Vimu Signal Process Library User Guide 1.0

Note:

1,This digital signal processing library, written in the standard C language, is not better optimized for any platform.

January 17, 2022

Update History:

20220328 Added filter section description

1. Window Function

In order to reduce spectral energy leakage, the signal can be truncated using different interception functions, which are called window functions, referred to as windows. The following are the more common window functions.

N is the length of the window, and hn is the window coefficient after the calculation is completed.

1.1. Rectangle

void rectangle(int32_t N, double *hn);

1.2. Bartlett

void bartlett(int32_t N, double *hn);

1.3. Triangular

Unlike bartlett, the start and end points are not zeros

void triangular(int32_t N, double *hn);

1.4. Cosine

void cosine(int32_t N, double *hn);

1.5. Hanning

void hanning(int32_t N, double *hn);

1.6. Bartlett Hanning

void bartlett_hanning(int32_t N, double *hn);

1.7. Hamming

void hamming(int32_t N, double *hn);

1.8. Blackman

void blackman(int32_t N, double *hn);

1.9. Blackman–Harris

void blackman_Harris(int32_t N, double *hn);

1.10. **Tukey**

alpha: Range [0,1]

Return 0 failed, 1success

char tukey(int32_t N, double *hn, double alpha);

1.11. Nuttall

void Nuttall(int32_t N, double *hn);

1.12. FlatTop

void FlatTop(int32_t N, double *hn);

1.13. Bohman

void Bohman(int32_t N, double *hn);

1.14. Parzen

void Parzen(int32_t N, double *hn);

1.15. Lanczos

void Lanczos(int32_t N, double *hn);

1.16. Kaiser

beta: Range [0,700]

void kaiser(int32_t N, double *hn, double beta);

1.17. Gauss

alpha Range[2,50]

Return 0 failed, 1success

char gauss(int32_t N, double *hn, double alpha);

1.18. Dolph Chebyshev

at: Range[1,300]

1.18.1

N: Must an even number

char dolph_chebyshev_fft(int32_t N, double *hn, double at);

1.18.2

dolph_chebyshev The function uses ifht, so N is odd, and when N is large, it will be slow.

void dolph_chebyshev(int32_t N, double *hn, double at);

1.19. The range of parameters for the window function

Gets the range of arguments for tukey, kaiser, gauss, and dolph_chebyshev.

void getpararange(WINDOW_STYLE window, double *min, double *max);

1.20. Calculate Window Functions based on window type and window parameters

type: window type

N: window length

alpha_beta: window parameters

Normalize: 1：Normalization 0：Non-normalization

char window(WINDOW_STYLE type, int32_t N, double *hn, double alpha_beta, char normalize);

1.21. Add Window for Data

type: window type

x: the data for add window

N: window length

alpha_beta: window parameters

Normalize: 1：Normalization 0：Non-normalization

return the coherent gain of the window function

double window_i(WINDOW_STYLE type, double *x, int32_t N, double alpha_beta, char normalize);

2. FFT

2.1 FFT

x_Re：real part array

x_Im：imaginary part array

N： FFT/ array length(Should be an integer multiple of 2)

flag: 0 Base 2 fft

1 Base 4 fft

2 Split-radix fft

3 If N is an integer power of 4, base 4 fft is used; otherwise, split base fft is used. (recommend)

void fft(double *x_Re,double *x_Im, uint32_t N, char flag);

2.2 IFFT

x_Re：real part array

x_Im：imaginary part array

N： FFT/ array length(Should be an integer multiple of 2)

flag: 0 Base 2 fft

1 Base 4 fft

2 Split-radix fft

3 If N is an integer power of 4, base 4 fft is used; otherwise, split base fft is used. (recommend)

void ifft(double *x_Re,double *x_Im, uint32_t N, char flag);

2.3 Cycle Signal FFT

FFT calculation of cycle signal, the result of FFT need to divide N； Xa(k)=X(k)/N

x_Re：real part array

x_Im：imaginary part array

N： FFT/ array length(Should be an integer multiple of 2)

flag: 0 Base 2 fft

1 Base 4 fft

2 Split-radix fft

3 If N is an integer power of 4, base 4 fft is used; otherwise, split base fft is used. (recommend)

void fft_signal(double *x_Re,double *x_Im, uint32_t N,uint32_t Real_N,char flag);

2.4 FFT Amplitude

x_Re：real part array

x_Im：imaginary part array

af：amplitude array

N：FFT/ array length(Should be an integer multiple of 2)

rms：1,RMS method 0, Amplitude method

void Fft_Amplitude (double *x_Re,double *x_Im,double *af, uint32_t N, unsigned char rms);

2.5 FFT Amplitude Unit Conversion

Convert unit V to dBV,dBmV,dBmW.

af：amplitude array

len：array length

R：the ohms of calculate dbw

void Fft_Amplitude_Conver_Type(double* af, uint32_t len, double R,FFT_Amplitude_Ref_Type type);

2.6 FFT Phase

x_Re：real part array

x_Im：imaginary part array

af：amplitude array(Call Fft_Amplitude first and then call Fft_Phase)

pf：Phase array

N：array length

ref_type: Phase Unit:　degree：FFT_Phase_Ref_Deg；radian：FFT_Phase_Ref_Rad

void Fft_Phase (double *x_Re, double *x_Im, double* af, double *pf, uint32_t N, FFT_Phase_Ref_Type ref_type);

2.7 Real Sequence FFT

x_Re：real part array

x_Im：imaginary part array(Input all is 0)

N：FFT/ array length(Should be an integer multiple of 2)

flag: 0　Base 2 fft

1　Base 4 fft

2　Split-radix fft

3　If N is an integer power of 4, base 4 fft is used; otherwise, split base fft is used. (recommend)

void real_fft(double *x_Re,double *x_Im,uint32_t N,char flag);

2.8 Cycle Real Sequence Signal FFT

FFT calculation of cycle signal, the result of FFT need to divide N；Xa(k)=X(k)/N

x_Re：real part array

x_Im：imaginary part array

N：　FFT/ array length(Should be an integer multiple of 2)

flag: 0　Base 2 fft

1　Base 4 fft

2　Split-radix fft

3　If N is an integer power of 4, base 4 fft is used; otherwise, split base fft is used. (recommend)

void real_fft_signal(double *x_Re, double *x_Im, uint32_t N, uint32_t Real_N, char flag);

3. FIR Filter

3.1 Base Filter

According to the basic filter definition，Perform filtering

hn：h Array of FIR Filter

N：The length of FIR Filter

date：The processes data by filter

num：The length of processes data

void filters(double *hn, int32_t N, int32_t *date, int32_t num);

void filters(double *hn, int32_t N, double *date, int32_t num);

3.2 FHT Fast filtering algorithm

Use FHT algorithm to accelerate the processing speed of filtered data;

In addition to using the FHT algorithm, filters_overlap_add_fht also uses overlapping addition, which is useful when the length of the date is much larger than the length of hn.

hn：h Array of FIR Filter

N：The length of FIR Filter

date：The processes data by filter

num：The length of processes data

fht_n：The length of overlapping addition(Should be an integer multiple of 2). Generally, the fht_n is more than twice the length of the short sequence hn

void filters_fht(double *hn, int32_t N, int32_t *date, int32_t fht_n);

void filters_fht(double *hn, int32_t N, double *date, int32_t fht_n);

void filters_overlap_add_fht(double *hn, int32_t N, int32_t *date, int32_t num, int32_t fht_n);

void filters_overlap_add_fht(double *hn, int32_t N, double *date, int32_t num, int32_t fht_n);

3.3 FFT Fast filtering algorithm

Use FFT algorithm to accelerate the processing speed of filtered data;

In addition to using the FFT algorithm, filters_overlap_add_fht also uses overlapping addition, which is useful when the length of the date is much larger than the length of hn.

hn：h Array of FIR Filter

N：The length of FIR Filter

date：The processes data by filter

num：The length of processes data

fht_n：The length of overlapping addition(Should be an integer multiple of 2). Generally, the fht_n is more than twice the length of the short sequence hn

void filters_fft(double *hn, int32_t N, int32_t *date, int32_t fft_n);

void filters_fft(double *hn, int32_t N, double *date, int32_t fft_n);

void filters_overlap_add_fft(double *hn, int32_t N, int32_t *date, int32_t num, int32_t fft_n);

void filters_overlap_add_fft(double *hn, int32_t N, double *date, int32_t num, int32_t fft_n);

4. IIR Filter

4.1 Base Filter

b: The coefficient of the polynomial of the filter molecule, the length is (m+1), m is the order of the polynomial of the filter molecule;

bn: The length of the polynomial of the filter molecule, bn=m+1

a: The coefficient of the filter denominator polynomial, the length is (n+1), n is the order of the filter denominator polynomial;

an: The length filter denominator polynomial, an=n+1

x: Array, length len. The input sequence of the filter is stored at the beginning, and the output sequence of the filter is stored at the end;

len: Integer variable. The length of the input and output sequences.

void filters(const std::vector<double> b, const std::vector<double> a, std::vector<double>& x, int32_t len);

void filters(double* b, int32_t bn, double* a, int32_t an, int32_t* x, int32_t len);

void filters(double* b, int32_t bn, double* a, int32_t an, double* x, int32_t len);

4.2 FHT Fast filtering algorithm

Use FHT algorithm to accelerate the processing speed of filtered data;

In addition to using the FHT algorithm to speed up filtering, filters_overlap_add_fht also uses overlapping addition, which is useful when the length of the date is much larger than the length of the filter.

b: The coefficient of the polynomial of the filter molecule, the length is (m+1), m is the order of the polynomial of the filter molecule;

bn: The length of the polynomial of the filter molecule, bn=m+1

a: The coefficient of the filter denominator polynomial, the length is (n+1), n is the order of the filter denominator polynomial;

an: The length filter denominator polynomial, an=n+1

x: Array, length len. The input sequence of the filter is stored at the beginning, and the output sequence of the filter is stored at the end;

len: Integer variable. The length of the input and output sequences.

fht_n：The length of overlapping addition(Should be an integer multiple of 2). Generally, the fht_n is more than twice the length of the short sequence b and a.

void filters_overlap_add_fht(double* b, int32_t bn, double* a, int32_t an, int32_t* x, int32_t len, int32_t fht_n);

void filters_overlap_add_fht(double* b, int32_t bn, double* a, int32_t an, double* x, int32_t len, int32_t fht_n);