# FRACLAB

## 0.1 AtanH — Arctangent variation

Author: Paulo Goncalves

Generates an arc-tangent trajectory

**USAGE :**

```
Ht=AtanH(N,h1,h2,shape);
```

**INPUT PARAMETERS :**

- N : Positive integer Sample size of the trajectory
- h1 : Real scalar First value of the arc-tangent trajectory
- h2 : Real scalar Last value of the arc-tangent trajectory
- shape : real in [0,1] smoothness of the trajectory shape = 0 : constant piecewise (step function) shape = 1 : linear

**OUTPUT PARAMETERS :**

- Ht : real vector [1,N] Time samples of the arc-tangent trajectory

**SEE ALSO: :**

**EXAMPLE: :**

```
Ht = AtanH(1024,0,1,0.9) ;
```

## 0.2 FWT — 1D Forward Discrete Wavelet Transform

**USAGE :**
[wt,index,length]=FWT(Input,NbIter,f1,[f2])

**INPUT PARAMETERS :**

- Input : real matrix [1,n] or [n,1] Contains the signal to be decomposed.
- NbIter : real positive scalar Number of decomposition Levels to compute
- f1 : Analysis filter
- f2 : real unidimensional matrix [m,n] Synthesis filter. Useful only for biorthogonal transforms. If not precised, the filter f1 is used for the synthesis.

**OUTPUT PARAMETERS :**

- wt : real matrix Wavelet transform. Contains the wavelet coefficients plus other informations.

- index : real matrix [1,NbIter+1] Contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces
- length : real matrix [1,NbIter+1] Contains the dimension of each projection

**DESCRIPTION :**

This routine computes discrete wavelet transforms of a 1D real signal. Two transforms are possible : Orthogonal and Biorthogonal

**INTRODUCTION :**

The discrete wavelet transform of Input is a projection on multiresolution Spaces. The number of scales NbIter tells how many convolutions are computed. Each convolution is followed by a downsampling of the output. For example, if the original signal size is 500, the resulting size of the projection after the first iteration is 250. Each iteration consists then in two convolution/downsampling steps. One is high-pass (H) and the other one is low-pass (L). Except for the last iteration, the low-pass output is used as the input of the next iteration. Thus, only the high-pass is stored in wt except at the last iteration where both the outputs are stored. This explains why the wti array dimension is equal to NbIter+1. The last index index(NbIter+1) is the index of firts element of the last low-pass projection. Two types of filters are available : Quadrature Mirror Filters (Orthogonal) or Conjugate Quadrature Filters (Biorthogonal). Each one allows perfect reconstruction of the signal but only CQF pairs can be symetric. The advantage of QMF is that synthesis and reconstruction filters are the same.

**PARAMETERS :**

Input must be a real unidimensional matrix. NbIter is the number of scales computed. It must be a positive integer greater than one and should be smaller than log2(max(size(Input))) but this is not necessary. f1 is the linear FIR filter used for the analysis and might be obtained with MakeQMF() or MakeCQF() f2 is the linear FIR filter to use for the reconstruction. It is only necessary if f1 has been obtained with MakeCQF(). wt is the wavelet decomposition structure. The next two parametres must be used to read the wavelet coefficients. index contains the indexes of the first coefficient of each output. length contains the dimension of each output.

**ALGORITHM DETAILS :**

Convolutions are computed through discrete linear convolutions in time domain. No FFT is used. The signal is mirrored at its boundaries. The wavelet structure contains all the informations for the reconstruction: wt(1) : size of the original signal wt(2) : Number of iterations wt(3) : Number of causal coefficients of the synthesis filter wt(4) : Number of anticausal coefficients of the synthesis filter then the Synthesis filter coefficients and finaly the wavelet coefficient are stored .

**EXAMPLES :**

```
a=rand(1,250);
q=MakeQMF('daubechies',4);
[wt,wti,wtl] = FWT(a,6,q);
M=WTMultires(wt);
plot(M(2,:));
//Then to suppress the Lowest Frequency component and then reconstruction:
for i=1:wtl(6),wt(wti(6)+i-1)=0;end;
result=IWT(wt);
```

**REFERENCES :**

Meyer Y. : Wavelets, Algorithms & Applications, SIAM. Meyer Y. : Ondelettes et Operateurs (I) : Hermann, Paris Daubechies I. : Ten Lectures on Wavelets, CBMS-NSF Regional conference series in applied mathematics.

AUTHOR : Author: Bertrand Guiheneuf

**SEE ALSO :**

IWT, MakeQMF, MakeCQF, WTStruct, WTNbScales, WTMultires

# 0.3 FWT2D 2D Forward Disrete Wavelet Transform

Author: Bertrand Guiheneuf

This routine computes discrete wavelet transforms of a 2D real signal. Two transforms are possible : Orthogonal and Biorthogonal

**USAGE :**

[wt,index,length]=FWT2D(Input,NbIter,f1,[f2])

**INPUT PARAMETERS :**

- Input : real matrix [m,n] Contains the signal to be decomposed.
- NbIter : real positive scalar Number of decomposition Levels
- f1 : Analysis filter
- f2 : real unidimensional matrix [m,n] Synthesis filter

**OUTPUT PARAMETERS :**

- wt : real matrix Wavelet transform. Contains all the datas of the decomposition.
- index : real matrix [NbIter,4] Contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces
- length : real matrix [NbIter,2] Contains the dimensions of each projection

**DESCRIPTION :**

**INTRODUCTION :**

The 2D discrete wavelet transform of Input is a projection on 2D multiresolution Spaces. The number of scales NbIter tells how many convolutions are computed. Each convolution is followed by a downsampling of the signal in both direction. For example, if the original matrix is (256,512), a resulting projection after the first iteration is (128,256). In 2D, there are 4 projections for each iteration corresponding to 2 projections in the row directions and 2 in the column direction. In each direction, the 2 projections are obtained through the convolutions with a low-pass filter and its associated high-pass filter. The projections are then HL HH LH LL where the first letter represents the filter used for the row filtering and the second letter is the filter used for column filtering. H is High-Pass filter and L Low-pass filter. Except for the last level where the four convolutions are kept, the LL output is always used as the input of the following iteration. Two types of filters are available : Quadrature Mirror Filters (Orthogonal) or Conjugate Quadrature Filters (Biorthogonal). Each one allows perfect reconstruction of the signal but only CQF pairs can be symetric. The advantage of QMF is that synthesis and reconstruction filters are the same.

**PARAMETERS :**

Input must be a real matrix. All dimensions are allowed but for a 1D vector, FWT is best suited. NbIter is the number of scales computed. It must be a positive integer greater than one and should be smaller then log2(max(size(Input))) but this is not necessary. f1 is the linear FIR filter used for the analysis and might be obtained with MakeQMF() or MakeCQF() f2 is the linear FIR filter to use for the reconstruction. It is only necessary if f1 has been obtained with MakeCQF(). wt is the wavelet decomposition structure. The next two parametres must be used to read the wavelet coefficients. index contains the indexes of the first coefficient of each output. At each scale Scale, the output indexes are: index(Scale,1) : HL index(Scale,2) : LH index(Scale,3) : HH index(Scale,4) : LL on the last scale and 0 otherwise length contains the dimensions (height, width) of each output at a given Iteration.

**ALGORITHM DETAILS :**

Convolutions are computed through discrete linear convolutions in time domain. No FFT is used. The signal is mirrored at its boundaries. The wavelet structure (wt) is a vector and NOT a 2D matrix. It contains all the informatiosn for the reconstruction: wt(1) : height of the original signal wt(2) : width of the original signal wt(3) : Number of iterations wt(4) : Number of causal coefficients of the synthesis filter wt(5) : Number of anticausal coefficients of the synthesis filter then the Synthesis filter coefficients and finaly the wavelet coefficient are stored .

**EXAMPLES :**

```
a=rand(256,256);
q=MakeQMF('daubechies',4);
[wt,wti,wtl] = FWT2D(a,3,q);
V=WT2Dext(wt,1,2);
viewmat(V);
//Then to suppress the Lowest Frequency component and then reconstruction:
index=0;
for i=1:wtl(3,1),
  for j=1:wtl(3,2),wt(wti(3,4)+index)=0;end
end;
result=IWT2D(wt);
```

**REFERENCES :**

Meyer Y. : Wavelets, Algorithms & Applications, SIAM. Meyer Y. : Ondelettes et Operateurs (I) : Hermann, Paris Daubechies I. : Ten Lectures on Wavelets, CBMS-NSF Regional conference series in applied mathematics.

**SEE ALSO :**

IWT2D, MakeQMF, MakeCQF, WT2Dext, WT2DVisu, WT2DStruct

## 0.4 GeneWei Generalized Weierstrass function

Author: Paulo Goncalves

Generates a Generalized Weierstrass function

**USAGE :**

```
[x,Ht]=GeneWei(N,ht,lambda,tmax,randflag)
```

**INPUT PARAMETERS :**

- N : Positive integer Sample size of the synthesized signal
- ht : Real vector or character string ht : real vector of size [1,N]: each element prescribes the local Holder regularity of the function. All elements of ht must be in the interval [0,1]. ht : character string : contains the analytic expression of the Holder trajectory (e.g. '0.5*sin(16*t) + 0.5')
- lambda : positive real Geometric progression of the Weierstrass function. Default value is lambda = 2.
- tmax : positive real Time support of the Weierstrass function. Default value is tmax = 1.
- randflag : flag 0/1 flag = 0 : deterministic Weierstrass function flag = 1 : random Weierstrass process Default value is randflag = 0

**OUTPUT PARAMETERS :**

- x : real vector [1,N] Time samples of the Weierstrass function
- Fj : real vector [1,N] Holder trajectory of the Weierstrass function

**SEE ALSO: :**

**EXAMPLE :**

```
[x,Ht] = GeneWei(1024,'abs(sin(16*t))',2,1,0) ;
```

## 0.5 IWT 1D Inverse Discrete Wavelet Transform

Author: Bertrand Guiheneuf

This routine computes inverse discrete wavelet transforms of a real signal. Two inverse transforms are possible : Orthogonal and Biorthogonal

**USAGE :**
[result]=IWT2D(wt,[f])
**INPUT PARAMETERS :**

- wt : real unidimensional matrix [m,n] Contains the wavelet transform (obtained with FWT).
- f : real unidimensional matrix [m,n] Synthesis filter.

**OUTPUT PARAMETERS :**

- result : real unidimensional matrix Result of the reconstruction.

**DESCRIPTION :**

**INTRODUCTION :**
The wavelet transform is an invertible linear transform. This routines is the inverse transform. For details on the algorithm procedure, see FWT.
**PARAMETERS :**
Input must be a real matrix. It's generally obtained with FWT but can be created "by hand". In that case, it's strongly recommended to decompose a null signal with FWT. f is the linear FIR filter to use for the reconstruction. It is only necessary if the analysis filter had been obtained with MakeCQF() and the reconstruction filter had not been passed to FWT. If not specified the filter given in the synthesis is used. (See FWT). result is the reconstructed signal. It has the same dimension as the original one.
**ALGORITHM DETAILS :**
Convolutions are computed through discrete linear convolutions in time domain. No FFT is used. The signal is mirrored at its boundaries.
**EXAMPLE :**

```
a=rand(1,256);
q=MakeQMF('daubechies',4);
wt,wti,wtl = FWT(a,8,q);
wt=abs(wt);
result=IWT(wt);
```

**REFERENCES :**

Meyer Y. : Wavelets, Algorithms & Applications, SIAM. Meyer Y. : Ondelettes et Operateurs (I) : Hermann, Paris Daubechies I. : Ten Lectures on Wavelets, CBMS-NSF Regional conference series in applied mathematics.

**SEE ALSO :**

FWT, MakeQMF, MakeCQF, WTMultires, WTStruct

## 0.6 IWT2D 2D Inverse Disrete Wavelet Transform

Author: Bertrand Guiheneuf

This routine computes inverse discrete wavelet transforms of a 2D real signal. Two inverse transforms are possible : Orthogonal and Biorthogonal

**USAGE :**

[result]=IWT2D(wt,[f])

**INPUT PARAMETERS :**

- wt : real unidimensional matrix [m,n] Contains the wavelet transform (obtained with FWT2D).
- f : real unidimensional matrix [m,n] Synthesis filter.

**OUTPUT PARAMETERS :**

- result : real matrix Result of the reconstruction.

**DESCRIPTION :**

**INTRODUCTION :**

The wavelet transform is an invertible linear transform. This routines is the inverse transform. For details on the algorithm procedure, see FWT2D.

**PARAMETERS :**

Input must be a real matrix. It's generally obtained with FWT2D but can be created "by hand". In that case, it's strongly recommended to decompose a null signal with FWT2D. f is the linear FIR filter to use for the reconstruction. It is only necessary if the analysis filter had been obtained with MakeCQF() and the reconstruction filter had not been passed to FWT2D. If not specified the filter given in the synthesis is used. (See FWT2D). result is the reconstructed signal. It has the same dimensions as the original one.

**ALGORITHM DETAILS :**

Convolutions are computed through discrete linear convolutions in time domain. No FFT is used. The signal is mirrored at its boundaries.

**EXAMPLE :**

```
a=rand(256,256);
q=MakeQMF('daubechies',4);
wt,wti,wtl = FWT2D(a,8,q);
wt=abs(wt);
result=IWT2D(wt);
```

**REFERENCES :**

Meyer Y. : Wavelets, Algorithms & Applications, SIAM. Meyer Y. : Ondelettes et Operateurs (I) : Hermann, Paris Daubechies I. : Ten Lectures on Wavelets, CBMS-NSF Regional conference series in applied mathematics.

**SEE ALSO :**

FWT2D, MakeQMF, MakeCQF, WT2Dext, WT2DVisu

## 0.7 Koutrouvelis _ Stable Law parameters estimation (Koutrouvelis method)

Author: Lotfi Belkacem

This routine estimates parameters of a stable law using the Koutrouvelis (1985) method.

**USAGE :**

[alpha,beta,mu,gamma]=Koutrouvelis(data)

**INPUT PARAMETERS :**

- proc : real vector [size,1] corresponding to the data sample.

**OUTPUT PARAMETERS :**

- alpha : real positive scalar between 0 and 2. This parameter is often referred to as the characteristic exponent.
- beta : real scalar between -1 and +1. This parameter is often referred to as the skewness parameter.
- mu : real scalar This parameter is often referred to as the location parameter. It is equal to the expectation when alpha is greater than 1.
- gamma : real positive scalar. This parameter is often referred to as the scale parameter. It is equal to the standard deviation over two squared when alpha equal 2.

**EXAMPLE :**

```
[proc1,inc1]=sim_stable(1,0,0,1,10000);
//generates a standard 1-stable motion (Cauchy process).
[alpha,beta,mu,gamma]=Koutrouvelis(inc1);
//estimates parameters of the previous simutated 1-stable random
//variable inc1.
```

**REMARQUE :**

Skewness and location parameters are badly estimated with this methode.

## 0.8 McCulloch ____ Stable law parameters estimation (McCulloch method)

Author: Lotfi Belkacem

This routine estimates parameters of a stable law using the Mc-Culloch (1985) method.

**USAGE :**
[param,sd_param]=McCulloch(data)
**INPUT PARAMETERS :**

- data : real vector [size,1] corresponding to the data sample.

**OUTPUT PARAMETERS :**

- param : real vector [4,1] corresponding to the four estimated parameters of the fited stable law. the order is respectively alpha (characteristic exponent), beta (skewness parameter), mu (location parameter), gamma (scale parameter)
- sd_param : real vector [4,1] corresponding to estimated standard deviation of the four previous parameters.

**EXAMPLE :**

```
//generates a standard 1.5-stable motion.
[proc1.5,inc1.5]=sim_stable(1.5,0,0,1,10000);
//estimates parameters of the previous simutated 1.5-stable
//random variable inc1.5 To visualize the estimates parameters
//or their sd-deviations use respectively  param or sd_param.
[param,sd_param]=McCulloch(inc1.5);

alpha=param(1), beta=param(2), mu=param(3), gamma=param(4).
sd_alpha=sd_param(1), sd_alphabeta=sd_param(2),
sd_alphamu=sd_param(3), sd_gamma=sd_param(4).
```

**REMARQUE :**
Skewness parameter and its sd-deviation estimations are not very accurate. Specially when the characteristic exponent is arround 2.

## 0.9 WT2DStruct _____ Retrieve the Structure of a 2D DWT

Author: Bertrand Guiheneuf

This routine retrieve the structure informations contained in a 2D Wavelet Transform.

**USAGE :**
[ScIndex, ScLength]=WT2DStruct(wt)
**INPUT PARAMETERS :**

- wt : real unidimensional matrix [m,n] Contains the wavelet transform (obtained with FWT2D).

**OUTPUT PARAMETERS :**

- index : real matrix [NbIter,4] Contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces
- length : real matrix [NbIter,2] Contains the dimensions of each projection

**DESCRIPTION :**

**INTRODUCTION :**
This routine is used to retreive the structure information of a wavelet transform. It must be used in all routine that might work on a wavelet transform whose structure is not passed as an imput parameter. (That should be the case of all routines taking a Wavelet Transform as input parameter to minimize the input).see FWT2D.

**PARAMETERS :**
Input must be a real matrix. It's generally obtained with FWT2D. It contains the wavelet transform. index contains the indexes of the first coefficient of each output. At each scale Scale, the output indexes are: index(Scale,1) : HL index(Scale,2) : LH index(Scale,3) : HH index(Scale,4) : LL on the last scale and 0 otherwise length contains the dimensions (height, width) of each output at a given Iteration.

**EXAMPLE :**

```
a=rand(256,256);
q=MakeQMF('daubechies',4);
wt = FWT2D(a,8,q);//(a few days pass...)
[wti, wtl]=WT2DStruct(wt);
wtl
```

**SEE ALSO :**
FWT2D, IWT2D, WT2Dext, WT2DVisu

## 0.10 WT2DVisu Visualise a 2D Multiresolution

Author: Bertrand Guiheneuf

This routine constructs a matrix that shows all the wavelet coefficients of a 2D matrix.

**USAGE :**
[V]=WT2DVisu(wt)

**INPUT PARAMETER :**

- wt : real unidimensional matrix [m,n] Contains the wavelet transform (obtained with FWT2D).

**OUTPUT PARAMETER :**

- V : real matrix [m,n] Contains a matrix to be visualized directly

**DESCRIPTION :**

**INTRODUCTION :**
This routine is used to display all the scales and all the frequency components of a wavelet transform.

**PARAMETERS :**
wt must be a real matrix. It's generally obtained with FWT2D. V the wavelet coefficents.

**EXAMPLE :**

```
a=rand(256,256);
q=MakeQMF('daubechies',4);
wt = FWT2D(a,8,q);
V=WT2DVisu(wt);
viewmat(V);
```

**SEE ALSO :**
FWT2D, IWT2D, WT2Dext,

## 0.11 WT2Dext Extract a Projection from a 2D WT

Author: Bertrand Guiheneuf

This routine extracts a projection from the wavelet transform of a 2D matrix.

**USAGE :**
[V]=WT2Dext(wt, Scale, Num)

**INPUT PARAMETER :**

- wt : real unidimensional matrix [m,n] Contains the wavelet transform (obtained with FWT2D).
- w Scale : real scalar Contains the scale level of the projection to extract.
- w Num : real scalar Contains the number of the output to extract in level Scale (between 1 and 4)

**OUTPUT PARAMETER :**

- V : real matrix [m,n] Contains the matrix to be visualized directly

**DESCRIPTION :**

**INTRODUCTION :**
At each scale, a wavelet transform contains 4 outputs (HL, HH, LH and HH at the last scale). This routine is used to extract a particular component at a desired scale.

**PARAMETERS :**
wt must be a real matrix. It's generally obtained with FWT2D. It contains the wavelet transform coefficients. Num is 1,2,3, or 4 (at the last scale). Each number coresponds to a particular 2D frequency component.

- 1 : HL High frequency in row direction, Low in column direction.
- 2 : HH High frequency in row direction, High in column direction.
- 3 : LH Low frequency in row direction, High in column direction.

- 4 : LL Low frequency in row direction, Low in column direction. Only for the last scale (equals 0 for the other scales).

  V the wavelet coefficents at scale Scale with fequency component given by Num

**EXAMPLE :**

```
a=rand(256,256);
q=MakeQMF('daubechies',4);
wt = FWT2D(a,8,q);
V=WT2Dext(wt,2,2);
viewmat(V);
```

**SEE ALSO :**
FWT2D, IWT2D, WT2DVisu,

## 0.12 WTMultires _ Construct a 1D Multiresolution Representation

Author: Bertrand Guiheneuf

This routine constructs a matrix that shows the projections of the signal on each multiresolution subspace

**USAGE :**
[V]=WTMultires(wt)
**INPUT PARAMETER :**

- wt : real unidimensional matrix Contains the wavelet transform (obtained with FWT).

**OUTPUT PARAMETER :**

- V : real matrix [Nbiter,n] Contains the projections on the Multiresolution. Each line is a projection on a subspace different "low-pass" space Vj

**DESCRIPTION :**

**INTRODUCTION :**
This routine is used to display all the scales of a wavelet transform. The projections are different from the wavelet coefficients as they represent "filtered" signals. Here each projection
**PARAMETERS :**
wt must be a real matrix containing the wavelet coefficients but also misc informations such as the original signal dimension and the reconstruction filter. It's generally obtained with FW. V Is the matrix containing the projection of the signal (decomposed in wt) on each Multiresolution subspace. The Nbiter first ones are the projections on the details subspaces. The last one is the projection on the last trend subspace.
**EXAMPLE :**

```
x=0.1:0.005:1;
s=(x.^0.7) .* sin( x.^(-2)) ;
q1 q2=MakeCQF(1);
wt = FWT(s,3,q1,q2);
V=WTMultires(wt);
plot(V');
```

**SEE ALSO :**
FWT, IWT, WTStruct,

## 0.13 WTStruct _____ Retrieve a 1D Discrete Wavelet Structure.

Author: Bertrand Guiheneuf

This routine retrieves the structure informations contained in a 1D Wavelet Transform.

**USAGE :**

```
[ScIndex, ScLength]=WT2DStruct(wt)
```

**INPUT PARAMETERS :**

- wt : real unidimensional matrix [1,n] Contains the wavelet transform (obtained with FWT).

**OUTPUT PARAMETERS :**

- index : real matrix [1,NbIter] Contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces
- length : real matrix [1,NbIter] Contains the dimensions of each projection

**DESCRIPTION :**

**INTRODUCTION :**
This routine is used to retreive the structure information of a wavelet transform. It must be used in all routine that might work on a wavelet transform whose structure is not passed as an imput parameter. (That should be the case of all routines taking a Wavelet Transform as input parameter to minimize the input).

**PARAMETERS :**
Input must be a real matrix. It's generally obtained with FWT. It contains the wavelet transform. index contains the indexes of the first coefficient of each output. The first "NbIter" indexes are the indexes (in wt) of the "high-pass" subspaces projections (Wj), the last one is the last "low-pass" projection (Vj); length contains the dimension of each output.

**EXAMPLE :**

```
a=rand(1,256);
q=MakeQMF('daubechies',4);
wt = FWT2D(a,8,q);
[wti, wtl]=WTStruct(wt);
wtl
```

**SEE ALSO :**
FWT2D, IWT2D, WT2Dext, WT2DVisu

## 0.14 alphagifs Holder function estimation using IFS

Author: Khalid Daoudi

Estimates the pointwise Holder exponents of a 1-D real signal using the GIFS method.

**USAGE :**
[Alpha, Ci]=wave2gifs(sig, limtype)

**INPUT PARAMETERS :**

- sig : Real vector [1,n] or [n,1] Contains the signal to be analysed.
- limtype : Character string Specifies the type of limit you want to use. You have the choice between 'slope' and 'cesaro'.

**OUTPUT PARAMETERS :**

- Alpha : Real vector Contains the estimated Holder function of the signal.
- Ci : Real matrix Contains the GIFS coefficients obtained using the Schauder basis.

**DESCRIPTION :**

**PARAMETERS :**

- sig is a real vector [1,n] or [n,1] which contains the signal to be analysed.
- limtype is a character string Specifies the type of limit you want to use. You have the choice between 'slope' and 'cesaro'.
- Alpha is a real vector which contains the estimated Holder function of the signal i.e the estimated pointwise Holder exponent a each point of the given signal.
- Ci is a real matrix which contains the GIFS coefficients obtained as the ration between (synchrounous) Schauder basis coefficients at succesive scales.

**ALGORITHM DETAILS :**
The algorithm uses the GIFS method to estimate the Holder exponent at each point of a given signal. The first step of this method consists in computing the coefficients of the GIFS whose attractor is the given signal. In the second step, we replace each coefficient which absolute value is greater than 1 (resp. smaller than 1/2) by 1 (resp. by 1/2). We then perform the computation of the limit that yields the estimated Holder function using the chosen type of limit.

**SEE ALSO: :**
gifs and prescalpha

**EXAMPLE: :**

```
//Synthesis of an fbm with exponent H=0.7 (of size 1024 samples):
x = fmblevinson(1024,0.7) ;
//Estimation of The Holder function :
Alpha = alphagifs(x,'slope');
plot(Alpha)
```

## 0.15 bbch — beneath-beyond concave hull

Author: Christophe Canus

This C_LAB routine determines the concave hull of a function graph using the beneath-beyond algorithm.

**USAGE :**

```
[rx,ru_x]=bbch(x,u_x)
```

**INPUT PARAMETERS :**

- x : real vector [1,N] or [N,1] Contains the abscissa.
- u_x : real vector [1,N] or [N,1] Contains the function to be regularized.

**OUTPUT PARAMETERS :**

- rx : real vector [1,M] Contains the abscissa of the regularized function.
- ru_x : real vector [1,M] Contains the regularized function.

**DESCRIPTION :**

**PARAMETERS :**
The abscissa x and the function u_x to be regularized must be of the same size [1,N] or [N,1]. The abscissa rx and the concave regularized function ru_x are of the same size [1,M] with M<=N.

**ALGORITHM DETAILS :**
Standard beneath-beyond algorithm.

**EXAMPLES :**

```
h=.3;beta=3;
N=1000;
// chirp singularity (h,beta)
x=linspace(0.,1.,N);
u_x=abs(x).^h.*sin(abs(x).^(-beta));
plot2d(x,u_x);
[rx,ru_x]=bbch(x,u_x);
plot2d(rx,ru_x,style=5);
plot2d(x,abs(x).^h,style=3);
```

**REFERENCES :**
None.SH See Also linearlt (C_LAB routine).

## 0.16 binom — binomial measure synthesis

Author: Christophe Canus

This C_LAB routine synthesizes a large range of pre-multifractal measures related to the binomial measure paradigm (deterministic, shuffled, pertubated, and mixing of two binomials: lumping and sum) and computes linked theoretical functions (partition sum function, Reyni exponents function, generalized dimensions, multifractal spectrum).

**USAGE :**

```
[varargout [,optvarargout]]=binom(p0,str,varargin,[optvarargin])
```

**INPUT PARAMETERS :**

- p0 : strictly positive real scalar Contains the weight of the binomial.
- str : string Contains the type of ouput.
- varargin : variable input argument Contains the variable input argument.
- optvarargin : optional variable input arguments Contains optional variable input arguments.

**OUTPUT PARAMETERS :**

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

**DESCRIPTION :**

**PARAMETERS :**

The binomial measure is completly characterized by its weight p0. This first parameter must be $>0.$ and $<1.$ (the case of p0=.5 corresponds to the Lebesgue measure). The second parameter str is a variable string used to determine the desired type of output. There are six suffix strings ('meas' for measure, 'cdf' for cumulative distribution function, , 'pdf' for probability density function, 'part' for partition sum function, 'Reyni' for Reyni exponent function , 'spec' for multifractal spectrum) for the deterministic binomial measure and a lot of possibly composed prefix strings for related measures ('shuf' for shuffled, 'pert' for pertubated, 'lump' for lumping , 'sum' for sum, 'sumpert' for sum of pertubated, and so on) which can be added to the first ones to form composed strings. For example, 'lumppertmeas' is for the synthesis of the lumping of 2 pertubated binomial pre-multifractal measures and 'sumspec' is for the computation of the multifractal spectrum of the sum of two binomials. Note that all combinaisons of strings are not implemented yet. When a string containing suffix string 'meas' is given as second input, a pre-multifractal measure mu_n (first output argument) is synthesized on the dyadic intervals I_n (second optional output argument) of the unit interval. In that case, the third input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real vectors mu_n (and I_n if used) is equal to 2n (so be aware the stack size ;-)). This option is implemented for the deterministic ('meas'), shuffled ('shufmeas') and pertubated ('pertmeas') binomial, and also for the mixing (lumping or sum) of two deterministic ('lumpmeas' and 'summeas') or pertubated ('lumppertmeas' and 'sumpertmeas') binomial measures. When a string containing prefix 'shuf' is given as second input, the synthesis is made for a shuffled binomial measure. At each level of the multiplicative cascade and for all nodes of the corresponding binary tree, the weight is chosen uniformly among p0 and 1-p0. This option is implemented only for the binomial measure ('shufmeas'). When a string containing prefix 'pert' is given as second input, the synthesis is made for a pertubated binomial measure. In that case, the fourth input argument is a strictly positive real scalar epsilon which contains the pertubation around weights. The weight is an independant random variable identically distributed between p0-epsilon and p0+epsilon which must be $>0.$, $<1.$ This option is implemented only for the binomial measure ('pertmeas') and the mixing (lumping and sum) of two binomial measures ('lumppertmeas' and 'sumpertmeas'). When replacing suffix string

'meas' with suffix string 'cdf', respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument). When string 'part' is given as second input, the partition sum function znq of multifractal measure is computed as sole output argument. In that case, the third input argument is a strictly positive real (integer) vector vn which contains the resolutions, and the fourth input argument is a real vector q which contains the measure exponents. The size of the output real matrix znq is equal to size(q)*size(vn). This option is implemented only for the binomial measure. When string 'Reyni' is given as second input, the Reyni exponents function tq (and the generalized dimensions Dq if used) of the multifractal measure is computed as first output argument (and second optional output argument if used). In that case, the third input argument is a real vector q which contains the measure's exponents. The size of the output real vector tq is equal to size(q)). This option is implemented only for the binomial measure. When a string containing suffix string 'spec' is given as second input, the multifractal spectrum f_alpha (second output argument) is synthesized on the Hoelder exponents alpha (first output argument). In that case, the third input argument is a strictly positive real (integer) scalar N which contains the number of Hoelder exponents. The size of both output real vectors alpha and f_alpha is equal to N. This option is implemented only for the binomial measure ('spec') and the mixing (lumping and sum) of two binomial measures ('lumpspec' and sumspec').

**ALGORITHM DETAILS :**

For the deterministic binomial, the pre-multifractal measure synthesis algorithm is implemented is a iterative way (supposed to run faster than a recursive one). For the shuffled or the pertubated binomial, the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). Note that the shuffled binomial measure is not conservative.

**EXAMPLES :**

```
p0=.2;
n=10;
// synthesizes a pre-multifractal binomial measure
[mu_n,I_n]=binom(p0,'meas',n);
plot(I_n,mu_n);
// synthesizes the cdf of a pre-multifractal shuffled binomial measure
F_n=binom(p0,'shufcdf',n);
plot(I_n,F_n);
e=.19;
// synthesizes the pdf of a pre-multifractal pertubated binomial measure
p_n=binom(p0,'pertpdf',n,e);
plot(I_n,p_n);
xbasc();
vn=[1:1:8];
q=[-5:.1:+5];
// computes the partition sum function of a binomial measure
znq=binom(p0,'part',vn,q);
mn=zeros(max(size(q)),max(size(vn)));
for i=1:max(size(q))
   mn(i,:)=-vn*log(2);
end
plot2d(mn',log(znq'));
// computes the Reyni exponents function of a binomial measure
tq=binom(p0,'Reyni',q);
plot(q,tq);
N=200;
q0=.4;
// computes the multifractal spectrum of the lumping of two binomial measures
[alpha,f_alpha]=binom(p0,'lumpspec',N,q0);
```

```
plot(alpha,f_alpha);
```

**REFERENCES :**

"Multifractal Measures", Carl J. G. Evertsz and Benoit B. MandelBrot. In Chaos and Fractals, New Frontiers of Science, Appendix B. Edited by Peitgen, Juergens and Saupe, Springer Verlag, 1992 pages 921-953.
"A class of Multinomial Multifractal Measures with negative (latent) values for the "Dimension" f(alpha)", Benoit B. MandelBrot. In Fractals' Physical Origins and Properties, Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum Press, New York, 1989 pages 3-29.

**SEE ALSO :**

sbinom, multim1d, multim2d, smultim1d, smultim2d (C_LAB routines). MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).

## 0.17 contwt Continuous L2 wavelet transform

Author: Paulo Goncalves

Computes a continuous wavelet transform of a 1-D signal (real or complex). The scale operator is unitary with respect to the L2 norm. Two closed form wavelets are available: the Mexican Hat or the Morlet Wavelet (real or analytic). For arbitrary analyzing wavelets, numerical approximation is achieved using a Fast Mellin Transform.

**USAGE :**

```
[wt,scale,f,scalo,wavescaled]=contwt(x,[fmin,fmax,N,wvlt_length])
```

**INPUT PARAMETERS :**

- x : Real or complex vector [1,nt] or [nt,1] Time samples of the signal to be analyzed.
- fmin : real scalar in [0,0.5] Lower frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.
- fmax : real scalar [0,0.5] and fmax $>$ Upper frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.
- N : positive integer. number of analyzing voices. When not specified, this parameter forces the program to interactive mode.
- wvlt_length : scalar or vector specifies the analyzing wavelet: 0: Mexican hat wavelet (real) Positive real integer: real Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Positive imaginary integer: analytic Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Real valued vector: waveform samples of an arbitrary bandpass function.

**OUTPUT PARAMETERS :**

- wt : Real or complex matrix [N,nt] coefficients of the wavelet transform.
- scale : real vector [1,N] analyzed scales
- f : real vector [1,N] analyzed frequencies
- scalo : real positive valued matrix [N,nt] Scalogram coefficients (squared magnitude of the wavelet coefficients wt )
- wavescaled : Scalar or real valued matrix [length(wavelet at coarser scale)+1,N]

  Dilated versions of the analyzing wavelet

**DESCRIPTION :**

**PARAMETERS :**

- x : signal to be analyzed. Real or complex vector
- fmin : lower frequency bound of the analysis. fmin is real scalar comprised in [0,0.5]
- fmax : upper frequency bound of the analysis. fmax is a real scalar comprised in [0,0.5] and fmax $>$ fmin
- N : number of analyzing voices geometrically sampled between minimum scale fmax/fmax and maximum scale fmax/fmin.
- wvlt_length : specifies the analyzing wavelet: 0: Mexican hat wavelet (real). The size of the wavelet is automatically fixed by the analyzing frequency Positive real integer: real Morlet wavelet of size 2*wvlt_length+1) at finest scale (1) Positive imaginary integer: analytic Morlet wavelet of size 2*|wvlt_length|+1) at finest scale 1. The corresponding wavelet transform is then complex. May be usefull for event detection purposes. Real valued vector: corresponds to the time samples waveform of any arbitrary bandpass function viewed as the analyzing wavelet at any given scale. Then, an approximation of the scaled wavelet versions is achieved using the Fast Mellin Transform (see dmt and dilate).
- wt : coefficients of the wavelet transform. X-coordinated corresponds to time (uniformly sampled), Y-coordinates correspond to frequency (or scale) voices (geometrically sampled between fmax (resp. 1) and fmin (resp. fmax / fmin ). First row of wt corresponds to the highest analyzed frequency (finest scale).
- scale : analyzed scales (geometrically sampled between 1 and fmax /fmin
- f : analyzed frequencies (geometrically sampled between fmax and fmin . f corresponds to fmax/scale
- scalo : Scalogram coefficients (squared magnitude of the wavelet coefficients wt )
- wavescaled : If wvlt_length is a real or Imaginary pure scalar, then wavescaled equal wvlt_length . If wvlt_length is a vector (containing the waveform samples of an arbitrary analyzing wavelet), then wavescaled contains columnwise all scaled version of wvlt_length used for the analysis. In this latter case, first element of each column gives the effective time support of the analyzing wavelet at the corresponding scale. wavescaled can be used for reconstructing the signal (see icontwt)

**ALGORITHM DETAILS :**

The wavelet transform of x is computed via convolutions of dilated and translated versions of a single function called the ”mother wavelet”. The scales are given by the dilatation factor. As the scales are not absolute, the scale factor is determined through the specification of the minimum and maximum frequency of the decomposition considered as a time/frequency transform. The maximum frequency might not be greater than the Nyquist Frequency i.e. 0.5 as the wavelet at this scale would be undersampled (and therefore would create aliasing). The number of scales tells how many convolutions are computed. The bigger it is, the slower the transform is. The frequency (or scale) axis is geometrically sampled. The resulting scales and frequencies values can be obtained as output parameters. The meaning of the wavelet length parameter is manyfold. When non zero integer, it tells the routine to use a real Morlet Wavelet and gives its length at scale 1 (maximum frequency). When it is a positive imaginary integer, the analytic Morlet wavelet is used. If wvlt_length = 0, the Mexican Hat is used. The resulting wavelet transform is then real but has a quite poor frequency resolution. If wvlt_length is a real vector, it corresponds to the analyzing wavelet waveform in time at any arbitrary scale. Dilated and compressed version of it (according to the range [ fmin , fmax] are computed directly from wvlt_length using a Fast Mellin Transform. For all choices of wavelet, approximative reconstruction of the decomposed signal is possible (see icontwt).

**SEE ALSO: :**

icontwt, contwtmir and cwt

**EXAMPLE: :**

```
//Signal synthesis
x = morlet(0.1,128) ;
//A Morlet (of size 2*8+1 samples ) wavelet transform
```

```
[wtMorlet,scale,f,scaloMorlet] = contwt(x,0.01,0.5,128,8) ;
viewmat(scalomor,1:257,1:128);
//Compared with a Mexican hat wavelet transform
[wtMex,scale,f,scaloMex] = contwt(x,0.01,0.5,128,0) ;
viewmat(scaloMex,1:257,1:128);
```

## 0.18 contwtmir _ Continuous L2 wavelet transform with mirroring

Author: Paulo Goncalves

Computes a continuous wavelet transform of a mirrored 1-D signal (real or complex). The scale operator is unitary with respect to the L2 norm. Two closed form wavelets are available: the Mexican Hat or the Morlet Wavelet (real or analytic). For arbitrary analyzing wavelets, numerical approximation is achieved using a Fast Mellin Transform.

**USAGE :**

```
[wt,scale,f,scalo,wavescaled]=contwtmir(x,[fmin,fmax,N,wvlt_length])
```

**INPUT PARAMETERS :**

- x : Real or complex vector [1,nt] or [nt,1] Time samples of the signal to be analyzed.
- fmin : real scalar in [0,0.5] Lower frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.
- fmax : real scalar [0,0.5] and fmax $>$ Upper frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.
- N : positive integer. number of analyzing voices. When not specified, this parameter forces the program to interactive mode.
- wvlt_length : scalar or vector specifies the analyzing wavelet: 0: Mexican hat wavelet (real) Positive real integer: real Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Positive imaginary integer: analytic Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Real valued vector: waveform samples of an arbitrary bandpass function.

**OUTPUT PARAMETERS :**

- wt : Real or complex matrix [N,nt] coefficient of the wavelet transform.
- scale : real vector [1,N] analyzed scales
- f : real vector [1,N] analyzed frequencies
- scalo : real positive valued matrix [N,nt] Scalogram coefficients (squared magnitude of the wavelet coefficients wt )
- wavescaled : Scalar or real valued matrix [length(wavelet at coarser scale)+1,N]

  Dilated versions of the analyzing wavelet

**DESCRIPTION :**

**PARAMETERS :**

- x : signal to be analyzed. Real or complex vector

- fmin : lower frequency bound of the analysis. fmin is real scalar comprised in [0,0.5]
- fmax : upper frequency bound of the analysis. fmax is a real scalar comprised in [0,0.5] and fmax > fmin
- N : number of analyzing voices geometrically sampled between minimum scale fmax/fmax and maximum scale fmax/fmin.
- wvlt_length : specifies the analyzing wavelet: 0: Mexican hat wavelet (real). The size of the wavelet is automatically fixed by the analyzing frequency Positive real integer: real Morlet wavelet of size 2*wvlt_length+1) at finest scale (1) Positive imaginary integer: analytic Morlet wavelet of size 2*|wvlt_length|+1) at finest scale 1. The corresponding wavelet transform is then complex. May be usefull for event detection purposes. Real valued vector: corresponds to the time samples waveform of any arbitrary bandpass function viewed as the analyzing wavelet at any given scale. Then, an approximation of the scaled wavelet versions is achieved using the Fast Mellin Transform (see dmt and dilate).
- wt : coefficient of the wavelet transform. X-coordinated corresponds to time (uniformly sampled), Y-coordinates correspond to frequency (or scale) voices (geometrically sampled between fmax (resp. 1) and fmin (resp. fmax / fmin ). First row of wt corresponds to the highest analyzed frequency (finest scale).
- scale : analyzed scales (geometrically sampled between 1 and fmax /fmin
- f : analyzed frequencies (geometrically sampled between fmax and fmin . f corresponds to fmax/scale
- scalo : Scalogram coefficients (squared magnitude of the wavelet coefficients wt )
- wavescaled : If wvlt_length is a real or Imaginary pure scalar, then wavescaled equal wvlt_length . If wvlt_length is a vector (containing the waveform samples of an arbitrary analyzing wavelet), then wavescaled contains columnwise all scaled version of wvlt_length used for the analysis. In this latter case, first element of each column gives the effective time support of the analyzing wavelet at the corresponding scale. wavescaled can be used for reconstructing the signal (see icontwt)

**ALGORITHM DETAILS :**

The overall details of the algorithm are similar to those of contwt . The difference stems from the mirror operation applied to the signal before computing the wavelet transform to minimize border effects. At each scale j the analyzed signal is mirrored at its both extremities. The number of added samples at both sides is equal to scale(j)* wvlt_length (the half length of the analyzing wavelet at this particular scale). After convolution of the mirrored signal with the analyzing wavelet, the result is truncated to the actual size of the initial signal.

**SEE ALSO: :**

contwt, icontwt, cwt

**EXAMPLE: :**

```
//Signal synthesis
x = fbmlevinson(1024,0.8) ;
//Regular Wavelet transform
[wt_nomirror,scale,f] = contwt(x,2^(-6),2^(-1),128,8) ;
viewmat(abs(wt_nomirror),[1 1 24]) ;
//Compared with a mirrored  wavelet transform
[wt_mirror,scale,f] = contwtmir(x,2^(-6),2^(-1),128,0) ;
viewmat(abs(wt_mirror),[1 1 24]) ;
```

## 0.19 contwtspec _ Continuous L2 wavelet based Legendre spectrum

Author: Paulo Goncalves

Estimates the multifractal Legendre spectrum of a 1-D signal from the wavelet coefficients of a L2 continuous decomposition

**USAGE :**

```
[alpha,f_alpha,logpart,tau] = contwtspec(wt,scale,Q[,FindMax,ChooseReg])
```

**INPUT PARAMETERS :**

- wt : Real or complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of contwt or contwtmir))
- scale : real vector [1,N_scale] Analyzed scale vector
- Q : real vector [1,N_Q] Exponents of the partition function
- FindMax : 0/1 flag. FindMax = 0 : estimates the Legendre spectrum from all coefficients FindMax = 1 : estimates the Legendre spectrum from the local Maxima coefficients of the wavelet transform Default value is FindMax = 1
- ChooseReg : 0/1 flag or integer vector [1,N_reg], (N_reg <= N_scale) ChooseReg = 0 : full scale range regression ChooseReg = 1 : asks online the scale indices setting the range for the linear regression of the partition function. ChooseReg = [n1 ... nN_reg] : scale indices for the linear regression of the partition function.

**OUTPUT PARAMETERS :**

- alpha : Real vector [1,N_alpha], N_alpha <= N_Q Singularity support of the multifractal Legendre spectrum
- f_alpha : real vector [1,N_alpha] Multifractal Legendre spectrum
- logpart : real matrix [N_scale,N_Q] Log-partition function
- tau : real vector [1,N_Q] Regression function

**SEE ALSO: :**
contwt, cwtspec, cwt, dwtspec, FWT

**EXAMPLE: :**

```
N = 2048 ; H = 0.7 ; Q = linspace(-4,4,11) ;
[x] = fbmlevinson(N,H) ;
[wt,scale] = contwtmir(x,2^(-8),2^(-1),16,8) ;
[alpha,f_alpha,logpart,tau] = contwtspec(wt,scale,Q,1,1) ;
plot(alpha,f_alpha),
```

## 0.20 cwt Continuous Wavelet Transform

Author: Bertrand Guiheneuf

This routine computes the continuous wavelet transform of a real signal. Two wavelets are available: the Mexican Hat or the Morlet Wavelet.

**USAGE :**

```
[wt,scales,freqs]=cwt(sig,fmin,fmax,nbscales,[wvlt_length])
```

**INPUT PARAMETERS :**

- sig : real vector [1,n] or [n,1] Contains the signal to be decomposed.
- fmin : real positive scalar Lowest frequency of the wavelet analysis
- fmax : real positive scalar Highest frequency of the wavelet analysis
- nbscales : integer positive scalar Number of scales to compute between the lowest and the highest frequencies.
- wvlt_length : real positive scalar (optionnal) If equal to 0 or not specified, the wavelet is the Mexican Hat and its length is automaticaly choosen. Otherwise, Morlet's wavelet is used and it's length at scale 1 is given by wvlt_length

**OUTPUT PARAMETERS :**

- wt : complex matrix [nbscales,n] Wavelet transform. The first line is the finer scale ( scale 1 ). It is real if the Mexican Hat has been used, complex otherwise.
- scales : real vector [1,nbscales] Scale corresponding to each line of the wavelet transform.
- freqs : real vector [1,nbscales] Frequency corresponding to each line of the wavelet transform.

**DESCRIPTION :**

**PARAMETERS :**
The wavelet transform of sig is computed via convolutions of dilated and translated versions of a single function called the "wavelet". The scales are given by the dilatation factor. As the scales are not absolute, the scale factor is determined through the specification of the minimum and maximum frequency of the decomposition considered as a time/frequency transform. The maximum frequency might not be greater than the Nyquist Frequency i.e. 0.5 as the wavelet at this scale would be undersampled. The number of scales tells how many convolutions are computed. The bigger it is, the slower the transform is. The frequency (or scale) axis is logarithmicaly sampled. The resulting scales and frequencies values can be obtained as output parameters. The meaning of the wavelet length parameter is twofold. If non zero, it tells the routine to use a Morlet Wavelet and gives its length at scale 1 (maximum frequency). Otherwise (zero or not specified), the Mexican Hat is used. The resulting wavelet transform is then real but has a quite poor frequency resolution.

**ALGORITHM DETAILS :**
Convolutions are computed through discrete linear convolutions in time domain. No FFT is used. The linear filters are obtained by a sampling of the wavelet after dilatation. The signal is mirrored at its boundaries.

## 0.21 cwtspec — Continuous L1 wavelet based Legendre spectrum

Author: Paulo Goncalves

Estimates the multifractal Legendre spectrum of a 1-D signal from the wavelet coefficients of a L1 continuous decomposition

**USAGE :**

```
[alpha,f_alpha,logpart] = cwtspec(wt,scale,Q[,FindMax,ChooseReg])
```

**INPUT PARAMETERS :**

- wt : Real or complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of cwt)
- scale : real vector [1,N_scale] Analyzed scale vector
- Q : real vector [1,N_Q] Exponents of the partition function
- FindMax : 0/1 flag. FindMax = 0 : estimates the Legendre spectrum from all coefficients FindMax = 1 : estimates the Legendre spectrum from the local Maxima coefficients of the wavelet transform Default value is FindMax = 1
- ChooseReg : 0/1 flag or integer vector [1,N_reg], (N_reg <= N_scale) ChooseReg = 0 : full scale range regression ChooseReg = 1 : asks online the scale indices setting the range for the linear regression of the partition function. ChooseReg = [n1 ... nN_reg] : scale indices for the linear regression of the partition function.

**OUTPUT PARAMETERS :**

- alpha : Real vector [1,N_alpha], N_alpha <= N_Q Singularity support of the multifractal Legendre spectrum
- f_alpha : real vector [1,N_alpha] Multifractal Legendre spectrum
- logpart : real matrix [N_scale,N_Q] Log-partition function
- tau : real vector [1,N_Q] Regression function

**SEE ALSO: :**
cwt, contwtspec, contwt, dwtspec

**EXAMPLE: :**

```
N = 2048 ; H = 0.7 ; Q = linspace(-4,4,11) ;
[x] = fbmlevinson(N,H) ;
[wt,scale] = cwt(x,2^(-8),2^(-1),16,8) ;
[alpha,f_alpha,logpart,tau] = cwtspec(wt,scale,Q,1,1) ;
plot(alpha,f_alpha),
```

## 0.22 cwttrack — Continuous L2 wavelet based Holder exponent estimation

Author: Paulo Goncalves

Estimates the local or global Holder exponent of a 1-D signal from its L2 continuous wavelet transform ( output of contwt(mir) ). In some cases, the global Holder exponent can also be refered to as the long range dependance parameter

**USAGE :**

```
[HofT] = cwttrack(wt,scale,whichT,FindMax,ChooseReg,radius,DeepScale,Show)
```

**INPUT PARAMETERS :**

- wt : Real or complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of contwt)
- scale : real vector [1,N_scale] Analyzed scale vector

- whichT : Integer whichT, when non zero specifies the time position on the signal where to estimate the local Holder exponent. When whichT is zero, the global scaling exponent (or LRD exponent) is estimated.
- FindMax : 0/1 flag. FindMax = 0 : estimates the Holder exponents (local or global) from all coefficients of the wavelet transform FindMax = 1 : estimates the Holder exponents (local or global) from the local Maxima coefficients of the wavelet transform Default value is FindMax = 1
- ChooseReg : 0/1 flag or integer vector [1,N_reg], (N_reg <= N_scale) ChooseReg = 0 : full scale range regression ChooseReg = 1 : scale range is choosed by the user, clicking with the mouse on a regression graph. ChooseReg = [n1 ... nN_reg] : imposes the scale indices for the linear regression of the wavelet coefficients versus scale in a log-log plot Default value is ChooseReg = 0
- radius : Positive integer. The local maxima line search is restricted to some neighbourhood of the analyzed point. Basically, this region is defined by the cone of influence of the wavelet. radius allows to modulate the width of the cone. Default value is cone = 8 .
- DeepScale : strictly positive integer. DeepScale tells the maxima line procedure how depth in scale to scan from step to step. Default value is DeepScale = 1
- Show 0/1 flag. Show = 1 : display the maxima line trajectory and the log-log regression graph Show = 0 : no display

**OUTPUT PARAMETERS :**

- HofT : Real scalar. Local or global Holder exponent estimated

**ALGORITHM DETAILS :**
The maxima line search follows the two steps:

- all local maxima are found using a standard gradient technique
- local maxima are connected along scales by finding the minimum Lobatchevsky distance between two consecutive maxima lying beneath the cone of influence.

**SEE ALSO: :**
cwttrack_all, contwtspec, contwt, dwtspec

**EXAMPLE: :**

```
N = 1024 ;
[x] = GeneWei(N,[ones(1,N/2)*0.2 ones(1,N/2)*0.8],2,1,1) ;
[wt,scale] = contwtmir(x,2^(-8),2^(-1),64,8*i) ;
HofT_1 = cwttrack(wt,scale,N/4,1,1)
HofT_2 = cwttrack(wt,scale,3*N/4,1,1)
```

## 0.23 cwttrack_all — Continuous L2 wavelet based Holder function estimation

Author: Paulo Goncalves

Estimates the Holder function of a signal from its continuous wavelet transform (L2 contwt). cwttrack_all merely runs cwttrack as many times as there are time samples to be analyzed

**USAGE :**

```
[HofT,whichT] = cwttrack_all(wt,scale,FindMax,ChooseReg,radius,DeepScale,dT)
```

**INPUT PARAMETERS :**

- wt : Real or complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of contwt)
- scale : real vector [1,N_scale] Analyzed scale vector
- whichT : Integer whichT, when non zero specifies the time position on the signal where to estimate the local Holder exponent. When whichT is zero, the global scaling exponent (or LRD exponent) is estimated.
- FindMax : 0/1 flag. FindMax = 0 : estimates the Holder exponents (local or global) from all coefficients of the wavelet transform FindMax = 1 : estimates the Holder exponents (local or global) from the local Maxima coefficients of the wavelet transform Default value is FindMax = 1
- ChooseReg : 0/1 flag or integer vector [1,N_reg], (N_reg <= N_scale) ChooseReg = 0 : full scale range regression ChooseReg = 1 : scale range is choosed by the user, clicking with the mouse on a regression graph. ChooseReg = [n1 ... nN_reg] : imposes the scale indices for the linear regression of the wavelet coefficients versus scale in a log-log plot Default value is ChooseReg = 0
- radius : Positive integer. The local maxima line search is restricted to some neighbourhood of the analyzed point. Basically, this region is defined by the cone of influence of the wavelet. radius allows to modulate the width of the cone. Default value is cone = 8 .
- DeepScale : strictly positive integer. DeepScale tells the maxima line procedure how depth in scale to scan from step to step. Default value is DeepScale = 1
- dT 01 Integer. Sampling period for the Holder function estimate

**OUTPUT PARAMETERS :**

- HofT : Real scalar. Local or global Holder exponent estimated
- whichT Integer vector Time sampling vector

**SEE ALSO: :**
cwttrack

**EXAMPLE: :**

```
N = 2048 ;
[x] = GeneWei(N,linspace(0,1,N),1.2,1,1) ;
[wt,scale] = contwtmir(x,2^(-6),2^(-1),64,8*i) ;
[HofT,whichT] = cwttrack_all(wt,scale,1,0,8,1,(N/64)) ;
```

## 0.24 dilate — Dilation of a signal

Author: Paulo Goncalves

Computes dilated/compressed version of a signal using Fast Mellin transform.

**USAGE :**

```
[sscaled,mellin,beta] = dilate(s,a,[fmin,fmax,N])
```

**INPUT PARAMETERS :**

- s : real vector [1,nt] or [nt,1] Time samples of the signal to be scaled.
- a : real strictly positive vector [1,N_scale] Dilation/compression factors. a $<$ 1 corresponds to compression in time
- fmin : real scalar in [0,0.5] Lower frequency bound of the signal (necessary for the intermediate computation of the Mellin transform)
- fmax : real scalar [0,0.5] and fmax $>$ Upper frequency bound of the signal (necessary for the intermediate computation of the Mellin transform)
- N : positive integer. number of Mellin samples.

**OUTPUT PARAMETERS :**

- sscaled : Real matrix with N_scale columns Each column j (for j = 1 .. N_scale) contains the dilated/compressed version of s by scale a(j). First element of each column gives the effective time support for each scaled version of s.
- mellin : complex vector [1,N] Mellin transform of s.
- beta : real vector [1,N] Variable of the Mellin transform mellin.

**DESCRIPTION :**

**PARAMETERS :**

- s : signal to be analyzed. Real or complex vector. Size of s should be odd. If even, a zero sample is appended at the end of the signal
- a scale factor. Maximum allowed scale is determined by the spectral extent of the signal to be compressed: the spectral extent of the compressed signal can not go beyond the Nyquist frequency (1/2). There is no theoretical limit for the minimum allowed scale, other than the computational cost.
- fmin : lower frequency bound of the analysis. fmin is real scalar comprised in [0,0.5]
- fmax : upper frequency bound of the analysis. fmax is a real scalar comprised in [0,0.5] and fmax $>$ fmin
- N : number of Mellin samples. This number must be greater than some ammount determined by the spectral extent of the signal, to avoid aliasing in the Mellin domain.

**ALGORITHM DETAILS :**
This algorith uses a Fast Mellin Transform (dmt) to diagonalize the Scale operator. The algorithm runs as follows

- compute the Fourier-Mellin transform of the signal
- Multiply the result by a(-i.beta) (beta is the Mellin variable), for each values of scale a
- compute the inverse Fourier-Mellin transform to get the a-dilated version of s

**SEE ALSO: :**
dmt, idmt

**EXAMPLE: :**

```
// Signal synthesis

x = morlet(0.1,32) ;
plot(x)
//Dilation by a factor 2
[sscaled,mellin,beta] = dilate(x,2,0.01,0.5,256) ;
plot(sscaled(2:sscaled(1)))
//Compression by a factor 2
[sscaled,mellin,beta] = dilate(x,1/2,0.01,0.5,256) ;
plot(sscaled(2:sscaled(1)))
```

# 0.25 dimR2d — Regularization dimension of the surface of a 2d function

Author: Francois Roueff

Computes the regularization dimension of the surface of a 2d function. Two kernels are available: the Gaussian or the Rectangle.

**USAGE :**

```
[dim,handlefig]=dimR(x,sigma,voices,Nmin,Nmax,kernel,mirror,reg,graphs)
```

**INPUT PARAMETERS :**

- x : Real or complex matrix [nt,pt] Space samples of the signal to be analyzed.
- sigma : Real positive number Standard Deviation of the noise. Its default value is null (noisefree)
- voices : Positive integer. number of analyzing voices. When not specified, this parameter is set to 128.
- Nmin : Integer in [2,nt/3] Lower scale bound (lower width) of the analysing kernel. When not specified, this parameter is set to around nt/12.
- Nmax : Integer in [Nmin,2nt/3] Upper scale bound (upper width) of the analysing kernel. When not specified, this parameter is set to nt/3.
- kernel : String specifies the analyzing kernel: ”gauss”: Gaussian kernel (default) ”rect”: Rectangle kernel
- mirror : Boolean

  specifies wether the signal is to be mirrorized for the analyse (default: 0).
- reg : Boolean

  specifies wether the regression is to be done by the user or automatically (default: 0).
- graphs : Boolean:

  specifies wether the regularized graphs have to be displayed (default: 0).

**OUTPUT PARAMETERS :**

- dim : Real Estimated regularization dimension.
- handlefig : Integer vector Handles of the figures opened during the procedure.

**DESCRIPTION :**
This function is the same as dimR but adapted to 2d signals. For a more complete explanation of the regularization dimension, one can refer to: ”A regularization approach to fractionnal dimension estimation”, F. Roueff, J. Levy-Vehel, submitted to Fractal 98 conference. The regularized graphs of x are computed via convolutions of x with dilated versions of the kernel at different scales. The lengths of the regularized graphs are computed via convolutions of x with the derivatives of the dilated versions of the kernel. The regularization dimension is computed either via an automatic range regression or via a regression by hand on the loglog plot of the lengths versus scales. If sigma is strictly positive, an estimation of the lengths without noise is used for the regression. These lengths are displayed in red while those of the noisy signal are in black. They should seperate at fine scales. When one specifies the range regression, the loglog plot of the lengths versus scales appears. Above are either increments (when sigma is null) or a loglog plot of the noise prevalence in the lengths. One selects the scale range of the regression. In the case of noisefree signals, select a scale region with stable increments. In the case of a strictly positive sigma, select a scale region where the noise prevalence is not too close to 1 (0 in log10): it should correspond to an approximately linear

region for the red estimations. The number of scales (voices) tells how many convolutions are computed. The bigger it is, the slower the computation is. The scale axis is geometrically sampled (i.e. its log is arithmetically sampled). The gaussian kernel should give a better result but the rectangle is faster. As a general rule, be careful of the size of the input signal and of the maximal size of the kernel (Nmax x Nmax) to avoid too long computing times.

**SEE ALSO: :**

cwttrack, cwtspec.

**EXAMPLE: :**

```
//Signal synthesis
x = GeneWei(100,0.6,2,1.0,0);
y = GeneWei(100,0.4,3,1.0,1);
w = x'*y;
plot3d((1:100)/100,(1:100)/100,w)
//Dimension of the graph with a regression by hand
dim = dimR2d(w,0,25,5,30,'gauss',0,1,0);
```

## 0.26 dmt — Discrete Mellin transform of a vector

Author: Paulo Goncalves

Computes the Fast Mellin transform of a signal.

**USAGE :**

```
[mellin,beta] = dmt(s,[fmin,fmax,N])
```

**INPUT PARAMETERS :**

- s : real vector [1,nt] or [nt,1] Time samples of the signal to be transformed.
- fmin : real scalar in [0,0.5] Lower frequency bound of the signal
- fmax : real scalar [0,0.5] and fmax > Upper frequency bound of the signal
- N : positive integer. number of Mellin samples.

**OUTPUT PARAMETERS :**

- mellin : complex vector [1,N] Mellin transform of s.
- beta : real vector [1,N] Variable of the Mellin transform mellin.

**DESCRIPTION :**

**PARAMETERS :**

- s : signal to be transformed. Real or complex vector.
- fmin : lower frequency bound of the analysis. fmin is real scalar comprised in [0,0.5]
- fmax : upper frequency bound of the analysis. fmax is a real scalar comprised in [0,0.5] and fmax > fmin
- N : number of Mellin samples. This number must be greater than some ammount determined by the spectral extent of the signal, to avoid aliasing in the Mellin domain.

**ALGORITHM DETAILS :**
The fast Mellin transform can be simply interpreted as a FFT applied to a geometrically sampled vector.

**SEE ALSO: :**
idmt, dilate

**EXAMPLE: :**

```
//Signal synthesis
x = morlet(0.1,32) ;
plot(x)
//Computation of the Mellin transform
[mellin,beta] = dmt(x,0.01,0.5,128) ;
plot(beta,abs(mellin))
```

## 0.27 dwtspec ——— Discrete wavelet based Legendre spectrum

Author: Paulo Goncalves

Estimates the multifractal Legendre spectrum of a 1-D signal from the wavelet coefficients of a discrete decomposition

**USAGE :**

```
[alpha,f_alpha,logpart] = dwtspec(wt,Q[,ChooseReg])
```

**INPUT PARAMETERS :**

- wt : Real vector [1,N] Wavelet coefficients of a discrete wavelet transform (output of FWT)
- Q : real vector [1,N_Q] Exponents of the partition function
- ChooseReg : 0/1 flag or integer vector [1,N_reg], (N_reg <= N_scale) ChooseReg = 0 : full scale range regression ChooseReg = 1 : asks online the scale indices setting the range for the linear regression of the partition function. ChooseReg = [n1 ... nN_reg] : scale indices for the linear regression of the partition function.

**OUTPUT PARAMETERS :**

- alpha : Real vector [1,N_alpha], N_alpha <= N_Q Singularity support of the multifractal Legendre spectrum
- f_alpha : real vector [1,N_alpha] Multifractal Legendre spectrum
- logpart : real matrix [N_scale,N_Q] Log-partition function
- tau : real vector [1,N_Q] Regression function

**SEE ALSO: :**
cwtspec, FWT, WTStruct, MakeQMF, flt, iflt

**EXAMPLE: :**

```
N = 2048 ; H = 0.3 ; Q = linspace(-4,4,11) ;
[x] = fbmlevinson(N,H) ;
qmf = MakeQMF('daubechies',2) ;
[wt] = FWT(x,log2(N),qmf) ;
[alpha,f_alpha,logpart,tau] = dwtspec(wt,Q,1) ;
plot(alpha,f_alpha),
```

## 0.28 fbmfwt — Discrete wavelet based synthesis of a fBm

Author: Paulo Goncalves

Generates a 1/f Gaussian process from a discrete wavelet transform

**USAGE :**

```
[x] = fbmfwt(N,H,[noctave,Q,randseed]) ;
```

**INPUT PARAMETERS :**

- N : Positive integer Sample size of the fBm
- H : Real in [0,1] Holder exponent
- noctave : integer Maximum resolution level (should not exceeed log2(N))
- Q : real vector. Analyzing QMF (e.g. Q = MakeQMF('daubechies',4) )
- randseed : real scalar Random seed generator

**OUTPUT PARAMETERS :**

- x : real vector [1,N] Time samples of the 1/f Gaussian process

**ALGORITHM DETAILS :**
Generates a 1/f Gaussian process by inverting a discrete wavelet transform. Step 1: generates y a [1,N] i.i.d. standard Gaussian noise Step 2: computes the discrete wavelet coefficients y Step 3: weigth the wavelet coefficients y with the corresponding scale power law Step 4: invert the weighted discrete wavelet transform

**SEE ALSO: :**
fbmlevinson, synth2, FWT, MakeQMF

**EXAMPLE: :**

```
Q = MakeQMF('daubechies',4) ;
[x] = fbmfwt(1024,0.5,10,Q) ;
[wt,scale,f] = contwt(x,2^(-8),2^(-1),64,8) ;
[H] = cwttrack(wt,scale,0,1,1,8,1,1) ;
```

## 0.29 fbmlevinson — Levinson synthesis of a fractional Brownian motion

Author: Paulo Goncalves

Generates a Fractional Brownian Motion (fBm) using Cholesky/Levinson factorization

**USAGE :**

```
[x,y,r] = fbmlevinson(N,H,[seed])
```

**INPUT PARAMETERS :**

- N : Positive integer Sample size of the fBm
- H : Real in [0,1] Holder exponent
- seed : real scalar Random seed generator

**OUTPUT PARAMETERS :**

- x : real vector [1,N] Time samples of the fBm
- y : real vector [1,N] Vector of N i.i.d. white standard Gaussian r.v.'s (input process of the generator)
- r : real vector [1,N] First row of the var/cov Toeplitz matrix R of the increment process w[k] = x[k+1] - x[k].

**ALGORITHM DETAILS :**
Generates a Fractional Brownian Motion using Levinson algorithm to triangularize the covariance matrix. R = E W * W' being the variance/covariance matrix of the fBm increment W[n ; shift] = X[n+shift] - X[n-shift], R = L*L', with L the lower left triangle matrix (Choleski or Levinson factorization). Then, we pose Z = L(-1) * W <=> W = L * Z with Rz the var/cov matrix of the process Z, Rz = E { Z * Z' } Rz = E { L(-1) * W * W' * (L(-1))' } Rz = L(-1) * R * (L(-1))' Rz = L(-1) * L * L' * (L(-1))' Rz = I (identity) Thus, Z is a white Gaussian noise with unit variance.
**SEE ALSO: :**
mbmlevinson
**EXAMPLE: :**

```
[x,y,r] = fbmlevinson(1024,0.8) ;
```

## 0.30 fft1d _____ Operates a column-wise direct or inverse FFT

Author: Paulo Goncalves

Operates a column-wise direct or inverse FFT on a matrix

**USAGE :**

```
Y = fft1d(X,DirInv) ;
```

**INPUT PARAMETERS :**

- X : Real or complex valued matrix [rx,cx]
- DirInv : +1 / -1 flag -1 Direct Fast Fourier Transform +1 Inverse Fast Fourier Transform

**OUTPUT PARAMETERS :**

- Y : Real or complex valued matrix [rx,cx] Each column of Y contains the FFT (resp IFFT) of the corresponding column of X

**SEE ALSO: :**
fft
**EXAMPLE: :**

```
//Matrix synthesis:
t = linspace( 0,1,128 ) ;
f0 = [4 8 16 32]
X = sin( 2*%pi*t(:)*f0 ) ;
Y = abs( fft1d( X , -1 ) ) ;
Y = [Y(65:128,:) ; Y(1:64,:)] ;
f = linspace(-64,63,128) ;
plot2d(f(ones(4,1),:)',Y) ;
```

## 0.31 findWTLM — Finds local maxima lines of a CWT

Author: Paulo Goncalves

Finds the local maxima of a continuous wavelet transform

**USAGE :**

```
[maxmap] = findWTLM(wt,scale[,depth])
```

**INPUT PARAMETERS :**

- wt : Complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of FWT or contwt)
- scale : real vector [1,N_scale] Analyzed scale vector
- depth : real in [0,1] maximum relative depth for the peaks search. Default value is 1 (all peaks found)

**OUTPUT PARAMETERS :**

- maxmap : 0/1 matrix [N_scale,N] If maxmap(m,n) = 0 : the coefficient wt(m,n) is not a local maximum If maxmap(m,n) = 1 : the coefficient wt(m,n) is a local maximum

**SEE ALSO: :**
contwt, cwt

**EXAMPLE: :**

```
N = 2048 ; H = 0.3 ; Q = linspace(-4,4,11) ;
[x] = fbmlevinson(N,H) ;
[wt,scale] = cwt(x,2^(-6),2^(-1),36,0) ;
[maxmap] = findWTLM(wt,scale) ;
```

## 0.32 flt — Fast Legendre transform

**USAGE :**

```
[u,s] = flt(x,y[,ccv])
```

**INPUT PARAMETERS :**

- x : real valued vector [1,N] samples support of the function y
- y : real valued vector [1,N] samples of function y = y(x)
- ccv : optional argument to choose between convex (ccv = 0) and concave (ccv = 1) envelope. Default value is ccv = 1 (concave)

**OUTPUT PARAMETERS :**

- u : real valued vector [1,M] Legendre transform of input y. Note that, since u stems from the envelope of y, in general M <= N.
- s : real valued vector [1,M] Variable of the Legendre transform of y.

**DESCRIPTION :**
Computes the Legendre transform of y y*(s) = sup_{x in X}[s.x - y(x)]

**EXAMPLE: :**

```
//Function synthesis
m0 = .55 ; m1 = 1 - m0 ;
m2 = .95 ; m3 = 1 - m2 ;
q = linspace(-20,20,201) ;
tau1 = - log2(exp(q.*log(m0)) + exp(q.*log(m1))) ;
tau2 = - log2(exp(q.*log(m2)) + exp(q.*log(m3))) ;
tau3 = min(tau1 , tau2) ;

//Legendre Transforms
[u1,s1] = flt(q,tau1) ;
[u2,s2] = flt(q,tau2) ;
[u3,s3] = flt(q,tau3) ;

// Vizualisation
plot2d(s3,u3,17) ;
plot2d(s1,u1,18,'001') ;
plot2d(s2,u2,19,'001') ;
```

AUTHOR : Paulo Goncalves

## 0.33 gauss Gaussian window

Author: Paulo Goncalves

Returns a Gaussian window

**USAGE :**

```
Win = gauss(N[,A])
```

**INPUT PARAMETERS :**

- N : Positive integer Number of points defining the time support of the window
- A : Real positive scalar Attenuation in dB at the end of the window (10(-A)). Default value is A = 2.

**OUTPUT PARAMETERS :**

- Win : real vector [1,N] Gaussian window in time.

**SEE ALSO: :**
mexhat, morlet

**EXAMPLE: :**

```
t = linspace(-1,1,128) ;
Win1 = gauss(128,2) ;
Win2 = gauss(128,5) ;

plot2d([t(:) t(:)],[Win1(:) Win2(:)],[17 19])
```

## 0.34 gifs2wave ___ wavelet coefficients from new GIFS coefficients

Author: Khalid Daoudi

Computes the wavelet c0efficients of the synthetic 1-D real signal from its new GIFS coefficients.

**USAGE :**

```
[wt_new]=gifs2wave(Ci_new,wt,wt_idx,wt_lg)
```

**INPUT PARAMETERS :**

- Ci_new : Real matrix Contains the new GIFS coefficients
- wt : Real matrix contains the wavelet coefficients (obtained using FWT)
- wt_idx : Real matrix [1,n] contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces
- wt_lg : Real matrix [1,n] contains the dimension of each projection

**OUTPUT PARAMETERS :**

- wi_new : Real matrix Contains the new wavelet coefficients plus other informations.

**DESCRIPTION :**

**PARAMETERS :**

**ALGORITHM DETAILS :**

**SEE ALSO: :**
wave2gifs

**EXAMPLE: :**

## 0.35 gifseg — Replaces nodes of the diadic tree by a ceratin unique value.

Author: Khalid Daoudi

Replaces at each scale the left (resp. right) nodes of the diadic tree, associated to the GIFS coefficients, that belong to [cmin,cmax] by a ceratin unique value.

**USAGE :**

```
[Ci_new, marks, L]=gifseg(Ci,[cmin,cmax,epsilon])
```

**INPUT PARAMETERS :**

- Ci : Real matrix Contains the GIFS coefficients (obtained using FWT)
- cmin : Real scalar [1,n] Specifies the minimal value of the Ci's to be considered (cmin=0 by default)
- cmax : Real scalar [1,n] Specifies the maximal value of the Ci's to be considered (cmin=0 by default)
- epsilon : real scalar Specifies the maximal error desied on the Ci's approximation.

**OUTPUT PARAMETERS :**

- Ci_new : Real matrix Contains the the new GIFS coefficients.
- marks : Real vector Contains the segmentation marques. length(marks)-1 is the number of segmented parts.
- L : Real matrix A structure containing the left and right lambda_i's corresponding to each segmented part.

**DESCRIPTION :**

**PARAMETERS :**

**ALGORITHM DETAILS :**

**SEE ALSO: :**
hist, wave2gifs

**EXAMPLE: :**

## 0.36 holder2d — holder exponents of a measures defined on 2D real signal

**USAGE :**

```
[holder]=holder2d(Input,[Meas],[Res],[Ref],[RefMeas])
```

**INPUT PARAMETERS :**

- Input : real matrix [m,n] Contains the signal to be analysed.
- Meas : string Analysing measure. Must choosen be in {”sum”, ”var”, ”ecart”, ”min”, ”max”, ”iso”, ”riso”, ”asym”, ”aplat”, ”contrast”, ”lognorm”, ”varlog”, ”rho”, ”pow”, ”logpow”, ”frontmax”, ”frontmin”, ”diffh”, ”diffv”, ”diffmin”, ”diffmax”} (default : ”sum”)
- res : Number of resolutions used for the computation. (default : 1)
- Ref : real matrix [m,n] Contains the reference signal i.e. the signal on which the reference measure will be computed. Input and Ref must have the same dimensions.
- RefMeas : string Reference measure. (default : ”sum”)

**OUTPUT PARAMETERS :**

- holder : real matrix [m,n] Contains the Holder exponents.

**DESCRIPTION :**

**INTRODUCTION :**
This routines computes holder exponents by regressing an analysing measure (in a log-log plot) at different scales. Given a pixel, one defines a (square) window around it. The window size is called the resolution. The specified measure (or capacity) is then evaluated on the set defined by the window. For example, in the case of the ”sum” measure, at resolution 2, a 5x5 square center on a pixel p0 is extracted from the input image. The mean of the gray levels of the obtained pixels defines the measure at pixel p0 and resolution 2. The measure type is thus given by the input parameter Meas whereas the actual measure is obtained through the input signal. In the case of a simple measure analysis, the regression is computed with respect to the size of the window, this corresponds to comparing the analysing measure to the Lebesgue measure. Nevertheless, it is possible to compute the regression by comparison with a reference measure given by the last two parameters.

AUTHOR : Author: Pascal Mignot - Bertrand Guiheneuf

## 0.37 icontwt Inverse Continuous L2 wavelet transform

Author: Paulo Goncalves

Computes the inverse continuous wavelet transform: reconstructs a 1-D signal from its wavelet coefficients. The scale operator is unitary with respect to the L2 norm.

**USAGE :**

```
[x_back]=icontwt(wt,f,wl_length)
```

**INPUT PARAMETERS :**

- wt : Real or complex matrix [N,nt] coefficient of the wavelet transform
- f : real vector of size [N,1] or [1,N] which elements are in /[0,0.5], in decreasing order.
- wl_length : scalar or matrix specifies the reconstruction wavelet: 0: Mexican hat wavelet (real) Positive real integer: real Morlet wavelet of size 2*wl_length+1) at finest scale 1 Positive imaginary integer: analytic Morlet wavelet of size 2*wl_length+1) at finest scale 1 Real valued matrix with N columns: each column contains a dilated versions of an arbitrary synthesis wavelet.

**OUTPUT PARAMETERS :**

- x_back : Real or complex vector [1,nt] Reconstructed signal.

**DESCRIPTION :**

**PARAMETERS :**

- wt : coefficient of the wavelet transform. X-coordinated corresponds to time (uniformly sampled), Y-coordinates correspond to frequency (or scale) voices (geometrically sampled between fmax (resp. 1) and fmin (resp. fmax / fmin ). First row of wt corresponds to the highest analyzed frequency (finest scale). Usually, wt is the output matrix wt of contwt .
- scale : analyzed scales (geometrically sampled between 1 and fmax /fmin. Usually, scale is the output vector scale of contwt .
- wl_length : specifies the synthesis wavelet: 0: Mexican hat wavelet (real). The size of the wavelet is automatically fixed by the analyzing frequency Positive real integer: real Morlet wavelet of size 2*wl_length+1) at finest scale (1) Positive imaginary integer: analytic Morlet wavelet of size 2*|wl_length|+1) at finest scale 1. The corresponding wavelet transform is then complex. May be usefull for event detection purposes. Real valued matrix: usually, for reconstruction wl_length is the output matrix wavescaled from contwt.

**ALGORITHM DETAILS :**
The reconstruction algorithm Inverse Wavelet Transform , proceeds by convolving the wavelet coefficients (obtained from contwt ) by the synthesis wavelet. As we deal with continuous wavelet decomposition, the analyzing wavelet and its dual for reconstruction are the same (continuous basis). This operation is iterated at each analyzed scale j yielding N corresponding band-passed signal versions. The reconstructed signal is the scale weighting sum of these N vectors.

**SEE ALSO: :**
contwt, contwtmir

**EXAMPLE: :**

```
//Signal synthesis
x = morlet(0.1,64) ;
t = 1:129 ;
//A Morlet (of size 2*8+1 samples ) wavelet transform
[wtMorlet,scale,f,scaloMorlet] = contwt(x,0.01,0.5,128,8) ;
viewmat(scaloMorlet,1:129,f,[1 1 24]) ;
//Reconstruction with the same synthesis wavelet
[x_back] = icontwt(wtMorlet,f,8) ;
plot([t(:) t(:)],[x(:) x_back(:)]) ;
```

## 0.38 idmt — Inverse Discrete Mellin transform

Author: Paulo Goncalves

Computes the Inverse Fast Fourier-Mellin transform of a signal.

**USAGE :**

```
[x,t] = idmt(mellin,beta,[M])
```

**INPUT PARAMETERS :**

- mellin : complex vector [1,N] Fourier-Mellin transform to be inverted. For a correct inversion of the Fourier-Mellin transform, the direct Fourier-Mellin transform mellin must have been computed from fmin to 0.5 cycles per sec.
- beta : real vector [1,N] Variable of the Mellin transform mellin.
- M : positive integer. Number of time samples to be recovered from mellin.

**OUTPUT PARAMETERS :**

- x : complex vector [1,M] Inverse Fourier-Mellin transform of mellin.
- t : time variable of the Inverse Fourier-Mellin transform x.

**DESCRIPTION :**
The Inverse Fourier-Mellin transform can be viewed as an Inverse Fast Fourier Transform which result is assumed geometrically sampled. To recover the initial time signal, a Discrete Inverse Fourier Transform is applied to this geometrically Fourier representation. Important The Inverse Fourier-Mellin transform is correct only if the direct Fourier-Mellin transform has been computed from fmin to 0.5 cycles per sec.

**SEE ALSO: :**
dmt, dilate

**EXAMPLE: :**

```
//Signal synthesis
x = morlet(0.1,32) ;
plot(x)
//Computation of the Mellin transform
[mellin,beta] = dmt(x,0.01,0.5,128) ;
plot(beta,abs(mellin))
//Computation of the Inverse Mellin transform

[y,t] = idmt(mellin,beta,65) ;
plot(t,abs(x-y))
```

## 0.39 integ — Approximate 1-D integral

Author: Paulo Goncalves

Approximate 1-D integral. integ(y,x) approximates the integral of y with respect to the variable x

**USAGE :**

```
SOM = integ(y[,x])
```

**INPUT PARAMETERS :**

- y : real valued vector or matrix [ry,cy] Vector or matrix to be integrated. For matrices, integ(Y) computes the integral of each column of Y
- x : row-vector [ry,1] Integration path of y. Default value is (1:cy)

**OUTPUT PARAMETERS :**

- SOM : real valued vector [1,cy] Finite sum approximating the integral of y w.r.t the integration path x

**SEE ALSO: :**
integ2d

**EXAMPLE: :**

```
//Cumulative Normal Distribution
sigma = 1 ; N = 100 ;
x = logspace(log10(0.001),log10(3),N/2) ;
x = [ -fliplr(x) x ] ;
y = 1/sqrt(2*pi) * exp( -(x.^2)./2 ) ;
plot(x,y)
for n = 1:N
  PartialSom(n) = integ( y(1:n),x(1:n) ) ;
end
xbasc();plot2d(x,PartialSom,-1)
```

## 0.40 isempty Checks if a matrix is empty

Author: Paulo Goncalves

isempty True for empty matrix. isempty(x) returns %T if x is an empty array and %F otherwise. An empty array has no elements, that is prod(size(X))==0.

**USAGE :**

```
isempty(x)
```

**INPUT PARAMETERS :**

- x : Real or complex valued matrix [rx,cx]

**SEE ALSO: :**
all

## 0.41 lambdak k's lambda functions for pseudoAW

Author: Paulo Goncalves

Computes the parametrizing function lambdak defining the Affine Wigner distributions.

**USAGE :**

```
[y] = lambdak(u,k)
```

**INPUT PARAMETERS :**

- u : real vector [1,n] Argument of the function lambdak.
- k : real scalar Parameter of the lambdak function. K=-1 corresponds to the Unterberger distribution; K=0 corresponds to the Bertrand distribution; K=0.5 corresponds to the D-Flandrin distribution; K=2 corresponds to the Wigner-Ville distribution on analytic signals.

**OUTPUT PARAMETERS :**

- y : real vector [1,n] Result of the function lambdak .

**SEE ALSO: :**
pseudoAW

**EXAMPLE: :**

```
x = linspace(-10,10,101) ;
y0 = lambdak(x,-1) ;
y1 = lambdak(x,2) ;
plot(y0)
plot(y1)
```

## 0.42 lepskiiap lepskii adaptive procedure

Author: Christophe Canus

This C_LAB routine is an implementation of the Lepskii's adaptive procedure. This algorithm selects the "best" estimator which balances the bias-variance tradeoff in a sequence of noisy and biased estimators theta_hat_j of a non-random parameter theta with the assumption that when j increases, bias b_j increases as variance sigma2_j decreases.

**USAGE :**

```
[K_star,j_hat,I_c_j_min,I_c_j_max,E_c_j_hat_min,E_c_j_hat_max]=..
lepskiiap(theta_hat_j,[sigma2_j,K])
```

**INPUT PARAMETERS :**

- theta_hat_j : real vector [1,J] or [J,1] Contains the sequence of estimators.
- sigma2_j : strictly positive real vector [1,J] or [J,1] Contains the sequence of variances.
- K : strictly positive real scalar Contains the confidence constant.

**OUTPUT PARAMETERS :**

- K_star : strictly positive real scalar Contains the optimal confidence constant.
- j_hat : strictly positive real (integer) scalar Contains the selected index.
- I_c_j_min : real vector [1,J] Contains the minimum bounds of the confidence intervals.
- I_c_j_max : real vector [1,J] Contains the maximum bounds of the confidence intervals.
- E_c_j_hat_min : real scalar Contains the minimum bound of the selected intersection interval.

o E_c_j_hat_max : real scalar Contains the maximum bound of the selected intersection interval.

**DESCRIPTION :**

**PARAMETERS :**

The sequence of variances sigma_j must be stricly positive, decreasing when j increases and of the same size than theta_hat_j. When no sequence of variances is given as input or when it is uniformly equal to 0, the algorithm computes the sequence of variances as sigma2_j=1./j. The default value for epsilon is 1./[1:J]. The confidence constant K must be >=1. For the meaning of the ouput parameters, see next section.

**ALGORITHM DETAILS :**

Define the sequence of confidence intervals I_c_j=[theta_hat_j-K*sigma_j,theta_hat_j+K*sigma_j], the sequence of their decreasing intersections E_c_j and j_hat as the largest index j such as that E_c_j is non void. The best estimator with respect to the Lepskii's adaptive procedure is selected as theta_hat_j_hat in E_c_j_hat. The two parameters to be handled are the sequence of variances sigma2_j and the confidence constant K. sigma2_j can be any sequence dominating the estimator variance. Choosing a smaller K speeds up the selection and results to smaller j_hat.

**EXAMPLES :**

**MATLAB :**

```
T=33;
% linear model
f_t=linspace(0,1,T);
% jump for t=floor(3/4*T)
f_t(floor(3/4*T):T)=2*f_t(floor(3/4*T):T);
% Wiener process
W_t=randn(1,T);
sigma=.1;
Y_t=f_t+sigma*W_t;
subplot(2,1,1);
plot(f_t);hold on;plot(Y_t);
title('White noise model Y(t)');
xlabel('index: t');
ylabel('Y(t)=f(t)+\sigma W(t)');
% estimation for t=t_0=floor(T/2)
t_0=floor(T/2)+1;
Y_t=f_t+sigma*W_t;
for t=1:floor(T/2)
  f_hat_t(t)=mean(Y_t(t_0-t:t_0+t));
end
% Lespkii's adaptive procedure
[K_star,t_hat,I_c_t_min,I_c_t_max,E_c_t_hat_min,E_c_t_hat_max]=lepskiiap(f_hat_t,.005*1
% plot and disp results
plot(t_0,Y_t(t_0),'k*');
plot(t_0-t_hat,Y_t(t_0-t_hat),'kd');
plot(t_0+t_hat,Y_t(t_0+t_hat),'kd');
subplot(2,1,2);
plot(f_hat_t);
hold on;
plot(I_c_t_max,'r^');
plot(I_c_t_min,'gV');
title(['estimator \theta_t(t_0) vs. index t with t_0=',num2str(floor(T/2)+1)]);
xlabel('index: t');
```

```
ylabel('estimator: \theta_t(t_0)');
plot(t_hat,E_c_t_hat_min,'ko');
plot(t_hat,E_c_t_hat_max,'ko');
disp(['linear estimation of f_t for t=t_0=',num2str(t_0)]);
disp(['selected index t=',num2str(t_hat)]);
disp(['estimated f_t_0 in [',num2str(E_c_t_hat_min),',',num2str(E_c_t_hat_min),']']);
```

**SCILAB :**

```
//
```

**REFERENCES :**
To be published..SH See Also monolr (C_LAB routine).

## 0.43 linearlt linear time legendre transform

Author: Christophe Canus

This C_LAB routine the Legendre transform of a function using the linear time Legendre transform algorithm.

**USAGE :**

```
[s,u_star_s]=linearlt(x,u_x)
```

**INPUT PARAMETERS :**

- x : real vector [1,N] or [N,1] Contains the abscissa.
- y : real vector [1,N] or [N,1] Contains the function to be transformed.

**OUTPUT PARAMETERS :**

- s : real vector [1,M] Contains the abscissa of the regularized function.
- u_star_s : real vector [1,M] Contains the Legendre conjugate function.

**DESCRIPTION :**

**PARAMETERS :**
The abscissa x and the function u_x to be transformed must be of the same size [1,N] or [N,1]. The abscissa s and the Legendre conjugate function u_star_s are of the same size [1,M] with M<=N.

**ALGORITHM DETAILS :**
The linear time Legendre transform algorithm is based on the use of a concave regularization before slopes' computation.

**EXAMPLES :**

```
x=linspace(-5.,5.,1024);
u_x=-1+log(6+x);
plot2d(x,u_x);
//looks like a Reyni exponents function, isn't it ?
[s,u_star_s]=linearlt(x,u_x);
plot2d(s,u_star_s);
```

**REFERENCES :**
None.

**SEE ALSO :**
bbch (C_LAB routine).

## 0.44 mbmlevinson _____ Levinson synthesis of a multifractional Brownian motion

Author: Paulo Goncalves

Generates a Multi-Fractional Brownian Motion (mBm) using Cholesky/Levinson factorization

**USAGE :**

```
[x,y,r] = mbmlevinson(N,H,[seed])
```

**INPUT PARAMETERS :**

- N : Positive integer Sample size of the fBm
- H : Real vector [1,N] of character string H real vector: contains the Holder exponents at each time. Each element in [0,1]. H character string: analytic expression of the Holder function (e.g. 'abs(0.5 * ( 1 + sin(16 t) ) )')
- seed : real scalar Random seed generator

**OUTPUT PARAMETERS :**

- x : real vector [1,N] Time samples of the mBm
- y : real vector [1,N] Vector of N i.i.d. white standard Gaussian r.v.'s (input process of the generator)
- r : real matrix [N,N] Matrix containing columnwise each first row of the var/cov Toeplitz matrices R(n) of the non-stationary increment process w[n] = x[n+1] - x[n].

**ALGORITHM DETAILS :**
For each time n, a fbm process with constant Holder exponent H[n/] is synthesized over N points (see fbmlevinson). Only the sample at rank n is kept. As a result of this computationally expensive procedure, only small sample sizes of mBms can be generated (typically less than 1024 samples).

**SEE ALSO: :**
mbmlevinson

**EXAMPLE: :**

```
[x,y,r] = mbmlevinson(512,AtanH(512,2,1,0.5)) ;
plot(x) ;
```

# 0.45 mcfg1d — Continuous large deviation spectrum estimation on 1d measure

Author: Christophe Canus

This C_LAB routine estimates the continuous large deviation spectrum on 1d measure.

**USAGE :**

```
[alpha,fgc_alpha,[pc_alpha,epsilon_star,eta,alpha_eta_x]]=..
    mcfg1d(mu_n,[S_min,S_max,J],progstr,ballstr,N,epsilon,..
    contstr,adapstr,kernstr,normstr,I_n])
```

**INPUT PARAMETERS :**

- mu_n : strictly positive real vector [1,N_n] or [N_n,1] Contains the 1d measure.
- S_min : strictly positive real scalar Contains the minimum size.
- S_max : strictly positive real scalar Contains the maximum size.
- J : strictly positive real (integer) scalar Contains the number of scales.
- progstr : string Contains the string which specifies the scale progression.
- ballstr : string Contains the string which specifies the type of ball.
- N : strictly positive real (integer) scalar Contains the number of Hoelder exponents.
- epsilon : strictly positive real vector [1,N] or [N,1] Contains the precisions.
- contstr : string Contains the string which specifies the definition of continuous spectrum.
- adapstr : string Contains the string which specifies the precision adaptation.
- kernstr : string Contains the string which specifies the kernel form.
- normstr : string Contains the string which specifies the pdf's normalization.
- I_n : strictly positive real vector [1,N_n] or [N_n,1] Contains the intervals on which the pre-multifractal 1d measure is defined.

**OUTPUT PARAMETERS :**

- alpha : real vector [1,N] Contains the Hoelder exponents.
- fgc_alpha : real matrix [J,N] Contains the spectrum(a).
- pc_alpha : real matrix [J,N] Contains the pdf('s).
- epsilon_star : strictly positive real matrix [J,N] Contains the optimal precisions.
- eta : strictly positive real vector [1,J] Contains the sizes.
- alpha_eta_x : strictly positive real matrix [J,N_n] Contains the coarse grain Hoelder exponents.

**DESCRIPTION :**

**PARAMETERS :**
The continuous large deviation spectrum (alpha,fgc_alpha) is estimated for J sizes eta_j and for the precision vector epsilon by taking into account the resolution of the 1d measure mu_n. The minimum size S_min sets the equivalent size eta_1 in the unit interval at which the first spectrum is estimated. eta_1 is equal to S_min*eta_n where eta_n is related to the resolution of the 1d measure (eta_n=N_n{-1} when all intervals are of equal size else it is max(|I_n|{-1}). It must be >=1. The default value for S_min is 1. The maximum size S_max sets the equivalent size eta_J in the unit interval at which the last spectrum is estimated. eta_J is equal to S_max*eta_n. It must be >=S_min. The default value for S_max is 1. The number of scales J sets the number of computed spectra. The bigger J is, the slower the computation is. It must be >=1. The

default value for J is 1. The scale progression string progstr specifies the type of scale discretization. It can be 'dec' for decimated, 'log' for logarithmic or 'lin' for linear scale. The default value for progstr is 'dec'. The ball string ballstr specifies the type of ball B_eta(x). It can be 'asym' for asymmetric, 'cent' for centered or 'star' for three times bigger asymmetric ball. The default value for ballstr is 'asym'. The number N sets the discretization of the Hoelder exponents interval. They are linearly spaced between alpha_eta_min and alpha_eta_max which are the minimum and maximum values of the coarse grain Hoelder exponents at size eta. The bigger N is, the slower the computation is. It must be >=1. The default value for N is 100. The precision vector epsilon sets the precisions at which the spectrum is estimated. It must be of size [1,N] or [N,1]. When no precision vector is given as input or when it is uniformly equal to 0, the algorithm determines the optimal precisions vector epsilon_star. The default value for epsilon is zeros(1,N). The continuous string contstr specifies the definition of continuous spectrum. It can be equal to 'hnokern' for definition without precision and kernel or 'hkern' for definition with precision and kernel. The default value for contstr is 'hkern'. The precision adaptation string adapstr specifies the local adaptation of the precision w.r.t. the Hoelder exponents alpha. It can be equal to 'maxdev' for maximum deviation or 'maxadaptdev' for maximum adaptive deviation. The default value for adapstr is 'maxdev'. The kernel string kernstr specifies the kernel. It can be equal to 'box' for boxcar, 'tri' for triangle, 'mol' for mollifier, 'epa' for epanechnikhov or 'gau' for gaussian kernel. The default value for kernstr is 'gau'. The normalization string normstr specifies the type of pdf's normalization conducted before double log-normalization. It can be equal to 'nonorm' for no normalization conducted, 'suppdf' for normalization w.r.t the supremum of pdf's, 'infsuppdf' for normalization w.r.t the infimum and the supremum of pdf's. The default value for normstr is 'suppdf'. The intervals vector I_n can be useful when the intervals on which the pre-multifractal 1d measure is defined are not of equal size (not implemented yet). The pdf of the coarse grain Hoelder exponents matrix or vector pc_alpha, the optimal precisions matrix or vector epsilon_star, the sizes vector eta and the coarse grain Hoelder exponents matrix or vector alpha_eta_x can be obtained as outputs parameters.

**ALGORITHM DETAILS :**

The coarse Hoelder exponents are estimated on each point x of the unit interval discretization by summing interval measures into a sliding window of size eta containing x (which corresponds to ball B_eta(x)). The probability density function pc_alpha is obtained by integrating horizontal sections.

**EXAMPLES :**

```
// computation of pre-multifractal besicovitch measure: mu_n
// resolution of the pre-multifractal measure
n=10;
// parameter of the besicovitch measure
p_0=.4;
// synthesis of the pre-multifractal besicovitch 1d measure
[mu_n,I_n]=binom(p_0,'meas',n);
// continuous large deviation spectrum estimation: fgc_alpha
// minimum size, maximum size & # of scales
S_min=1;S_max=8;J=4;
// # of hoelder exponents, precision vector
N=200;epsilon=zeros(1,N);
// estimate the continuous large deviation spectrum
[alpha,fgc_alpha,pc_alpha,epsilon_star]=mcfg1d(mu_n,[S_min,S_max,J],'dec','cent',N,epsi
// plot the Continuous Large Deviation spectrum
plot2d(a,f,[6]);
xtitle(["Continuous Large Deviation spectrum";" "],"alpha","fgc(alpha)");
```

**REFERENCES :**

To be published.

**SEE ALSO :**

mch1d, fch1d, fcfg1d, cfg1d (C_LAB routines). MFAG_continuous, MFAG_epsilon, MFAG_eta, MFAG_epsilon_eta (Matlab and/or Scilab functions).

## 0.46 mdfl1d Discrete Legendre spectrum estimation on 1d measure

Author: Christophe Canus

This routine estimates the discrete Legendre Spectrum on 1d measure.

**USAGE :**

```
[alpha,f_alpha]=mdfl1d(mu_n,N,n)
```

**INPUT PARAMETERS :**

- mu_n : strictly positive real vector [1,nu_n] Contains the pre-multifractal measure.
- N : strictly positive real (integer) scalar Contains the number of Hoelder exponents.
- n : strictly positive real (integer) scalar Contains the final resolution.

**OUTPUT PARAMETERS :**

- alpha : real vector [1,N] Contains the Hoelder exponents.
- f_alpha : real vector [1,N] Contains the dimensions.

**DESCRIPTION :**

**PARAMETERS :**
The discrete Legendre spectrum f_alpha is estimated on the finite finer resolution of the pre-multifractal 1d measure mu_n. The three steps of the estimatation are:

- estimation of the partition function;
- estimation of the Reyni exponents;
- estimation of the Legendre transform.

**ALGORITHM DETAILS :**

The discrete partition function is estimated by coarse-graining masses mu_n into non-overlapping boxes of increasing diameter (box method). If nu_n is a power of 2, 2^n corresponds to the coarser scale. The reyni exponents are estimated by least square linear regression. The Legendre transform of the mass exponent function is estimated with the linear-time Legendre transform.

**SEE ALSO :**
mdzq1d,mdzq2d,reynitq,linearlt,mdfl2d.

## 0.47 mdfl2d Discrete Legendre spectrum estimation on 2d measure

Author: Christophe Canus

This routine estimates the discrete Legendre spectrum on a pre-multifractal 2d measure.

**USAGE :**

```
[alpha,fl_alpha]=mdfl2d(mu_n,N,n)
```

**INPUT PARAMETERS :**

- mu_n : strictly positive real matrix [nux_n,nuy_n] Contains the pre-multifractal measure.
- N : strictly positive real (integer) scalar Contains the number of Hoelder exponents.
- n : strictly positive real (integer) scalar Contains the final resolution.

**OUTPUT PARAMETERS :**

- alpha : real vector [1,N] Contains the Hoelder exponents.
- fl_alpha : real vector [1,N] Contains the dimensions.

**DESCRIPTION :**

**PARAMETERS :**

The discrete Legendre spectrum fl_alpha is estimated on the finite finer resolution of the 2d measure mu_n. The three steps of the estimation are:

- estimation of the discrete partition function;
- estimation of the Reyni exponents;
- estimation of the Legendre transform.

**ALGORITHM DETAILS :**
The discrete partition function is estimated by coarse-graining masses mu_n into non-overlapping boxes of increasing diameter (box method). If nux_n and nuy_n are power of 2, 2n corresponds to the coarser scale. The Reyni exponents are estimated by least square linear regression. The Legendre transform of the mass exponent function is estimated with the linear-time Legendre transform.

**SEE ALSO :**
mdznq1d,mdznq2d,reynitq,linearlt,mdfl1d.

## 0.48 mdznq1d ——— Discrete partition function estimation on 1d measure

Author: Christophe Canus

This routine computes the discrete partition function on a pre-multifractal 1d measure.

**USAGE :**

```
[mznq]=mdznq1d(mu_n,n,q)
```

**INPUT PARAMETERS :**

- mu_n : strictly positive real vector Contains the pre-multifractal measure.
- n : strictly positive real (integer) vector Contains the resolutions.
- q : strictly positive real vector Contains the exponents.

**OUTPUT PARAMETERS :**

- mznq : real matrix [size(q),size(n)] Contains the partition function.

**DESCRIPTION :**

**PARAMETERS :**

The discrete partition function mznq is computed on the pre-multifractal 1d measure mu_n. The vector of resolutions n and the vector of exponents q sets the size of the output real matrix mznq to size(q)*size(n).

**ALGORITHM DETAILS :**

The discrete partition function mznq is computed by coarse-graining masses mu_n into non-overlapping boxes of increasing diameter (box method). If nu_n is a power of 2, n corresponds to the resolution.

**SEE ALSO :**
mdzq2d, reynitq, linearlt, mdfl1d, mdfl2d

## 0.49 mdznq2d _____ Discrete partition function estimation on 2d measure

Author: Christophe Canus

This routine computes the discrete partition function on a pre-multifractal 2d measure.

**USAGE :**

```
[mznq]=mdznq2d(mu_n,n,q)
```

**INPUT PARAMETERS :**

- mu_n : strictly positive real matrix Contains the pre-multifractal measure.
- n : strictly positive real (integer) vector Contains the resolutions.
- q : strictly positive real vector Contains the exponents.

**OUTPUT PARAMETERS :**

- mznq : real matrix [size(q),size(n)] Contains the discrete partition function.

**DESCRIPTION :**

**PARAMETERS :**

The discrete partition function mznq is computed on the pre-multifractal 2d measure mu_n. The vector of resolutions n and the vector of exponents q sets the size of the output real matrix mznq to size(q)*size(n).

**ALGORITHM DETAILS :**

The discrete partition function mznq is computed by coarse-graining masses mu_n into non-overlapping boxes of increasing diameter (box method). If nux_n and nuy_n are power of 2, n corresponds to the resolution.

**SEE ALSO :**
mdznq1d, reynitq, linearlt, mdfl1d, mdfl2d

## 0.50 mexhat Mexican hat wavelet

Author: Paulo Goncalves

Computes a Mexican Hat wavelet (seconde derivative of the gaussian).

**USAGE :**

```
[wavelet,alpha] = mexhat(nu)
```

**INPUT PARAMETERS :**

- nu : real scalar between 0 and 1/2 Central (reduced) frequency of the wavelet.

**OUTPUT PARAMETERS :**

- wavelet : real vector [1,2*N+1] Mexican Hat wavelet in time.
- alpha : real scalar Attenuation exponent of the Gaussian enveloppe of the Mexican Hat wavelet.

**SEE ALSO: :**
morlet, contwt

**EXAMPLE: :**

```
//wavelet synthesis

wavelet1 = mexhat(0.05) ; plot(wavelet1)

wavelet2 = mexhat(0.2) ;plot(wavelet2)
```

# 0.51 monolr — monovariate linear regression

Author: Christophe Canus

This C_LAB routine provides six different algorithms to proceed linear regression on monovariate data: least square, weighted least square, penalized least square, multiple least square, maximum likelyhood and Lepskii's adaptive procedure least square, in one sole routine.

**USAGE :**

```
[a_hat,[b_hat,y_hat,e_hat,sigma2_e_hat,optvarargout]=..
                                    monolr(x,y,[lrstr,optvarargin])
```

**INPUT PARAMETERS :**

- x : real vector [1,J] or [J,1] Contains the abscissa.
- y : real vector [1,J] or [J,1] Contains the ordinates to be regressed.
- lrstr : string Contains the string which specifies the type of linear regression to be used.
- optvarargin : Contains optional variable input arguments. Depending on the choice of linear regression, the fourth parameter can be
- w : strictly positive real vector [1,J] or [J,1] If weighted least square is chosen, contains the weights.
- I : strictly positive real (integer) scalar If penalized least square is chosen, contains the number of iterations.
- sigma2_j : strictly positive real vector [1,J] or [J,1] If Lepskii's adaptive procedure least square is chosen, contains the sequence of variances.

  The fifth parameter can be
- m : real scalar If penalized least square is chosen, contains the mean of the normal weights.
- K : strictly positive real scalar If Lepskii's adaptive procedure least square is chosen, contains the confidence constant.

  The sixth parameter can be
- s : strictly positive real scalar If penalized least square is chosen, contains the variance of the normal weights.

**OUTPUT PARAMETERS :**

- a_hat : real scalar or vector [1,J] Contains the estimated slope.
- b_hat : real scalar or vector [1,J] Contains the estimated ordimate at the origin.
- y_hat : real vector [1,J] or [1,(J+2)*(J-1)/2] Contains the regressed ordinates.
- e_hat : real vector [1,J] or [1,(J+2)*(J-1)/2] Contains the residuals.
- sigma2_e_hat : real scalar Contains the residuals' variance (that is, the mean square error).
- optvarargout : Contains optional variable output arguments. If Lepskii's adaptive procedure least square is chosen, the parameters are
- K_star : strictly positive real scalar Contains the optimal confidence constant.
- j_hat : strictly positive real (integer) scalar Contains the selected index.
- I_c_j_min : real vector [1,J] Contains the minimum bounds of the confidence intervals.
- I_c_j_max : real vector [1,J] Contains the maximum bounds of the confidence intervals.
- E_c_j_hat_min : real scalar Contains the minimum bound of the selected intersection interval.
- E_c_j_hat_max : real scalar Contains the maximum bound of the selected intersection interval.

**DESCRIPTION :**

**PARAMETERS :**

The abscissa x and the ordinate y to be regressed with must be of the same size [1,J] or [J,1]. The linear regression string lrstr specifies the type of linear regression used. It can be 'ls' for least square, 'wls' for weighted least square, 'pls' for penalized least square, 'mls' for multiple least square (that is for j varying from 1 to J), 'ml' for maximum likelyhood, 'lapls' for Lepskii's adaptive procedure least square. The default value for lrstr is 'ls'. The weights w or the sequence of variances sigma2_j must be strictly positive and of size [1,J] or [J,1]. For the meaning of the variable optional input parameters sigma2_j and K, see lepskiiap (Lepskii's Adaptive Procedure) C_LAB routine's help. The number of iterations I must be >=2. The variance of the normal weights s must be strictly positive. If multiple least square, maximum likelyhood or Lepskii's adaptive procedure least square is chosen, the estimated slope a_hat and the ordinate at the origin b_hat are vectors of size [1,J], resp. the regressed ordinates y_hat and the residuals e_hat vectors are of size [1,(J+2)*(J-1)/2] (as they contains results for multiple linear regression, be aware of that when vizualising them :-), see examples), otherwise there are scalars, resp. vectors of size [1,J]. For maximum likelyhood, multiple least square linear regressions are proceeded in order to obtain variance estimates. Then maximum likelyhood linear regression is proceeded (corresponding results are found in a_hat(1), b_hat(1), y_hat(1:J), e_hat(1:J) and sigma2_e_hat(1), see examples). For the meaning of the variable optional output parameters K_star, j_hat, I_c_j_min, I_c_j_max, E_c_j_max, and E_c_j_max, see lepskiiap (Lepskii's Adaptive Procedure) C_LAB routine's help.

**ALGORITHM DETAILS :**

For the details of the Lepskii's adaptive procedure, see lepskiiap (Lepskii's Adaptive Procedure) C_LAB routine's help.

**EXAMPLES :**

**MATLAB :**

```
J=32;
x=1+linspace(0,1,J);
% Wiener process
W=randn(1,J);
epsilon=.1;
y=x+epsilon*W;
% least square
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat]=monolr(x,y);
plot(x);hold on;plot(y);plot(y_hat,'kd');
plot(epsilon.*W);hold on;plot(e_hat);
title('least square');
disp('type return');
pause;
clf;
% weighted least square
epsilon=linspace(.05,.5,J);
y=x+epsilon.*W;
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat]=monolr(x,y,'wls',1./epsilon);
plot(x);hold on;plot(y);plot(y_hat,'kd');
plot(epsilon.*W);hold on;plot(e_hat);
title('weighted least square');
disp('type return');
pause;
clf;
% penalized least square
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat]=monolr(x,y,'pls',30);
```

```
plot(x);hold on;plot(y);plot(y_hat);
title('penalized least square');
disp('type return');
pause;
clf;
% multiple least square
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat]=monolr(x,y,'mls');
plot(x);hold on;plot(y)
start_j=0;
hold on;
for j=2:J
  plot([1:j],y_hat(start_j+1:start_j+j),'k');
  disp(['estimated slope a_hat =',num2str(a_hat(j))]);
  disp('type return');
  pause;
  start_j=start_j+j;
  j=j+1;
end
clf
% maximum likelyhood
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat]=monolr(x,y,'ml');
plot(x);hold on;plot(y_hat(1:J),'kd');
plot(epsilon.*W);hold on;plot(e_hat(1:J));
clf;
% Lespkii's adaptive procedure
epsilon=.01;
y(1:16)=x(1:16)+epsilon*W(1:16);
y(16:32)=2*x(16:32)+epsilon*W(16:32);
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat,K_star,j_hat,I_c_j_min,I_c_j_max,E_c_j_hat_min,E_
plot(a_hat);
hold on;
plot(I_c_j_max,'rˆ');
plot(I_c_j_min,'gV');
title('LAP: estimator vs. index');
xlabel('index: j');
ylabel('estimator: \theta_j');
plot(j_hat,E_c_j_hat_min,'ko');
plot(j_hat,E_c_j_hat_max,'ko');
```

**SCILAB :**

```
//
```

**REFERENCES :**

To be published.

**SEE ALSO :**

lepskiiap (C_LAB routine).

## 0.52 morlet — Morlet wavelet

Author: Paulo Goncalves

Computes a Morlet wavelet.

**USAGE :**

```
[wavelet,alpha] = morlet(nu,[N,analytic])
```

**INPUT PARAMETERS :**

- nu : real scalar between 0 and 1/2 Central (reduced) frequency of the wavelet
- N : Positive integer Half length of the wavelet transform. Default value corresponds to a total length of 4.5 periods.
- analytic : boolean (0/1) under Matalb or (%F/%T) under Scilab. 0 or %F : real Morlet wavelet 1 or %T : analytic Morlet wavelet

**OUTPUT PARAMETERS :**

- wavelet : real or complex vector [1,2*N+1] Morlet wavelet in time.
- alpha : real scalar Attenuation exponent of the Gaussian enveloppe of the Morlet wavelet.

**SEE ALSO: :**
mexhat, contwt

**EXAMPLE: :**

```
//wavelet synthesis
wavelet1 = morlet(0.1,64) ;plot(wavelet1)

wavelet2 = morlet(0.1) ;plot(wavelet2)
```

## 0.53 multim1d multinomial 1d measure synthesis

Author: Christophe Canus

This C_LAB routine synthesizes a large range of pre-multifractal measures related to the multinomial 1d measure (deterministic, shuffled, pertubated) and computes linked theoretical functions (partition sum function, Reyni exponents function, generalized dimensions, multifractal spectrum).

**USAGE :**

```
[varargout,[optvarargout]]=multim1d(b,p,str,varargin,[optvarargin])
```

**INPUT PARAMETERS :**

- b : strictly positive real (integer) scalar Contains the base of the multinomial.
- p : strictly positive real vector [1,b] Contains the weights of the multinomial.
- str : string Contains the type of ouput.
- varargin : variable input argument Contains the variable input argument.

- optvarargin : optional variable input arguments Contains optional variable input arguments.

**OUTPUT PARAMETERS :**

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

**DESCRIPTION :**

**PARAMETERS :**

The multinomial 1d measure is completly characterized by its base b and its weights p(i) (i=1 to b). The first parameter b must be $>1$. The second parameter must be a vector of size equal to b. The weights p(i) must be $>0.$, $<1.$ and their sum must be =1. (the case of p(i)=1/b corresponds to the Lebesgue measure) (i=1 to b). The third parameter str is a variable string used to determine the desired type of output. There are six suffix strings ('meas' for measure, 'cdf' for cumulative distribution function , 'pdf' for probability density function, 'part' for partition sum function, 'Reyni' for Reyni exponent function , 'spec' for multifractal spectrum) for the deterministic multinomial measure and two prefix strings for related measures ('shuf' for shuffled , 'pert' for pertubated) which can be added to the first ones to form composed strings. For example, 'shufmeas' is for the synthesis of a shuffled multinomial 1d pre-multifractal measure. Note that all combinaisons of strings are not implemented yet. When a string containing suffix string 'meas' is given as third input, a pre-multifractal measure mu_n (first output argument) is synthesized on the b-adic intervals I_n (second optional output argument) of the unit interval. In that case, the fourth input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real vectors mu_n (and I_n if used) is equal to bn (so be aware the stack size ;-)). This option is implemented for the deterministic ('meas'), shuffled ('shufmeas') and pertubated ('pertmeas') multinomial 1d measure. When a string containing prefix 'shuf' is given as third input, the synthesis is made for a shuffled multinomial measure. At each level of the multiplicative cascade and for all nodes of the corresponding binary tree, the vector of weights p is shuffled. This option is implemented only for the multinomial 1d measure ('shufmeas'). When a string containing prefix 'pert' is given as third input, the synthesis is made for a pertubated multinomial measure. In that case, the fifth input argument is a strictly positive real scalar epsilon which contains the pertubation around weights. The weights are independant random variables identically distributed between p(i)-epsilon and p(i)+epsilon which must be $>0.$, $<1.$ (i=1 to b). This option is implemented only for the multinomial 1d measure ('pertmeas'). When replacing suffix string 'meas' with suffix string 'cdf', respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument). When string 'part' is given as third input, the partition sum function znq of multifractal measure is computed as sole output argument. In that case, the fourth input argument is a strictly positive real (integer) vector vn which contains the resolutions, and the fifth input argument is a real vector q which contains the measure exponents. The size of the output real matrix znq is equal to size(q)*size(vn). This option is implemented only for the multinomial 1d measure. When string 'Reyni' is given as third input, the Reyni exponents function tq (and the generalized dimensions Dq if used) of the multifractal measure is computed as first output argument (and second optional output argument if used). In that case, the fourth input argument is a real vector q which contains the measure's exponents. The size of the output real vector tq is equal to size(q)). This option is implemented only for the multinomial 1d measure. When string 'spec' is given as third input, the multifractal spectrum f_alpha (second output argument) is synthesized on the Hoelder exponents alpha (first output argument). In that case, the fourth input argument is a strictly positive real (integer) scalar N which contains the number of Hoelder exponents. The size of both output real vectors alpha and f_alpha is equal to N. This option is implemented only for the multinomial 1d measure.

**ALGORITHM DETAILS :**

For the deterministic multinomial, the pre-multifractal measure synthesis algorithm is implemented is a iterative way (supposed to run faster than a recursive one). For the shuffled or the pertubated multinomial,

the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). In the case of the pertubated multinomial, the weights of each node are normalised by their sum for the measure to remain conservative. Note that the shuffled multinomial 1d measure is not conservative.

**EXAMPLES :**

```
b=3;
p=[.1 .3 .6];
n=8;
// synthesizes a pre-multifractal multinomial 1d measure
[mu_n,I_n]=multim1d(b,p,'meas',n);
plot(I_n,mu_n);
// synthesizes the cdf of a pre-multifractal shuffled multinomial 1d measure
F_n=multim1d(b,p,'shufcdf',n);
plot(I_n,F_n);
e=.09;
// synthesizes the pdf of a pre-multifractal pertubated multinomial 1d measure
p_n=multim1d(b,p,'pertpdf',n,e);
plot(I_n,p_n);
xbasc();
vn=[1:1:8];
q=[-5:.1:+5];
// computes the partition sum function of a multinomial 1d measure
znq=multim1d(b,p,'part',vn,q);
mn=zeros(max(size(q)),max(size(vn)));
for i=1:max(size(q))
   mn(i,:)=-vn*log(2);
end
plot2d(mn',log(znq'));
// computes the Reyni exponents function of a multinomial 1d measure
tq=multim1d(b,p,'Reyni',q);
plot(q,tq);
N=200;
// computes the multifractal spectrum of a multinomial 1d measure
[alpha,f_alpha]=multim1d(b,p,'spec',N);
plot(alpha,f_alpha);

.SH References
"Multifractal Measures", Carl J. G. Evertsz and Benoit
B. MandelBrot. In Chaos and Fractals, New Frontiers of Science,
Appendix B. Edited by Peitgen, Juergens and Saupe, Springer Verlag,
1992 pages 921-953.
"A class of Multinomial Multifractal Measures with negative
(latent) values for the "Dimension" f(alpha)", Benoit
B. MandelBrot. In Fractals' Physical Origins and Properties,
Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum
Press, New York, 1989 pages 3-29.
.SH See also
binom, sbinom, multim2d, smultim1d, smultim2d (C_LAB routines).
MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).
```

## 0.54 multim2d multinomial 2d measure synthesis

Author: Christophe Canus

This C_LAB routine synthesizes a large range of pre-multifractal measures related to the multinomial 2d measure (deterministic, shuffled, pertubated) and computes linked theoretical functions (partition sum function, Reyni exponents function, generalized dimensions, multifractal spectrum).

**USAGE :**

```
[varargout,[optvarargout]]=binom(bx,by,p,str,varargin,[optvarargin])
```

**INPUT PARAMETERS :**

- bx : strictly positive real (integer) scalar Contains the abscissa base of the multinomial.
- by : strictly positive real (integer) scalar Contains the ordonate base of the multinomial.
- p : strictly positive real vector [by,bx] Contains the weights of the multinomial.
- str : string Contains the type of ouput.
- varargin : variable input argument Contains the variable input argument.
- optvarargin : optional variable input arguments Contains optional variable input arguments.

**OUTPUT PARAMETERS :**

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

**DESCRIPTION :**

**PARAMETERS :**

The multinomial 2d measure is completly characterized by its abscissa base bx, ordonate base by and its weights p(i) (i=1 to bx*by). The first two parameters bx and by must be $>1$. The third parameter must be a vector of size equal to bx*by. The weights p(i) must be $>0.$, $<1.$ and their sum must be =1. (the case of p(i)=1/(bx*by) corresponds to the Lebesgue measure) (i=1 to bx*by). The fourth parameter str is a variable string used to determine the desired type of output. There are six suffix strings ('meas' for measure, 'cdf' for cumulative distribution function , 'pdf' for probability density function, 'part' for partition sum function, 'Reyni' for Reyni exponent function , 'spec' for multifractal spectrum) for the deterministic multinomial measure and two prefix strings for related measures ('shuf' for shuffled , 'pert' for pertubated) which can be added to the first ones to form composed strings. For example, 'shufmeas' is for the synthesis of a shuffled multinomial 2d pre-multifractal measure. Note that all combinaisons of strings are not implemented yet. When a string containing suffix string 'meas' is given as fourth input, a pre-multifractal measure mu_n (first output argument) is synthesized on the bx-adic and by-adic intervals I_nx and I_ny (second and third optional output argument) of the unit square. In that case, the fifth input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real matrix mu_n is equal to bxn*byn and the one of the output real vectors I_nx and I_ny (if used) is equal to bxn and byn (so be aware the stack size ;-)). This option is implemented for the deterministic ('meas'), shuffled ('shufmeas') and pertubated ('pertmeas') multinomial 2d measure. When a string containing prefix 'shuf' is given as fourth input, the synthesis is made for a shuffled multinomial measure. At each level of the multiplicative cascade and for all nodes of the corresponding binary tree, the vector of weights p is shuffled. This option is implemented only for the multinomial 2d measure ('shufmeas'). When a string containing prefix 'pert' is given as fourth input, the synthesis is made for a pertubated multinomial measure. In that case, the fifth input argument is a strictly positive real scalar epsilon which contains the pertubation around weights. The weights are independant random variables identically distributed between p(i)-epsilon and p(i)+epsilon which must be $>0.$, $<1.$ (i=1 to bx*by). This option is implemented only for the multinomial 2d measure ('pertmeas'). When replacing suffix string 'meas' with suffix string 'cdf',

respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument). When string 'part' is given as fourth input, the partition sum function znq of multifractal measure is computed as sole output argument. In that case, the fifth input argument is a strictly positive real (integer) vector vn which contains the resolutions, and the sixth input argument is a real vector q which contains the measure exponents. The size of the output real matrix znq is equal to size(q)*size(vn). This option is implemented only for the multinomial 2d measure. When string 'Reyni' is given as third input, the Reyni exponents function tq (and the generalized dimensions Dq if used) of the multifractal measure is computed as first output argument (and second optional output argument if used). In that case, the fifth input argument is a real vector q which contains the measure's exponents. The size of the output real vector tq is equal to size(q)). This option is implemented only for the multinomial 2d measure. When string 'spec' is given as fourth input, the multifractal spectrum f_alpha (second output argument) is synthesized on the Hoelder exponents alpha (first output argument). In that case, the fifth input argument is a strictly positive real (integer) scalar N which contains the number of Hoelder exponents. The size of both output real vectors alpha and f_alpha is equal to N. This option is implemented only for the multinomial 2d measure.

**ALGORITHM DETAILS :**

For the deterministic multinomial, the pre-multifractal measure synthesis algorithm is implemented is a iterative way (supposed to run faster than a recursive one). For the shuffled or the pertubated multinomial, the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). In the case of the pertubated multinomial, the weights of each node are normalised by their sum for the measure to remain conservative. Note that the shuffled multinomial 2d measure is not conservative.

**EXAMPLES :**

```
bx=2;
by=3;
p=[.05 .1; .15 .2; .24 .26];
n=5;
// synthesizes a pre-multifractal multinomial 2d measure
[mu_n,I_nx,I_ny]=multim2d(bx,by,p,'meas',n);
plot3d(I_nx,I_ny,mu_n);
// synthesizes the cdf of a pre-multifractal shuffled multinomial 2d measure
F_n=multim2d(bx,by,p,'shufcdf',n);
plot3d(I_nx,I_ny,F_n);
e=.049;
// synthesizes the pdf of a pre-multifractal pertubated multinomial 2d measure
p_n=multim2d(bx,by,p,'pertpdf',n,e);
plot3d(I_nx,I_ny,p_n);
xbasc();
vn=[1:1:8];
q=[-5:.1:+5];
// computes the partition sum function of a multinomial 2d measure
znq=multim2d(bx,by,p,'part',vn,q);
mn=zeros(max(size(q)),max(size(vn)));
for i=1:max(size(q))
   mn(i,:)=-vn*log(2);
end
plot2d(mn',log(znq'));
// computes the Reyni exponents function of a multinomial 2d measure
tq=multim2d(bx,by,p,'Reyni',q);
plot(q,tq);
N=200;
// computes the multifractal spectrum of a multinomial 2d measure
[alpha,f_alpha]=multim2d(bx,by,p,'spec',N);
```

```
plot(alpha,f_alpha);
```

**REFERENCES :**
”Multifractal Measures”, Carl J. G. Evertsz and Benoit B. MandelBrot. In Chaos and Fractals, New Frontiers of Science, Appendix B. Edited by Peitgen, Juergens and Saupe, Springer Verlag, 1992 pages 921-953. ”A class of Multinomial Multifractal Measures with negative (latent) values for the ”Dimension” f(alpha)”, Benoit B. MandelBrot. In Fractals’ Physical Origins and Properties, Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum Press, New York, 1989 pages 3-29. .SH See also binom, sbinom, multim1d, smultim1d, smultim2d (C_LAB routines). MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).

## 0.55 nextpowQ — Rounds a number to the up-nearest power of an integer

Author: Paulo Goncalves

Rounds a number x to the up-nearest power of an integer Q

**USAGE :**

```
[xup2Q,powQ] = nextpowQ(x[,Q])
```

**INPUT PARAMETERS :**

- x : Real positive number
- Q : Positive integer. Default value is Q = 2

**OUTPUT PARAMETERS :**

- xup2Q : Positive integer x rounded to the closest power of Q
- powQ : Positive integer xup2Q = powQQ.

**SEE ALSO: :**
log, log2

## 0.56 oscillsing — Oscillating Singularity synthesis

Author: Paulo Goncalves

Generates oscillating singularities located in the interval [0 .. 1]

**USAGE :**

```
[x,Fj,Fs] = oscillsing(alpha,beta,sing_pos,N,show) ;
```

**INPUT PARAMETERS :**

- alpha : Real positive vector [1,n_sing] or [n_sing,1] Holder strenghts of the singularities
- beta : Real positive vector [1,n_sing] or [n_sing,1] Chirp exponents of the singularities
- sing_pos : Real vector [1,n_sing] or [n_sing,1] Location of the singularities in the interval [0 .. 1]
- N : Integer Sample size for the synthesized signal
- show : flag 0/1 flag = 0 : no display flag = 1 : displays the instantaneous frequencies and the synthesized signal

**OUTPUT PARAMETERS :**

- x : real vector [1,N] Time samples of the synthesized signal
- Fj : real matrix [N,n_sing] instantaneous frequencies (each column of Fj contains the frequency chirp of each singularity)
- Fs : real sampling frequency

**SEE ALSO: :**

**EXAMPLE: :**

```
[x,Fj,Fs] = oscillsing([1/2 1 2],[1 2 4],[-0.5 0 0.5],256,1) ;
```

## 0.57 prescrib _____ Generation of signals with prescribed Holder function

Author: Khalid Daoudi

Using the GIFS method, this routine generates a continous function with prescribed Holder function, while interpolating a set of point.

**USAGE :**

```
[x,y]=prescrib(Interp_pts, Holder_funct, nbr_iter)
```

**INPUT PARAMETERS :**

- Interp_pts : Real matrix [n,2] Contains the interpolation points in the format : abscissa-ordinate.
- Holder_funct : Character string Specifies the Holder function you want to prescribe. It must have the form of compositions of matlab functions of variable t ('2*sqrt(1-t)' for instance). The use of the variable t is crucial. For shake of simplicity, this variable t is supposed to vary in [0,1].
- nbr_iter : integer Number of iteration wanted in the generation process of the GIFS attractor.

**OUTPUT PARAMETERS :**

- x : Real vector Contains the abscissa of the attractor graph.
- y : Real vector Contains the ordinates of the attractor graph.

**DESCRIPTION :**

**PARAMETERS :**

- Interp_pts is a real matrix [n,2] containing the cordinates of the interpolation points.
- Holder_funct is a character string specifying the Holder function you want to prescribe. This means that GIFS attrcator will have, at a point t, a Holder exponent equal to the value of this function at pint t.
- nbr_iter is the number of iteration wanted in the generation process of the GIFS attractor.
- x and y contain the cordinates of the GIFS attractor.

**ALGORITHM DETAILS :**
Generalized Iterated Functions Systems (GIFS) are a generalization of the usual IFS. This generalization consists in allowing the contarations to change at each step (scale) of the attractor generation process. We also allow their number and their support to change. Here, we use the GIFS to construct continuous function with prescribed local regularity. More precisely, if H(t) is the prescribed Holder function, then for each j=1,...,nbr_iter-1, and for each k=0,...,pow(m,j)-1, the GIFS coefficient c_kj is definied as : c_kj = pow(m,H(k*pow(m,-j))), where m+1 is the number of interpolation points. The resulting attractor is the graph of a continuous function F such that the Holder exponent of F, at each point t, is H(t). Moreover, if {(t_i, y_i), i=1,...,m+1} is the set of interpolation points, then F(t_i)=y_i for each i=1,...,m+1.

**SEE ALSO: :**
gifs and alphagifs

**EXAMPLE: :**

```
I = [0 0
    .5 1
     1 0];
[x,y] = prescrib(I,'abs(sin(3*pi*t))',10);
plot(x,y)
//[x,y] is the graph of a continuous function F which
//interpolates {(0,0); (0.5 1); (1,0)} and such that
// the Holder exponent of F, at each point t, is abs(sin(3*pi*t)).
```

## 0.58 pseudoAW — Pseudo affine Wigner distribution

Author: Paulo Goncalves

Computes a Pseudo Affine Wigner Distributions of a 1-D signal (real or complex).

**USAGE :**

```
[tfr,scale,f,wt] = pseudoAW(x,K,[wave,smooth,fmin,fmax,N]);
```

**INPUT PARAMETERS :**

- x : Real or complex vector [1,nt] or [nt,1] Time samples of the signal to be analyzed.
- K : real scalar Parameter of the pseudo affine Wigner distributions. K = -1 : pseudo Unterberger K = 0 : pseudo Bertrand K = 1/2 : pseudo D-Flandrin K = 2 : pseudo affine Wigner-Ville, etc ...

- wvlt_length : positive integer specifies the analyzing wavelet: 0: Mexican hat wavelet (real) Positive real integer: real Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Positive imaginary integer: analytic Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Default value is the Mexican hat wavelet ( wvlt_length = 0 )
- smooth : positive integer half length of the time smoothing window. SMOOTH = 0 corresponds to the Pseudo affine Wigner distribution with no time-smoothing. Default value is mooth = 0.
- fmin : real in [0,0.5] Lower frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.
- fmax : real in [0,0.5] and fmax $>$ Upper frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.
- N : positive integer. number of analyzing voices. When not specified, this parameter forces the program to interactive mode.

**OUTPUT PARAMETERS :**

- tfr : Real matrix [N,nt] time-frequency distribution
- scale : real vector [1,N] analyzed scales
- f : real vector [1,N] analyzed frequencies
- wt : real or complex matrix [N,nt] matrix of the wavelet coefficients (intermediate step)

**DESCRIPTION :**

**PARAMETERS :**

- K : fixes the function lambda_K(u) = K ( (exp(-u)-1)/(exp(-Ku)-1) )(1/(k-1)) used in the generalized affine convolution to define the K-order pseudo affine Wigner distribution.
- smooth : fixes the ammount of smooth in time of the distribution. This ammount can vary continuously from an unsmoothed pseudo affine Wigner distribution up to a maximum smoothness corresponding to a scalogram (squared magnitude of the intermediate wavelet coefficients)
- N : number of analyzing voices geometrically sampled between minimum scale fmax/fmax and maximum scale fmax/fmin.
- tfr : Samples of the pseudo affine Wigner distribution. X-coordinated corresponds to time (uniformly sampled), Y-coordinates correspond to frequency (or scale) voices (geometrically sampled between fmax (resp. 1) and fmin (resp. fmax / fmin ). First row of tfr corresponds to the highest analyzed frequency (finest scale).
- scale : analyzed scales (geometrically sampled between 1 and fmax /fmin
- f : analyzed frequencies (geometrically sampled between fmax and fmin . f corresponds to fmax/scale
- wt : coefficients of the intermediate-step wavelet transform. X-coordinated corresponds to time (uniformly sampled), Y-coordinates correspond to frequency (or scale) voices (geometrically sampled between fmax (resp. 1) and fmin (resp. fmax / fmin ). First row of wt corresponds to the highest analyzed frequency (finest scale).

**ALGORITHM DETAILS :**

A pseudo affine Wigner distribution requires to compute a continuous wavelet transform first. For each time, the corresponding column of the wavelet transform is affine convolved (generalized affine convolution defined through function lambda_K(u) ) with itself.

**SEE ALSO: :**

contwt, cwt and lambdak

**EXAMPLE: :**

```
//Signal synthesis
x = morlet(0.35,32)+morlet(0.1,32) ;
// K = -1 pseudo affine Wigner distribution with no time smoothing
```

```
[tfr,scale,f,wt] = pseudoAW(x,-1,12*i,0,0.01,0.5,128) ;
viewmat(tfr,1:length(x),f,[1 0 .5]) ;
//K = -1 time smoothed pseudo affine Wigner distribution
[tfr,scale,f,wt] = pseudoAW(x,-1,12*i,3,0.01,0.5,128) ;
viewmat(tfr,1:length(x),f,[1 0 0]) ;
```

## 0.59 regdim — Estimate the regularization dimension of a 1d or 2d sample.

Author: Francois Roueff

Computes the regularization dimension of the graph of a 1d or 2d sampled function. Two kernels are available: the Gaussian or the Rectangle.

**USAGE :**

```
dim = regdim(x,sigma,voices,Nmin,Nmax,kernel,mirror,reg,graphs)
```

**INPUT PARAMETERS :**

- x : 1d: Real vector [1,nt] or [nt,1] 2d: Real matrix [nt,pt] Time samples of the signal to be analyzed.
- sigma : Real positive number Standard Deviation of the noise. Its default value is null (noisefree)
- voices : Positive integer. number of analyzing voices. When not specified, this parameter is set to 128.
- Nmin : Integer in [2,nt/3] Lower scale bound (lower length) of the analysing kernel. When not specified, this parameter is set to 2.
- Nmax : Integer in [Nmin,2nt/3] Upper scale bound (upper length) of the analysing kernel. When not specified, this parameter is set to nt/3.
- kernel : String specifies the analyzing kernel: ”gauss”: Gaussian kernel (default) ”rect”: Rectangle kernel
- mirror : Boolean

  specifies wether the signal is to be mirrorized for the analyse (default: 0).
- reg : Boolean

  specifies wether the regression is to be done by the user or automatically (default: 0).
- graphs : Boolean:

  for one dimensional signals, it specifies wether the regularized graphs have to be displayed (default: 0). In two dimensional sugnals and for matlab only, all the regularized samples contours are plotted on a same figure.

**OUTPUT PARAMETERS :**

- dim : Real Estimated regularization dimension.
- handlefig (for Matlab only): Integer vector Handles of the figures opened during the procedure.

**DESCRIPTION :**

For a more complete explanation of the regularization dimension, one can refer to: ”A regularization approach to fractionnal dimension estimation”, F. Roueff, J. Levy-Vehel, submitted to Fractal 98 conference. The regularized graphs of x are computed via convolutions of x with dilated versions of the kernel at different scales. The lengths of the regularized graphs are computed via convolutions of x with the derivatives of the dilated versions of the kernel. The regularization dimension is computed either via an automatic range regression or via a regression by hand on the loglog plot of the lengths versus scales. If sigma is strictly positive, an estimation of the lengths without noise is used for the regression. These lengths are displayed in

red while those of the noisy signal are in black. They should seperate at fine scales. When one specifies the range regression, the loglog plot of the lengths versus scales appears. Above are either increments (when sigma is null) or a loglog plot of the noise prevalence in the lengths. One selects the scale range of the regression. In the case of noisefree signals, select a scale region with stable increments. In the case of a strictly positive sigma, select a scale region where the noise prevalence is not too close to 1 (0 in log10): it should correspond to an approximately linear region for the red estimations. The number of scales (voices) tells how many convolutions are computed. The bigger it is, the slower the computation is. The scale axis is geometrically sampled (i.e. its log is arithmetically sampled). The gaussian kernel should give a better result but the rectangle is faster.

**SEE ALSO: :**
cwttrack, cwtspec.

**EXAMPLE: :**

```
//          1D:
// Signal synthesis
x = GeneWei(1024,0.6,2,1.0,0);
plot(x);
//Dimension of the graph with a regression by hand
dim = regdim(x,0,128,10,500,'gauss',0,1,1);

//          2D
// Signal synthesis
z = GeneWei(200,0.6,2,1.0,0);
y = GeneWei(200,0.4,3,1.0,0);
w = z'*y;
plot3d(linspace(0,1,200),linspace(0,1,200),w);
//Dimension of the graph with a regression by hand
dim = regdim(w,0,25,10,50,'gauss',0,1);
```

## 0.60 reynitq — Reyni exponents estimation

Author: Christophe Canus

This routine estimates the Reyni exponents on a partition function.

**USAGE :**

```
[tq,[Dq]]=reynitq(znq,n,q)
```

**INPUT PARAMETERS :**

- znq : strictly positive real matrix Contains the partition function.
- n : strictly positive real (integer) vector Contains the resolutions.
- q : strictly positive real vector Contains the exponents.

**OUTPUT PARAMETERS :**

- tq : real vector [1,size(q)] Contains the discrete Legendre Spectrum.
- Dq : real vector [1,size(q)] Contains the generalized dimensions.

**DESCRIPTION :**

**PARAMETERS :**

The mass exponents tq and the generalized dimensions Dq (if used) are computed on the partition function znq. The input real matrix znq must be of height size(q) and of width size(n).

The output real vectors tq and Dq (if used) are of size size(q).

**ALGORITHM DETAILS :**

The mass exponent function tq by least mean square linear-fit.

**SEE ALSO :**

mdzq1d,mdzq2d,linearlt,mdfl1d,mdfl2d.

## 0.61 sbinom stochastic binomial measure synthesis

Author: Christophe Canus

This C_LAB routine synthesizes two types of pre-multifractal stochastic measures related to the binomial measure paradigm (uniform law and lognormal law) and computes linked multifractal spectrum.

**USAGE :**

```
[varargout,[optvarargout]]=sbinom(str,varargin,[optvarargin])
```

**INPUT PARAMETERS :**

- str : string Contains the type of ouput.
- varargin : variable input argument Contains the variable input argument.
- optvarargin : optional variable input arguments Contains optional variable input arguments.

**OUTPUT PARAMETERS :**

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

**DESCRIPTION :**

**PARAMETERS :**

The first parameter str is a variable string used to determine the desired type of output. There are four suffix strings ('meas' for measure, 'cdf' for cumulative distribution function , 'pdf' for probability density function, 'spec' for multifractal spectrum) and a two prefix strings for the type of stochastic measure ('unif' for uniform and 'logn' for lognormal) which must added to the first ones to form composed. For example, 'unifmeas' is for the synthesis of a uniform law binomial pre-multifractal measure and 'lognspec' is for the computation of the multifractal spectrum of a lognormal binomial measure. When a string containing suffix string 'meas' is given as second input, a pre-multifractal measure mu_n (first output argument) is synthesized on the dyadic intervals I_n (second optional output argument) of the unit interval. In that case, the third input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real vectors mu_n (and I_n if used) is equal to 2n (so be

aware the stack size ;-)). This option is implemented for the uniform law ('unifmeas') and the lognormal law ('lognmeas') binomial measures. When a string containing prefix 'unif' is given as second input, the synthesis or the computation is made for a uniform law binomial measure. In that case, the optional fourth input argument is a strictly positive real scalar epsilon which contains the pertubation around weight .5. The weight is an independant random variable identically distributed between epsilon and 1-epsilon which must be $>0.$, $<1$. The default value for epsilon is 0. When a string containing prefix 'logn' is given as second input, the synthesis or the computation is made for a lognormal law binomial measure. In that case, the optional fourth input argument is a strictly positive real scalar sigma which contains the standard deviation of the lognormal law. When replacing suffix string 'meas' with suffix string 'cdf', respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument). When a string containing suffix string 'spec' is given as second input, the multifractal spectrum f_alpha (second output argument) is synthesized on the Hoelder exponents alpha (first output argument). In that case, the third input argument is a strictly positive real (integer) scalar N which contains the number of Hoelder exponents. The size of both output real vectors alpha and f_alpha is equal to N. This option is implemented for the uniform law ('unifspec') and the lognormal law ('lognspec') binomial measures.

**ALGORITHM DETAILS :**

For the uniform and lognormal law binomial, the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). Note that the lognormal law binomial measure is not conservative.

**EXAMPLES :**

```
n=10;
// synthesizes a pre-multifractal uniform Law binomial measure
[mu_n,I_n]=sbinom('unifmeas',n);
plot(I_n,mu_n);
s=1.;
// synthesizes the cdf of a pre-multifractal lognormal law binomial measure
F_n=sbinom('logncdf',n,s);
plot(I_n,F_n);
e=.19;
// synthesizes the pdf of a pre-multifractal uniform law binomial measure
p_n=sbinom('unifpdf',n,e);
plot(I_n,p_n);
N=200;
// computes the multifractal spectrum of the lognormal law binomial measure
[alpha,f_alpha]=sbinom('lognspec',N,s);
plot(alpha,f_alpha);
```

**REFERENCES :**

"A class of Multinomial Multifractal Measures with negative (latent) values for the "Dimension" f(alpha)", Benoit B. MandelBrot. In Fractals' Physical Origins and Properties, Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum Press, New York, 1989 pages 3-29. "Limit Lognormal Multifractal Measures", Benoit B. MandelBrot. In Frontiers of Physics, Landau Memorial Conference, Proceeding of the Tel-Aviv Meeting, 1988. Edited by Errol Asher Gotsman, Yuval Ne'eman and Alexander Voronoi, New York Pergamon, 1990 pages 309-340.

**SEE ALSO :**

binom, multim1d, multim2d, smultim1d, smultim2d (C_LAB routines). MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).

## 0.62 sgifs Semi Generalized IFS generation

Author: Khalid Daoudi

This routine generates stochastical Semi-Generalized Iterated Functions Systems (SGIFS) attractors.

**USAGE :**

```
[x, y, Ci]=sgifs(Interp_pts, coefs, nbr_iter,law_type,var)
```

**INPUT PARAMETERS :**

- Interp_pts : Real matrix [n,2] Contains the interpolation points in the format : abscissa-ordinate.
- coefs : Real vector Contains the fundamental contractions ratios.
- nbr_iter : Integer Number of iterations wanted in the generation process of the SGIFS attractor.
- law_type : Character string Specifies the type of law desired. You have the choice between 'uniform' and 'normal'.
- var : Real scalar Rules the variance decrease across scales. At each scale j, the variance would be 1/pow(j,var).

**OUTPUT PARAMETERS :**

- x : Real vector Contains the abscissa of the attractor graph.
- y : Real vector Contains the ordinates of the attractor graph.
- Ci : Real vector Contains all the coefficients of the so generated GIFS.

**DESCRIPTION :**

**PARAMETERS :**

- Interp_pts is a real matrix [n,2] containing the interpolation points.
- coefs is a real vector containing the fundamental contractions ratios, i.e. coefs(1) (resp. coef(2)) would be the mean of the even (resp. odd) ceofficients.
- nbr_iter is the number of iterations wanted in the generation process of the SGIFS attractor.
- law_type is a character string which specifies the type of law desired for the GIFS coefficients.
- var is a real scalar ruling the variance decrease across scales. At each step j of the attractor generation, the variance of the chosen law would be 1/pow(j,var).
- [x,y] contains the resulting attractor.
- Ci is a real vector containing all the coefficients of the so generated GIFS.

**ALGORITHM DETAILS :**

Semi-Generalized Iterated Functions Systems (SGIFS) are a generalization of the usual IFS. This generalization consists in allowing the contarations to change at each step (scale) of the attractor generation process. Here, we use GIFS to construct stocastical SGIFS. More precisely, at each scale j, the GIFS coefficients c_kj, for k even (resp. odd), are a random variable of law law_type, of mean coefs(1) (resp. coefs(2)) and of variance 1/pow(j,var) Moreover, if $\{(t\_i, y\_i), i=1,...,n+1\}$ is the set of interpolation points, then any realisation of the attractor is the graph of a continuous function F such that : F(t_i)=y_i for each i=1,...,n+1.

**SEE ALSO: :**

fif, alphagifs and prescrib

**EXAMPLE: :**

```
I = [0 0
    .5 1
     1 0];
coefs = [.3 -.9];
[x,y,Ci] = sgifs(I,coefs,10,'uniform',1);
plot(x,y)
```

## 0.63 sim_stable Generation of stable random processes

Author: Lotfi Belkacem

This routine generates a stable random process and its increments using the Chambers, Mallows and Stuck (1976) algorithm.

**USAGE :**

```
[proc,inc]=sim_stable(alpha,beta,mu,gamma,size)
```

**INPUT PARAMETERS :**

- alpha : real positive scalar between 0 and 2. This parameter is often referred to as the characteristic exponent.
- beta : real scalar between -1 and +1. This parameter is often referred to as the skewness parameter.
- mu : real scalar This parameter is often referred to as the location parameter. It is equal to the expectation when alpha is greater than 1.
- gamma : real positive scalar. This parameter is often referred to as the scale parameter. It is equal to the standard deviation over two squared when alpha equal 2.
- size : integer positive scalar. size of the simulated sample.

**OUTPUT PARAMETERS :**

- proc : real vector [size,1] corresponding to the stable random process.
- inc : real vector [size,1] corresponding to the increments of the simulated process.

**EXAMPLES :**

```
//Example 1
//generates a standard stable random process with alpha=2, beta=0
//(symmetric), mu=0 and gamma=1.4142 which coinside with a standard
//gaussian process (Brownian motion). To visualize the process or
//the increments use  plot(proc) or plot(inc).

[proc,inc]=sim_stable(2,0,0,1.4142136,5000);

//Example 2 generates a standard 1.5-stable motion
[proc,inc]=sim_stable(1.5,0,0,1,5000);
```

## 0.64 smultim1d multinomial 1d measure synthesis -

Author: Christophe Canus

This C_LAB routine synthesizes two types of pre-multifractal stochastic measures related to the multinomial 1d measure (uniform law and lognormal law) and computes linked multifractal spectrum.

**USAGE :**

```
[varargout,[optvarargout]]=sbinom(b,str,varargin,[optvarargin])
```

**INPUT PARAMETERS :**

- b : strictly positive real (integer) scalar Contains the base of the multinomial.
- str : string Contains the type of ouput.
- varargin : variable input argument Contains the variable input argument.
- optvarargin : optional variable input arguments Contains optional variable input arguments.

**OUTPUT PARAMETERS :**

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

**DESCRIPTION :**

**PARAMETERS :**
The stochastic multinomial 1d measure is completly characterized by its base b. This first parameter must be $>1$.

The second parameter str is a variable string used to determine the desired type of output. There are four suffix strings ('meas' for measure, 'cdf' for cumulative distribution function q, 'pdf' for probability density function, 'spec' for multifractal spectrum) and a two prefix strings for the type of stochastic measure ('unif' for uniform and 'logn' for lognormal) which must added to the first ones to form composed. For example, 'unifmeas' is for the synthesis of a uniform law multinomial 1d pre-multifractal measure and 'lognspec' is for the computation of the multifractal spectrum of a lognormal multinomial 1d measure. When a string containing suffix string 'meas' is given as second input, a pre-multifractal measure mu_n (first output argument) is synthesized on the b-adic intervals I_n (second optional output argument) of the unit interval. In that case, the third input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real vectors mu_n (and I_n if used) is equal to bn (so be aware the stack size ;-)). This option is implemented for the uniform law ('unifmeas') and the lognormal law ('lognmeas') multinomial 1d measures. When a string containing prefix 'unif' is given as second input, the synthesis or the computation is made for a uniform law multinomial 1d measure. In that case, the optional fourth input argument is a strictly positive real scalar epsilon which contains the pertubation around weight .5. The weight is an independant random variable identically distributed between epsilon and 1-epsilon which must be $>0.$, $<1$. The default value for epsilon is 0. When a string containing prefix 'logn' is given as second input, the synthesis or the computation is made for a lognormal law multinomial 1d measure. In that case, the optional fourth input argument is a strictly positive real scalar sigma which contains the standard deviation of the lognormal law. When replacing suffix string 'meas' with suffix string 'cdf', respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument). When a string containing suffix string 'spec' is given as second input, the multifractal spectrum f_alpha (second output argument) is synthesized on the Hoelder exponents alpha (first output argument). In that case, the third input argument is a strictly positive real (integer) scalar N which contains the number of Hoelder exponents. The size of both output real vectors alpha and f_alpha is equal to N. This option is implemented only for the lognormal law ('lognspec') multinomial 1d measures.

**ALGORITHM DETAILS :**

For the uniform and lognormal law multinomial 1d, the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). Note that the lognormal law multinomial 1d measure is not conservative.

**EXAMPLES :**

```
n=10;
// synthesizes a pre-multifractal uniform Law multinomial 1d measure
[mu_n,I_n]=smultim1d(b,'unifmeas',n);
plot(I_n,mu_n);
s=1.;
// synthesizes the cdf of a pre-multifractal lognormal law multinomial 1d
measure
F_n=smultim1d(b,'logncdf',n,s);
plot(I_n,F_n);
e=.19;
// synthesizes the pdf of a pre-multifractal uniform law multinomial 1d measure
p_n=smultim1d(b,'unifpdf',n,e);
plot(I_n,p_n);
N=200;
// computes the multifractal spectrum of the lognormal law multinomial 1d
measure
[alpha,f_alpha]=smultim1d(b,'lognspec',N,s);
plot(alpha,f_alpha);
```

**REFERENCES :**

"A class of Multinomial Multifractal Measures with negative (latent) values for the "Dimension" f(alpha)", Benoit B. MandelBrot. In Fractals' Physical Origins and Properties, Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum Press, New York, 1989 pages 3-29. "Limit Lognormal Multifractal Measures", Benoit B. MandelBrot. In Frontiers of Physics, Landau Memorial Conference, Proceeding of the Tel-Aviv Meeting, 1988. Edited by Errol Asher Gotsman, Yuval Ne'eman and Alexander Voronoi, New York Pergamon, 1990 pages 309-340.

**SEE ALSO :**

binom, sbinom, multim1d, multim2d, smultim2d (C_LAB routines). MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).

## 0.65 smultim2d _____ multinomial 2d measure synthesis -

Author: Christophe Canus

This C_LAB routine synthesizes two types of pre-multifractal stochastic measures related to the multinomial 2d measure (uniform law and lognormal law).

**USAGE :**

```
[varargout,[optvarargout]]=sbinom(bx,by,str,varargin,[optvarargin])
```

**INPUT PARAMETERS :**

- bx : strictly positive real (integer) scalar Contains the abscissa base of the multinomial.
- by : strictly positive real (integer) scalar Contains the ordonate base of the multinomial.
- str : string Contains the type of ouput.
- varargin : variable input argument Contains the variable input argument.
- optvarargin : optional variable input arguments Contains optional variable input arguments.

**OUTPUT PARAMETERS :**

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

**DESCRIPTION :**

**PARAMETERS :**

The stochastic multinomial 2d measure is completly characterized by its abscissa base bx, ordonate base by. These first two parameters must be $>1$.

The third parameter str is a variable string used to determine the desired type of output. There are four suffix strings ('meas' for measure, 'cdf' for cumulative distribution function q, 'pdf' for probability density function, 'spec' for multifractal spectrum) and a two prefix strings for the type of stochastic measure ('unif' for uniform and 'logn' for lognormal) which must added to the first ones to form composed. For example, 'unifmeas' is for the synthesis of a uniform law multinomial 2d pre-multifractal measure and 'lognspec' is for the computation of the multifractal spectrum of a lognormal multinomial 2d measure. When a string containing suffix string 'meas' is given as third input, a pre-multifractal measure mu_n (first output argument) is synthesized on the bx-adic and by-adic intervals I_nx and I_ny (second and third optional output argument) of the unit square. In that case, the fourth input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real matrix mu_n is equal to bxn*byn and the one of the output real vectors I_nx and I_ny (if used) is equal to bxn and byn (so be aware the stack size ;-)). This option is implemented for the uniform law ('unifmeas') and the lognormal law ('lognmeas') multinomial 2d measures. When a string containing prefix 'unif' is given as third input, the synthesis or the computation is made for a uniform law multinomial 2d measure. In that case, the optional fourth input argument is a strictly positive real scalar epsilon which contains the pertubation around weight .5. The weight is an independant random variable identically distributed between epsilon and 1-epsilon which must be $>0.$, $<1$. The default value for epsilon is 0. When a string containing prefix 'logn' is given as third input, the synthesis or the computation is made for a lognormal law multinomial 2d measure. In that case, the optional fifth input argument is a strictly positive real scalar sigma which contains the standard deviation of the lognormal law. When replacing suffix string 'meas' with suffix string 'cdf', respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument).

**ALGORITHM DETAILS :**

For the uniform and lognormal law multinomial 2d, the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). Note that the lognormal law multinomial 2d measure is not conservative.

**EXAMPLES :**

```
n=5;
bx=2;
by=3;
// synthesizes a pre-multifractal uniform Law multinomial 2d measure
[mu_n,I_nx,I_ny]=smultim2d(bx,by,'unifmeas',n);
mesh(I_nx,I_ny,mu_n);
s=1.;
// synthesizes the cdf of a pre-multifractal lognormal law multinomial 2d
measure
F_n=smultim2d(bx,by,'logncdf',n,s);
mesh(I_nx,I_ny,F_n);
e=.19;
// synthesizes the pdf of a pre-multifractal uniform law multinomial 2d measure
p_n=smultim2d(bx,by,'unifpdf',n,e);
mesh(I_nx,I_ny,p_n);
```

**REFERENCES :**

"A class of Multinomial Multifractal Measures with negative (latent) values for the "Dimension" f(alpha)", Benoit B. MandelBrot. In Fractals' Physical Origins and Properties, Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum Press, New York, 1989 pages 3-29. "Limit Lognormal Multifractal Measures", Benoit B. MandelBrot. In Frontiers of Physics, Landau Memorial Conference, Proceeding of the Tel-Aviv Meeting, 1988. Edited by Errol Asher Gotsman, Yuval Ne'eman and Alexander Voronoi, New York Pergamon, 1990 pages 309-340.

**SEE ALSO :**
binom, sbinom, multim1d, multim2d, smultim1d (C_LAB routines). MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).

## 0.66 sortup ____ Sorts the elements of an array in increasing order

Author: Paulo Goncalves

Sorts the elements of an array in increasing order

**USAGE :**

```
[yup,kup] = sortup(x[,how])
```

**INPUT PARAMETERS :**

- x : Real valued array [rx,cx]
- how : option argument '*' : x is treated as x(:). sortup returns a [rx,cx] array 'c' : x is treated column-wise. sortup returns a [rx,cx] array which columns are sorted in increasing order 'r' : x is treated in row. sortup returns a [rx,cx] array which rows are sorted in increasing order Default value is '*'

**OUTPUT PARAMETERS :**

- yup : Real matrix [rx,cx] Sorted elements of x
- kup : Integer matrix [rx,cx] Indices of the sorted elements of x

**SEE ALSO: :**
sort

**EXAMPLE: :**

```
[y,x] = sort(rand(4,4)) ;
x
xSortAll = sortup(x,'*')
xSortCol = sortup(x,'c')
xSOrtRow = sortup(x,'r')
```

## 0.67 stable_cov Covariation of two jointly symmetric Alpha-Stable random variables

Author: Lotfi Belkacem

This routine estimates the covariation of two jointly symmetric alpha-stable random variables.

**USAGE :**

```
[cov]=stable_cov(data1,data2)
```

**INPUT PARAMETERS :**

- data1 : real vector [size,1] corresponding to the the first data sample.
- data2 : real vector [size,1] corresponding to the second data sample.

**OUTPUT PARAMETERS :**

- sm : real scalar corresponding to the estimation the covariation of data1 on data2.

**DESCRIPTION :**
The covariation of two jointly symmetric alpha stable random variables is defined only for alpha between 1 and 2. It designed to replace the covariance when the latter is not defined (alpha<2). Unfortunately, it lacks some of the desirable properties of the covariance (not symmetric, ...). It is however, a useful quantity and appears naturally in many settings, for example, in the context of linear regression..SH Example for two given signals S1 and S2, cov=stable_cov(S1,S2); estimates the covariation of S1 on S2.

## 0.68 stable_sm ____ Spectral measure of a bivariate Stable random vector

Author: Lotfi Belkacem

This routine estimates a normalized spectral measure of a bivariate stable random vector.

**USAGE :**

```
[theta,sm]=stable_sm(data1,data2)
```

**INPUT PARAMETERS :**

- data1 : real vector [size,1] corresponding to the the first data sample.
- data2 : real vector [size,1] corresponding to the second data sample.

**OUTPUT PARAMETERS :**

- theta : real vector corresponding to the the input argument of the estimated spectral measure. Components of the vector theta are varying between 0 and 2*PI.
- sm : real vector corresponding to the estimation of the normalized spectral measure of the bivariate vector (data1,data2).

**EXAMPLE :**
for two given signals S1 and S2, [theta,sm]=stable_sm(S1,S2); estimates the normalized spectral measure of the data vector (S1,S2). To visualize it use plot(theta,sm).

# 0.69 stable_test — stable law conformicy test

Author: Lotfi Belkacem

This routine tests the stability property of a signal.

**USAGE :**

```
[param,sd_param]=stable_test(maxr,data)
```

**INPUT PARAMETERS :**

- maxr : integer positive scalar. maximum resolution witch depend on the size of the sample.
- data : real vector [size,1] corresponding to the data sample (increments of the signal).

**OUTPUT PARAMETERS :**

- param : real matrix [maxr,4] corresponding to the four estimated parameters of the fited stable law at each level of resolution. param(i,:), for i=1, ...maxr, gives respectively alpha(characteristic exponent), beta (skewness parameter), mu (location parameter), gamma (scale parameter) estimated at the resolution i.
- sd_param : real matrix [maxr,4] corresponding to the estimated standard deviations of the four previous parameters at each level of resolution. sd_param(i,:), for i=1, ...maxr, gives respectively standard deviation of alpha, beta, mu and gamma estimated at the resolution i.

**DESCRIPTION :**
The stability test consists on estimating parameters of a fited alpha-satble law at different level of resolution. the variable is said to be stable if the charateristic exponent alpha remains approximatively constant at different resolution, and the scale parameter follows a scaling law with exponent (1/alpha)-1. .SH Example

```
[proc1_5,inc1_5]=sim_stable(1.5,0,0,1,20000);
[param,sd_param]=stable_test(7,inc1_5);
alpha=param(:,1);
m=(1:7)';
lnm=log(m);
plot2d(m,alpha,1,'111','alpha',[1,0,7,2]);
gamma=param(:,4);
lngamma=log(gamma);
plot(lnm,lngamma);
[a,b,sig]=reglin(lnm',lngamma');
slope=a
th_slope=1/1.5-1
```

- we generate a standard 1.5-stable motion and its increments.
- we test the stability property of the previous simutated 1.5-stable random variable "inc1_5" at 7 resolutions.
- we list estimated alpha at different scales.
- we visualize the stability of the shape parameter alpha.
- we list estimated gamma at different scales.
- we visualize the scaling law of the scale parameter gamma with a log-log plot in the space (scale,scale parameter).

- we compute the slope ”a” of the fited line which will be compared to (1/alpha-1).

:

## 0.70 strfbm — Structure Function of a Brownian Field

Author: B. Pesquet-Popescu (ENS-Cachan)

Creates the structure function of an isotropic fBm

**USAGE :**

```
[Y] = strfbm(x,y,H)
```

**INPUT PARAMETERS :**

- x : Real vector [1,N] vertical coordinate
- y : Real scalar [1,M] horizontal coordinate
- H : Real in [0,1] Hurst parameter

**OUTPUT PARAMETERS :**

- Y : Matrix [N,M] Matrix containing the values of the structure function

**SEE ALSO: :**
synth2

**EXAMPLE: :**

```
x = 1:128 ;
y = 1:128 ;
[Y] = strfbm(x,y,0.8) ;
```

## 0.71 symcori — Symmetrization of a periodic 2D correlation field

Author: B. Pesquet-Popescu (ENS-Cachan)

Symmetrization of a periodic 2D correlation field

**USAGE :**

```
Ss = symcori(S) ;
```

**INPUT PARAMETERS :**

- S : Matrix [N/2+1 , N] Periodic 2D correlation field S(1:N/2+1,1:N) of a complex 2D NxN field. Values of S(1,N/2+2:N) may be arbitrary.

**OUTPUT PARAMETERS :**

- Ss : Matrix [N , N] Symetrized correlation field

**SEE ALSO: :**
synth2, strfbm

## 0.72 synth2 — Stationary Increments 2D Process

Author: B. Pesquet-Popescu (ENS-Cachan)

Incremental Fourier synthesis method for processes with stationary increments of order (0,1) and (1,0)

**USAGE :**

```
[B] = synth2(M,H,core)
```

**INPUT PARAMETERS :**

- M : Positive integer Vertical/Horizontal dimension of the generated field
- H : Real in [0,1] parameter of the structure function (e.g. : Hurst parameter)
- core : string Name of the structure function of type core(x,y,H) with x,y : vertical/horizontal coordinates

**OUTPUT PARAMETERS :**

- B : real matrix [N,N] Synthesized random field

**REFERENCES :**
L.M. Kaplan, C.C. J Kuo : IEEE Tran. on IP, May 1996 (extended version).

**SEE ALSO: :**
fbmlevinson, fbmcwt, fbmfwt, mbmlevinson

**EXAMPLE: :**

```
[B] = synth2(128,0.8,'strfbm') ;
viewmat(B)
```

## 0.73 viewWTLM — Vizualises the local maxima lines of a CWT

Author: Paulo Goncalves

Displays the local maxima of a continuous wavelet transform

**USAGE :**

```
viewWTLM(maxmap[,scale,wt])
```

**INPUT PARAMETERS :**

- maxmap : 0/1 matrix [N_scale,N] Indicator matrix of the local wavelet coefficients maxima
- scale : real vector [1,N_scale] Analyzed scale vector
- wt : Complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of FWT or contwt)

**SEE ALSO: :**
findWTLM, viewmat, contwt, cwt

**EXAMPLE: :**

```
N = 2048 ; H = 0.3 ; Q = linspace(-4,4,11) ;
[x] = fbmlevinson(N,H) ;
[wt,scale] = cwt(x,2^(-6),2^(-1),36,0) ;
[maxmap] = findWTLM(wt,scale)

viewWTLM(maxmap,scale,wt) ,
```

## 0.74 viewmat — Vizualisation of a matrix

**USAGE :**

```
viewmat(Z [,X,Y])
```

**INPUT PARAMETERS :**

- Z : Real valued matrix [ny,nx] 2-D matrix to be displayed
- X : Real vector [1,nx] or [nx,1] x-axis
- Y : Real vector [1,ny] or [ny,1] Controls the vertical axis. y forces the vertical axis to be numbered from bottom to top in the increasing order. When not specified, the coordinate system is set to its ”Cartesian” axes mode. The coordinate system origin is at the lower left corner. The x axis is horizontal and is numbered from left to right. The y axis is vertical and is numbered from bottom to top.
- type = 0 : image
- type = 1 : pseudo color
- type = 2 : contour plot
- type = 3 : mesh plot
- type = 4 : shaded surface with lighting
- scale = 0 : linear dynamic
- scale = 1 : logarithmic dynamic
- level : scalar setting the minimum level of the display 0 < level < +1 for linear scale 0 dB < level < Infty dB for logarithmic scale

  Scilab version: cmd is ineffective and frozen to [1 0 0] .

**REMARK :**
viewmat changes the color map

**SEE ALSO: :**
plot3d, grayplot

**EXAMPLE: :**

```
//Signal synthesis:
x = oscillsing(1,1,0,128) ;
X = x(:)*x(:)' ;
//Matrix vizualisation:

viewmat(abs(X))
```

AUTHOR : Author: Paulo Goncalves - Bertrand Guiheneuf

## 0.75 wave2gifs — Computation of IFS coef. with Discrete Wavelet coefficients

Author: Khalid Daoudi

Computes the GIFS cefficients of a 1-D real signal as the ratio between (synchrounous) wavelets coefficients at successive scales. You have to compute the wavelet coefficients of the given signal (using FWT) before using wave2gifs.

**USAGE :**

```
[Ci,Ci_idx,Ci_lg,pc0,pc_ab]=wave2gifs(wt,wt_idx,wt_lg, [M0,a,b])
```

**INPUT PARAMETERS :**

- wt : Real matrix [1,n] Contains the wavelet coefficients (obtained using FWT).
- wt_idx : Real matrix [1,n] Contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces (obtained also using FWT).
- wt_lg : Real matrix [1,n] Contains the dimension of each projection (obtained also using FWT).
- M0 : Real positive scalar If specified, each GIFS coefficient whose absolute value belong to ]1,M0[ will be replaced by 0.99 (keeping its signe).
- a,b : Real positive scalars The routine gives the percentage of the Ci's whose absolute value belong to ]a,b[ (if not specified, ]a,b[=]0,2[).

**OUTPUT PARAMETERS :**

- Ci : Real matrix Contains the GIFS coefficients plus other informations.
- Ci_idx : Real matrix Contains the the indexes of the first Ci at each scale (the finest scale is 1).
- Ci_lg : Real matrix Contains the length of Ci's at each scale.
- pc0 : Real scalar Gives the percentage of vanishing Ci's
- pc_ab : Real scalar Gives the percentage of Ci's which belong to ]a,b[

**DESCRIPTION :**

**PARAMETERS :**

- wt is a real matrix which is a structure containg the wavelet coefficients and other informations. It is obtained using FWT.

- wt_idx is a real vector which contains the indexes (in wt) of the first wavelet coefficient a each scale. For instance, wt(wt_idx(1) : wt_idx(2)-1) is a vector containing the wavelet coefficients a the finest scale.
- wt_lg is a real vector which contains the length of wavelet coefficients a each scale. For instance, wt_lg(1) is the number of the wavelet coefficients a the finest scale.
- M0 is a real positive scalar such that each GIFS coefficient (ci) whose absolute value belong to ]1,M0[ will be replaced by 0.99*signe(ci).
- a and b are two real positive scalars. The routine gives the percentage of the Ci's whose absolute values belong to ]a,b[ (if not specified, ]a,b[=]0,2[).
- Ci is a real matrix which contains the GIFS coefficients, the size of the signal in Ci(lenght(Ci)) and the number of scales used in the wavelet decomposition in Ci(lenght(Ci)-1).
- Ci_idx is a real matrix which ontains the the indexes of the first Ci at each scale. For instance, Ci(Ci_idx(j) : Ci_idx(j)+ Ci_lg(j) - 1) is a vector containing the GIFS coefficients at scale j (the finest scale is j=1).
- Ci_lg is a real vector which contains the length of GIFS coefficients a each scale. For instance, Ci_lg(1) is the number of the wavelet coefficients a the finest scale.
- pc0 is a real scalar which gives the percentage of vanishing GIFS coefficients.
- pc_ab is a real scalar which gives the percentage of GIFS coefficients which belong to ]a,b[.

**ALGORITHM DETAILS :**

**SEE ALSO: :**

FWT and MakeQMF.

**EXAMPLE: :**

# Index

V



W