by d by 2), first (nprop by d) matrix shows selection rates of proposal standard deviations for each component. Second matrix represents how often selected proposal standard deviations are accepted for each coordinate. Calculated based on samples from the last adaptation point. If *adp* is FALSE,

returns NULL.
num.adp - vector of length d, shows number of produced adaptations for each coordinate. If *adp* is FALSE, returns NULL.
last.adp - iteration where last adaptation occurred (if *adp*=TRUE).

EXAMPLE:

```
# Sample from 2 dimensional Gaussian mixture
library(mvtnorm)
targSigma1 = rbind(c(6.25,0),c(0,6.25))
targSigma2 = rbind(c(6.25,0),c(0,0.25))
targMean1 = c(5, 0)
targMean2 = c(15, 0)

log.dens.target = function(theta, mu1, mu2, sig1, sig2)
{
  log( 0.5*dmvnorm(theta,mu1,sig1) + 0.5*dmvnorm(theta,mu2,sig2) )
} # Note: this function can evaluate many thetas at the same time (theta can be a matrix),
  # therefore vectorization is possible.

sig.mat = matrix(rep(c(0.25,0.5,1,2,4),2),ncol=2)
CMTM.res = CMTM_sampling(log.dens.target,10000,c(0,0),sig.mat,2.9,adp=TRUE,vec=TRUE,
                         monitor=TRUE,1000,mu1=targMean1,mu2=targMean2,sig1=targSigma1,
                         sig2=targSigma2)

par(mfrow=c(2,1))
plot(CMTM.res$samples[,1],type='l')
plot(CMTM.res$samples[,2],type='l')

CMTM.res$sig.mat

# Compare CPU time for vectorized and non-vectorized versions:
system.time({
CMTM.res = CMTM_sampling(log.dens.target,1000,c(0,0),sig.mat,2.9,adp=TRUE,vec=FALSE,
                         monitor=FALSE,1000,mu1=targMean1,mu2=targMean2,sig1=targSigma1,
                         sig2=targSigma2)
})

system.time({
CMTM.res = CMTM_sampling(log.dens.target,1000,c(0,0),sig.mat,2.9,adp=TRUE,vec=TRUE,
                         monitor=FALSE,1000,mu1=targMean1,mu2=targMean2,sig1=targSigma1,
                         sig2=targSigma2)
})
```