Auxiliary Program functions/subroutines

# Auxiliary Program functions/subroutines of the class TEUTILS\_CLASS

This class **contains** the following routines:

# function new\_unit() result (result) -- Returns a new unit for a new file to be open

# subroutine iswap(x,y) -- Swaps the values of two integers

# subroutine iswap\_vec(D,x,y) -- Swaps the values of two integer arrays of size D

# subroutine rswap(x,y) -- Swaps the values between two real variables

# subroutine rswap\_vec(x,y) -- Swaps the values between two real arrays of size D

# Interface Swap --- is an interface for iswap, iswap\_vec, rswap, rswap\_vec

# subroutine rperm(N0, N, p, IS) --- performs permutation of the array elements p from N0 to N

# subroutine BlockShuffle1D(N,x,m,tau,IS) --- performs block shuffling of the one-dimensional integer array X

# subroutine BlockShuffleND(N,ndim,x,m,tau,IS) --- performs block shuffling of the N-dimensional integer array X

# subroutine BlockShuffleND2(N,ndim,x,m,tau,IS) --- performs block shuffling of the N-dimensional real array X

# subroutine BlockShuffle1D2(N,x,m,tau,IS) --- performs block shuffling of the one-dimensional real array X

# subroutine shuffleND(n,ndim,x,is) --- performs simple shuffling of the N-dimensional integer array X

# subroutine shuffle1D(n,x,is) --- performs simple shuffling of the one-dimensional integer array X

# subroutine shuffleND2(n,ndim,x,is) --- performs simple shuffling of the N-dimensional real array X

# subroutine shuffle1D2(n,x,is) --- performs simple shuffling of the one-dimensional real array X

# FUNCTION psi(xx) RESULT(fn\_val) -- computes the derivative of the gamma function

# subroutine addStrings\_slow(a,b) -- Slow (depending on the machine) routine of adding two strings

# subroutine addStrings(a,b) --- Fast (depending on the machine) routine of adding two strings

# SUBROUTINE addStrings2(ST,STMAX,STLEN,ADST,ADLEN) --- Fast (depending on the machine) routine of adding two strings

# function toString(a) result(b) - it converts integer to string

# function toInteger(a) result(b) -- it converts string to integer

# subroutine UPCASE (STRING) --- it converts a string to upper case letters

# subroutine DOWNCASE (STRING) --- it converts a string to lower case letters

# function mean(X,n) result(r) --- computer the mean value of the elements of a vector

# function rmin1d(X) --- Finds the minimum of an array of real numbers

# function rmax1d(X) --- Finds the maximum of an array of real numbers

# function imax1d(X) result(r) --- Finds the maximum of an array of integer numbers

# function imin1d(X) result(r) --- Finds the minimum of an array of integer numbers

# subroutine sort(n,arr,indx) --- It sorts elements of an array (machine depending)

# subroutine sort2(n,r0,indx0,indx) --- It sorts elements of an array (machine depending)

# function vcopy(Xin,istart,M,istep,Sign) result(Xou) --- it copies elements of an array to another

# function getCovariance(Nframes,X, Y) result(Cov) --- it computes covariance matrix

# function find(i,list) result(r) --- it find the element i in the list of elements

# Subroutine statTest(N, r0, val\_in, val\_out, stat\_prob) -- performs a statistical test

# FUNCTION erfcc\_s(x); FUNCTION erfcc\_v(x) --- it computes the error function value (with interface erfcc)

# function square(x) result(r) -- computes the square of a number

# Subroutine TransferFunctionInn\_2d(z, r, ifunc) --- transfer function of Machine Learning (two-dimensional array)

# subroutine TransferFunctionInn\_1d(Z, R, ifunc) --- transfer function of Machine Learning (one-dimensional array)

# function TransferFunctionInn\_0d(z, ifunc) result(r) --- definition of the transfer function for Machine Learning

# Subroutine TransferFunctionInn\_derivative\_2d(z, r, ifunc) -- derivative of transfer function of Machine Learning (two-dimensional array)

# subroutine TransferFunctionInn\_derivative\_1d(Z, R, ifunc) --- derivative of transfer function of Machine Learning (one-dimensional array)

# function TransferFunctionInn\_derivative\_0d(z, ifunc) result(r) --- definition of the derivative of transfer function for Machine Learning

# subroutine getTimeLag(A\_n, Tau) --- compute the time lag from the correlation function

# Auxiliary Program functions/subroutines of the class TELinkList\_class

This class contains the following routines:

# subroutine LinkList(head,tail,temp) --- it builds up the link list

# function compareStrings(a,b) result(r) --- a function to compare two strings if they are equal or not

# subroutine LocalLinkList(head,tail,temp,visited\_state) --- builds up a local link list for each visited state

# subroutine freeList(head) --- deletes the link list and its heap memory storage

# DeleteList(head) -- it just deletes the link list

# function SymbolicShannonEntropy(head) result(Entropy) --- it computes the Shannon entropy of symbolic states

# subroutine SymbolicJointProbabilityTraj(head, total, prob, frequency) -- It computes the joint prob distribution of symbolic trajectory of states

# function getTotalStates(head) result(total) -- gets the total number of visited states

# function CountStates(head) result(Nstates) -- the same as 'getTotalStates'

# Program Control Variables

## Program Control Variables in the class SIFM\_KINDS

# This class contains the following definitions in the file 'defs.f90':

! Kind constants for system-independent precision specification.

! H. Kamberaj

! integer, parameter :: sifm\_real4 = kind(0.) !-- real single precision

! integer, parameter :: sifm\_real8 = kind(0.d0) !-- real double precision

!

!-- Kinds for specified integer ranges.

! integer, parameter :: sifm\_int1 = selected\_int\_kind(2) !-- 99 max

! integer, parameter :: sifm\_int2 = selected\_int\_kind(4) !-- 9,999 max

! integer, parameter :: sifm\_int4 = selected\_int\_kind(8) !-- 99,999,999 max

! integer, parameter :: sifm\_int8 = selected\_int\_kind(16) !-- 9,999,999,999,999,999 max

!

!-- Kind for working real precision.

! integer, parameter :: sifm\_real = sifm\_real8

!-- Big constants. These are less than huge so that we have some

!-- room for comparisons and other arithmetic without overflowing.

! real(sifm\_real), parameter :: rzero = 0.0\_sifm\_real

! real(sifm\_real), parameter :: one = 1.0\_sifm\_real

! real(sifm\_real), parameter :: toBits = one/log(2.0\_sifm\_real)

! real(sifm\_real), parameter :: half = 0.5\_sifm\_real

! real(sifm\_real), parameter :: M\_PI = 3.141592653589793238462643383279502884197\_sifm\_real

! real(sifm\_real), parameter :: M\_2PI = 6.283185307179586476925286766559005768394\_sifm\_real

! real(sifm\_real), parameter :: ToDeg = 180.0\_sifm\_real/M\_PI

! real(sifm\_real), parameter :: LOG\_ZERO=-30.0\_sifm\_real

! real(sifm\_real), parameter :: max\_exp = maxexponent(rzero)

! real(sifm\_real), parameter :: min\_exp = minexponent(rzero)

! real(sifm\_real), parameter :: kB = 0.00198614\_sifm\_real

! REAL(sifm\_real), PARAMETER :: SQRT2=1.41421356237309504880168872420969807856967\_sifm\_real

! real(sifm\_real), parameter :: PRTINY = tiny(rzero) \* 16.0\_sifm\_real

! real(sifm\_real), parameter :: PRBIG = huge(sifm\_real) / 16.0\_sifm\_real

! integer, parameter :: PIBIG = huge(sifm\_int8)/16

! character(len=1), dimension(26) :: Symbol = (/'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', &

! 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', &

! 's', 't', 'u', 'v', 'w', 'x', 'y', 'z' /)

#

# Parallelization

#

## Program Control Variables in the class TEMPI\_CLASS for the MPI version

# This class contains the following definitions in the file 'defs\_mpi.f90':

!type val\_t

! real(sifm\_real) :: value

! integer :: rank

!end type val\_t

! integer, save :: NumProcs = 1

! integer, save :: MyID = 0

! integer, parameter :: master = 0

! integer , parameter :: msgtag2 = 12

! integer, save, dimension ( MPI\_STATUS\_SIZE ) :: status

!contains

# subroutine MPI\_start() --- Start MPI communications

# subroutine MPI\_finish() --- Finalizes the MPI communications

# Random number generator routines

# Random number generator routines in the class TERANDOM\_CLASS

# This class contains the following definitions in the file 'random.f90':

# Contains:

# function RANDF(ig) result(random\_rtn) --- Returns a double precision uniformly distributed random number in (0,1) from the ig-th stream

# subroutine SetSeed (ig,s) -- sets seed value s for stream ig

# FUNCTION GAUSS\_rand(g) result(RND) --- generates a Gaussian random deviate of 0.0 mean and standard deviation 1 for the stream g\\

# subroutine rndnum\_iniall(nrnd, rngc, mynod) --- it initialize all needed for random number generators:

# --- whether a sequence of Gaussians or u(0,1)

# --- number of streams

# --- and the processor ID (MPI version)