微目数字信号处理库使用说明书 1.1

说明：

1、该数字信号处理库，使用标准的 C 语言编写的，没有针对任何的平台进行更好的优化。

2022 年 1 月 17 号

更新历史：

20220328 增加滤波器部分说明

1. 窗函数

为了减少频谱能量泄漏，可采用不同的截取函数对信号进行截断，截断函数称为窗函数，简称为窗。下面是比较常见的窗函数计算函数。

N 为窗的长度，hn 为计算完成后的窗系数。

1.1. 矩形窗

void rectangle(int32_t N, double *hn);

1.2. 三角窗

void bartlett(int32_t N, double *hn);

1.3. 三角窗

与 bartlett 不同点是，起点和终点不是 0

void triangular(int32_t N, double *hn);

1.4. 余弦窗

void cosine(int32_t N, double *hn);

1.5. 汉宁窗

void hanning(int32_t N, double *hn);

1.6. 三角汉宁窗

void bartlett_hanning(int32_t N, double *hn);

1.7. 海明窗

void hamming(int32_t N, double *hn);

1.8. 布莱克曼窗

void blackman(int32_t N, double *hn);

1.9. Blackman–Harris

void blackman_Harris(int32_t N, double *hn);

1.10. 图基窗

alpha 的范围[0,1]

返回 0 失败；1 成功

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

1.11. Nuttall 窗

void Nuttall(int32_t N, double *hn);

1.12. 平顶窗

void FlatTop(int32_t N, double *hn);

1.13. Bohman 窗

void Bohman(int32_t N, double *hn);

1.14. Parzen window 窗

void Parzen(int32_t N, double *hn);

1.15. 兰乔斯法窗

void Lanczos(int32_t N, double *hn);

1.16. 凯塞—贝塞尔窗

beta 的范围[0,700]

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

1.17. 高斯窗

alpha 的范围[2,50]

返回 0 失败；1 成功

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

1.18. 道尔夫-切比雪夫窗

at 的范围[1,300]

当盘瓣的最大值固定时，道尔夫-切比雪夫窗可使主瓣幅度为最小；

该窗函数的特点是所有的盘瓣振幅都相等。

1.18.1

N 必须为偶数

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

1.18.2

dolph_chebyshev 该函数使用了 ifft，所以 N 为奇数，并且 N 很大时，速度会很慢。

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

1.19. 窗函数的参数范围

获取 tukey，kaiser，gauss 和 dolph_chebyshev 的参数可取值范围。

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

1.20. 根据窗类型和窗参数计算对应的窗函数

type 窗类型

N 窗长度

alpha_beta 窗参数

normalize 1：归一化 0：不归一化

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

1.21. 加窗处理

type 窗类型

x 加窗的数组

N 窗长度

alpha_beta 窗参数

normalize 1：归一化 0：不归一化

返回窗口函数的相干增益 gain

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

2. FFT

2.1 FFT

x_Re：实部数组

x_Im：虚部数组

N： FFT/数组长度(应该为 2 的整数倍)

flag: 0 基 2 fft

1 基 4 fft

2 分裂基 fft

3 如果 N 是 4 的整数次幂，采用基 4 fft；否则采用分裂基 fft。（建议）

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

2.2 IFFT

x_Re：实部数组

x_Im：虚部数组

N： FFT/数组长度(应该为 2 的整数倍)

flag: 0 基 2 fft

1 基 4 fft

2 分裂基 fft

3 如果 N 是 4 的整数次幂，采用基 4 fft；否则采用分裂基 fft。（建议）

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

2.3 周期信号 FFT

周期信号的 FFT 计算，需要将 FFT 的结果除以 N；Xa(k)=X(k)/N

x_Re：实部数组

x_Im：虚部数组

N： FFT 长度(应该为 2 的整数倍)

Real_N： 未填充 0 部分的数组长度；如果数组没有填 0，Real_N==N

flag: 0 基 2 fft

1 基 4 fft

2 分裂基 fft

3 如果 N 是 4 的整数次幂，采用基 4 fft；否则采用分裂基 fft。（建议）

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

2.4 FFT 幅频

x_Re：实部数组

x_Im：虚部数组

af：幅频数组

N： FFT/数组长度(应该为 2 的整数倍)

rms：1，有效值方式 0，幅值方式

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

2.5 FFT 幅频单位转换

将单位 V 的数据转换成 dBV，dBmV，dBmW。

af：幅频数组

len： 数组长度

R：计算 dbw 的阻抗电阻

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

2.6 FFT 相频

x_Re：实部数组

x_Im：虚部数组

af：幅频数组（先调用 Fft_Amplitude 再调用 Fft_Phase）

pf：相频数组

N： 数组长度

ref_type: 相频单位 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 实序列 FFT

x_Re：实部数组

x_Im：虚部数组(输入全部为 0)

N： FFT/数组长度(应该为 2 的整数倍)

flag: 0 基 2 fft

1 基 4 fft

2 分裂基 fft

3 如果 N 是 4 的整数次幂，采用基 4 fft；否则采用分裂基 fft。（建议）

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

2.8 周期实序列信号 FFT

周期信号的 FFT 计算，需要将 FFT 的结果除以 N；Xa(k)=X(k)/N

x_Re：实部数组

x_Im：虚部数组

N： FFT 长度(应该为 2 的整数倍)

Real_N： 未填充 0 部分的数组长度；如果数组没有填 0，Real_N==N

flag: 0 基 2 fft

1 基 4 fft

2 分裂基 fft

3 如果 N 是 4 的整数次幂，采用基 4 fft；否则采用分裂基 fft。（建议）

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

3. FIR 滤波

3.1 基本滤波

按照基本的滤波定义，来执行滤波处理

hn：FIR 滤波器 h 数组

N：FIR 滤波器长度 N

date：滤波处理数据

num： 滤波处理数据长度

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 快速滤波算法

使用 FHT 算法，加速滤波数据的处理速度；

除了使用 FHT 算法加速滤波，filters_overlap_add_fht 还使用了重叠相加法，该函数适用 date 的长度比 hn 的长度大很多的情况。

hn：FIR 滤波器 h 数组

N：FIR 滤波器长度 N

date：滤波处理数据

num： 滤波处理数据长度

fht_n：重叠相加(fht_n 必须是 2 的整数次幂),一般选取 fht_n 大于短序列 hn 长度的两倍以上

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 快速滤波算法

使用 FFT 算法，加速滤波数据的处理速度；

除了使用 FFT 算法加速滤波，filters_overlap_add_fft 还使用了重叠相加法，该函数适用 date 的长度比 hn 的长度大很多的情况。

hn：FIR 滤波器 h 数组

N：FIR 滤波器长度 N

date：滤波处理数据

num： 滤波处理数据长度

fft_n：重叠相加(fft_n 必须是 2 的整数次幂),一般选取 fft_n 大于短序列 hn 长度的两倍以上

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 滤波

4.1 基本滤波

b: 滤波器分子多项式的系数,长度为(m+1), m 滤波器分子多项式的阶数；

bn:滤波器分子多项式长度 bn=m+1

a: 滤波器分母多项式的系数,长度为(n+1),n 滤波器分母多项式的阶数；

an:滤波器分母多项式长度 an=n+1

x: 数组，长度为 len。开始时存放滤波器的输入序列，最后存放滤波器的输出序列；

len: 整型变量。输入序列与输出序列的长度。

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 快速滤波算法

使用 FHT 算法，加速滤波数据的处理速度；

除了使用 FHT 算法加速滤波，filters_overlap_add_fht 还使用了重叠相加法，该函数适用 date 的长度比滤波器的长度大很多的情况。

b: 滤波器分子多项式的系数,长度为(m+1), m 滤波器分子多项式的阶数；

bn:滤波器分子多项式长度 bn=m+1

a: 滤波器分母多项式的系数,长度为(n+1),n 滤波器分母多项式的阶数；

an:滤波器分母多项式长度 an=n+1

x: 数组，长度为 len。开始时存放滤波器的输入序列，最后存放滤波器的输出序列；

len: 整型变量。输入序列与输出序列的长度；

fft_n：重叠相加(fft_n 必须是 2 的整数次幂),一般选取 fft_n 大于短序列 hn 长度的两倍以上。

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);