类型说明

1. 结构体类型

1.1 ComplexNative

复数类型，表示一个复数（实部+虚部）。

```
typedef struct {
    double Real;          // 实部
    double Imaginary;     // 虚部
} ComplexNative;
```

- **Real**：复数的实部。
- **Imaginary**：复数的虚部。

1.2 SignalNative

时域信号数据结构体，描述一段采样信号。

```
typedef struct {
    double* Samples;      // 指向信号采样值数组的指针
    int Length;           // 采样点数
    double DeltaTime;     // 采样间隔（秒），等于1/采样率
    int64_t UnixTime;     // 信号起始时间（Unix时间戳，秒）
} SignalNative;
```

- **Samples**：信号数据数组，需动态分配和释放。
- **Length**：采样点总数。
- **DeltaTime**：采样间隔，常用如1/51200。
- **UnixTime**：信号起始时间戳，可用于同步。

1.3 RpmNative

转速数据结构体，描述一组转速随时间变化的数据。

```
typedef struct {
    double* RpmValues;    // 指向转速值数组的指针
    double* TimeValues;   // 指向对应时间点数组的指针
    int Length;           // 数据点数
} RpmNative;
```

- **RpmValues**：转速数据数组，单位rpm。

- **TimeValues**：每个转速点对应的时间（秒）。
- **Length**：数据点数量。

1.4 EdgeDetectorNative

脉冲信号边缘检测参数结构体。

```
typedef struct {
    int MinInterval;      // 最小脉冲间隔（采样点数），用于去除毛刺，默认10
    double offset;        // 电压偏置，默认0.0
    int edgeType;         // 边缘类型：0-上升沿，1-下降沿，2-双边，默认0
} EdgeDetectorNative;
```

- **MinInterval**：去毛刺用，典型值10。
- **offset**：电平偏置，典型为0。
- **edgeType**：0=上升沿，1=下降沿，2=双边。

1.5 RpmCalculationOptionsNative

转速计算参数结构体。

```
typedef struct {
    int PulsesPerRevolution;    // 每转脉冲数（编码器参数）
    int PlusesPerMeasurement;   // 每个RPM值的脉冲数，默认100
    int SkipEdges;              // 跳过的边缘数，默认100
    bool IsResample;            // 是否重采样，默认false
    double ResampleStartTime;   // 重采样起始时间，默认-1.0（用第一个时间）
    double ResampleEndTime;     // 重采样结束时间，默认-1.0（用最后一个时间）
    double ResampleInterval;    // 重采样时间间隔，默认1.0
} RpmCalculationOptionsNative;
```

- **PulsesPerRevolution**：编码器每转输出的脉冲数。
- **PlusesPerMeasurement**：每个RPM值计算用的脉冲数。
- **SkipEdges**：跳过的边缘数。
- **IsResample**：是否对结果重采样。
- **ResampleStartTime/EndTime/Interval**：重采样相关参数。

---

2. 枚举类型

2.1 AverageType

平均类型枚举。

```
typedef enum {
    Energy = 0, // 能量平均
    Mean = 1,   // 算术平均
    Max = 2     // 最大值平均
} AverageType;
```

2.2 WeightType

加权类型枚举。

```
typedef enum {
    None = 0, // 无加权
    A = 1,    // A计权
    B = 2,    // B计权
    C = 3     // C计权
} WeightType;
```

2.3 WindowType

窗函数类型枚举。

```
typedef enum {
    Rectangular = 0, // 矩形窗
    Hanning = 1      // 汉宁窗
} WindowType;
```

2.4 ScaleType

缩放类型枚举。

```
typedef enum {
    Linear = 0, // 线性输出
    Db = 1      // dB输出
} ScaleType;
```

2.6 OctaveType

倍频程类型枚举。

```
typedef enum {
    Octave,              // 1倍频程
    ThirdOctave,         // 1/3倍频程
    SixthOctave,         // 1/6倍频程
    TwelfthOctave,       // 1/12倍频程
    TwentyFourthOctave   // 1/24倍频程
} OctaveType;
```

2.7 EdgeType

边缘采样类型

```
typedef enum {
    Rising = 0, // 上升沿
    Falling = 1,// 下降沿
    Both = 2    // 双边
};
```

API说明

1. OverallLevelSpectral

功能：计算信号的声压总级（可选加权、缩放、窗函数等）。

函数声明：

```
int OverallLevelSpectral(
    SignalNative signalNative,
    int spectrumLines,
    double increment,
    double referenceValue,
    int windowType,
    int weightType,
    int scaleType,
    double** levelData,
    int* timeBins
);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号数据结构体，包含信号采样、长度、采样间隔等。
spectrumLines	int	谱线数，决定频谱分辨率，越大分辨率越高。
increment	double	分析步进，单位为秒。每隔多少秒计算一次声压级。
referenceValue	double	参考值，仅用于dB输出时的参考值（如声压级常用2e-5 Pa）。

参数名	类型	说明
windowType	int	窗函数类型：0-矩形窗，1-Hanning窗。
weightType	int	加权类型：0-A计权，1-B计权，2-C计权。
scaleType	int	缩放类型：0-线性输出，1-dB输出。
levelData	double**	输出参数，指向计算得到的声压级数据的指针。调用后需释放内存。
timeBins	int*	输出参数，指向时间分箱数量的指针。

调用范例：

```
SignalNative signal = {/*...*/};
double* levelData = nullptr;
int timeBins = 0;
int ret = OverallLevelSpectral(
    signal, 4096, 0.1, 2e-5, 1, 0, 1, &levelData, &timeBins
);
if (ret == 1) {
    for (int i = 0; i < timeBins; ++i) {
        printf("Level[%d]=%f\n", i, levelData[i]);
    }
    delete[] levelData;
}
```

2. OrderSection

功能：分析信号在特定阶次下的幅值随转速变化。

函数声明：

```
int OrderSection(
    SignalNative signalNative,
    RpmNative rpmNative,
    int spectrumLines,
    double targetOrder,
    double orderBandwidth,
    double rpmStep,
    double referenceValue,
    int windowType,
    int weightType,
    int scaleType,
    double** outOrderSection,
    double** outRpmPoints,
    int* rpmBins
);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号
rpmNative	RpmNative	输入转速数据
spectrumLines	int	谱线数
targetOrder	double	目标阶次
orderBandwidth	double	阶次带宽
rpmStep	double	转速步进（rpm）
referenceValue	double	dB参考值
windowType	int	窗函数
weightType	int	加权类型
scaleType	int	缩放类型
outOrderSection	double**	输出，阶次幅值数组，需释放
outRpmPoints	double**	输出，转速点数组，需释放
rpmBins	int*	输出，转速分箱数

调用范例：

```
// 假设已准备好 signalNative 和 rpmNative
double* orderSectionData = nullptr;
double* rpmOutData = nullptr;
int rpmBins = 0;
int ret = OrderSection(
    signalNative, rpmNative, 4096, 14.0, 0.5, 25.0, 1.0,
    1, 0, 1, &orderSectionData, &rpmOutData, &rpmBins
);
if (ret == 1) {
    for (int i = 0; i < rpmBins; ++i) {
        printf("RPM: %f, AMP: %f\n", rpmOutData[i], orderSectionData[i]);
    }
    delete[] orderSectionData;
    delete[] rpmOutData;
}
```

3. GetRms

功能：计算信号的均方根值（RMS）。

函数声明：

```
int GetRms(
    SignalNative signalNative,
    int weightType,
    double* outValue
);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号
weightType	int	加权类型
outValue	double*	输出，RMS值

调用范例：

```
double rmsValue = 0.0;
int ret = GetRms(signalNative, 1, &rmsValue);
if (ret == 1) {
    printf("RMS: %f\n", rmsValue);
}
```

4. GetMax

功能：计算信号的最大值。

函数声明：

```
int GetMax(
    SignalNative signalNative,
    double* outValue
);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号
outValue	double*	输出，最大值

调用范例：

```
double maxValue = 0.0;
int ret = GetMax(signalNative, &maxValue);
if (ret == 1) {
    printf("Max: %f\n", maxValue);
}
```

5. AveragedOctaveBandLevels

功能：对信号进行倍频程（Octave/1/3 Octave等）能量分析。

函数声明：

```
int AveragedOctaveBandLevels(
    SignalNative signalNative,
    int spectrumLines,
    double increment,
    double lowerFreq,
    double upperFreq,
    double referenceValue,
    int formatType,
    int octaveType,
    int averageType,
    int windowType,
    int weightType,
    int scaleType,
    double** levelData,
    double** centerFreqs,
    double** lowerFreqs,
    double** upperFreqs,
    int* levelBins
);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号
spectrumLines	int	谱线数
increment	double	分析步进
lowerFreq	double	最低频率
upperFreq	double	最高频率
referenceValue	double	dB参考值
formatType	int	输出格式（0-RMS等）

参数名	类型	说明
octaveType	int	倍频程类型
averageType	int	平均类型
windowType	int	窗函数
weightType	int	加权类型
scaleType	int	缩放类型
levelData	double**	输出，能量数组，需释放
centerFreqs	double**	输出，中心频率数组，需释放
lowerFreqs	double**	输出，下限频率数组，需释放
upperFreqs	double**	输出，上限频率数组，需释放
levelBins	int*	输出，频带数

调用范例：

```
double* levelData = nullptr;
double* centerFreqs = nullptr;
double* lowerFreqs = nullptr;
double* upperFreqs = nullptr;
int levelBins = 0;
int ret = AveragedOctaveBandLevels(
    signalNative, 4096, 0.15, 12, 26000, 2e-5,
    0, 1, 0, 1, 0, 1,
    &levelData, &centerFreqs, &lowerFreqs, &upperFreqs, &levelBins
);
if (ret == 1) {
    for (int i = 0; i < levelBins; ++i) {
        printf("CenterFreq: %f, Level: %f\n", centerFreqs[i], levelData[i]);
    }
    delete[] levelData;
    delete[] centerFreqs;
    delete[] lowerFreqs;
    delete[] upperFreqs;
}
```

6. 阶次谱分析

功能：计算信号的阶次谱。

函数声明：

```
int OrderSpectrum(
    SignalNative signalNative,
    RpmNative rpmNative,
    double maxOrder,
    double orderResolution,
    double oversamplingFactor,
    double lowerRpmThreshold,
    double upperRpmThreshold,
    int rpmStep,
    double referenceValue,
    int formatType,
    int windowType,
    int weightType,
    int scaleType,
    double** spectrumData,
    double** orderAxis,
    int* orderBins);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号数据结构体。
rpmNative	RpmNative	输入转速数据结构体。
maxOrder	double	阶次最大值（如12.0）。
orderResolution	double	阶次分辨率（如0.5）。
oversamplingFactor	double	过采样因子，提升阶次分辨率，典型为1.0。
lowerRpmThreshold	double	转速下限，低于该值的数据不参与分析。
upperRpmThreshold	double	转速上限，高于该值的数据不参与分析。
rpmStep	int	转速步进，单位为rpm。
referenceValue	double	参考值，仅用于dB输出时的参考值。
formatType	int	输出格式类型：0-RMS，1-Peak，2-Peak to Peak。
windowType	int	窗函数类型：0-矩形窗，1-Hanning窗。
weightType	int	加权类型：0-无加权，1-A计权，2-B计权，3-C计权。
scaleType	int	缩放类型：0-线性输出，1-dB输出。
spectrumData	double**	输出参数，阶次谱数据数组，需释放。
orderAxis	double**	输出参数，阶次坐标数组，需释放。
orderBins	int*	输出参数，阶次数量。

调用范例：

```
double* spectrumData = nullptr;
double* orderAxis = nullptr;
int orderBins = 0;
int ret = OrderSpectrum(
    signalNative,
    rpmNative,
    12.0, // maxOrder
    0.5, // orderResolution
    1.0, // oversamplingFactor
    0.0, // lowerRpmThreshold
    -1.0, // upperRpmThreshold
    25, // rpmStep
    1.0, // referenceValue
    0, // FormatTypeRMS
    1, // WindowTypeHanning
    0, // WeightTypeNone
    1, // ScaleTypeDb
    &spectrumData,
    &orderAxis,
    &orderBins);
if (ret == 1) {
    for (int i = 0; i < orderBins; ++i) {
        printf("Order: %f, Value: %f\n", orderAxis[i], spectrumData[i]);
    }
    delete[] spectrumData;
    delete[] orderAxis;
 }
```

7. AveragedSpectrumByIncrement

功能：分段计算信号的平均谱（可选格式、平均方式、加权、窗函数等）。

函数声明：

```
int AveragedSpectrumByIncrement(
    SignalNative signalNative,
    int spectrumLines,
    double increment,
    int formatType,
    int averageType,
    int weightType,
    int windowType,
    double** outSpectrum,
    int* outLength
);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号数据结构体，包含信号采样、长度、采样间隔等。
spectrumLines	int	谱线数，决定频谱分辨率，越大分辨率越高。
increment	double	分析步进，单位为秒。每隔多少秒计算一次平均谱。
formatType	int	输出格式类型：0-RMS，1-Peak，2-Peak to Peak。
averageType	int	平均类型：0-能量平均，1-算术平均，2-最大值平均。
weightType	int	加权类型：0-无加权，1-A计权，2-B计权，3-C计权。
windowType	int	窗函数类型：0-矩形窗，1-Hanning窗。
outSpectrum	double**	输出参数，指向计算得到的平均谱数据的指针。调用后需释放内存。
outLength	int*	输出参数，指向平均谱长度的指针。

调用范例：

```
SignalNative signal = {/*...*/};
double* spectrumResult = nullptr;
int outLength = 0;
int ret = AveragedSpectrumByIncrement(
    signal,
    4096,
    0.15,
    0, // FormatType::RMS
    0, // AverageType::Energy
    1, // WeightType::A
    1, // WindowType::Hanning
    &spectrumResult,
    &outLength
);
if (ret == 1) {
    for (int i = 0; i < outLength; ++i) {
        printf("Spectrum[%d]=%f\n", i, spectrumResult[i]);
    }
    delete[] spectrumResult;
}
```

8. GetEnvelope

功能：计算信号的包络线。

函数声明：

```
int GetEnvelope(
    SignalNative signalNative,
    double** outEnvelope,
    int* outLength
);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号
outEnvelope	double**	输出，包络谱数组，需释放
outLength	int*	输出，包络谱长度

调用范例：

```
double* envelope = nullptr;
int envelopeLen = 0;
int ret = GetEnvelope(signalNative, &envelope, &envelopeLen);
if (ret == 1) {
    for (int i = 0; i < envelopeLen; ++i) {
        printf("Envelope[%d]=%f\n", i, envelope[i]);
    }
    delete[] envelope;
}
```

9. GenerateTimeFrequencyColormapByIncrement

功能：生成信号的时间-频率色谱图（时频谱），用于分析信号在不同时间段的频谱分布。

函数声明：

```
int GenerateTimeFrequencyColormapByIncrement(
    SignalNative signalNative,
    int spectrumLines,
    double increment,
    double startTime,
    double endTime,
    double referenceValue,
    int formatType,
    int windowType,
    int weightType,
    int scaleType,
    double** colormapData,
    int* timeBins,
    int* frequencyBins
```

```
);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号数据结构体，包含信号采样、长度、采样间隔等。
spectrumLines	int	谱线数，决定频谱分辨率，越大分辨率越高。
increment	double	分析步进，单位为秒。每隔多少秒计算一次频谱。
startTime	double	分析起始时间，单位为秒。
endTime	double	分析结束时间，单位为秒，-1.0表示到信号末尾。
referenceValue	double	参考值，仅用于dB输出时的参考值。
formatType	int	输出格式类型：0-RMS，1-Peak，2-Peak to Peak。
windowType	int	窗函数类型：0-矩形窗，1-Hanning窗。
weightType	int	加权类型：0-无加权，1-A计权，2-B计权，3-C计权。
scaleType	int	缩放类型：0-线性输出，1-dB输出。
colormapData	double**	输出参数，指向色谱图数据的指针（按行优先，timeBins × frequencyBins），需释放。
timeBins	int*	输出参数，时间分箱数量。
frequencyBins	int*	输出参数，频率分箱数量。

调用范例：

```
double* colormapData = nullptr;
int timeBins = 0;
int frequencyBins = 0;
int ret = GenerateTimeFrequencyColormapByIncrement(
    signalNative,
    4096,
    0.15,
    0.0,
    -1.0,
    1.0, // referenceValue
    0,   // FormatType::RMS
    1,   // WindowType::Hanning
    0,   // WeightType::None
    1,   // ScaleType::Db
    &colormapData,
    &timeBins,
    &frequencyBins
);
if (ret == 1) {
```

```
    for (int i = 0; i < timeBins; ++i) {
        for (int j = 0; j < frequencyBins; ++j) {
            printf("T[%d] F[%d]=%f\n", i, j, colormapData[i * frequencyBins + j]);
        }
    }
    delete[] colormapData;
}
```

10. GenerateRpmFrequencyColormap

功能：生成信号的转速-频率色谱图（rpm-频谱），用于分析不同转速下的频谱分布。

函数声明：

```
int GenerateRpmFrequencyColormap(
    SignalNative signalNative,
    RpmNative rpmNative,
    int spectrumLines,
    double lowerRpmThreshold,
    double upperRpmThreshold,
    double rpmStep,
    double referenceValue,
    int formatType,
    int windowType,
    int weightType,
    int scaleType,
    double** colormapData,
    double** rpmData,
    double** freqData,
    int* rpmBins,
    int* frequencyBins
);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号数据结构体。
rpmNative	RpmNative	输入转速数据结构体。
spectrumLines	int	谱线数，决定频谱分辨率。
lowerRpmThreshold	double	转速下限。
upperRpmThreshold	double	转速上限，-1.0表示最大转速。
rpmStep	double	转速步进，单位为rpm。
referenceValue	double	参考值，仅用于dB输出时的参考值。
formatType	int	输出格式类型。

参数名	类型	说明
windowType	int	窗函数类型。
weightType	int	加权类型。
scaleType	int	缩放类型。
colormapData	double**	输出参数，色谱图数据指针（rpmBins × frequencyBins），需释放。
rpmData	double**	输出参数，转速轴数据指针，需释放。
freqData	double**	输出参数，频率轴数据指针，需释放。
rpmBins	int*	输出参数，转速分箱数量。
frequencyBins	int*	输出参数，频率分箱数量。

调用范例：

```
double* colormapData = nullptr;
double* rpmOutData = nullptr;
double* freqOutData = nullptr;
int rpmBins = 0;
int freqBins = 0;
int ret = GenerateRpmFrequencyColormap(
    signalNative,
    rpmNative,
    4096,
    0.0,
    -1.0,
    25.0,
    1.0, // referenceValue
    0,   // FormatType::RMS
    1,   // WindowType::Hanning
    0,   // WeightType::None
    1,   // ScaleType::Db
    &colormapData,
    &rpmOutData,
    &freqOutData,
    &rpmBins,
    &freqBins
);
if (ret == 1) {
    for (int i = 0; i < rpmBins; ++i) {
        for (int j = 0; j < freqBins; ++j) {
            printf("RPM: %f, Freq: %f, Value: %f\n", rpmOutData[i], freqOutData[j],
colormapData[i * freqBins + j]);
        }
    }
    delete[] colormapData;
    delete[] rpmOutData;
```

```
    delete[] freqOutData;
}
```

11. GenerateRpmOrderColormap

功能：生成信号的转速-阶次色谱图（rpm-Order谱），用于分析不同转速下各阶次的幅值分布。

函数声明：

```
int GenerateRpmOrderColormap(
    SignalNative signalNative,
    RpmNative rpmNative,
    double maxOrder,
    double orderResolution,
    double oversamplingFactor,
    double lowerRpmThreshold,
    double upperRpmThreshold,
    int rpmStep,
    double referenceValue,
    int formatType,
    int windowType,
    int weightType,
    int scaleType,
    double** colormapData,
    double** rpmAxis,
    double** orderAxis,
    int* rpmBins,
    int* orderBins
);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号数据结构体。
rpmNative	RpmNative	输入转速数据结构体。
maxOrder	double	阶次最大值。
orderResolution	double	阶次分辨率。
oversamplingFactor	double	过采样因子。
lowerRpmThreshold	double	转速下限。
upperRpmThreshold	double	转速上限。
rpmStep	int	转速步进，单位为rpm。
referenceValue	double	参考值，仅用于dB输出时的参考值。

参数名	类型	说明
formatType	int	输出格式类型。
windowType	int	窗函数类型。
weightType	int	加权类型。
scaleType	int	缩放类型。
colormapData	double**	输出参数，色谱图数据指针（rpmBins × orderBins），需释放。
rpmAxis	double**	输出参数，转速轴数据指针，需释放。
orderAxis	double**	输出参数，阶次轴数据指针，需释放。
rpmBins	int*	输出参数，转速分箱数量。
orderBins	int*	输出参数，阶次分箱数量。

调用范例：

```
double* colormapData = nullptr;
double* rpmAxis = nullptr;
double* orderAxis = nullptr;
int rpmBins = 0;
int orderBins = 0;
int ret = GenerateRpmOrderColormap(
    signalNative,
    rpmNative,
    12.0,    // maxOrder
    0.5,     // orderResolution
    1.0,     // oversamplingFactor
    0.0,     // lowerRpmThreshold
    -1.0,    // upperRpmThreshold
    25,      // rpmStep
    1.0,     // referenceValue
    0,       // FormatType::RMS
    1,       // WindowType::Hanning
    0,       // WeightType::None
    1,       // ScaleType::Db
    &colormapData,
    &rpmAxis,
    &orderAxis,
    &rpmBins,
    &orderBins
);
if (ret == 1) {
    for (int i = 0; i < rpmBins; ++i) {
        for (int j = 0; j < orderBins; ++j) {
            printf("RPM: %f, Order: %f, Value: %f\n", rpmAxis[i], orderAxis[j],
colormapData[i * orderBins + j]);
        }
    }
```

```
    delete[] colormapData;
    delete[] rpmAxis;
    delete[] orderAxis;
}
```

12. GetCrestFactor

功能：计算信号的Crest值（峰值因子，最大值与RMS之比）。

函数声明：

```
int GetCrestFactor(
    SignalNative signalNative,
    double* outValue
);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号数据结构体。
outValue	double*	输出参数，指向Crest值的指针。

调用范例：

```
double crestValue = 0.0;
int ret = GetCrestFactor(signalNative, &crestValue);
if (ret == 1) {
    printf("Crest Factor: %f\n", crestValue);
}
```

13. SignalSlice

功能：截取信号的指定时间段，返回新的信号结构体。

函数声明：

```
int SignalSlice(SignalNative signalNative, double startTime, double endTime, SignalNative*
outSignal);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入信号数据结构体。
startTime	double	起始时间，单位为秒。
endTime	double	结束时间，单位为秒。

参数名	类型	说明
outSignal	SignalNative*	输出参数，指向截取后信号的结构体指针，需释放其Samples。

调用范例：

```
SignalNative slicedSignal;
int ret = SignalSlice(signalNative, 1.0, 2.0, &slicedSignal);
if (ret == 1) {
    // 使用slicedSignal.Samples和slicedSignal.Length
    delete[] slicedSignal.Samples;
}
```

14. RpmSlice

功能：截取转速数据的指定时间段，返回新的转速结构体。

函数声明：

```
int RpmSlice(RpmNative rpmNative, double startTime, double endTime, RpmNative* outRpm);
```

参数说明：

参数名	类型	说明
rpmNative	RpmNative	输入转速数据结构体。
startTime	double	起始时间，单位为秒。
endTime	double	结束时间，单位为秒。
outRpm	RpmNative*	输出参数，指向截取后转速的结构体指针，需释放其RpmValues和TimeValues。

调用范例：

```
RpmNative slicedRpm;
int ret = RpmSlice(rpmNative, 1.0, 2.0, &slicedRpm);
if (ret == 1) {
    // 使用slicedRpm.RpmValues和slicedRpm.TimeValues
    delete[] slicedRpm.RpmValues; delete[] slicedRpm.TimeValues;
}
```

15. PulseToRpm

功能：将脉冲信号转换为转速数据。

函数声明：

```
int PulseToRpm(SignalNative signalNative, EdgeDetectorNative edgeDetectorNative,
RpmCalculationOptionsNative rpmCalculationOptionsNative, RpmNative* outRpm);
```

参数说明：

参数名	类型	说明
signalNative	SignalNative	输入脉冲信号数据结构体。
edgeDetectorNative	EdgeDetectorNative	边缘检测参数结构体。
rpmCalculationOptionsNative	RpmCalculationOptionsNative	转速计算参数结构体。
outRpm	RpmNative*	输出参数，转速数据结构体指针，需释放其RpmValues和TimeValues。

调用范例：

```
EdgeDetectorNative edgeOpt = {10, 0.0, 0};
RpmCalculationOptionsNative rpmOpt = {2, 10, 1, true, -1.0, -1.0, 0.1};
RpmNative rpmNative;
int ret = PulseToRpm(signalNative, edgeOpt, rpmOpt, &rpmNative);
if (ret == 1) {
    for (int i = 0; i < rpmNative.Length; ++i) {
        printf("Time: %f, RPM: %f\n", rpmNative.TimeValues[i], rpmNative.RpmValues[i]);
    }
    delete[] rpmNative.RpmValues; delete[] rpmNative.TimeValues;
}
```