Generic Functions for MAD# Post-Processing

Heather Savoy

January 23, 2015

This collection of functions will read data from MAD# databases, calculate likelihoods, and calculate posteriors. There are also plotting functions.

npLike

This function calculates non-parametric likelihood for a sample. Its arguments are **a matrix or data.frame object that holds the data** with different measurements in the columns with the rows being the realizations of those measurements [r-by-m] and **a vector containing the observed values** [m]. The value returned is a single value representing the likelihood of the sample.
Keep the number of measurements low, because the kernel density estimator only expects <5 dimensions. For steady state projects, if locations are within the length scale of correlation, they should be passed together into this function. If they are farther, they can be passed separately to another call of this function and the independent resulting likelihoods can be multiplied together. For time series, use some form of curve fitting to reduce the number of dimensions down to a few coefficients. Pass the coefficients to this function, as opposed to the time series (you can use the PCA functions written later in this file). Make sure to pass your observations' coefficients as well! NOTE: this function does not support observation error yet

```
npLike <- function(data,obs){
  #Calculate bandwidths
  bw <- npudensbw(data,bwmethod='normal-reference',ckertype='gaussian')
  #Calculate and return the probability of observation given realizations
  return(npudens(tdat=data,edat=t(as.matrix(obs)),bws=bw)$dens)
}
```

pLike

This function calculates parametric (normal assumption) likelihood for a sample. You can use this if the realizations of your measurements appear to be normally distributed. Its arguments are **a matrix or data.frame object that holds the data** with different measurements in the columns with the rows being the realizations of those measurements [r-by-m] and **a vector containing the observed values** [m]. The value returned is a single value representing the likelihood of the sample.

```
pLike <- function(data,obs){
  #Find the means of each variable
  mu <- apply(data,2,mean)
  #Find the covariance function
  sigma <- cov(data)
  #Find and return the density associated with the observations given the realizations
  return(dmvnorm(obs,mean=mu,sigma=sigma))
}
```

posterior

This function will calculate the (joint) posterior densities for the samples. The arguments are **a matrix the holds the prior samples** [s-by-p] with the different parameters in the columns and the rows being samples, and **a vector of likelihoods** of the samples. It returns a vector of posterior densities [s].

```
posterior <- function(prior,likelihood){
  prior_dens <- npudens(tdat=prior,
                        edat=prior,
                        bws=npudensbw(prior,bwmethod='normal-reference'))$dens
  posterior <- prior_dens*likelihood
  return(posterior/sum(posterior))
}
```

plotPosterior

This function will plot the prior versus the posterior. The arguments are **a matrix the holds the prior samples** [s-by-p] with the different parameters in the columns and the rows being samples, and **a vector of posterior densities** [s] of the samples. It returns a plot.

```
plotPosterior <- function(prior,posterior){
  plots <- plot(density(prior,weights=posterior), col="red")
  lines(density(prior))
  legend("topleft",legend=c("prior","posterior"),col=c("black","red"),pch=c("-","-"))
  return(plots)
}
```

timePCA

This function applies principal component analysis to a time series. The only argument is **a vector containing a time series**. The values returned are in a vector and are returns two rotation coefficients and two mean values. Only the first principal component is returned without assessing if other components are necessary to cover some proportion of the variance. By only using two segments and one principal component, only four coefficients are created for the likelihood calculations. This method reflects how PCA is implemented in MAD#, but is not the best method for all time series.

```
timePCA <- function(series){
  # split into 2 segments
  segments <- matrix(series,ncol=2)
  # apply PCA as if the two segments are two variables
  pca <- prcomp(segments,center=TRUE,scale=FALSE)
  # return the first component's coefficients and the means
  return(c(pca$rotation[,1],pca$center))
}
```

plotReconstruct

This function. . .

```r
plotReconstruct <- function(series,coeffs){
  recon <- as.vector(t(coeffs[1:2] %*%
               (t(coeffs[1:2]) %*%
                  (t(matrix(series,ncol=2)) - coeffs1[3:4])) + coeffs1[3:4]))
  plots <- plot(1:numTimeSteps,series,pch=16,
       main="Observed and Recontructed Time Series")
  points(1:numTimeSteps,recon,col="red")
  legend("topleft",legend=c("PC1","measured"),col=c("red","black"),pch=c(1,16))
  return(plots)
}
```