指导小组成员：

汤大侃　副教授
朱谦　副教授

摘要

在海上运动的环境监测系统中，通过传感网系统收集得到的原始数据不能直接被训练人员使用，因此设计这个软件将数据提取、计算、分析，并通过曲线图、柱状图、表格等形式展现出来。本文即描述了这样一种将数据可视化分析的一系列方法和软件实现。

在本软件系统中，通过曲线图可以描绘出风向、风速的数据值与时间的曲线关系。除了基本的数据操作处理，比如可以有误差时去除过大或过小值，在时间范围较长时截取时间段实现部分时间段分析从而可以提供横向和纵向的双重缩放查看功能外，进行数据分析时使用了数字滤波的方法对原始数据进行处理，提供了傅里叶变换和卡尔曼滤波两种方法。快速傅里叶变换和反变换加窗后组成有限字长数字滤波器和卡尔曼滤波器都可以使图像达到更加平滑和去噪声的目的。另外使用卡尔曼滤波方法使得数据的前后误差更小。本文对此也进行了较为详细的说明和分析。在数据统计处理中，使用了风速、风向的矢量计算方法让计算结果更符合物理实际。

软件实现的开发环境是基于 Windows 7 下的 Visual Studio 2010 的.Net Framework。

【关键词】数据可视化，快速傅里叶变换，卡尔曼滤波，矢量计算方法，C++

Abstract

In an Environment Monitoring System on Yachting Sports, raw data collected by the sensor networks could not be used by analysists directly. A software system should be designed to make use of the raw data, including extracting, calculating, analyzing and display them through curve graphs, bar graphs, tables, etc.. The article describes a seiries of methods and implementation of such kind of data visualization analysis.

In the process of this data visualization analysis, function of values of wind directions and wind speed to time series could be described through curve graphs. While the program provides basic data operations including excluding large or small values caused by mistakes and cutting time series while total time range is large make it possible to zoom in or out to watch the curve graphs vertically and horizonally, it also offers data analysis with Digital Fileters. Finite Impulse Reponse (FIR) Digital Filters supported by Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) with Window Methods and Kalman Filters are available to deal with data values collected. And Kalman Filtering has been used to cut down the error of values and to make the curves smoother. This article has made descriptions to the two filters in detail. In the statistical analysis of these data values, vector calculation method was used to make statistical results more applicable to their physical features.

The software has been developed based on the .Net Framework provided by Microsoft Visual Studio 2010 in Windows.

[Keywords] Data Visualization, Fast Fourier Transform (FFT), Inverse Fourier Transform (IFFT), Kalman Filtering, Vector Calculation Method, C++

第一章 绪论

1.1 选题背景

伦敦奥运会帆船比赛将于 2012 年 7 月 28 日在英国韦茅斯-波特兰港举行。由于韦茅斯的地理位置特殊，地形对风和潮汐流的影响都比较显著，风变规律和潮汐流的变化都存在一定特点。因此研制一套对风速风向进行数据分析的科研软件是非常有利于训练和备战的。

1.2 帆船运动概述

1.2.1 风对帆船运动的影响

帆船运动是靠风在帆上产生动力推动船前进的。风是帆船运动的最主要的条件。从事专业帆船运动的人士对气象变化，尤其是风的变化规律有着很深的理解和研究。对举办国在气象预报上也有很高的要求。通常比赛期间风力持续达到每秒 3 米以上时才适于比赛。而风力超过每秒 20 米时，就要考虑水上安全而停止比赛。另外，在一轮比赛的时间段内，风向摆动超过 50 度也要从比赛公平的角度考虑放弃比赛。

在大风天气中，气流对帆的压力较大，帆的浮升力增加，船的横向移动力也相对增大。为了减小船的横移力就只有加大前进力，也就是提高船的前速。提高船速必需加大风向角。如果不加大风向角行驶，船的速度减慢，船的横移力就会相对增大，帆船的绝对速度就会变慢。

在中、小风环境中，气流对帆的压力减小（比风大时），对船的横移力也相对减小。但由于水下的阻力不会因风速的变化而变化，这样船在风小时比风大时的横移力相对要小。因此船的速度相对的慢些，船的横移力矩也不会太大。所以，在中、小风的迎风直线驶时，可以追求较小的风向角。[1]

1.2.2 流对帆船运动的影响

水流和波浪都是海洋的自然特征，运动员如何利用好涌浪的力量产生更大的前进力，如何在计划航线时候考虑到水流的影响同样也是奥妙颇深的。但如果海域内水流过急，则会影响比赛的正常进行。

一般情况下，在顶流和下压流时可适当加大风向角以加快船速，在顺流和上压时可略减小风向角行驶。同样道理，我们在有涌浪的天气中必需加快船的速度，除了调整帆和身体的移动外，最有效的操作技术方法就是放大风向角来达到提高船速的目的。

以上关于海上环境与帆船运行的关系说明风速风向与帆船运动的密切。风的情况影响水流的情况，而两者直接决定运动员如何控制帆船上的拉杆和风向角。

1.3 研究目的

帆船运动主要是靠风在帆船上产生动力而推动船前进的，风是帆船运动的最主要条件。在比赛中，有效利用风是取得好成绩的保证，因此探索比赛场地的风变规律显得尤为重要，所以实现对风速风向的观测结果进行记录和数据分析是本课题的研究目的所在。

1.4 系统结构概述

本文描述的风向风速数据分析软件是“帆船帆板赛场环境监测和运动技术分析系统”的系统，位于整个系统的上层，向上提供用户与用户的数据操作接口，向下制定服务需求和技术规范。

1.4.1 整体监测和技术分析系统

整个系统分为两个部分，有场地监测系统和多参数帆船运动技术监测系统。其整体结构图如图 1-1 所示。

在场地监测系统中，数个浮标置于水面上，自动测量并记录风速风向数据。采集完成后将存储数据的 SD 卡取出放在电脑上即可读出数据并分析。

在多参数帆船运动机监测系统中，每个帆船上设置不同种类的传感器来采集不同的参数数据。根据技术要求，帆船运动技术检测包括 GPS 定位、船速和航程测量、船体姿态（倾角）测量、风力（风速风向）测量以及运动员心率监测等多项技术参数，未来还可能对帆船三角绳张力、横杆转动角度等其它技术监测。船上每个节点均内置 Zigbee 模块，通过其中一个称之为“网络协调器”的专用节点设备来创建船上多参数检测无线传感网络。船上的网关节点负责将数据存在 SD 卡上供回放使用，同时也将实时数据发给移动设备用于实时比赛显示。

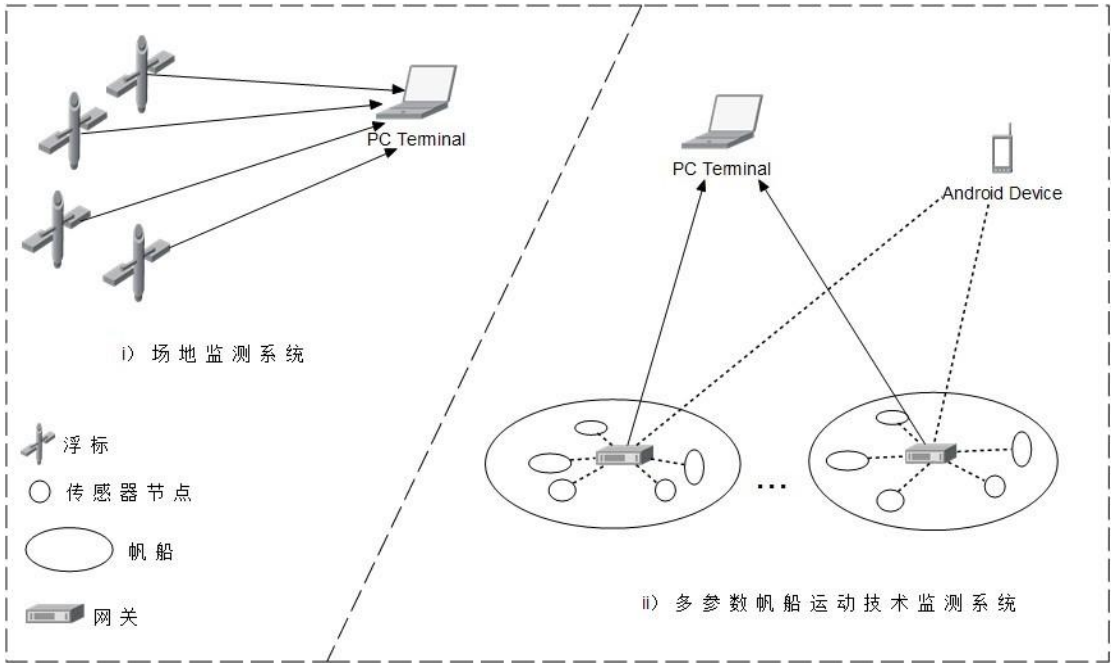


图 1-1 整体监测和运动技术分析系统

利用芯片的集成特性可以适当简化网络结构图，比如设计将“GPS 节点”和“网络协调器”合二为一，称为船上的“主节点”。每艘船上的“主节点”内部包含一个移动通信模块，这样就可以通过 2G/3G 公用移动网络收集每艘船的各项参数。

1.4.2 风向风速分析系统

风向风速分析系统位于场地监测系统中，数据来源于浮标上存储于 SD 卡中的数据，经过 PC 机处理分析后展示在终端上。

风向风速分析系统主要提供两个功能：图像显示处理和数据统计分析。将原始数据表经过幅度和时间上的截取和筛减后可以得到操作数据表，即可通过图像显示出来，又可以进行数理统计。另外任何阶段的数据操作对象都可以以相同的格式导出并且保存，以作为下次分析的原始数据表。风向风速分析系统的功能图可如图 1-2 所示。

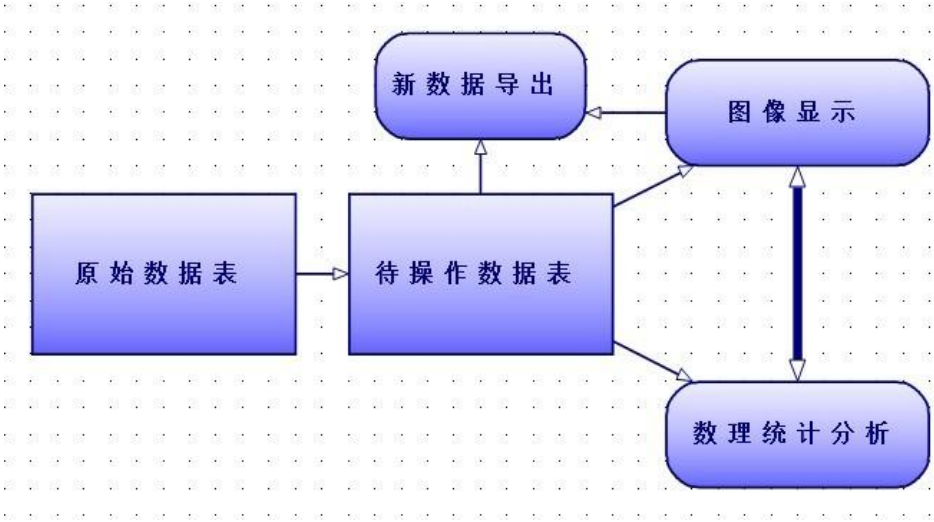


图 1-2 风向风速分析系统功能图

图像显示也提供数据平滑、滤波的操作，因此也会产生新的可用数据；数理统计结果不仅通过列表展示，也可以通过柱状图、直方图显示出来。

风向风速自动记录仪的采样率为 1Hz，即每小时有 3600 条记录生成。如果每天采集 3 个小时（通常是 3-5 个小时），则每台风速风向仪产生 10800 条记录，数据量庞大。所以分为以下操作步骤，如图 1-3 所示。

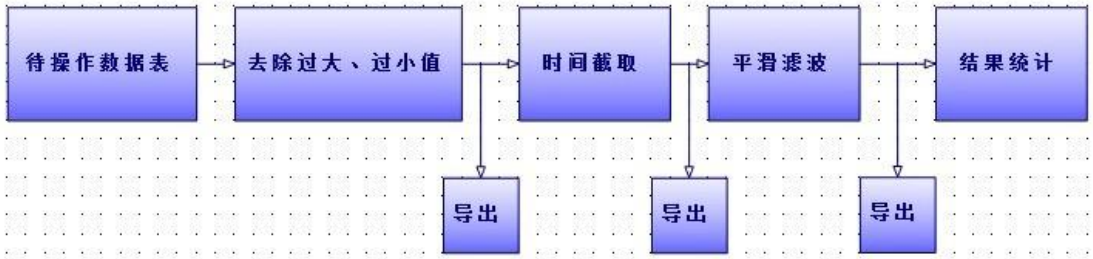


图 1-3 风向风速分析系统操作流程图

上图中，去除过大过小值作用是剔除明显的错误数据。这些错误数据一般都是由外界环境的干扰或者系统内部的紊乱导致的，这样的数据变化较为急剧，易于识别，如果某个记录中出现错误数据，则删除该条记录。

时间截取的作用是手动截取所需要的时间段，得到图像和统计结果。

数据的平滑滤波是指用低通滤波或者卡尔曼滤波来处理数据点，得到平滑滤波后的图像和数据。

第二章 理论基础

2.1 数据可视化技术

数据可视化是关于数据的视觉表现形式的研究。

数据可视化主要是利用图形化手段清晰、有效的传达与沟通信息。数据可视化与信息图形、科学可视化、信息可视化以及统计图形密切相关。当前在研究、教学和开发领域，数据可视化都是一个极为活跃而又关键的方面。“数据可视化”这条术语实现了成熟的科学可视化领域与较年轻的信息可视化领域的统一。

数据可视化起源于 1960s 计算机图形学，人们用计算机创建图形图表，将提取出来的数据可视化，把数据的各种属性和变量呈现出来。随着计算机硬件的发展，人们创建更复杂并且规模更大的数字模型，发展了数据采集设备和数据保存设备。人们同时也需要更高级的计算机图形学技术及方法来创建这些规模庞大的数据集。随着数据可视化平台的拓展，应用领域的增加，表现形式的不断变化，以及增加了诸如实时动态效果、用户交互使用等，数据可视化像所有新兴概念一样边界不断扩大。[2]

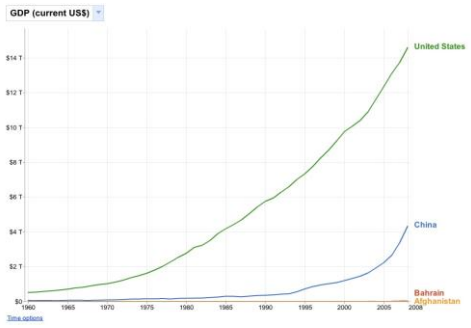


图 2-1 Google 的数据可视化图表

2.2 快速傅里叶变换及反变换

2.2.1 离散变换

离散变换描述的是离散数据在时域和频域之间进行的变换，比如将电压与时间的表示变成了幅度和频率以及相位与频率的表示；反过来也是如此。时域和频域对同样的数据提供互补信息。离散变换特别是离散余弦变换可以用在语音和视

频信号的数据压缩中，使得信号能够以小的带宽进行传输。离散变换也用于图像处理中，提供为模式识别简化的特征集。从频域到时域的变换如同从时域到频域的变换一样重要。

在很多有效的离散变换中，离散傅里叶变换（DFT）以及他的快速计算方法——快速傅里叶变换（FFT）是最知名的，也是最重要的算法之一。理由是他们允许在频域表示所有的信号，即便是最短的数据记录长度（<1s），截断的傅里叶频率分量要比任何其他指数型级数更好的表示数据，每个分量是正弦的，在通过线性系统时不会产生失真，因而构造了一个好的测试信号，并且可以快速计算FFT。另一个理由是傅立叶分析自1822年自傅里叶发表后就存在，因此傅里叶变换得到了高度的重视和开发，因而应用领域十分广泛。[3]

2.2.2 傅里叶级数和傅里叶变换

任何周期波形f(t)可以表示为无限正弦项、余弦项和一个常数之和，表示成下式中的傅里叶级数：

$$f(t)=a_0+\sum_{n=1}^{\infty}a_n\cos(n\omega t)+\sum_{n=1}^{\infty}b_n\sin(n\omega t) \tag{2.2.1}$$

其中t是一个独立的变量，常常表示时间，但是也可以表示距离或者任何其它的量；$f(t)$常常表示电压与时间的变化波形，但也可能是其他波形；$\omega=2\pi/T_p$称为一次谐波、基波或角频率，它与基频 f 通过$\omega=2\pi f$ 建立联系；T_p是波形的重复周期，

$$a_0=\frac{1}{T}\int_{-T_p/2}^{T_p/2}f(t)dt \tag{2.2.2}$$

是常数，它等于$f(t)$在一个周期上的时间平均。

$$a_n=\frac{1}{T_p}\int_{-T_p/2}^{T_p/2}f(t)\cos(n\omega t)dt \tag{2.2.3}$$

和

$$b_n=\frac{1}{T_p}\int_{-T_p/2}^{T_p/2}f(t)\sin(n\omega t)dt \tag{2.2.4}$$

频率$n\omega$称为ω的n次谐波，因此，无穷级数(2.2.1)式包括频率相关的正弦项和余弦项，这些正弦项和余弦项在正的谐波频率$n\omega$处的幅度a_n和b_n是不同的。这

种级数用指数形式表示可以写更易于处理：

$$f(t)=\sum_{n=-\infty}^{\infty} d_n e^{jn\omega t} \tag{2.2.5}$$

其中

$$d_n=\frac{1}{T_p}\int_{-T_p/2}^{T_p/2} f(t)e^{-jn\omega t}dt \tag{2.2.6}$$

当T_p增加时，谐波分量之间的间隔$1/T_p=\omega/2\pi$减小到$d\omega/2\pi$，最终变成零，这对英语从离散频率变量$n\omega$变到连续变量ω，幅度谱和相位谱变成连续的。因此，当$T_p\to\infty$时，$d_n\to d(\omega)$，通过这些修改，(2.2.6)事变成

$$d(\omega)=\frac{d\omega}{2\pi}\int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt \tag{2.2.7}$$

将上式除以$d\omega/2\pi$进行归一化，得

$$\frac{d(\omega)}{d\omega/2\pi}=F(j\omega)=\int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt \tag{2.2.8}$$

$F(j\omega)$是复数，称为傅里叶积分，也即傅里叶变换。

利用傅里叶反变换从频域到时域也是可能的，这时，

$$f(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty} F(j\omega)e^{j\omega t}d\omega=\int_{-\infty}^{\infty} F(j\omega)e^{j\omega t}df \tag{2.2.9}$$

2.2.3 离散傅里叶变换及其逆

在实际中，数据的傅里叶变换是用数字计算机而不是模拟处理得到的。由于模拟波形有无数量的相邻的点组成，表示所有这些值在实际中是不可能的。因此，模拟值必须以均匀的间隔进行抽样，然后将抽样值转换成数字二进制表示。抽样速率要高到足以正确的表示波形。理论上必须的抽样率是奈奎斯特抽样率，也就是$2f_{\max}$，其中$f_{\max}$是有效幅度值信号中最高频率的正弦分量的频率。

假定波形以均匀的时间间隔T进行抽样，得到N个抽样值的抽样序列$\{x(nT)\}=x(0),x(T),...,x[(N-1)T]$，其中$n$是从$n=0$到$n=N-1$的抽样数，数据值$x(nT)$在表示一个时间序列时是为实。那么$x(nT)$的 DFT 定义为频域中的一个复值序列$\{X(k\Omega)\}=X(0),X(\Omega),...,X[(N-1)\Omega]$，其中$\Omega=2\pi/(N-1)T$是一次谐

波频率，当$N \gg 1$时，$\Omega \simeq 2\pi / NT$。N 个实数据变换成 N 个复 DFT 值（频域）。DFT 值 $X(k)$ 为

$$X(k) = F_D[x(nT)] = \sum_{n=0}^{N-1} x(nT)e^{-jk\Omega nT}, k = 0,1,...,N-1 \qquad (2.2.10)$$

其中，F_D 表示离散傅里叶变换。在这个等式中 k 表示变换分量的谐波数，类似于(2.2.8)式的傅里叶变换。当$T < 0$或$t > (N-1)T$时 $f(t) = 0$，其中令 $x(nT) = f(t)$、$k\Omega = \omega$和 $nT = t$，这时(2.2.10)与(2.2.8)类似。将这些变量替换代入(2.2.8)式，且令 $dt = T$，用对谐波频率 kf_s 求和取代，其中 $f_s = 1/(N-1)T = 2\pi / \Omega$，

$$\sum_{n=0}^{N-1} x(nT)e^{-jk\Omega nT}T = F(j\omega) \qquad (2.2.11)$$

那么当$0 \le t \le (N-1)T$时，比较(2.2.10)和(2.2.11)式得

$$F(j\omega) = TX(k) \qquad (2.2.12)$$

上式表明傅里叶变换分量与 DFT 分量通过抽样间隔建立起了联系。傅里叶变换通过对 DFT 乘以抽样间隔而得到。

如果将 DFT 的第 k 个分量 $X(k)$ 与第 $(k+N)$ 个分量 $X(k+N)$ 进行比较，可以得到：

$$X(k) = \sum_{n=0}^{N-1} x(nT)e^{-jk\Omega nT} \qquad ($$

$$= \sum_{n=0}^{N-1} x(nT)e^{-jk2\pi n/N}$$

和

$$X(k+N) = \sum_{n=0}^{N-1} x(nT)e^{-jk2\pi n/N}e^{-jN2\pi n/N}$$

$$= \sum_{n=0}^{N-1} x(nT)e^{-jk2\pi n/N}e^{-j2\pi n}$$

$$= \sum_{n=0}^{N-1} x(nT)e^{-jk2\pi n/N} = X(k)$$

$X(k) = X(n+k)$ 的事实表明 DFT 是周期的，其周期为 N，这是 DFT 的循环特性，DFT 分量的值是重复的。如果$k = 0$，那么 $k+N = N$， $X(0) = X(N)$。

同理，DFT 也能从频域到时域进行离散变换，这可以由下式定义的离散傅里叶反变换（IDFT）来实现，

$$x(nT)=F_D^{-1}[X(k)]=\frac{1}{N}\sum_{k=0}^{N-1}X(k)e^{jk\Omega nT},n=0,1,...,N-1 \qquad (2.2.13)$$

其中 F_D^{-1} 表示离散傅里叶反变换。

显然，IDFT 与(2.2.9)式的傅里叶反变换类似。容易证明，傅里叶反变换可以由 IDFT 除以 T 得到。(2.2.13)式的有效性通过将 $x(nT)$ 代入到(2.2.10)式即可得到证明。

2.2.4 DFT 的计算复杂性和优化算法

2.2.4.1 DFT 计算的复杂性

DFT 计算需要很多的乘法和加法。例如对于 8 点序列做 DFT 计算（8 点 DFT），$X(k)$ 为

$$X(k)=\sum_{k=0}^{7}X(n)e^{-jk2\pi n/8},k=0,1,...,7 \qquad (2.2.13)$$

令 $2\pi k=K$，展开后得

$$\begin{aligned}X(k)=&x(0)e^{-jk0}+x(1)e^{-jk1}+x(2)e^{-jk2}+x(3)e^{-jk3}+x(4)e^{-jk4}\\&+x(5)e^{-jk5}+x(6)e^{-jk6}+x(7)e^{-jk7}\end{aligned} \qquad (2.2.14)$$

(2.2.14)式的右边包含了 8 项，每一项由指数项（复数）与另一项（实数或复数）相乘组成，将这些乘积项加到一起。因此，他们有 8 次复数乘法，7 次复数加法需要计算。对于 N 点 DFT，需要计算的谐波分量有 N 个，因此，8 点 DFT 的计算要求 64 次复数乘法和 56 次复数加法。

对于 N 点 DFT，需要计算的复数乘法和复数加法次数分别变成了 N^2 和 $N(N-1)$。如果 $N=1024$，那么大约需要一百万次复数加法和一百万次复数乘法。

注意到这些等式中有相当多的内在冗余运算，那么所要求的计算量是可以减少的。比如，$k=1$ 和 $n=2$ 时，有 $e^{-jk2\pi/8}=e^{-j\pi/2}$；当 $k=2$ 和 $n=1$ 时也有 $e^{-jk2\pi/8}=e^{-j\pi/2}$。

2.2.4.2 时域抽取的快速傅里叶变换算法（Cooley-Tukey 算法）

利用 DFT 中固有的计算冗余来减少计算量，可以加快计算速度。对于 1024 点 DFT，要求的计算量可以减少 204.8 倍。能够实现这种快速运算的算法称为快速傅里叶变换（FFT）。当算法在时域运用时称为时域抽取 FFT。第一个 DFT 算法是由 Cooley 和 Tukey 提出的，称为 Cooley-Tukey 算法。

首先对符号做简化处理，对于(2.2.10)，可以重写为

$$X_1(k)=\sum_{n=0}^{N-1}x_n e^{-j2\pi nk/N},k=0,1,...,N-1 \tag{2.2.15}$$

将因子 $e^{-j2\pi/N}$ 写为 W_N，即

$$W_N=e^{-j2\pi/N} \tag{2.2.16}$$

可将(2.2.15)式变成

$$X_1(k)=\sum_{n=0}^{N-1}x_n W_N^{kn},k=0,1,...,N-1 \tag{2.2.17}$$

设 $N=2^L$，将 $x(n)$ 按 n 的奇偶分成两组：

$$\begin{cases} x(2r)=x_1(r) \\ x(2r+1)=x_2(r),r=0,1,...,\dfrac{N}{2}-1 \end{cases}$$

则，

$$X(k)=DFT[x(n)]=\sum_{n=0}^{N-1}x(n)W_N^{nk}$$

$$=\sum_{\substack{n=0\\ n\text{为偶数}}}^{N-1}x(n)W_N^{nk}+\sum_{\substack{n=0\\ n\text{为奇数}}}^{N-1}x(n)W_N^{nk}$$

$$=\sum_{\substack{n=0\\ n\text{为偶数}}}^{N-1}x(n)W_N^{nk}+\sum_{\substack{n=0\\ n\text{为奇数}}}^{N-1}x(n)W_N^{nk}$$

$$=\sum_{r=0}^{\frac{N}{2}-1}x(2r)W_N^{2rk}+\sum_{r=0}^{\frac{N}{2}-1}x(2r+1)W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x_1(r) W_{\frac{N}{2}}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x_2(r) W_{\frac{N}{2}}^{rk}$$

$$= X_1(k) + W_N^k X_2(k)$$

式中， $X_1(k)$ 和 $X_2(k)$ 分别是 $x_1(n)$ 和 $x_2(n)$ 的 $N/2$ 的 DFT。 k 的取值范围是 $0,1,...,N/2-1$。因此 $X(k)=X_1(k)+W_N^k X_2(k)$ 只能计算出 $X(k)$ 的前一半值。后一半 $X(k)$ 值， $N/2, N/2+1, ..., N$：

利用

$$W_{N/2}^{r(N/2+k)} = W_{N/2}^{rk}$$

可得到

$$X_1(\frac{N}{2}+k) = \sum_{r=0}^{N/2-1} x_1(r) W_{N/2}^{r(N/2+k)} = \sum_{r=0}^{N/2-1} x_1(r) W_{N/2}^{rk} = X_1(k)$$

同理可得，

$$X_2(\frac{N}{2}+k) = X_2(k)$$

考虑到

$$W_N^{(N/2+k)} = W_N^{N/2} \cdot W_N^k = -W_N^k$$

及前半部分，

$$X(k) = X_1(k) + W_N^k X_2(k), k = 0,1,...,N/2-1$$

因此可得后半部分 $X(k)$

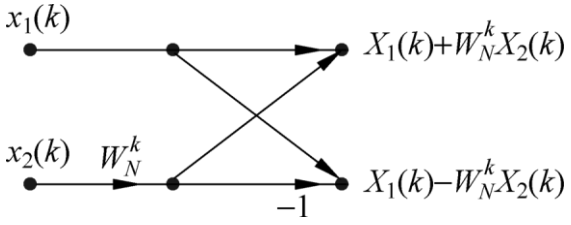
$$X(k+\frac{N}{2}) = X_1(k+\frac{N}{2}) + W_N^{k+N/2} X_2(k+\frac{N}{2})$$
$$= X_1(k) - W_N^k X_2(k), k = 0,1,...,N/2-1$$

综上得到：

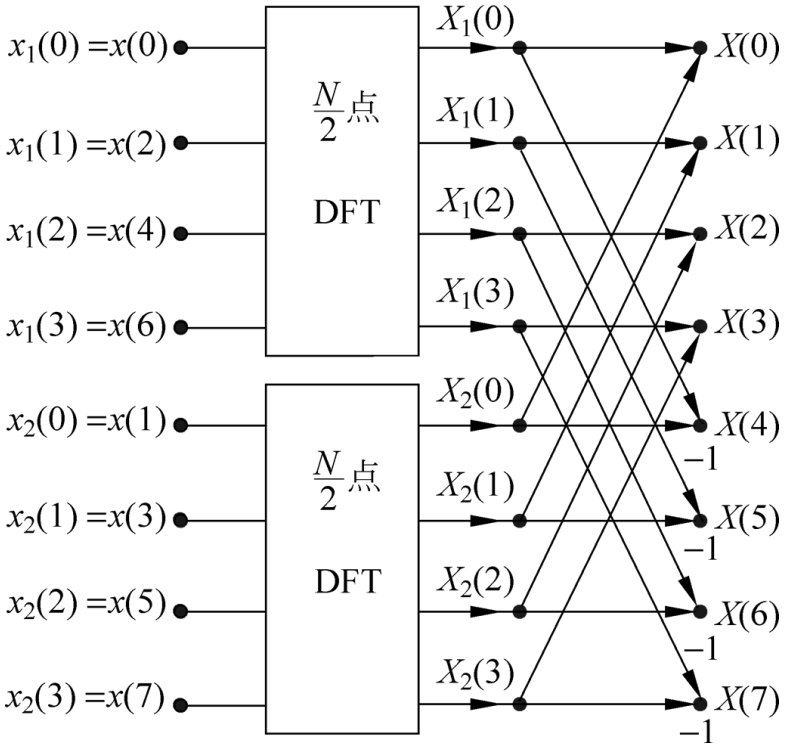
$$\begin{aligned} X(k) &= X_1(k) + W_N^k X_2(k) \\ X(k) &= X_1(k) - W_N^k X_2(k) \end{aligned} \tag{2.2.18}$$

上式称为蝶形运算表达式。

(2.2.18)式说明，只要求出 2 个 $N/2$ 点的 DFT，即 $X_1(k)$ 和 $X_2(k)$，再经过蝶形运算就可求出全部 $X(k)$ 的值，运算量大大减小。其信号流图如下图：



比如，当$N=8$时，分解为 2 个 4 点的 DFT，然后做 4 次蝶形运算即可求出所有 8 点 $X(k)$ 的值：



第一次分解后需要的运算量为 2 个 $N/2$ 的 DFT+ $N/2$ 蝶形：

复数乘法次数：$2\times(N/2)^2+N/2=N^2+N/2$

复数加法次数：$2\times(N/2)(N/2-1)+2\times N/2=N^2$

相比于 N 点 DFT 的运算量 N^2 和 $N(N-1)$，运算工作量减少了进一半。

若 $N/2$ 仍是偶数，则可进一步把每个 $N/2$ 点子序列再按其奇偶部分分解为两个 $N/2$ 点的子序列。以 $N/2$ 点序列 $x_1(r)$ 为例：

$$\left.\begin{aligned} x_1(2l)&=x_3(l)\\ x_1(2l+1)&=x_4(l)\end{aligned}\right\}\quad l=0,1,\ldots,\frac{N}{4}-1$$

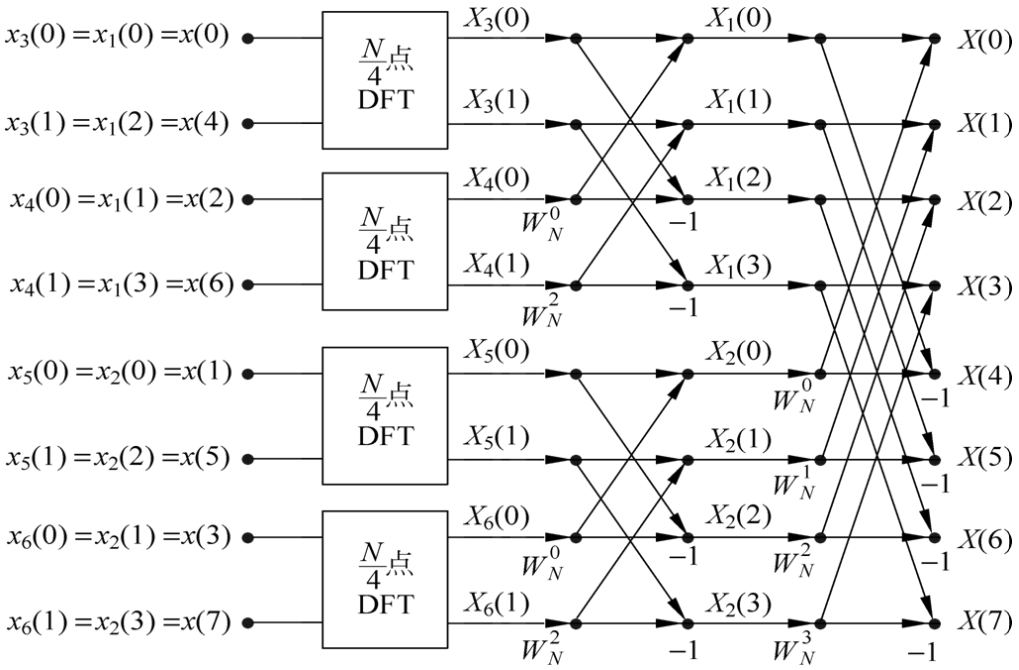
则有，

$$X_1(k)=\sum_{r=0}^{N/2-1}x_1(r)W_{N/2}^{rk}=\sum_{l=0}^{N/4-1}x_1(2l)W_{N/2}^{2lk}+\sum_{l=0}^{N/4-1}x_1(2l+1)W_{N/2}^{(2l+1)k}$$
$$=\sum_{l=0}^{N/4-1}x_3(l)W_{N/4}^{lk}+W_{N/2}^{k}\sum_{l=0}^{N/4-1}x_4(l)W_{N/4}^{lk}$$
$$=X_3(k)+W_{N/2}^{k}X_4(k),k=0,1,...,\frac{N}{4}-1$$

且

$$X_1\left(\frac{N}{4}+k\right)=X_3(k)-W_{N/2}^{k}X_4(k),k=0,1,...,\frac{N}{4}-1$$

由此可见，一个 $N/2$ 点 DFT 可以分解成两个 $N/4$ 点 DFT。同理也可对 $x_2(n)$ 进行同样的分解，求出 $X_2(K)$。同样以 8 点为例，第二次奇偶分解的蝶形运算信号流图为：



由按时间抽取法 FFT 的信号流图可知，当 $N=2^L$ 时，共有 L 级蝶形运算；每级都有 $N/2$ 个蝶形运算组成，而每个蝶形有 1 次复乘、2 次复加，因此每级运算都需 $N/2$ 次复乘和 N 次复加。这样 L 级蝶形运算总共需要：

复数乘法：
$$\frac{N}{2}\cdot L=\frac{N}{2}\log_2 N$$

复数加法：

$$N \cdot L = N \log_2 N$$

可以得到直接 DFT 与 FFT 计算量的比值为：

$$M = \frac{N^2}{\frac{N}{2}\log_2 N} = \frac{2N}{\log_2 N}$$

由式(2.2.13)可知，

$$x(n) = IDFT\left[X(k)\right] = \frac{1}{N}\sum_{k=0}^{N-1} X(k) W_N^{-nk} \tag{2.2.19}$$

由式(2.2.10)可知，

$$X(k) = DFT\left[x(n)\right] = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{2.2.20}$$

式(2.2.19)即 IFFT，比较(2.2.19)和(2.2.20)，可以看出 IFFT 只有一个因子$1/N$和指数的符号不同 FFT，因此只要进行很小的修改，FFT 算法就能用来计算 IFFT。

2.3 使用加窗方法的 FIR 数字滤波器

2.3.1 FIR 数字滤波器

一个滤波器实际上是一个系统或者网络。以一种期望的模式有选择的改变信号的波形、幅度-频率和（或）相位-频率特性。一般滤波的目的是为了改善信号的质量，例如消除或者减少噪声（正如本软件的目的）。[3]

图 2-2 给出了一个具有模拟输入信号和输出的信号的实时数字滤波器的简化框图。这个带限模拟信号被周期抽样，切转化成一系列数字$x(n)(n=0,1,...)$。数字处理器依据滤波器的计算算法，执行滤波运算、把输入系列$x(n)$映射到输出系列$y(n)$。DAC 把数字滤波后的输出转出成模拟值，这些模拟值接着被模拟滤波器平滑，并且消去不想要的高频分量。

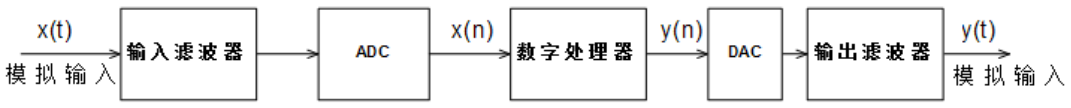


图 2-2 模拟输入和输出的试试数字滤波器简化框图

在本系统中，模拟输入为自然信号，通过电压表示后直接成为输入信号 $x(n)$，输出信号也直接是序列 $y(n)$。

数字滤波器分为两大类：无限冲击响应（IIR）和有限冲击响应（FIR）滤波器。对于 IIR 滤波器：

$$y(n)=\sum_{k=0}^{\infty}h(k)x(n-k)$$

对于 FIR 滤波器：

$$y(n)=\sum_{k=0}^{N-1}h(k)x(n-k)$$

如上所说，本系统都是有限序列长，所以使用 FIR 滤波器。

对于一个基本的 FIR 滤波器，

$$y(n)=\sum_{k=0}^{N-1}h(k)x(n-k)$$
$$H(z)=\sum_{k=0}^{N-1}h(k)z^{-k}$$

式中 $h(k)$ 是滤波器的冲击响应系数。$H(z)$ 是滤波器的传递函数，N 是滤波器的长度，即滤波器系数的数目。上式是 FIR 时域差分方程，使用非递归形式描述：当前输出信号 $y(n)$ 只是过去和当前的输入值 $x(n)$ 的函数。下式是滤波器的传递函数，提供了分析滤波器的一种方法，例如评估频率相应。

2.3.2 使用窗口方法的 FIR 滤波器设计

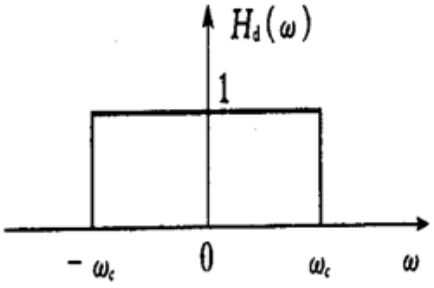
大多数 FIR 系数的计算方法中，唯一的目标是求 $h(n)$ 的值，使得导出的滤波器满足设计规范，例如幅度-频率相应和吞吐率的要求。求 $h(n)$ 的有效方法有几种，其中之一就是窗口方法。

在这种方法中，利用这样一个事实，滤波器的频率相应 $H_D(\omega)$ 和相应冲激相应 $h_D(n)$ 通过傅里叶反变换联系起来：

$$h_D(n)=\frac{1}{2\pi}\int_{-\pi}^{\pi}H_D(\omega)e^{j\omega n}d\omega \qquad (2.3.1)$$

下标 D 用来区别理想的和实际的冲激相应。如果已知$H_D(\omega)$，就可以通过计算傅里叶反变换求得$h_D(n)$。

对于本系统，使用的是如 2.3.2 的理想低通相应：



其中ω_c是截止频率，且频率刻度被归一化为$T=1$。通过令响应从$-\omega_c$到ω_c，我们简化积分运算。这样，冲激相应为

$$\begin{aligned} h_D(n) &= \frac{1}{2\pi}\int_{-\pi}^{\pi} 1\times e^{j\omega n}d\omega \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j\omega n}d\omega \\ &= \frac{2f_c\sin(n\omega_c)}{n\omega_c}, n\neq 0, -\infty \le n \le \infty \\ &= 2f_c, n=0 \end{aligned} \qquad (2.3.2)$$

在本系统中，窗口加在 FFT 后的频域里，低通后进行反变换，既可以处理得到低通滤波后的数据和图形。

2.4 卡尔曼滤波算法

2.4.1 概述

1960 年，卡尔曼发表了用递归方法解决离散数据线性滤波问题的论文（A New Approach to Linear and Prediction Problems）。在这篇文章里，一种克服维纳滤波的新方法被提出，也就是现在人们所说的卡尔曼滤波方法。卡尔曼滤波应用非常广发并且功能强大，它可以估计信号的过去和当前状态，甚至预估计将来的状态，即使是在不知道模型的性质情况下。

从本质上说，滤波就是一个信号处理和变换（去除或减弱不想要的成分，增强所需成分）的过程，这个过程既可以通过硬件或软件实现。卡尔曼滤波属于软件实现。基本思想是：以最小均方误差为最佳估计准则，采用信号与噪声的状态

空间模型，利用前一时刻的估计值和当前时刻的观测值来更新对状态变量的估计，求出当前时刻的估计值，算法根据建立的系统方程和观测方程对需要处理的信号做满足最小均方误差的估计。

2.4.2 滤波模型的建立

卡尔曼滤波器包括两个主要过程：预估与校正。预估过程主要是利用时间更新方程建立对当前状态的先验估计，及时向前推算当前状态变量和误差协方差估计的值，以便为下一个时间状态构造先验估计值；校正过程负责反馈，利用测量更新方程在预估过程的先验估计值及当前测量变量的基础上建立起对当前状态的改进的后验估计。这样的一个过程，我们称之为预估－校正过程，对应的这种估计算法称为预估－校正算法。以下给出离散卡尔曼滤波的时间更新方程和状态更新方程。[4]

时间更新方程：

$$\hat{X}_k^- = A\hat{X}_{k-1}^- + B\hat{U}_{k-1} \tag{2.4.1}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{2.4.2}$$

状态更新方程：

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-2} \tag{2.4.3}$$

$$\hat{X}_k = \hat{X}_k^- + K_k(Z_k - H\hat{X}_k^-) \tag{2.4.4}$$

$$P_k = (I - K_k H)P_k^- \tag{2.4.5}$$

上式中各变量说明如下：

X_k 是系统状态变量（-代表先验；^代表估计）。

A：作用在 X_{k-1} 上的 n×n 状态变换矩阵。

B：作用在控制向量 U_{k-1} 上的 n×1 输入控制矩阵。

H：m×n 观测模型矩阵，它把真实状态空间映射成观测空间 P_k。

P_k^-：为 n×n 先验估计误差协方差矩阵。

P_k：为 n×n 后验估计误差协方差矩阵。

I：n×n 阶单位矩阵。

K_k：n×m 阶矩阵，称为卡尔曼增益或混合因数，作用是使后验估计误差协方差最小。

对于系统模型是非线性的情形，使用扩展的卡尔曼滤波（Extended Kalman Filter）器给出解法。同 Taylor 级数类似，面对非线性关系时，我们可以通过求过程方程和量测方程的偏导来线性化，并且计算当前估计量。不同于基本卡尔曼滤波（KF）过程，扩展的卡尔曼滤波（EKF）过程中的因子矩阵（A,W,H,K）是时刻变化的，因此加下标 k（表示 k 时刻）以示标记。EKF 的与 KF 的主要区别在于非线性情形下需要进行线性化处理，且因子矩阵一般都随时间变化（与 k 有关）。但是经线性变换后系统噪声及量测噪声不再服从高斯分布。

2.5 风速风向的矢量统计分析

2.5.1 风速风向的传统平均方法

算术平均

算数平均方法指的是把规定时间间隔内对风速风向的采样结果求代数和除以测量次数，其平均结果代表测量点位置的风，计算式为：

$$\bar{F} = \sum_{i=1}^{n} f_i / n \tag{2.5.1}$$

式中左边为风速或者风向的算数平均值，f_i 为每次的采样值，n 是规定时间段内的采样次数。

这种方法在进行风向平均时，如果采样值在 0° 两边摆动时，必需进行特殊处理。对于平面直角所标西，通常采用将第一象限的角度加上 360° 再与第四象限的角度相加的方法，由于所得结果可能大于 360°，也可能小于 360°，还需进一步判断才能得到正确结果。[5]

滑动平均

滑动平均仍属于算术平均，但它是按特定的方式变形之后的算术平均。采样来源和要求与算术平均相同，其计算公式为：

$$\bar{F} = F_{n-1} + \frac{1}{N}(F_n - \bar{F}_{n-1}) \tag{2.5.2}$$

式中，$\bar{F}$ 为新的滑动平均值；$\bar{F}_{n-1}$ 为上一次的滑动平均值；F_n 为新的采样值；$1/N$ 为滑动加权系数。

若将式(2.5.2)进行变换则成为：

$$\bar{F} = \frac{N-1}{N}\bar{F}_{n-1} + \frac{1}{N}F_n \tag{2.5.3}$$

若 N 取值交大，则 $N-1 \approx N$，则平均值可表示为：

$$\bar{F} = \bar{F}_{n-1} + \frac{1}{N}F_n \tag{2.5.4}$$

公式(2.5.4)即是滑动平均的实用公式，新的采样值 F_n 对最终平均值的影响可以通过变动 N 值进行调整， N 值越大最后一次采样值对于平均结果的影响越小。每个采样值在平均值中的影响随着时间的延长越来越小，新的采样值至权为零的采样值间的时间即是滑动平均值的间隔时间。

在进行风向滑动平均时通常将公式(2.5.4)变换如下形式：

$$\bar{d} = d_{n-1} + k\bullet(d_n - d_{n-1}) \tag{2.5.5}$$

上式中 $d_n - d_{n-1}$ 是风向的变化项，即当得到新的采样值时，首先计算两者的差，然后乘以一个系数，再与原来的平均值相加得到新的平均值，虽然可以避免新的采样值改变象限的问题，但必须进行复杂的判断。

2.5.2 传统平均方法存在的问题

在以往的测量风速风向数据处理方法中，滑动平均值的工作方式就如电子线路中的电阻-电容滤波器，利用电路时间常数，对高频波动产生幅度衰减，对低频波动产生相位滞后，滑动平均得到的结果，把变化的风向风速值进行了平滑处理，这种平均方法显示的数据，在最初的一段数据内都是不准确的，只有在较长时间段后才显示较为正确的风速风向的结果。

由于风向在很短的两次采样内出现变化等于或接近 180° 的情况是存在的，滑动平均方法仍然要处理这种情况。并且处理起来比算数平均更为麻烦。因为一旦出现就要立即解决，可能出现不正常的情况，因为 270° 与 90° 平均的结果的确是 0° 。而算术平均却可以将众多的数据混合在一起统计，把错误掩盖在平均结果中以较大的数据混合在一起的统计，把错误掩盖在平均结果中以较大误差的形式表现出来。随着采样间隔的增大，这种情况将明显增加，因而，在采用滑动平均方法时，采样间隔必需很短。

算术平均计算的特点是每次采样值对于平均值的影响是相同的。其缺点在于，必需等待一个平均时间才能得到一次平均结果。如果按照世界气象组织要求每秒采样一次，不论取 10 min 还是取 2 min 时段，都可能出现风向变化接近或超过 180°的情况。对于算术平均来说，风速往往不会出现问题，风向就可能出现错误。两次间隔为 1s 时的连续采样值正好相差或接近 180°，造成风向平均值不可判断的情况很少，但在 2 min（1 min）间隔的 120（60）个采样值或 10 min 间隔的 600 个采样值中就不能得以保障。在小尺度湍流作用下，风向在短时间内会频繁的变化，由此得到的平均值结果必然会包含较大的误差以致错误。

测量时，不论风速是否为零，风向标总会显示一个角度，在风速为零时，风向肯定是一个错误的值，因为这个时候没有风向可言，因此计算时必须将其舍弃。这样，无论是算数平均还是滑动平均，规定的采样间隔和样本数就不符合规定了。对于两次采样数据风向过 0°的情况，算术平均方法只要判断准确和处理得当也可以得到正确的结果。而滑动平均方法必须在新的采样值前乘以一个小数，这就会带来问题，因为相乘之后，新的采样值可能从一个象限的值变为其它象限的值，对于第四象限的值就无法判断两次采样之之间是否过北（0°）了。这就必须采取更加复杂的判断方法了。

2.5.3 风速风向的矢量平均方法

实际上，风速风向是一个统一的参数，在测量中却成了两个互相独立的值，且单位和量纲都不一样，风参数的完整描述最确切的表达应该是风的矢量。设风的方向为 d，速度为 V，则风矢量 $\vec{F}(d,V)$ 可用图 2-3 表示。

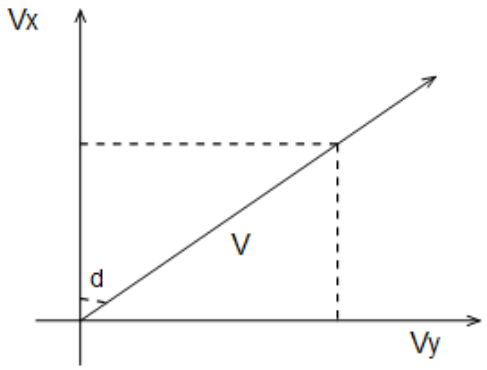


图 2-3 风矢量 $\vec{F}(d,V)$

图中 d 为风向，V 为风速，风矢量 $\vec{F}$ 可用 V_x 和 V_y 两个分量表示其大小，其计算公式是：

$$\begin{cases} V_x = V\cos d \\ V_y = V\sin d \end{cases} \tag{2.5.6}$$

矢量平均方法是将规定时段的多组风速分量，先用代数法对相同分量进行平均求出两个平均值 $\overline{V}_x$ 和 $\overline{V}_y$，即：

$$\begin{cases} \overline{V}_x = \sum_{i=1}^{n} V_{xi} / n \\ \overline{V}_y = \sum_{i=1}^{n} V_{yi} / n \end{cases} \tag{2.5.7}$$

其中 $i = 1,2,3,.....,n$，n 为采样的次数。

然后将两个风速分量的平均值合成风向风速，就可得到用矢量法合成的风向风速值。风速 V 的计算公式是：

$$V = \sqrt{V_x^2 + V_y^2} \tag{2.5.8}$$

合成风向 d 的公式是：

$$d = \arctan\frac{V_y}{V_x} \tag{2.5.9}$$

值得注意的是，由于风向是在 360° 变化的，必需根据两个风速分量的不同方向进行判断，在规定南北分量气流向北为正值，向南为负值；东西分量气流向东为正值，向西为负值的情况下，其判断方法如下：

若 $V_x > 0, V_y > 0$，则 d 的值不变；若 $V_x < 0, V_y < 0$ 或 $V_x < 0, V_y > 0$，则 d 的值加 180°；若 $V_x > 0, V_y < 0$，则 d 的值加 360°。

用矢量法计算风向风速的平均值，其特点是能够始终保持着风向和风速的密切关系，在两个风速分量都为零时，其矢量可表示为 $\vec{F}(0,0)$，这就避免了算术平均和滑动平均对风速为零风向数值无法处理，必需舍弃而使在平均时段内采样间隔和次数不符合要求的情况。同时，在平均过程中也不再会出现风向过 0° 的问题了。因此，用矢量法计算风向风速的平均值，将是很好的选择。

2.6 C++特性和接口

2.6.1 Windows 下 C++开发环境

C++语言特性

C++是在面向对象的程序设计思想的基础上提出来的新一代编程语言，是 C 语言的升级版，保留了 C 语言所具有的所有优点，并且在此基础上增加了面向对象的机制，因其是从 C 语言发展而来，所以与 C 兼容，用 C 语言编写的程序基本上不经过任何修改就能用于 C++的编译。总体上将，C++对 C 的“增强”主要表现在以下两个方面：一是在原来面向过程的基础上，对 C 的功能做了不少扩充；二是增加了新的面向对象的机制。

其中面向对象的程序设计相比于以往面向过程的程序设计思想是一个质的飞跃，它可以用这样的公式来描述：

程序 = 算法 + 数据结构

其主要思想是将我们日常所见到的失误所拥有的共同特性分成一个个的类，再以这些类为基础，具体化出一个个的对象，最终的操作是对这些对象进行的。因此，可以抽象出另外一个公式：

程序 = （对象 + 对象 + 对象 + •••） + 消息

公式中消息的作用就是实现对对象的控制。面向对象的程序设计方法还拥有信息隐蔽性、可继承性、重用性以及多态性等多种优点。

由此可见，面向对象的程序设计更加有利于大型应用软件的便知，也体现 C++语言的优势所在。

Visual Studio 2010 开发环境

Visual Studio2010 是由微软公司推出的，目前最流行的 Windows 平台应用程序开发环境。其集成开发环境（IDE）的界面被重新设计和组织，变得更加简单明了。于此同时，它带来了.Net Framework 4.0 和 Mictosoft Visual Studio 2010 CTP，并且支持开发面向 Windows 7 的应用程序。除了自身推出的 Microsoft SQL Server 外，它还支持 IBM DB2 和 Oracle 数据库。

Visual Studio2010 除了延续之前版本的经典功能外，还开发支持许多新功能，包括 C＃4.0 中的动态类型和动态编程、支持多显示器、Office 和 TDD，可以用来创建 Ribbon 界面，以及新增了基于.Net 平台的 F＃。此外，它还增强和完善了 Visual Studio IDE；改进了 Visual Basic 和 C＃语言，使开发人员可以根据自己的爱好选择任意一种语言；改进了代码编写的速度，简化了 Web 开发；改进了对 WPF

和 Silverlight 应用程序的开发。最重要的变化是，它对核心 Visual C++功能的完善，为并行编程提供了更多支持，以充分利用高性能的多核系统。

.Net Framework 框架

.Net Framework 又称.Net 框架，是由微软开发，并致力于敏捷软件开发、快速应用开发、平台无关性和网络透明化的软件开发平台。.Net 框架是以一种采用系统虚拟机运行的编程平台，以通用语言运行库为基础，支持多种语言（C++、C＃、VB、Python 等）的开发。同时，.Net 也为应用程序接口提供了新功能和开发工具。这些革新使得程序设计院可以同时进行 Windows 应用软件和网络应用软件以及组建和 Web 服务的开发。.Net 提供了一个新的反射性的且面向对象程序设计编程接口，其设计得足够通用化从而使许多不同高级语言都得以被汇集。

.Net 框架 4.0 于 2010 年 4 月发布，旨在实现下列目标： 提供一个一致的、面向对象的编程环境，而无论对象代码是在本地存储和执行，还是在本地执行但是在 Internet 上分布，或者是在远程执行的；提供一个将软件部署和版本控制冲突最小化的代码执行环境；提供一个可提高代码（包括由未知的或不完全受信任的第三方创建的代码）执行安全性的代码执行环境；提供一个可消除脚本环境或解释环境的性能问题的代码执行环境；使开发人员的经验在面对类型大不相同的应用程序时保持一致；按照工业标准生成所有通信，以确保基于.Net Framework 的代码可与任何其他代码集成。

2.6.2 CMsChart 类

MSChart 控件

图标由于其直观明了的特性，在实际应用中十分广泛。我们常常希望数据能通过图标来显示其特性。例如在 Delphi 和 C++ Builder 编程中，我们可以很方便地实现数据图表。MSChart 是 Windows 系统中 Visual Studio 自带的一个 ACTIVEX 控件，功能强大，应用广泛，具有以下特点：支持随机数据和随机数组，动态显示；支持所有主要的图标类型；支持三维显示。

MSChart 图标中将数据分组显示。一组相关的数据在绘图中称为一个“系列”，一张图标可由一个或多个系列构成。

若 MSChart 控件被复制一个以为数组数据，则它所显示的数据图形只绘制出一个系列的图表。若一维数组中的数据被绘制为直方图，则数组中的数据会被逐一绘制为直方图中的彩条。若一维数组中的数据被绘制为饼图，则数组会被逐一绘制为饼图中的彩色扇形。

若 MSChart 控件被赋值一个二维数组数据，则它所显示的数据图形将绘制出多个系列的图表。若二维数组中的数据被绘制为直方图，则数组中的数据分组逐一绘制为直方图中的彩条。若二维数组中的数据被绘制为圆饼图，则 MSChart 控件将绘制出多个圆饼图，每个系列的数据会被逐一绘制为饼图中的彩色扇形。

使用 MSChart 可画如下图表：二维或三维条形图、二维或三维折线图、二维或三维面积图、二维或三维阶梯形图、二维或三维联合图表、二维饼图、二维坐标图等。

要使用 MSChart 控件画图，得先将数据存入 MSChart 的 Data 中，然后对其它图形标题、背景、图例、注脚等方面定义，就可以得到所期望的数据分析图形。

MSChart 接口函数

假设 MSChart 关联的变量为 m_Chart。

```
m_Chart.SetTitleText();// 设置标题
// 下面两句改变背景色
m_Chart.GetBackdrop().GetFill().SetStyle(1);
m_Chart.GetBackdrop().GetFill().GetBrush().GetFillColor().Set(255, 255, 255);
// 显示图例
m_Chart.SetShowLegend(TRUE);
m_Chart.SetColumn(1);
m_Chart.SetColumnLabel();
// 栈模式
// m_Chart.SetStacking(TRUE);
// Y 轴设置
VARIANT var;
//不自动标注 Y 轴刻度
m_Chart.GetPlot().GetAxis(1,var).GetValueScale().SetAuto(FALSE);
// Y 轴最大刻度
m_Chart.GetPlot().GetAxis(1,var).GetValueScale().SetMaximum(100);
// Y 轴最小刻度
m_Chart.GetPlot().GetAxis(1,var).GetValueScale().SetMinimum(0);
// Y 轴刻度 5 等分
m_Chart.GetPlot().GetAxis(1,var).GetValueScale().SetMajorDivision(5);
// 每刻度一个刻度线
m_Chart.GetPlot().GetAxis(1,var).GetValueScale().SetMinorDivision(1);
```

```
m_Chart.GetPlot().GetAxis(1,var).GetAxisTitle().SetText("小时"); // Y 轴名称
m_Chart.SetColumnCount(3); // 3 条曲线
// 线色
m_Chart.GetPlot().GetSeriesCollection().GetItem(1).GetPen().GetVtColor().Set(0,
0, 255);
// 线宽(对点线图有效)
m_Chart.GetPlot().GetSeriesCollection().GetItem(1).GetPen().SetWidth(50);
// 数据点类型显示数据值的模式(对柱柱状图和点线图有效)
// 0: 不显示 1: 显示在柱状图外
// 2: 显示在柱状图内上方 3: 显示在柱状图内中间 4: 显示在柱状图内下方
m_Chart.GetPlot().GetSeriesCollection().GetItem(1).GetDataPoints().GetItem(-1)
.GetDataPointLabel().SetLocationType(0);

//设置数据
m_Chart.SetRowLabel();设置坐标显示
m_Chart.GetDataGrid().SetData();设置纵坐标值
//改变显示类型
m_Chart.SetChartType();
```

本软件系统的数据画图模块均使用的是 MSChart 控件和 CMsChart 类。

2.6.3 FFTW 库

FFTW 介绍

FFTW（the Faster Fourier Transform in the West）是由麻省理工学院计算机科学实验室超级计算技术组开发的一套离散傅里叶变换（DFT）的计算库，开源、高效和标准 C 语言编写的代码使其得到了非常广泛的应用，Intel 的数学库和 Scilib（类似于 Matlab 的科学软件）都使用 FFTW 做 FFT 计算。

FFTW 是计算离散 Fourier 变换（DFT）的快速 C 程序的一个完整集合。

1. 它可计算一维或多维、实和复数据以及任意规模的 DFT；甚至包括正弦/余弦变换和离散哈特莱变换（DHT）。
2. FFTW 输入数据长度任意。
3. FFTW 支持任意多维数据。
4. FFTW 支持 SSE、SSE2、Altivec 和 MIPS 指令集。

5. FFTW 还包含对共享和分布式存储系统的并行变换。

FFTW 不是采用固定算法计算变换，而是根据具体硬件和变换参数来调整使用不同算法，以期达到最佳效果。因此，变换被分成两个阶段。首先 FFTW 规划针对目标计算机的最快变换的计算途径，并生成一个包含此信息的数据结构。然后，对输入数据进行变换。改规划可以被多次使用。在一个典型的高性能应用中，总是在执行相同参数条件的任务，因而，相对复杂但结果就显得过于费时。基于此 FFTW 提供基于启发式和先例的快速初始化。总的来说，FFTW 的一个显著特点就是，对某一参数类型的单词变换优势不大，但对于参数相同的多次变换其有更快的平均速度。

FFTW 为了加快用户的使用集成素的，提供了三种不同层次的接口：

1. 连续数据的单一变换的接口。
2. 计算多重和步进阵列数据的高级接口。
3. 支持通用数据布局、多重和步进的顶级接口。[6]

FFTW 基本数据类型

任何调用 FFTW 的程序都要包含它的头文件 fftw3.h 以及它的 dll 库文件。

fftw_complex 默认由两个 double 组成，在内存中顺序排列，实部在前，虚部在后，即 typedef double fftw_complex[2]。FFTW 文档指出如果有一个支持 C99 标准的 C 编译器（如 gcc），可以在#include <fftw3.h>前加入#include <complex.h>，这样一来 fftw_complex 就被定义为本机复数类型，而且与上述 typedef 二进制兼容（指内存排列），经测试不能用在 Windows 下。C++有一个复数模板类 complex<T>，在头文件<complex>下定义。C++标准委员会最近同意该类的存储方式与 C99 二进制兼容，即顺序存储，实部在前，虚部在后，该解决方案在所有主流标准库实现中都能正确工作。所以实际上可以用 complex <double> 来代替 fftw_complex，比如有一个复数数组 complex<double> *x，则可以将其类型转换后作为参数传递给 fftw：reinterpret_cast<fftw_complex*>(x)。测试如下：开两个数组 fftw_complex x1[2]和 complex<double> x2[2]，然后赋相同值，在调试模式下可以看到它们的内存排列是相同的。complex<T>类数据赋值的方式不是很直接，必须采用无名对象方式 x2[i] = complex <double>(1,2) 或成员函数方式 x2[i].real(1);x2[i].imag(2);不能直接写 x2[i].real=1;x2[i].imag=2。fftw_complex 赋值方式比较直接：x1[i][0]=1;x1[i][1]=2。最后，考虑到数据对齐，最好使用 fftw_malloc 分配内存，所以可以将其返回的指针转换为 complex <double> *类型使用（比如赋值或读取等），变换时再将其转换为 fftw_complex*。[7]

FFTW 函数接口

1. 一维复数据的 DFT

```
fftw_plan fftw_plan_dft_1d(int n, fftw_complex *in, fftw_complex *out,int sign, unsigned flags);
```

n 为数据个数，可以为任意正整数，但如果为一些小因子的乘积计算起来可以更有效，不过即使 n 为素数算法仍然能够达到$O(n\log n)$的复杂度。FFTW 对$N=2^a3^b5^c7^d11^e13^f$的变换处理的最好，其中$e+f=0/1$，其它幂指数可以为任意值。

如果 in 和 out 指针相同则为原位运算，否则为非原位运算。

sign 值为 FFT_FORWARD（-1）时表示正变换；值为 FFT_BACKWARD（+1）时表示逆变换。实际上就是变换公式中指数项的符号.需要注意的是 FFTW 的逆变换结果没有除以 N ，即数据正变换再反变换后是原来的 N 倍。

flags 参数一般情况下是 FFT_MEASURE 或 FFT_ESTIMATE。前者表示 FFTW 会先计算一些 FFT 并测量所用时间，以便为大小为 n 的变换寻找最优的计算方法。依据机器配置和变换的大小（n），这个过程耗费约数秒（时钟 clock 精度）。FFT_ESTIMATE 则相反，它直接构造一个合理的但可能是次最优的方案。总体来说，如果你的程序需要进行大量相同大小的 FFT，并且初始化时间不重要，可以使用 FFT_MEASURE，否则使用 FFT_ESTIMATE。后者模式下 in 和 out 数组中的值会被覆盖，所以应该在用户初始化输入数据 in 之前完成。

2. 多维复数据的 DFT

```
fftw_plan fftw_plan_dft_2d(int n0, int n1,fftw_complex *in, fftw_complex*out,int sign, unsigned flags);
fftw_plan fftw_plan_dft_3d(int n0, int n1, int n2,fftw_complex *in, fftw_complex *out,int sign, unsigned flags);
fftw_plan fftw_plan_dft(int rank, const int *n,fftw_complex *in, fftw_complex *out,int sign, unsigned flags);
```

多维数据的 DFT 同一维数据的 DFT 用法类似，数组 in/out 为行优先方式存储。Fft_plan_dft()是一个通用的复 DFT 函数，可以执行一维、二维或多维复 DFT。比如对于图像（二维数据），其变换为 fft_plan_dft_2d(height,width,85),因为原始图像数据为 height*width 的矩阵，即第一维(n_0)为行数 height。

3. 一维实数据的 DFT

```
fftw_plan fftw_plan_dft_r2c_1d(int n, double *in, fftw_complex *out,unsigned flags);
fftw_plan fftw_plan_dft_c2r_1d(int n, fftw_complex *in, double *out,unsigned flags);
```

r2c 版本：实输入数据，复 Hermitian 输出，正变换。

c2r 版本：复 Hermitian 输入数据，实输出数据，逆变换。

n：逻辑长度，不必为物理长度。由于实数据的 DFT 具有 Hermitian 对称性，所以只需要计算 n/2+1（向下取整）个输出就可以了。比如对于 r2c，输入 in 有 n 个数据，输出 out 有 floor(n /2)＋1 个数据（floor 为向下取整函数）。对于原位运算，in 和 out 为同一数组（out 须强制类型转换），所以其必须足够大以容纳所有数据，长度为 $2\times(n/2+1)$，in 数组的前 n 个数据为输入数据，后面的数据用来保存输出。

c2r 逆变换在任何情况下（不管是否为原位运算）都破坏输入数组 in，如果有必要可以通过在 flags 中加入 FFTW_PRESERVE_INPUT 来阻止，但这会损失一些性能，而其这个标志位目前在多维实 DFT 中已不被支持。

比如对于 n=4，in=[1 2 3 4]，out=[10 -2+2i -2 -2-2i]，out 具有共轭对称性，out 的实际内存为 10，0，-2，2，-2，0，共 3 个复数数据。对于 n=5，in=[1 2 3 4 5]，out=[15 -2.5+3.44i -2.5+0.81i -2.5-0.81i -2.5-3.44i]，out 的实际内存为 15，0，-2.5，3.44，-2.5，0.81，共 3 个复数数据。

实数据 DFT 中，首个变换数据为直流分量，是实数；如果 n 为偶数，由 Nyquist 采样定理，第 n/2 个变换数据也为实数；所以可以把这两个数据组合在一起，即将第 n/2 个变换数据作为第 0 个变换数据的虚部，这样一来输入 数组就和输出数组相等（n=n/2*2）。一些 FFT 的实现就是这么做的，但 FFTW 没有这么做，因为它并不能很好地推广到多维 DFT 中，而且存储空间的节省也是非常小以至于可以忽略不计。

一个一维 c2r 和 r2c DFT 的替代接口可以在 r2r 接口中找到，它利用了半复数输出类型（即实部和虚部分开放在不通的区域里），使输出数组具有和输入数组同样的长度和类型。该接口在多维变换中用处不大，但有时可能会有一些性能的提升。

4. 多维实数据的 DFT

```
fftw_plan fftw_plan_dft_r2c_3d(int n0, int n1, int n2,double *in,fftw_complex
```

```
*out,unsigned flags);
fftw_plan fftw_plan_dft_r2c(int rank, const int *n,double *in, fftw_complex
*out,unsigned flags);
```

用法大致与一维情况相同。

本分析系统软件中操作对象都是实数据，并且是一维的，因此主要使用的是一维 DFT 及其逆变换函数。

第三章 数据分析软件程序设计与实现

3.1 系统模块与功能

图 1-2 和图 1-3 已经描述了整个系统的结构图，在实际实现中，必需对系统做更为详细地模块划分模块才能实现有助于编程实现。

系统的模块细分如图 3-1 所示。

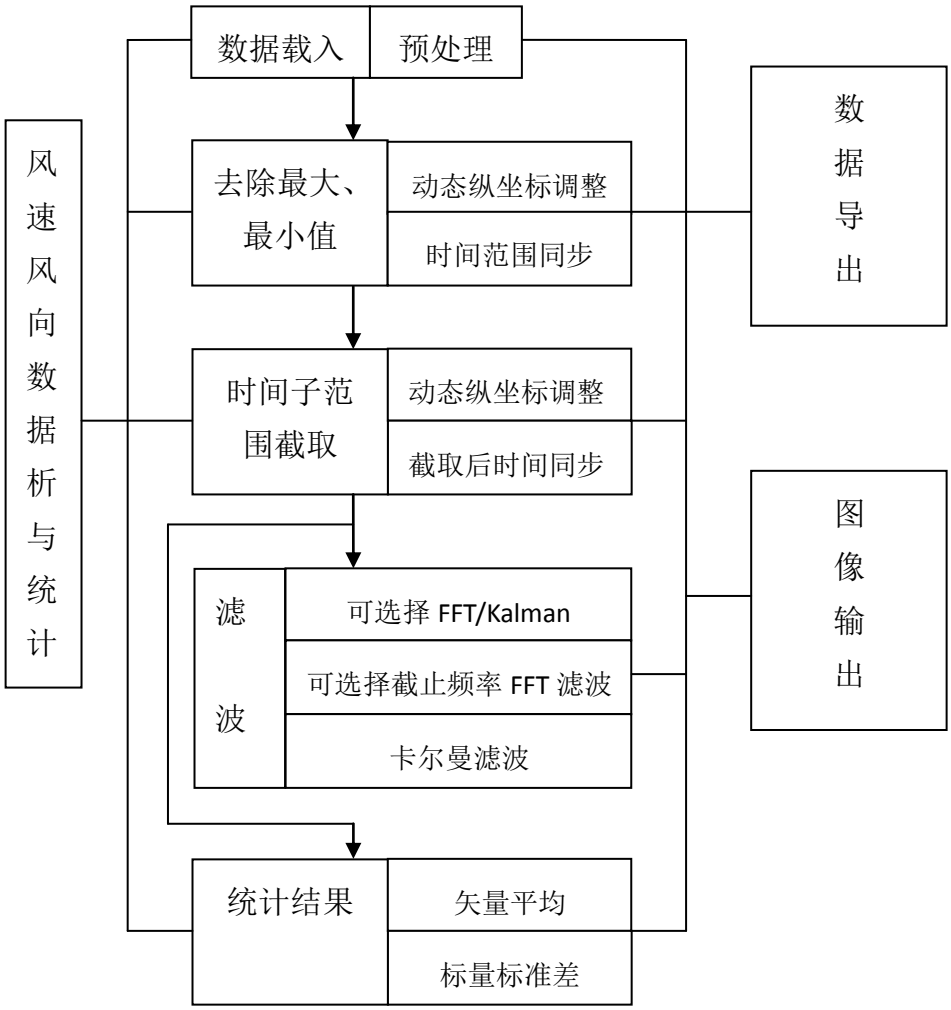


图 3-1 风速风向数据分析与统计系统模块细分

3.2 底层数据封装与接口实现

数据通过 SD 卡中转，搜集到的风速风向数据按照固定格式存在 SD 卡上，然后将其读出。数据格式定义如下：

字段	格式	定量说明
日期	yyyy-mm-dd	e.g. 2012-05-22
时间	hh:mm:ss	e.g. 10:20:00
纬度	xxxx.xxxx	之前有 N 标识
经度	xxxx.xxxx	之前有 E 标志
相对风向	xx.xx	0~30
相对风速	xx.xx	0~360

相对应的，在程序中格式化为图 3-2（左）：

```
class CFrameString
{
public:
    char* date;
    char* time;
    char* weidu;
    char* jingdu;
    char* wind_speed;
    char* wind_dir;
public:
    int year;
    int month;
    int day;
    int hour;
    int minu;
    int sec;
    CTime TimeFormat();
};
```

```
class CFrameList
{
public:
    //CFrameString frameString;
    CFrameEle* head;
    CFrameEle* current;
    char nodeId[5];
    fstream file;
    int nRow;//0~nRow
public:
    CFrameList();
    ~CFrameList();
    void FileToList();
    void setNodeId(CString fileName);
    CFrameString* getFrameString(int index);

    //return new list after splitting
    CFrameList* getNewFrameList(CTime tFrom, CTime tTo);
    void ListScanner();
};
```

图 3-2 数据元格式定义（左）；数据集格式和接口定义（右）

后面的数据操作都从这个数据类中读入/读出操作。由于数据是一个由这种数据类型构成的链表，所以还需定义一个数据集类来操作这些数据元，并且向上提供操作的接口，如图 3-2（右）。

其中 FileToList()函数提供将文件转化为数据集的操作；getFrameString()输入值是整形变量，返回值是当前 index 的数据元数据；ListScanner()对转化的数据集做预处理的扫描过程，如果遇到错误的时间记录或重复的时间记录，则予以删除操作。这样可以为上层调用提供稳定、可靠的数据操作。

3.3 最大、最小过滤值处理

3.3.1 动态纵坐标自动调整

对于任何一串或多串数据表集，倘若固定纵坐标刻度，则可能会出现两种错误：一是最大值小于数据表中的最大值或最小值大于数据表中的最小值；二是数据表的值域远小于固定值的值域。对于第一种情况，数据不能完全显示出来或者出现错误显示，因为其超出了阈值；对于第二种情况，可见数据图像非常小，影响视觉观察和分析。因此采取自动纵坐标范围调整的算法。

为了在数据表值域范围之内，首先要得到数据表的最大值和最小值，然后动态的设定纵坐标的值：

```
double speedMinimum = GetMinimum(flist[0], flist[1], flist[2], flist[3], 0) * 9/10;
double dirMinimum = GetMinimum(flist[0], flist[1], flist[2], flist[3], 1) * 9/10;
double speedMaximum = GetMax(flist[0], flist[1], flist[2], flist[3], 0) * 1.1;
double dirMaximum = GetMax(flist[0], flist[1], flist[2], flist[3], 1) * 1.1;
m_ChartUp.GetPlot().GetAxis(1,var).GetValueScale().SetMinimum(speedMinimum);     // Y轴最小刻度
m_ChartUp.GetPlot().GetAxis(1,var).GetValueScale().SetMaximum(speedMaximum);
m_ChartDown.GetPlot().GetAxis(1,var).GetValueScale().SetMinimum(dirMinimum);
m_ChartDown.GetPlot().GetAxis(1,var).GetValueScale().SetMaximum(dirMaximum);
```

图 3-3 动态纵坐标值代码

流程图为：

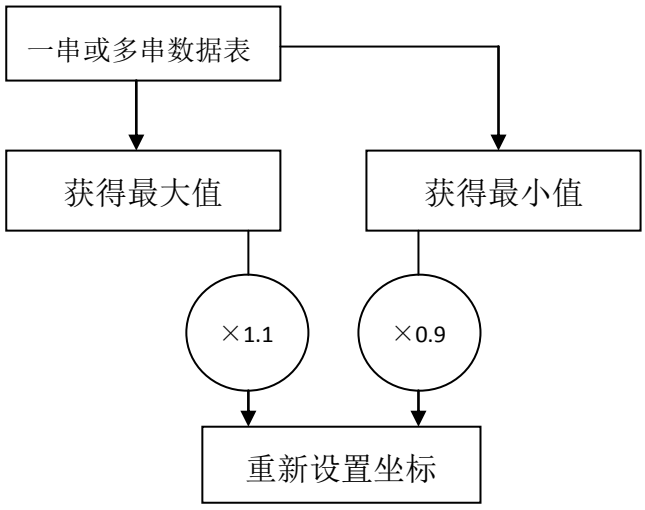


图 3-4 动态坐标值设置流程

采用以上的算法，可以动态设置坐标范围。数据不作处理前的范围如图 3-5（左）显示，默认最大值和最小值为规定 30 和 0。新设置最大值和最小值为 9 和 8，对其进行去除值操作后，纵坐标范围如图 3-5（右）所示。如此可以充分观察和分析曲线变化。

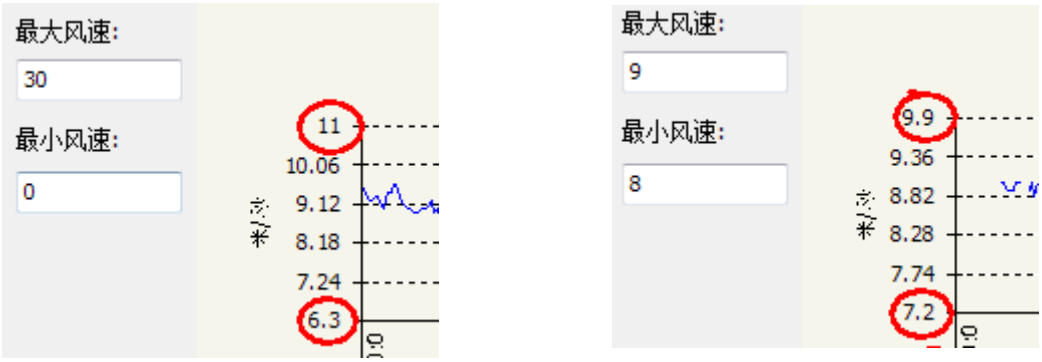


图 3-5 去除值前坐标轴范围（左）；去除值后坐标值动态范围（右）

3.3.2 多数据表时间同步

当数据表只有一串时不存在这样的问题，因为时间起始值和结束值都只有一个；而当数据集有两个或以上个数时，制定时间范围就成为一个问题。为了将每条数据集都显示出一张图上，对时间范围做一下处理：

$$TimeFrom = Min(one, two, three, four)$$
$$TimeTo = Max(one, two, three, four)$$

即起始值为所有数据集中最小值，结束值为所有数据集中最大值。

3.3.3 去除不正常数据值

对于设定数据范围之外的数据，可以选择不将其显示出来来达到过滤的目的。通过改变 SetData()函数的最后一个入参的值（为 0 为显示，为 1 为不显示）来完成这一功能。

```
if(iSpeed>m_nMaxUp||iSpeed<m_nMinUp)
{
   m_ChartUp.GetDataGrid().SetData(i,j , (m_nMaxUp-m_nMinUp)/(3.14/2)*atan(iSpeed)+m_nMinUp, 1);
   continue;
}
m_ChartUp.GetDataGrid().SetData(i,j, iSpeed, 0);
```

图 3-6 利用判断和参数设置过滤过大、过小值

例如过滤前的显示如图 3-7（左），滤值后的值如图 3-7（右）。

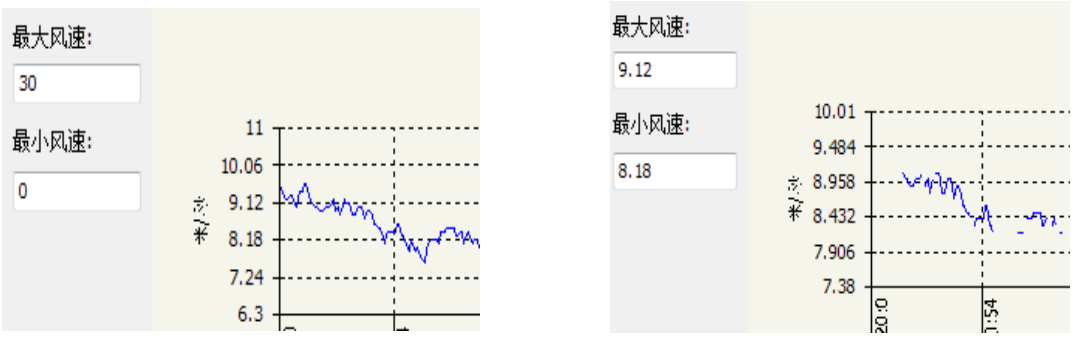


图 3-7 去除最大、最小值前（左）和去除后（右）

3.4 多数据表时间段截取的同步

每次重绘时程序都使用 DrawChart()函数进行图形绘制，在绘制时都会对入参数据集中的时间进行检查并计算其时间段。对于多个数据表的持续时间段，可如图 3-8 实例：

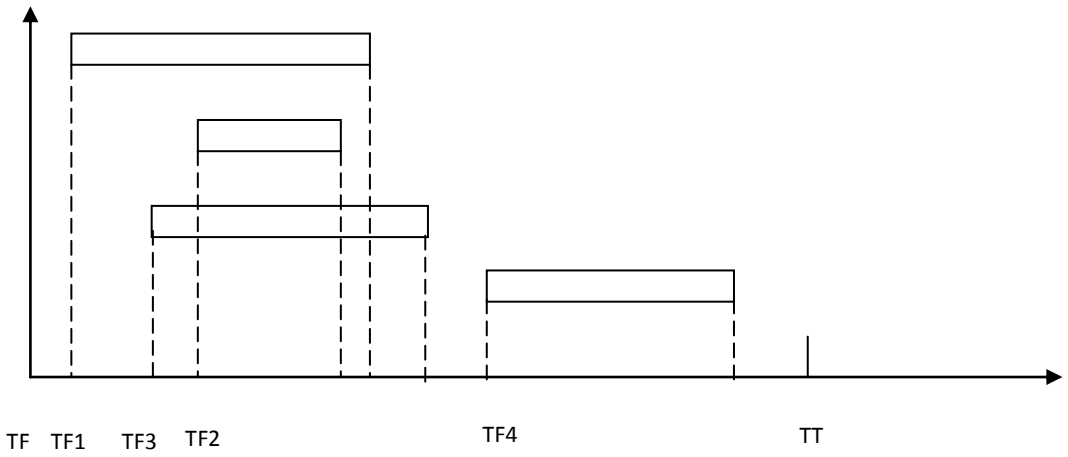


图 3-8 多数据表始末时间值与截取的关系

由于每一段新的得到数据值的起点至少在整体截取时间段最小值的右侧，即 $TF \le TF_i$，终点至少在整体截取时间段最大值的左侧，即 $TT_i \le TT$（它们是同时得到的），所以可以将它们独立考虑（数据集之间不受影响）。以数据集 1 为例。

首先判断 TF 和 TF1 之间的数据，$TF \le TF_i$，首先对 $TF \le n \le TF_i$ 内的数据都置

为不显示（即 SetData()函数最后一个参数为 1），然后对$TF_i \le n \le TT_i$内的值置为原值，并显示，最后对$TT_i \le n \le TT$ 内的值再次置为不显示。如此对每个数据集都做此操作，则可实现截取时间段时间同步的问题。

其程序代码为：

```
for(i = 1; i< segOne; i++)
{
   m_ChartUp.GetDataGrid().SetData(i,j, 0, 1);
}
for(int k = 0;k<=framelist[j-1]->nRow;k++,i++)
{
   char* speed = framelist[j-1]->getFrameString(k)->wind_speed;
   double iSpeed = atof(speed);
   if(iSpeed>m_nMaxUp||iSpeed<m_nMinUp)
   {
      m_ChartUp.GetDataGrid().SetData(i,j , (m_nMaxUp-m_nMinUp)/(3.14/2)*atan(iSpeed)+m_nMinUp, 1);
      continue;
   }
   m_ChartUp.GetDataGrid().SetData(i,j, iSpeed, 0);
}
for(; i<= nRow; i++)
{
   m_ChartUp.GetDataGrid().SetData(i,j, 0, 1);
}
```

图 3-9 时间截取同步的代码模块

三个红色框内即算法中三个步骤。其截取的效果图如图 3-10。

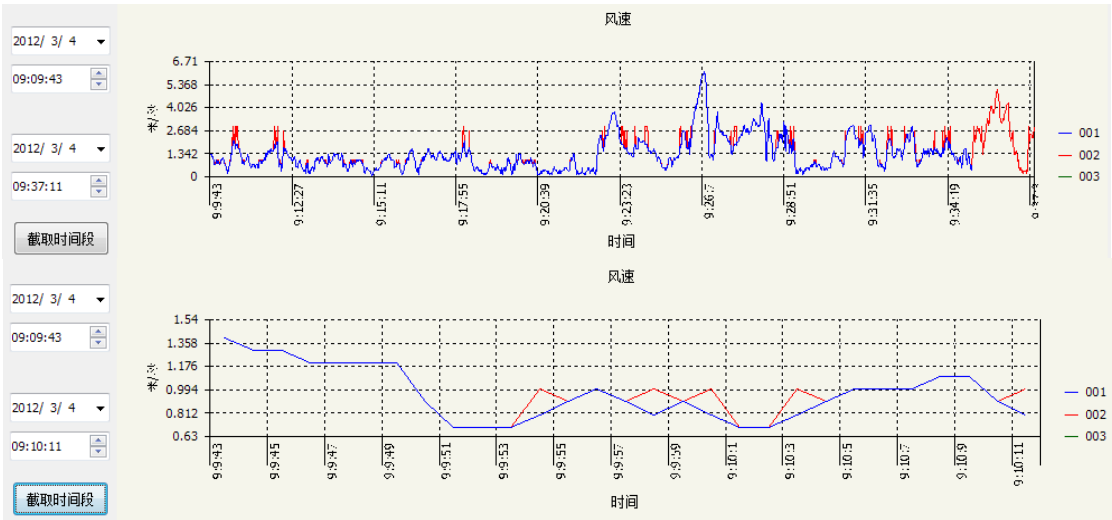


图 3-10 三个数据集时间截取同步前（上）和截取后（下）

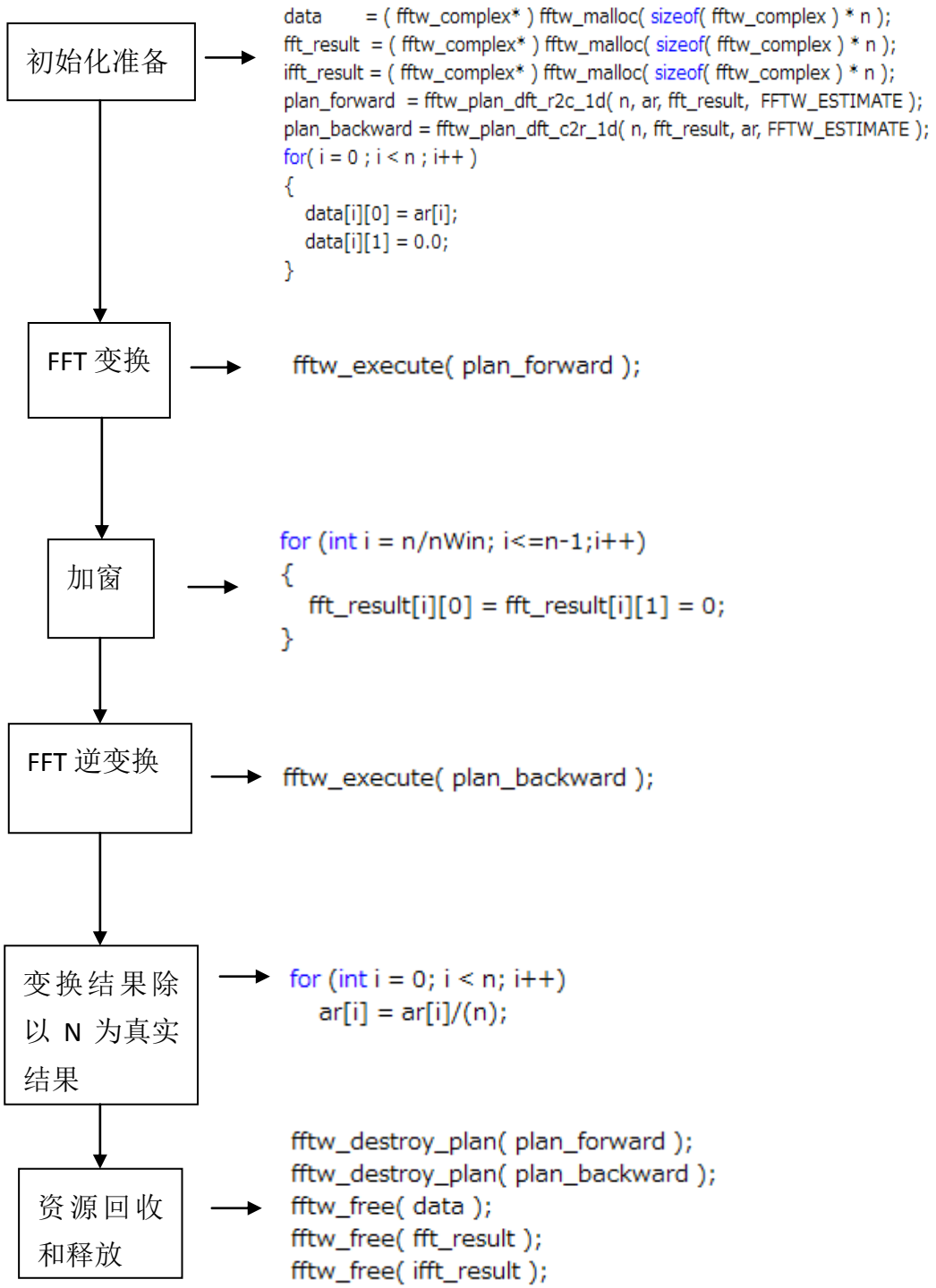
3.5 可选择截止频率的 FFT 和卡尔曼滤波图像处理

滤波功能为用户提供两种滤波方式选择。可通过下拉框进行选择。

3.5.1 可选择截止频率的 FFT 滤波处理

采集到的数据都是连续的实数据，所以使用FFTW库中的一维实数据函数fftw_plan_dft_r2c_1d()进行FFT正变换,使用fftw_plan_dft_c2r_1d()进行FFT逆变换，两者之间加窗函数进行频域滤波，这样就组成了采用矩形窗的。

为了方便程序使用FFTW，同时封装上自己的窗函数，另写一个头文件fft.h，实现fftLow()方法：



使用加窗FFT滤波的效果如图3-11所示。

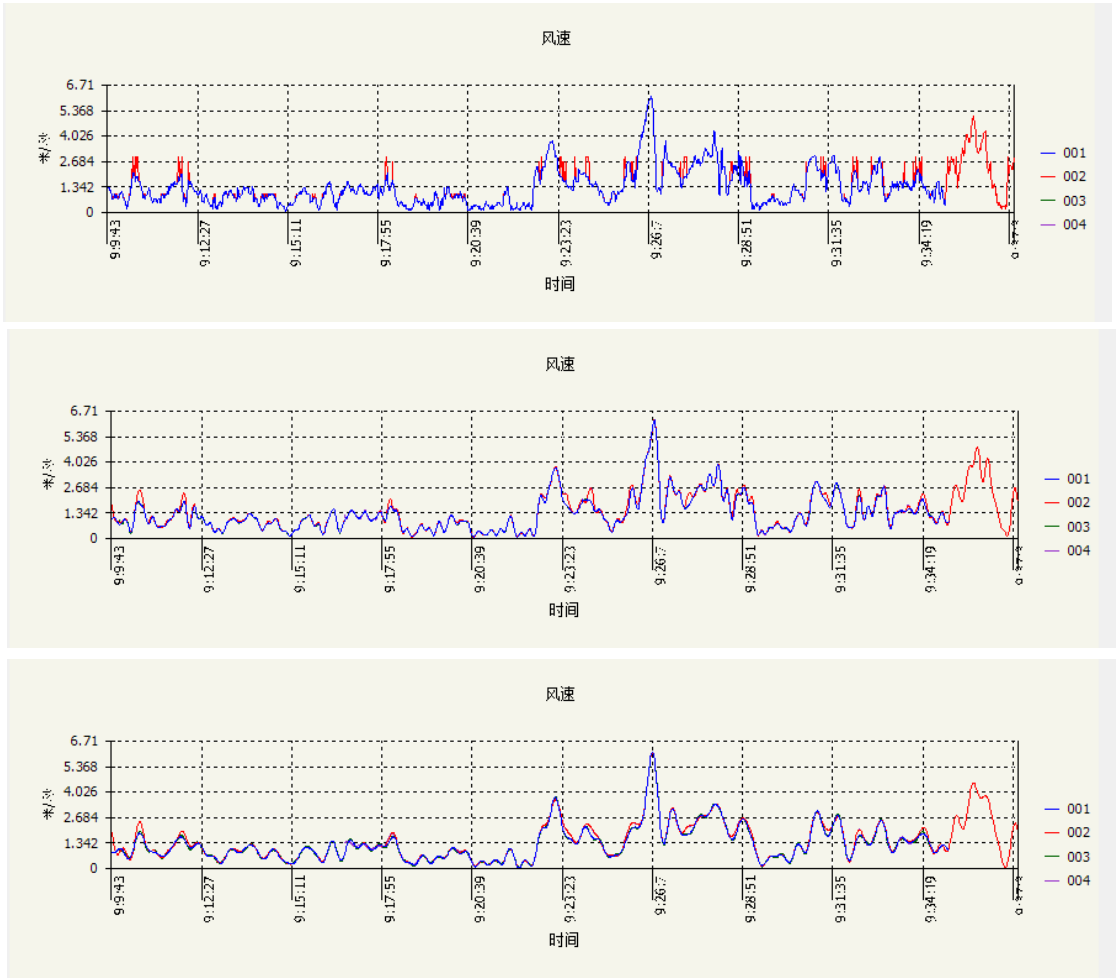


图3-11 滤波前曲线图（上）；n/16加窗（中）；n/25加窗（下）

n/M、n/M表示对于一个n点的序列，做了FFT变换后n个频域值，加上式3.5.1的矩形窗：

$$F(\omega)=\begin{cases}1, \omega \le n/M \\ 0, n/M < \omega \le n\end{cases} \tag{3.5.1}$$

因此M越大时，截止频率n/M就越小，通带越窄，滤波效果就越明显。从图3.5.1中也可以看出n/25截止频率滤波比n/16截止频率滤波明显、平滑了许多，因为它滤掉了更多的高频成分。

3.5.2 卡尔曼滤波处理

对于一维的数据，卡尔曼滤波的结构会比多维的简单很多。每次按照预估计

的协方差值和实际测量值会得到一个新的估计值，同时得到新的协方差值，然后利用这个协方差值和下一个测量值会得到新的估计值，这就是利用迭代和递归的卡尔曼滤波算法。[8]

本系统中按照的计算规则如下所示：

$$x = x \tag{3.5.2}$$

$$p = p + q \tag{3.5.3}$$

$$k = p / (p + r) \tag{3.5.4}$$

$$x = x + k \times (\text{测量值} - x) \tag{3.5.5}$$

$$p = (1 - k) \times p \tag{3.5.6}$$

式 3.5.2 和 3.5.3 代表的卡尔曼滤波器的预测值。由于本算法中不加入控制状态，所以显得较为简单，后面三个式子计算更新后的方差值和估计值。计算后，卡尔曼滤波器状态得到刷新。代码如下：

```
typedef struct {
  double q; //process noise covariance
  double r; //measurement noise covariance
  double x; //value
  double p; //estimation error covariance
  double k; //kalman gain
} kalman_state;
```

（1）

```
kalman_state kalman_init(double q, double r, double p, double intial_value)
{
  kalman_state result;
  result.q = q;
  result.r = r;
  result.p = p;
  result.x = intial_value;

  return result;
}
```

（2）

```
void kalman_update(kalman_state* state, double measurement)
{
  //prediction update
  //omit x = x
  state->p = state->p + state->q;

  //measurement update
  state->k = state->p / (state->p + state->r);
  state->x = state->x + state->k * (measurement - state->x);
  state->p = (1 - state->k) * state->p;
}
```

（3）

图 3-12 卡尔曼滤波模块代码

图 3-12 中，（1）内是对卡尔曼滤波器的定义，x 是状态值（filtered value），q 是过程噪声（process noise），r 是实际噪声（actual noise），p 是预估计误差（estimated error），k 是卡尔曼增益（Kalman Gain）。

（2）是对滤波器的初始化过程，定义状态值、噪声值、误差的初始值。通常将状态值初始化为输入数组的第一个元素。

（3）就是卡尔曼得到新值和新状态值的过程，对应于式 3.5.4 到 3.5.6。

卡尔曼滤波模块对外提供的接口是输入一个数组，输出滤波后的数组，保存在输入数组中，通过图 3-13 的代码对其封装：

```
void kalman_filter(double ar[], int n)
{
  kalman_state state;

  state=kalman_init(0.06,4.0,0.46,ar[0]);

  for (int i = 1;i<100;i++)
  {
    kalman_update(&state,ar[i]);
  }
  state = kalman_init(state.q,state.r,state.p,ar[0]);
  for (int i = 1;i<n;i++)
  {
    kalman_update(&state,ar[i]);
    ar[i]=state.x;
  }
}
```

图 3-13 卡尔曼滤波算法对外接口实现

根据式(2.4.1)到式(2.4.5)，卡尔曼滤波器的输入参数有很多，由于现实中环境的难以测量，这些参数很难得到。因此使用预先计算的方法进行预估计，得出在大致区间内的状态值，然后将这些值作为输入初始状态值重新进行滤波。考虑到实际中处理的点非常多，预处理所有的点即降低了效率又没有必要（后面的计算已经趋于稳定在固定范围内），所以只选择计算前 100 或几百个点得到初始状态值，然后作为正式的输入状态参数重新滤波。

根据上述算法，对一个包含 20 个随机数据的序列进行卡尔曼滤波。为了确定初始输入状态值，先自选输入状态作为输入参数，然后对 20 个数据做卡尔曼滤波，得到较为合理的输入状态值，然后重新计算，得到的新的滤波输出结果。

输入和输出值如下图：

```
41.000000    41.000000
18467.000000   2163.719165
6334.000000   2644.146849
26500.000000   5392.412144
19169.000000   6979.518872
15724.000000   7986.912293
11478.000000   8389.097688
29358.000000   10804.789897
26962.000000   12666.159263
24464.000000   14025.314128
5705.000000   13066.782795
28145.000000   14803.850412
23281.000000   15780.450331
16827.000000   15901.016862
9961.000000   15216.704059
491.000000   13520.246075
2995.000000   12307.696962
11942.000000   12265.567252
4827.000000   11408.615383
5436.000000   10720.546837
```

图 3-14 20 个数据的卡尔曼滤波输入输出值

从上图 3-14 看到，滤波效果还是比较明显的。从原始值 41.0 到 18467.0 的变化中，变化非常打，经过滤波器后的输出为 2163.7，明显得到了较为平滑的值。然后从 18467.0 到 6334.0 的原始值变换中，经过滤波器，输出变成了 2644.4，使得变小的趋势得到了缓解，同样得到较为平滑的值。

利用上述算法得到的滤波效果如图 3-15 和图 3-16。

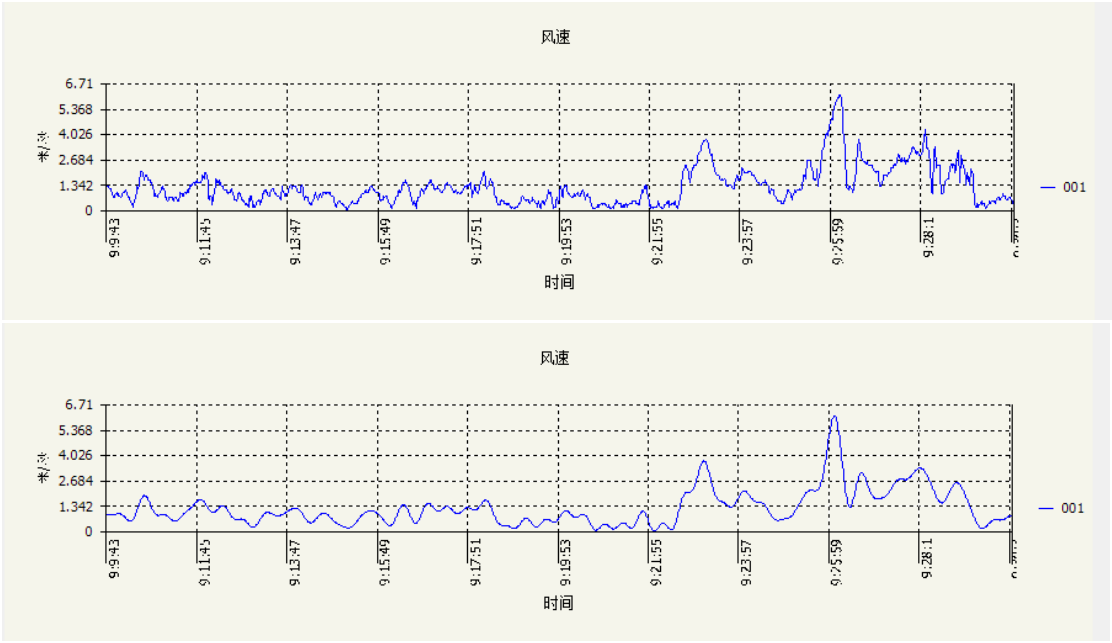


图 3-15 对风速数据卡尔曼滤波前后对比

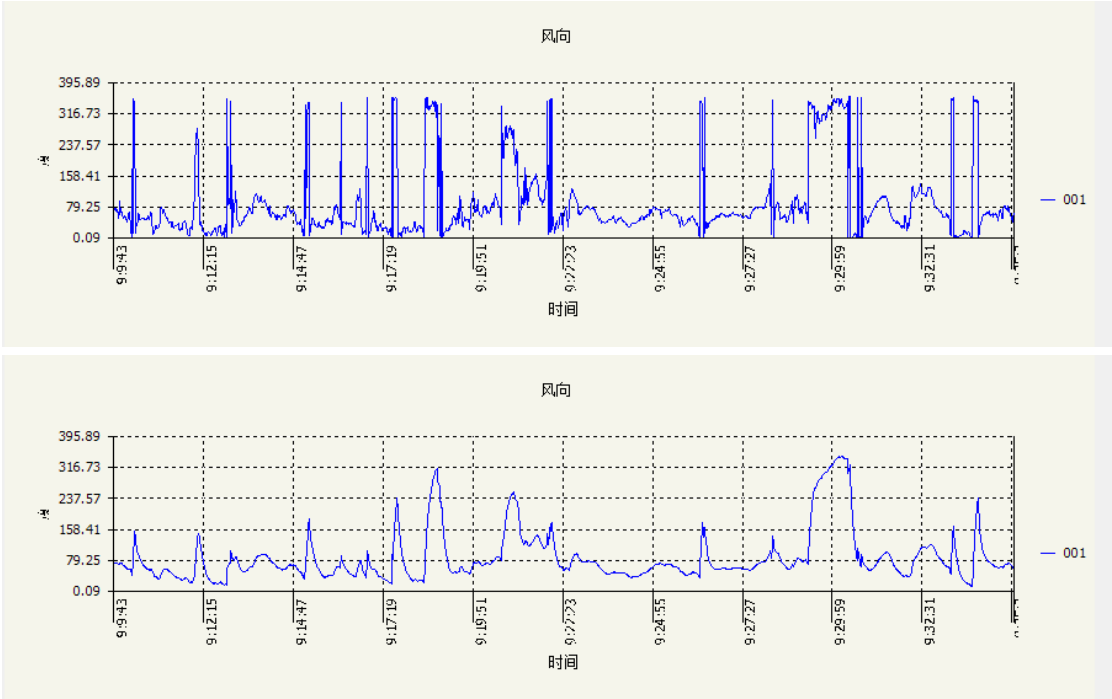


图 3-16 风向数据的卡尔曼滤波前后对比

代码中看到，上图中的许多毛刺都被滤掉，变得更为平滑，这正是因为对于每一个点卡尔曼滤波器都对其重新进行了计算，在之前的较为准确的状态参数基础上。虽然滤波不如 FFT 那样很明显基于频域，但是数据更为可靠。

与图 3-11 相比，卡尔曼滤波是自适应的滤波，因此数据更为可信。FFT 基于所有的数据点进行计算，而卡尔曼是根据上一个和之前的计算结果进行计算，特别适合具有惯性特征的滤波问题，这也是它广泛应用于传感器数据融合、导弹追踪、图像跟踪等领域，这些系统有共同的特性，就是数据具有很强的惯性和连续性。正因为如此，在传感器数据分析中，特别的，在海上风向风速数据分析中，使用卡尔曼滤波器是非常有意义的。

3.6 数据的标量和矢量统计分析及输出

将式 2.5.6 到 2.5.9 描述的矢量统计方法程序实现：

```
for (int i = 0; i <= flist->nRow; i++)
{
   tempSpeed = atof(flist->getFrameString(i)->wind_speed);
   tempDir = atof(flist->getFrameString(i)->wind_dir);
   if (tempSpeed <= m_nMaxUp && tempSpeed >= m_nMinUp && tempDir <= m_nMaxDown && tempDir >= m_nMinDown)
   {
      float rad = atof(flist->getFrameString(i)->wind_dir) * 3.1415 / 180;
      float V = atof(flist->getFrameString(i)->wind_speed);
      Vxi = V*cos(rad);
      Vyi = V*sin(rad);
      _Vxi += Vxi;
      _Vyi += Vyi;
   }
   else
   {
      nExclude++;
      continue;
   }
}
_Vxi /= flist->nRow + 1 - nExclude;
_Vyi /= flist->nRow + 1 - nExclude;
if (i == 0)//speed
{
   return sqrt(_Vxi*_Vxi + _Vyi * _Vyi);
}
if(i == 1)
{
   float degree = atan(_Vyi/_Vxi);
   degree = degree *180 / 3.1415;
   if( _Vxi > 0 && _Vyi > 0)
      return degree;
   else if (_Vxi < 0 && _Vyi < 0)
      return degree + 180;
   else if (_Vxi < 0 && _Vyi > 0)
      return degree + 180;
   else if (_Vxi > 0 && _Vyi < 0)
      return degree + 360;
}
```

（2.6.7）

（2.6.8）

（2.6.9）

图 3-17 矢量统计算法的程序实现

第一个红框内是风速、风向相乘的矢量，通过角度分解得到；红框二内的功能是返回矢量统计结果的风速值；红框三内返回矢量统计结果的风向值，返回前针对不同的象限做了判断。

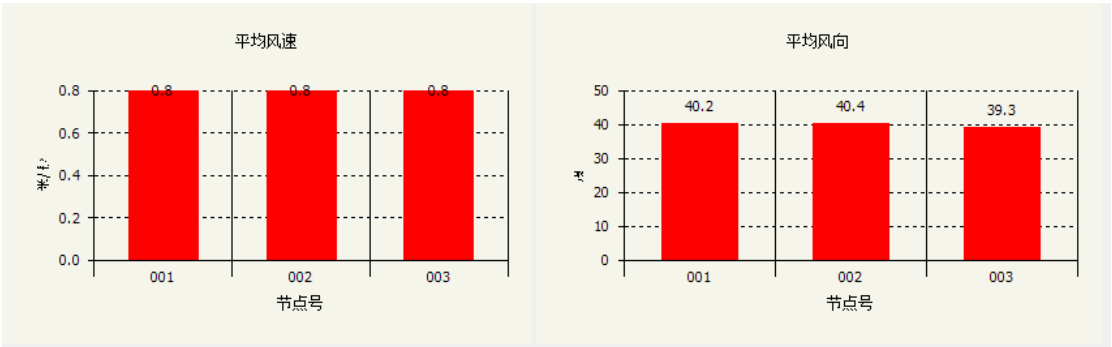
另外，统计标准差时使用矢量统计，而是使用传统意义上的标准差：

```
   if (tempSpeed <= m_nMaxUp && tempSpeed >= m_nMinUp && tempDir <= m_nMaxDown && tempDir >= m_nMinDown)
   {
      v = atof(flist->getFrameString(i)->wind_speed);
      s += (v-sum)*(v-sum);
   }
   else
   {
      nExclude++;
      continue;
   }
}
s /= flist->nRow+1-nExclude;
s = sqrt(s);
```

图 3-18 标准差的标量算法

红框一内是算出每一项的差值（平方），框二先是算出整个的方差，然后开根号得到返回值标准差。

数据统计值同时通过图表和列表表格显示。图表显示矢量平均风向、风速值；表格显示的是各节点最大或最小的风速、风向值以及它们的标准差。如图 3-19 所示。



节点号	最大风速	最小风速	风速标准差	最大风向	最小风向	风向标准差
001	2.1	0.0	0.4	359.2	0.1	85.2
002	2.9	0.0	0.5	359.2	0.1	85.2
003	2.1	0.0	0.4	359.2	0.1	86.2

图 3-19 数据统计图

3.7 数据的导出

数据会以原格式导出到文件中，使用 fstream 输入、输出文件流。将处理后的数据集导出至文件中：

```
fhandle<<m_framelist->getFrameString(i)->date
<<','<<m_framelist->getFrameString(i)->time<<",N"
<<','<<m_framelist->getFrameString(i)->weidu<<",E"
<<','<<m_framelist->getFrameString(i)->jingdu
<<','<<m_framelist->getFrameString(i)->wind_speed
<<','<<m_framelist->getFrameString(i)->wind_dir
<<endl;
```

图 3-20 导出数据集至文件

第四章 系统稳定性和效率分析

4.1 系统稳定性分析

在系统使用的初期表现的并不是很好。其中之一主要表现为程序会不抛错而崩溃。调试中发现，程序在底层函数里抛出异常，主要错误出在临界位置的数据元，多数据集指针没有初始化等。好在系统的模块化实现，使得底层的修改不需要牵连上层的应用。经过测试和实际使用反馈，目前底层的数据和函数接口都已经非常稳定，这样上层调用也不会出现问题。

4.2 系统计算量和资源使用分析

4.2.1 计算量分析

由于每条数据集都有上千至上万条数据，因此系统的计算量还是比较大的。涉及到计算的主要有取最大（小）值，傅里叶变换（逆变换）滤波，卡尔曼滤波，矢量计算等。其中傅里叶变换和卡尔曼滤波的时间不能控制，而其它计算却可以通过算法进行优化。

系统交付使用之处运行速度较慢，经检查发现大部分画图、数据统计都会频繁调用 GetMax()和 GetMinimum()函数来计算最大最小值。而每次要利用函数返回值的时候都显示调用此函数，由于每计算一次都需要将数据表扫一遍，此重复操作很浪费时间，因此将最大、最小值先保存在临时变量（存放在栈里）里，这样只需扫一遍然后从内存（栈）里使用这个值速度会大大提高。事实也证明这一优化的可用性。

4.2.2 系统资源使用分析

除了计算时使用的大量资源外，原始数据表集的保存和子集的保存都需要较大的内存。如此调用的内存如果不回收会造成很大的资源占用和浪费，甚至出现泄漏的危险。由于外部接口提供新建数据集的方法调用接口，使用接口新建数据集的化不好用 delete 函数释放。因为倘若在函数内部 delete，会导致上层在使用前数据集的寿命就已经结束；若在外部手动 delete，则由于不可见性不能判断其大小和长度。

再往下分析，每个数据集都是一个 CFrameList 类，无论是函数怎么操作，其返回和输入及操作的都是 CFrameList 对象，对象使用结束之后系统会自动调用其析构函数，因此选择在 CFrameList 类的析构函数里将内存释放。由于析构函数是最后执行的，因此不会影响数据集的使用。

```
CFrameList::~CFrameList()
{
  file.close();
  CFrameEle* tmp;
  while (current != NULL)
  {
    tmp = current->next;
    delete current;
    current = tmp;
  }
  //delete
}
```

图 4-1 在类的析构函数中释放资源

第五章 总结与展望

5.1 系统概述总结

本系统开发和应用于国家帆船队进行海上风速风向的数据测量和研究。文章按照系统需求、理论基础和实现过程的主要步骤来进行阐述。

首先在系统需求的论述中，根据客户的切实需求和风速风向实际的物理意义确定了开发的方向，尤其是数据的处理和使用要求。确定了采用加窗的傅里叶变换组成数字滤波器处理密集的数据集，并选择适合传感器数据处理的卡尔曼滤波器处理数据。另外，确定了使用矢量计算方法计算风速风向，符合其物理条件和意义。

其次，根据需求论述了系统的主要原理，说明了数据可视化的意义和开发环境还有一些类库的接口特性。另外，着重阐述了数据处理的理论分析，包括离散傅里叶变换（DFT）及其逆（IDFT），加窗方法和它们组成的有限字长数字滤波器（FIR），还有卡尔曼滤波器的基本算法和原理。除此以外还有矢量算法计算风速风向的方法和意义。

最后对系统的实现做了分析和说明，提供了关键性处理代码和算法的实现。实际运行结果也在这一章进行了展示。另外对程序的稳定性分析和效率计算也进行了说明，并提供了提高计算和运行速度的一些方法。

做为总结，示例按照去除错误值、时间段截取、数据滤波、统计分析四个流程步骤执行，如图 5-1 所示。

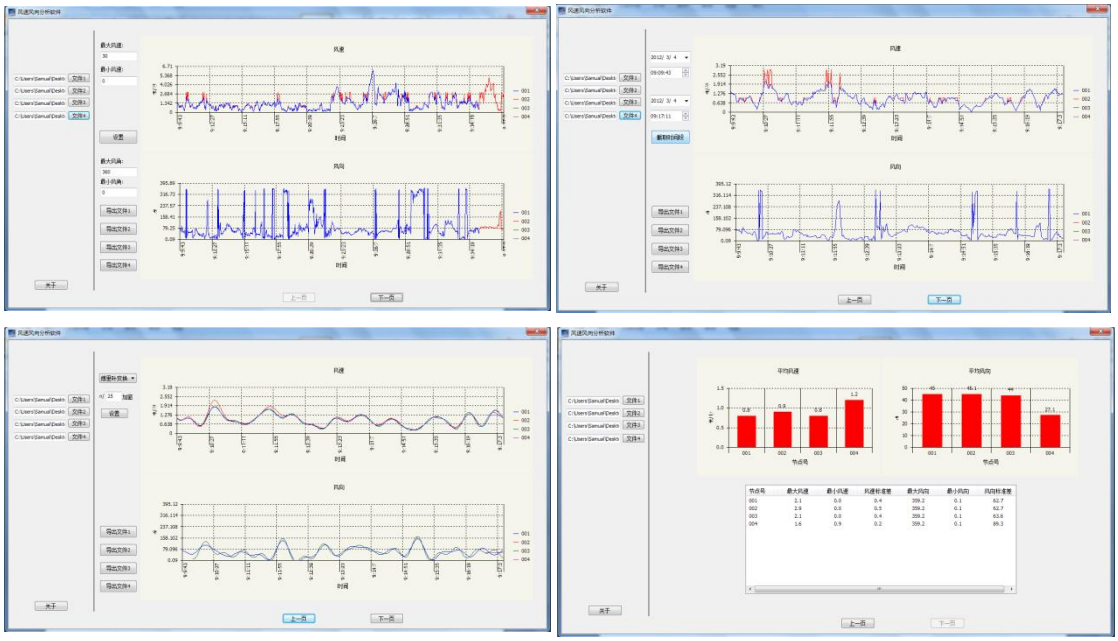


图 5-1 系统工作步骤图

5.2 系统展望

本软件系统作为一款数据分析与统计软件，应该提供更快速、功能更丰富的应用。由于现实需求中最多只有 4 个浮标采集风速风向数据，所以程序中最多限量 4 个输入数据集。为了扩大分析范围，应该支持更多数据集的分析，而随着数据集的增多，应该有更好的算法支持其成倍的计算量增长。

在界面美观上，可以提供更友好的界面设计，像类似于 Web 风格的动态的曲线图，可以提供用户更友好的操作方式和使用方法。

由于该软件主要功能是数据分析和统计。在数字滤波的处理过程中，目前卡尔曼滤波器的初始参数只是“穷举法”以后的有效猜想值。而更准确的这需要基于事实环境的测量和估算。

系统中的很多算法，包括数字处理，图像接口等，都可以扩展为其它数据的统计分析，而不仅局限于风速风向的测量。所以从软件工程的角度来说，该软件系统可以设计接口来处理和显示其它各种类型的数据。

参考文献

[1]. 葛艳，孟庆春，魏振钢，高云，闫传军，《帆船直线航行比赛最优路径动态规划方法研究》，控制与决策，第 20 卷，第 12 期

[2]. 刘勘，周晓铮，周铜汝，《数据可视化的研究与发展》，计算机工程，第 28 卷，第 8 期

[3]. Emmanuel C. Ifeachor, Barrie W. Jervis, “Digital Signal Processing: A Practical Approach, Second Edition” （《数字信号处理实践方法》（第二版），罗鹏飞，杨世海，等译），电子工业出版社

[4]. 彭丁聪，《卡尔曼滤波的基本原理及应用》，软件导刊，2009 年 11 月，第 8 卷，第 11 期

[5]. 沈彦燕，袁峰，张静，刘书友，《矢量平均法测量地面风》，气象水文海洋仪器，2008 年 6 月，第二期

[6]. Matteo Frigo, Steven G. Johnson, “The Design and Implementation of FFTW3”, Proceedings of the IEEE, 93(2): 216–231, 2005. special issue on ”Program Generation, Optimization, and Platform Adaptation”

[7]. Matteo Frigo, Steven G. Johnson, “FFTW: An Adaptive Software Architecture for the FFT”, Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing 3: 1381–1384. DOI:10.1109/ICASSP.1998.681704

[8]. Interactive Matter, “Filtering Sensor Data With a Kalman Filter”, 2009 年 12 月

致谢

能够完成此毕业设计和毕业论文的过程主要归功于实验室的朱谦老师和汤大侃老师，两位老师用自己的身体力行教会我们工程类学生所应该具备的精益求精和严谨的修养和学习习惯。非常感谢他们在毕业设计的完成中提供的指导性的建议和学术帮助以及在毕业论文的完成过程中的督促和意见。

感谢凌力、钱松荣老师，感谢他们在大二和大三的时候带我进入实验室参与项目，帮助我有了今天的成长。也感谢通信系的其他老师在专业学习中提供的指导和帮助。

感谢实验室学姐苏菲、项目组同仁们，没有那些组会和通宵，这个项目也不会完成。

感谢所有我所认识和熟悉的大学同学。范静远，齐普，张健尧，张时超，蔡荣锡，还有任重 2 班的那些老友，没有你们我的大学生活怎么会如此丰富。

特别的感谢给李童童，谢谢你的远程陪伴和鼓励。

最后还要感谢我的父母，谢谢你们一直的支持和鼓励。

攻读学位期间发表的论文：

[1]. **Xiangyu Hu**, Songrong Qian, “IOT Application System with Crop Growth Models in Facility Agriculture”, Proc. 6th Int’l Conf. Computer Sciences and Convergence Information Technology, Nov. 2011.