Geodetic Data Modelling System: Thesis

Daniel Fernandez
991672

Michael Still
964076

Blake Swadling
982087

Kristy Van Der Vlist
983118

Nick Wheatstone
983131

Geodetic Data Modelling System: Thesis

by Daniel Fernandez, Michael Still, Blake Swadling, Kristy Van Der Vlist, and Nick Wheatstone

Abstract

This thesis discusses the theory and implementation of the Geodetic Data Modelling System (**GDMS**), a software time series analysis application designed specifically for the analysis of Global Positioning Satellite (GPS), Very Long Baseline Interferometry (VLBI) and Satellite Laser Ranging (SLR) data. The year long project carried out by the authors, involved first prototyping a system in MATLAB, before implementing a full version in ANSI ISO C++. This thesis starts with a brief discussion on the history of time series analysis, and an introduction to the **GDMS**. It is then further divided into five main chapters, the first of which, provides a comprehensive look at the theory behind all process employed in the system. These include time domain and frequency domain analysis, along with interpolation, windowing and the user interface. The following chapter discusses design methodology, the implementation of the prototype and of the **GDMS** itself. In addition, issues which arose are examined, along with suggestions for future enhancements that could be made to this system. The fourth chapter discusses the testing methodology, which is an integral part of any systems software design and the fifth chapter discusses the documentation process. The final chapter reflects on the project as a whole in terms of meeting the project team's objectives and expanding their body of knowledge.

Table of Contents

List of Equations

List of Tables

List of Figures

Acknowledgments

Daniel Fernandez would like to thank:- Karen for her patience and support, as well as my family and friends for their support and encouragement.

Michael Still would like to thank:- My wife Catherine and my son Andrew.

Blake Swadling would like to thank:- My wife Linda for her support and my two beautiful children Michaela and Ezra.

Kristy Van Der Vlist would like to thank:- My mother and the guys at work. A special thanks goes to Emma and Sam for all their help and support throughout the year.

Nick Wheatstone would like to thank:- My Mother and father for providing support throughout the year, especially the hectic last month.

The group would like to give special thanks to Professor Peter Morgan for all his time, effort, patience and support throughout the year.

Chapter 1. Introduction

Introduction

Time series analysis of discrete signals is not a new concept. Texts on this topic were printed as early as 1924. It was not, however, until the digital computer appeared, in the latter half of the 20th century, that momentum in this field increased dramatically. The FORTRAN programming language has enabled complex mathematics to be written on a computer. Many routines and texts appeared around this time. The focus of these applications was in the areas of oil and space exploration, further significant texts emerged from Australia in the nineteen sixties and seventies on Fourier analysis of time series and its applications.

Published works such as Numerical Recipes emerged, firstly in FORTRAN, then C and C++. This is accepted as the standard reference today for Numerical Analysis algorithms, covering least squares analysis, interpolation, statistics and frequency analysis techniques. Time Series View (TSVIEW) is a more recent example of a computerised package dealing time series analysis and was developed at the Massachusetts Institute of Technology (MIT). It is a front end to MATLAB providing time series analysis in the time domain. TSVIEW also provides an intuitive, interactive graphical user interface thereby allowing the easy analysis of a given data set.

MIT is an institution with significant influence in the area of time series analysis. It is also home to the GAMIT/GLOBK suite of programs for the analysis of Global Positioning System (GPS), Very Long Baseline Interferometry (VLBI) and Satellite Laser Ranging (SLR) data. It is worthwhile to take a brief look at the functionality of this suite, paying particular attention to the CVIEW module. Its features include the following:

1. Selecting or spanning a region of points for closer inspection (i.e. zooming).
2. The ability to display, with various symbols, data that is used in the solution. This also involves data that does not correctly fit a particular model and is therefore weighted out of the solution.
3. The ability to weight and un-weight points.
4. The ability to changes the value of any number of points.
5. Allows the displacement of elementary statistics and regression coefficients.
6. The testing of different processing strategies.
7. A variable number of display windows, ranging from one to 10.
8. Quick and flexible use of file pointers.

The geodetic data sets used in this analysis are extensive, covering in the order of ten years of measurements from stations around the globe. The GPS, VLBI, and SLR data are susceptible to various conditions, potentially affecting the validity of the data. Some of these conditions include:-

- Ionospheric delay.
- Temperature fluctuations.
- Other considerations including projected satellite orbits.

With the ability to model certain situations, taking into consideration factors such as those above, conclusions can be drawn regarding, among other things, tectonic plate movement, the level of continental drift and weather conditions.

Whilst both TSVIEW and CVIEW are useful tools for analysis of this type of data, they are not without their short comings. MATLAB is expensive to purchase, and this has prohibited some institutions from being able to use the TSVIEW package which requires this program to run. In addition, TSVIEW uses the MATLAB algorithms to calculate its data transformations and whilst this guarantees that all mathematical functionality has been rigorously tested and validated, these routines are not specifically optimised for the analysis of geodetic data. Similarly, the CVIEW package is not without its problems. Although it is faster than TSVIEW in calculating a solution this package has grown as a set of individual modules that have been added over time, and the whole system is preserved inside X Windows wrappers. This has had the effect of making the package itself almost impossible to maintain in recent times. CVIEW also suffers from the problem that its user interface is dated and is not considered particularly user friendly. This has meant that there is a steep leaning curve required for any user that may wish to use this product. One of the most important limitations of both these packages is, however, that they both lack any support for frequency domain analysis which is imperative for the analysis of geodetic data.

Consequently, the final year project developed by Daniel Fernandez, Michael Still, Blake Swadling, Kristy Van Der Vlist and Nick Wheatstone known as The Geodetic Data Modelling System (**GDMS**) was created for the specific purpose of remedying the short falls of the existing geodetic data analysis applications. This system aims to provide and expand upon the features included in both TSVIEW and CVIEW. The **GDMS** provides time domain analysis in the form of Linear Least Squares Regression modelling. In addition, several interpolation and Gaussian filter (windowing) algorithms have been implemented, thereby allowing the data to be transformed into the frequency domain for analysis with such tools as Fast Fourier Transforms (FFT) and Power Spectral Density Plots (PSD). By allowing analysis in both the time and frequency domains the **GDMS** gives the user greater flexibility in determining the general data trend as well as the ability to detect and remove any erroneous data points that may appear in the given data set. The **GDMS** also provides an intuitive user interface capable of, among other things graphing, spanning and display multiple windows.

This thesis aims to provide a comprehensive insight in the theory behind the processing techniques that have been employed in the **GDMS**. Where appropriate alternative theories and algorithms are presented and compared. The design methodology used in the implementation is discussed, along with the the prototype and each specific algorithm with reasons for why they were chosen. Issues that arose throughout this process are also highlighted, ranging from design changes during the project, to difficulties with the implementation of various algorithms. In each section, we examine the potential improvements and enhancements that could be made to the various modules. A background on the testing approach is provided. As projects increase in size, standardised testing procedures becomes paramount. Here we detail the approach used for the testing of the various modules of the project. This paper closes by providing among other things, what we believe to be a measure of the success of the project, both in terms of the project itself and also in terms of our enhanced knowledge and experience that resulted from undertaking this task.

Chapter 2. Theory

Time Domain Analysis

Introduction

Analysis of discrete signals in the time domain is essential to the understanding of geodetic data produced by GPS, VLBI and SLR. By applying a linear least squares regression model to the system underlying data trends can be ascertained, such as, the average rate of continental drift. Linear regression analysis also allows such phenomenon as Random Walk and white noise, both of which which are known to affect each type of geodetic data to differing degrees, to be detected and removed. This, in turn, enables the geodetic community to gain a greater understanding of the elements that may affect the accuracy of a given data set. In addition, an analysis of residuals resulting from a linear regression not only give a measure of how accurately the model approximates the data set it can also aid in the detection and deletion of erroneous points.

Theory

Linear Least Squares Regression Modelling

One of the major problems in discrete signal analysis is that these systems are almost always inconsistent, that is, there is no exact solution to the equation of the line

Equation 2-1. Equation of the Line

$$y = mx + c$$

where **m** is the slope and **c** is the y-intercept. This is usually due to the fact that there are more data points in the system than those required to solve this equation. Linear least squares regression modelling is a method used to calculate **m** and **c**, such that, it is the closest approximation **y** for the given set of observations in the system, also known as a line of best fit. In the analysis of geodetic data the line of best fit is given as the solution to the matrix equation

Equation 2-2. Linear Least Squares Regression Equation 1a

$$X = -(A^T PA)^{-1} A^T PL$$

where **X** is the corrections to the apriori estimates

Equation 2-3. Corrections to Apriori Matrix

$$\begin{pmatrix} m \\ c \end{pmatrix}$$

and **n** is the number of observations in the system. The **A** matrix is the know as the design matrix

Equation 2-4. Design Matrix

$$\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \cdots & \cdots \\ x_n & 1 \end{pmatrix}$$

P is the **n** x **n** weighting matrix and **L** is the apriori matrix

Equation 2-5. Apriori Matrix 1a

$$\begin{pmatrix} y_{c_1} - y_{o_1} \\ y_{c_2} - y_{o_2} \\ \cdots \\ \cdots \\ y_{c_n} - y_{o_n} \end{pmatrix}$$

When carrying out least squares regression analysis on geodetic data it is common to set the initial apriori estimates of the slope and y-intercept to zero, thereby making all value of $\mathbf{L}_c = 0$. By applying this initial constraint the apriori matrix becomes

Equation 2-6. Apriori Matrix 1b

$$\begin{pmatrix} -y_{o_1} \\ -y_{o_2} \\ \cdots \\ \cdots \\ -y_{o_n} \end{pmatrix}$$

and hence Equation 2 becomes

Equation 2-7. Linear Least Squares Regression Equation 1b

$$V = AX - L_0$$

Another common constraint use in this type of analysis is to set y-intercept (**c**) to the point of the first time observation in the given system. This is desirable because of the time format used in geodetic data systems, that is, a decimal number where the numerator corresponds to the year and number of days into the year is denominator. For example, the date value 2002.0014 corresponds to the Gregorian calender date of 12:00, 1 January 2002. If x-axis was not constrained, the y-intercept would always be given at **x** = 0, which corresponds to a date of 4000 BC (approximately 500 years before the Bronze age), thus rendering this value virtually meaningless. Therefore, the x-axis is constrained by setting **x** = **t** - $\mathbf{t}_0$ where $\mathbf{t}_0$ is the first time observation in the given system. This has the effect of making the **c** value more meaningful and simplifying any graph of this data.

Weighting Matrices - Variance Co-variance and Random Walk

It is not uncommon in geodetic data, especially data taken over a long period of time (i.e. a number of years) for some of the value to be less reliable than others. This can be due to the failure or replacement of data gathering equipment or natural phenomena such as earthquakes. In addition, random or white noise is known to affect different types of geodetic data to varying degrees. This type of signal distortion is truly random and has a zero mean (μ) and a variance (σ), so that, over a sufficently large data set its effects are cancelled out. The problem occors when smaller systems are used and the white noise is not cancelled out, this can distort the mean value of the given system. It is important, therefore, to have some mechanism of ensuring that this potentially erroneous data does not influence the linear regression analysis in an adverse manner. In order to do this what is known as a Variance Co-variance (VCV) weighting matrix is employed.

A VCV weighting matrix is a strictly diagonal matrix where correlation is not admitted, that is $\mathbf{P}_{ij}$ = 0 where $\mathbf{i} \neq \mathbf{j}$ which models a standard Gaussian distribution (also known as a bell curve).

Equation 2-8. Standard Gaussian Distribution

$$y_i = n(\mu\sigma^2)$$

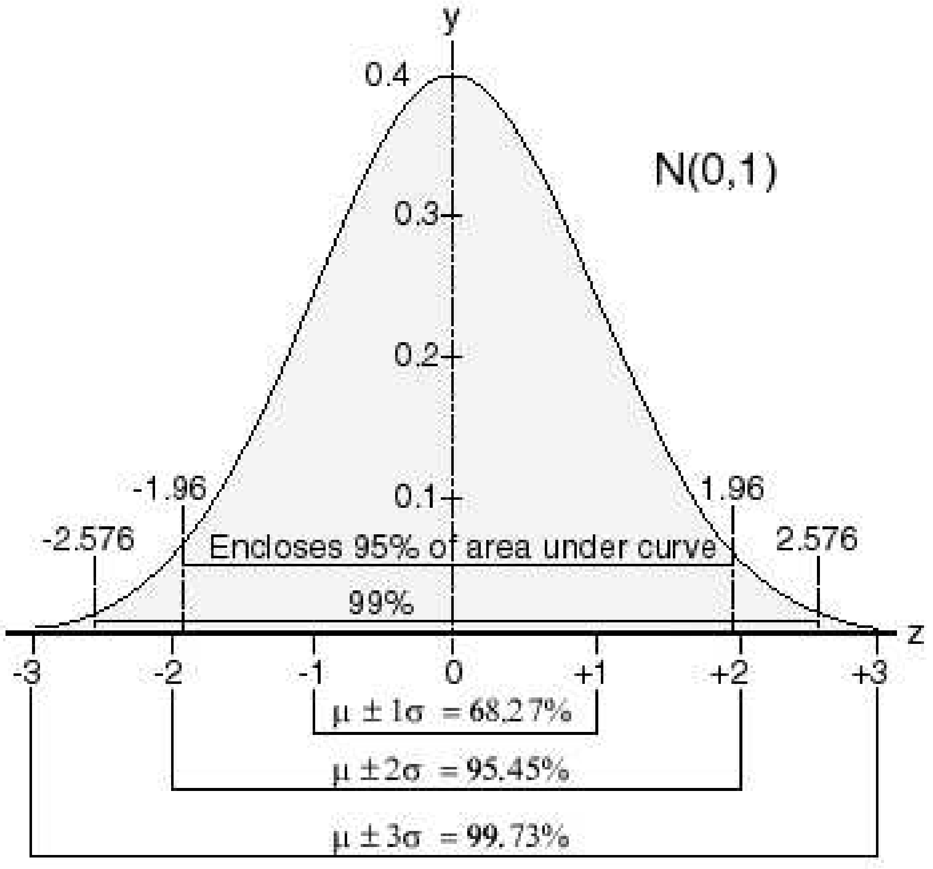

Figure 2-1. Diagram of Standard Gaussian Distribution

This method works by pre-multiplying the observation values with a number that represents its "correctness" (see equation 2-2). Therefore the weighting value specified on the diagonal of the VCV matrix should be a percentage value ranging from one to zero, whereby a value of one means the given data point fully participates in the calculation and a value of zero indicates that the data point was erroneous and is discarded.

Due to the nature the of method of weighting data values, great care must be taken so that the linear regression model calculated is not distorted. This is due to the fact that, by pre-multiplying the observation data with a weighting matrix their values are essentially being changed. If too few erroneous data points are un-weighted, the least squares line calculated will not accurately represent the general data trend. Similarly, weighting out too may data points will cause the same error to occur. For example, one method of weighting an inconsistent system would be to assign a weighting value according to standard deviation, that is, for all points within standard deviation above or below the mean would be given a weight of 1. Similarly, points within two standard deviations, but greater than one standard deviation would be weighted 0.75 and any point two standard deviations above or below the mean would be given a weight of 0.5. Finally, all other points would be weighted to zero. This model, however, has been known to unnaturally distort the data when applied to geodetic systems. This is due to the fact that the data maybe skewed, that is where the third moment is

Equation 2-9. Skewness

$$s = \frac{E(x-\mu)^3}{\sigma^3}$$

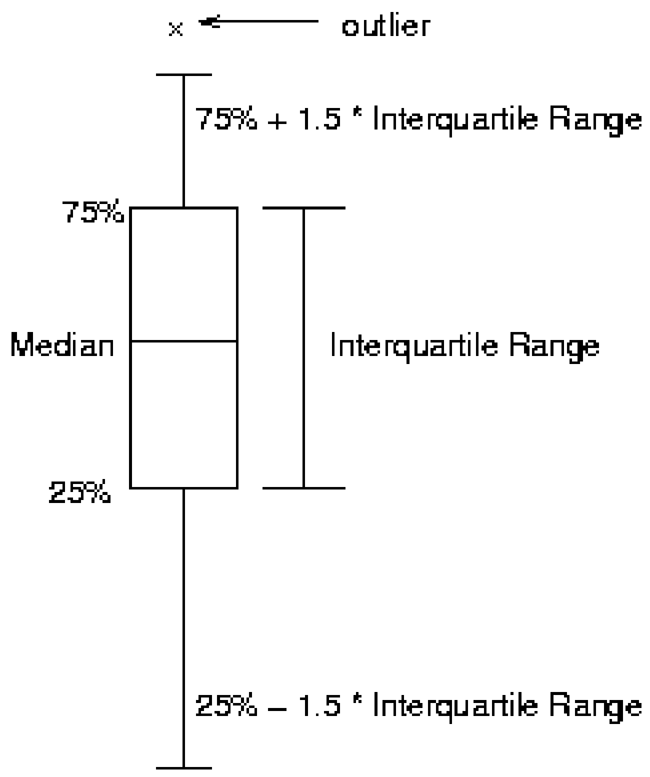
outlier
75% + 1.5 * Interquartile Range
75%
Median
Interquartile Range
25%
25% – 1.5 * Interquartile Range

Figure 2-2. Box Plot of a Skewed Data Set

where **s** $\neq 0$ and/or affected by kurtosis where the forth moment is

Equation 2-10. Kurtosis

$$k = \frac{E(x-\mu)^4}{\sigma^4}$$

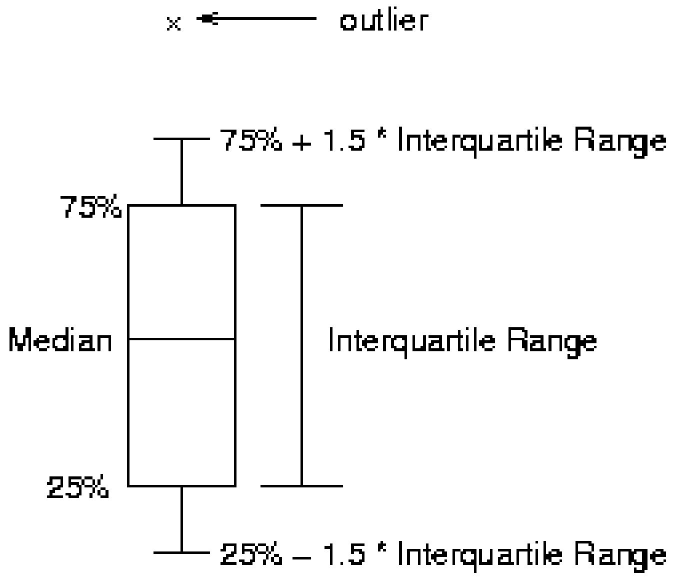
outlier
75% + 1.5 * Interquartile Range
75%
Median
Interquartile Range
25%
25% – 1.5 * Interquartile Range

Figure 2-3. Box Plot of a Data Set affected by Kurtosis

for **k** $\neq$ 3. By applying this model, these anomalies will only be accentuated in the resulting linear regression model and either too many data points will be weighted less than one or too few points will be weighted out. A far safer method of assigning a weighting matrix is to simply weight any data point greater than three standard deviations above or below the mean to zero and all other values as one. This ensures the resulting model will be calculated using a standard Gaussian distribution, and thus will be unaffected by these phenomenon.

Weighting matrices will also remove any underlying noise that may be affecting the system, one such trend is known as Random Walk, which is a phenomenon that the geodetic community believes occurs, particularly in GPS data. Detection of this trend has become possible in recent times because it is believed that Random Walk can only be accurately seen in systems that span at least six years, or preferably 10 years and data sets of this length have not become available until now. If it is assumed that the data follows a standard Gaussian distribution (see figure 2-1) that is, the observation values are evenly distributed then the expected Random Walk values are

Table 2-1. Table Of Random Walk Values

Time	1	2	...	n
RW	R_1	$R_1 + R_2$	...	$R_1 + R_2 + ... + R_n$

and thus the weighting matrix is

Equation 2-11. Random Walk Weighting Matrix

$$P_{rw} = \left(\sum_{norm} + \sum_{rw}\right)^{-1}$$

It is anticipated that using such a weighting matrix will remove any underlying Random Walk trend that may be affecting the data. At this time, however, this is some debate as to which equation should be used in order to determine the random walk co-efficients and therefore the optimal weighting matrix. As a result, there is limited support for this type of calculation within the scope of the project.

Residuals and Determining VCV Weighting Models

Residual space is another important aspect of Time Domain analysis. In essence, a residual is vector that represents the distance between the observed value and the least squares regression line of best fit. The residual vectors are therefore calculated by the matrix equation

Equation 2-12. Residual Equation 1a

$$V = AX + L$$

Where **L** is specified by equation 2-5. If the same constraint is applied to the apriori matrix as was used in the linear least squares model, that is, it is assumed that the initial model has a slope and y-intercept of 0, the residual equation becomes

Equation 2-13. Residual Equation 1b

$$V = AX - L_O$$

At the very least, the residual matrix shows how well the regression line approximates the given system, that is, the smaller the values of **V**, the better the fit of the model. In addition, if the residuals are transformed into the Frequency Domain via a Fast Fourier Transform, an optimal model would give a totally flat frequency response. Another property of residuals is that their sum should always be equal to zero. This is a very useful property for testing the validity of any residual calculation.

Calculating residuals are also essential for determining the VCV weighting model that should be applied to the given system. As it has been seen, calculating an optimal VCV weighting model relies heavily on determining the standard deviation of a system according to a Gaussian distribution. One method achieving this is to calculate the new weighting matrix as

Equation 2-14. Weighting Matrix Equation 1a

$$P = \sum{}^{-1} = N$$

where **N** is given by

Equation 2-15. Weighting Matrix Equation 1b

$$N = A^T PA$$

This method leads to the new weighting matrix Σ that is not strictly diagonal, this is undesirable becuase it is computationally much slower to calculate. For example, if the matrix equation $\mathbf{A}^T\mathbf{P}$ was to be caulculated where **A** was a **n** x 2 matrix and **P** was a **n** x **n** weighting matrix. If the weighting matrix was strictly diagonal then then it would take 2 x **n** multiplication operations to compute. Conversely, if **P** was a non-diagonal matrix then it would take 2 x $\mathbf{n}^2$ and 2 x $\mathbf{n}^2$ addition operations to compute the same equation.

Because of the slowness of using a weighting matrix that is not diagonal, another method of calculating a weighting matrix was sought. This method works entirely in the residual space and ensures that the weighting matrix is always diagonal. In this calculation, a line of best fit is calculated using equation 2-7 and the residuals are calculated using equation 2-13. The residual matrix is then sorted in ascending order. From there the median and interquartile range can be calculated. The threshold of three standard deviations above or below the mean, assuming a standard Gaussian distribution, can be calculated by adding 1.5 x the interquartile range to the 75% limit and subtracting 1.5 x the interquartile range from the 25% limit, resulting in a box and whiskers plot.

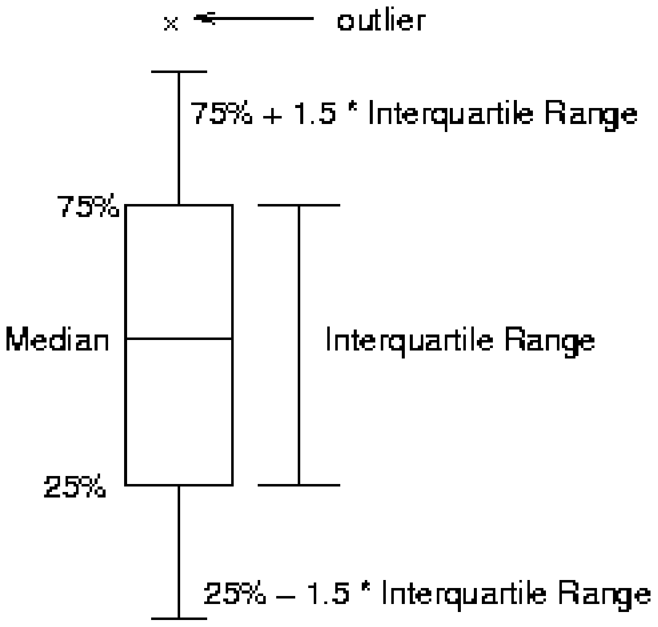
outlier
75% + 1.5 * Interquartile Range
75%
Median
Interquartile Range
25%
25% – 1.5 * Interquartile Range

Figure 2-4. Box Plot of Data Set Re-weighting Using Resdiuals

Therefore, any residual with a value greater than three standard deviations above the mean is considered to be erroneous and will be weighted out.

Conclusion

Time domain analysis of discrete geodetic signals is a powerful tool in determining data trends. Linear Regression Modelling can be employed to establish any general patterns in the data and give an indication of any data anomalies. It has also been seen that by using a weighting matrix, erroneous data can be removed, thereby allowing the most accurate regression model to be fitted. Weighting matrices can also be employed to detect and remove any underlying data trends, such as Random Walk that may be adversly affecting the system. Time Domain analysis allows a given system to be modelled in the residual space. This is important for may reasons, firstly, the residuals give an indication of how well the regression model actually fits the observation data in both the time and frequency domains. In addition, the residual space allows an optimal VCV weighting matrix to be calculated, such that it is always guaranteed to be strictly diagonal. This vastly reduces the computational time required to calculate a least squares solution.

Interpolation

Introduction

One problem with GPS datasets is that there are often gaps in the data, which may skew data results and prevent transformation into the frequency domain. Another problem is that the time scales in GPS data sets are often non-linear. Interpolation solves these problems by filling the gaps in the data. A large number of interpolation methods exists, all having their advantages and disadvantages for different applications. As a result, the **GDMS** offers six different interpolation methods:-

1. Nearest Neighbour
2. Linear
3. Cubic Spline
4. Natural Spline
5. Newton Divided Difference
6. Least Squares

Theory

Theoretically, each of the datasets that we analyse are sampled once a day at exactly 12:00. This is, however, not always possible for reasons such as such as equipment failure, extreme weather and in some case local politics. The Fourier transform algorithms available to us require that the dataset be regular. This means that if a dataset is missing a point or has an irregular sample rate then it can not be transferred to the frequency domain. The dataset therefore, must be made regular either by removing points or interpolating new points.

This irregular sampling rate has been compounded by having the datasets stored on a non-linear time scale. The data that we have been provided has been sampled once a day at noon as timed by an atomic clock, thereby allowing an accurate sample rate. The sample date/time, however, has then been stored in decimal year format to eight significant figures, and this creates a problem. During a regular year one day is 0.00273972, except one in every four years is a leap year and the length of a day then becomes 0.00273224. Also at the start of each year a rounding occurs so that the first reading of the year occurs at XXXX.0014. This rounding creates the illusion of a different time distance between the last day of one year and the first day of the next. Due to the nature of this problem, it is necessary to transform the data scale to be independent of calender years before carrying out interpolation.

The new time scale that we convert to is what is known as a truncated Julian day. The Julian day system uses an integer day count since the first of January 4714 BC. This, however, produces extremely large numbers which could potentially cause loss of accuracy due to machine limitations. To prevent this we have used the first of January 1901 as our start date. The existing decimal dates then converted to truncated Julian day and rounded to the nearest whole number. Once the timescale is linear, interpolation can be carried out on the data without fear of loss of data or returning inaccurately interpolated points.

Different interpolation methods have different strengths and weaknesses. On an arbitrary dataset it is impossible to tell which interpolation method will be the most accurate, and as such six different interpolation methods have been provided. Each will be outlined below, including details on its operation, strengths and weaknesses.

Nearest Neighbour Interpolation

Nearest neighbour interpolation is the simplest method provided. This method sets the value of any new point to the value of the nearest point on the original dataset. The advantage of this is that any added points will be of the same approximate value as nearby points. There are a number of disadvantages with nearest neighbour approximation. Firstly, any new points will not follow any linear or frequency trends in the data, which could lead to inaccuracy of models both in the time and frequency domains. In addition, if new points lie close to outliers then the points generated can be very inaccurate.

Linear Interpolation

Linear interpolation is widely used as it is simple, yet generally produces better results than nearest neighbour. In this interpolation method, each new point is placed on a line between the two adjacent points. The Equation for adding a new point is:

Equation 2-16. Linear Interpolation Between Points

$$y = y_{prev} + \frac{y_{next} - y_{prev}}{x_{next} - x_{prev}} * (x - x_{prev})$$

Linear interpolation has several advantages. Firstly, each added point has a value that is approximately the same as nearby value thus obtaining values significantly out of range are very rare. Secondly, the added point will follow any local linear trends between the two points. The disadvantages of linear interpolation stem from the interpolation method only working with the two adjacent points. If the line between these two points does not follow the overall trend of the data then loss of accuracy occurs.

Cubic Spline Interpolation

The **GDMS** includes one of the many implementations of cubic spline interpolation. This specific implementation, has no particular name and is given the generic title of cubic spline interpolation. The only difference between this method and a natural spline is the choice of second derivatives used at the end points. With this implementation the values of the second derivative for the end points are set the same as the second derivatives of the second to end points.

To produce a spine for a dataset of **n+1** points you have to produce **n** separate cubics. Each of these cubic should have each end match up exactly with the points to either side. And on the same point the two adjacent cubic should have shared first and second derivatives. Each of these cubics can be described by the equation:

Equation 2-17. Equation of a cubic (Gerald and Wheatley, 1999)

$$y_i = a(x - x_0)^3 + b(x - x_0)^2 + c(x - x_0) + d$$

Because each of these cubics begins at a any existing point we can say: $\mathbf{d_i = y_i}$

where $\mathbf{h_i=(x_{i+1} - x_i)}$, is the width of the the interval.

Also if we make **S** the set of second derivatives; then through algebraic simplification we can get:

Equation 2-18. Elements of a cubic (Gerald and Wheatley, 1999)

$$a_i = \frac{S_{i+1} - S_i}{6h_i}$$

$$b_i = \frac{S_i}{6h_i}$$

$$c_i = \frac{y_{i+1} - y_i}{h_i} - \frac{2h_i S_i + h_i S_{i+1}}{6}$$

With cubic splines, solving for **S** is achieved by solving:

Equation 2-19. Solving for S (Gerald and Wheatley, 1999)

$$\begin{bmatrix} 3h_0 + 2h_1 & h_1 & & & \\ h_1 & 2(h_1 + h_2) & h_2 & & \\ & h_2 & 2(h_2 + h_3) & & \\ & & \dots & \dots & \\ & & & h_{n-2} & 2h_{n-2} + 3h_{n-1}) \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \dots \\ S_{n-1} \end{bmatrix} =$$

$$6\begin{bmatrix} f[x_1, x_2] - f[x_0, x_1] \\ f[x_2, x_3] - f[x_1, x_2] \\ f[x_3, x_4] - f[x_2, x_3] \\ \dots \\ f[x_{n-1}, x_n] - f[x_{n-2}, x_{n-1}] \end{bmatrix}$$

The two end values, $\mathbf{S}_0$ and $\mathbf{S}_n$, are then set to equal $\mathbf{S}_1$ and $\mathbf{S}_n$ respectively.

A full mathematical derivation can be found in Gerald & Wheatley (1999), page 238.

Splines can provide extremely accurate results when the original sample rate is notable greater than the frequency of fluctuation in the data. However if the sample rate is less than half the frequency of change (including signal noise) in the data then the results can be erratic. Splines also have a problem when it comes to large gaps in the dataset. Because the gap between two points is represented by a cubic, large gaps result in peaks or troughs in the dataset. The final problem with spline is that the end sections are inaccurate due to arbitrary methods used to assign the derivative of the end points. In the case of cubic splines, the end points tend to be too curved and it is for this reason that two cubic spline methods have been provided for this application.

Natural Spline Interpolation

Natural splines are a type of cubic spline, originating in ancient times, that is before the birth of Christ, when drafts-men and builders would create a smooth curve by pegging a flexible piece of wood between a number of points. As with all cubic spline implementations, the interpolation is achieved by inserting a cubic between each two adjacent points. Equalising the derivatives has the effect of making the resulting interpolation appear smooth and visually pleasing. For this reason splines are often used in graphics.

In the mathematical process of building splines it is impossible to infer the derivatives of the two end points of a dataset. It is the values assigned to these end points that determine the type of spine. In a natural spline the end points are set to 0. This gives a result similar to the ancient wooden splines used by builders. Mathematically this produces a result where the end segments can be too straight.

To produce a spine for a dataset of **n+1** points you have to produce n separate cubics. Each of these cubic should have each end match up exactly with the points to either side. And on the same point the two adjacent cubic should have shared first and second derivatives.

Each of these cubics can be described by the equation:

Equation 2-20. Equation of a cubic (Gerald and Wheatley, 1999)

$$y_i = a(x - x_0)^3 + b(x - x_0)^2 + c(x - x_0) + d$$

Because each of these cubics begins at a point we can say: $\mathbf{d_i = y_i}$

Where $\mathbf{h_i=(x_{i+1} - x_i)}$, is the width of the the interval.

Also if we make **S** the set of second derivatives; then through algebraic simplification we can get:

Equation 2-21. Elements of a cubic (Gerald and Wheatley, 1999)

$$a_i = \frac{S_{i+1} - S_i}{6h_i}$$

$$b_i = \frac{S_i}{6h_i}$$

$$c_i = \frac{y_{i+1} - y_i}{h_i} - \frac{2h_i S_i + h_i S_{i+1}}{6}$$

With natural splines solving for **S** is done by solving:

Equation 2-22. Solving for S (Gerald and Wheatley, 1999)

$$\begin{bmatrix} 2(h_0+h_1) & h_1 & & & \\ h_1 & 2(h_1+h_2) & h_2 & & \\ & h_2 & 2(h_2+h_3) & & \\ & & \dots & \dots & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \dots \\ S_{n-1} \end{bmatrix} =$$

$$6\begin{bmatrix} f[x_1,x_2]-f[x_0,x_1] \\ f[x_2,x_3]-f[x_1,x_2] \\ f[x_3,x_4]-f[x_2,x_3] \\ \dots \\ f[x_{n-1},x_n]-f[x_{n-2},x_{n-1}] \end{bmatrix}$$

The two end values, $\mathbf{S}_0$ and $\mathbf{S}_n$, are then set to zero.

A full mathematical derivation can be found in Gerald & Wheatley (1999), page 238.

Newton Divided Difference Interpolation

Mathematicians have long know that a nth order difference table can be used to accurately recreate a polynomial of degree n. Newton divided differences uses a similar method to approximate new points within an tributary dataset.

The Newton divided differences method is based on building a table of divided differences. The divided difference between two points in a dataset is defined as:

Equation 2-23. Standard divided difference (Gerald and Wheatley, 1999)

$$f[x_1,x_2] = \frac{f_2 - f_1}{x_2 - x_1}$$

Likewise higher order divided differences can be defined as:

Equation 2-24. Divided difference of differences (Gerald and Wheatley, 1999)

$$f[x_0,x_1,\dots,x_n] = \frac{f[x_1,x_2,\dots,x_n] - f[x_0,x_1,\dots,x_{n-1}]}{x_n - x_0}$$

These divided differences are built into a table as follows:

Equation 2-25. Divided difference table (Gerald and Wheatley, 1999)

x_i	f_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
x_0	f_0	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$
x_1	f_1	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	
x_2	f_2	$f[x_2, x_3]$		
x_3	f_3			

This table can then be used to generate an equation:

Equation 2-26. Newton divided differences (Gerald and Wheatley, 1999)

$$
\begin{aligned}
P_n(x) = {} & f_0 + (x - x_0) f[x_0, x_1] + (x - x_0)(x - x_1) f[x_0, x_1, x_2] + \\
& (x - x_0)(x - x_1)(x - x_3) f[x_0, x_1, x_2] + \ldots + \\
& (x - x_0)(x - x_1) \ldots (x - x_{n-1}) f[x_0, x_1, x_2]
\end{aligned}
$$

A full mathematical derivation can be found in Gerald & Wheatley (1999), page 229.

Because divided differences estimates a curve by using polynomial, using a polynomial of too low or high a degree can result in errors. The accuracy available by using the double precision floating point format also creates a limit.

To detect and limit the errors caused by these factors the error estimates can be calculated. As the divided difference table grows it calculates the rough error for a table of size one less. If the error estimate has shrunk then the program continues building the table to a higher order. If the error has grown then the program stops building the table.

The error for a divided difference of a given order is approximately equal to the change that will be incurred by increasing the order of the divided difference table by one. So for an order n newton divided difference interpolation the mathematical definition of the error is:

A full mathematical derivation can be found in Gerald & Wheatley (1999), page 229.

Conclusion

Interpolation methods provide a useful toolkit that can be used to remove defects within the dataset. They can be used to fill holes in the time series, allowing the general trend of these missing points to be seen as well Fourier transforms to be applied to the given data system. No one interpolation method, however, is guaranteed to correctly estimate the missing data points for every data set, due to the nature of the individual characteristics of the gaps. It is for this reason that six different interpolation methods have been provided.

Windowing

Introduction

Windowing is the technique whereby an infinite signal is truncated in the time domain. Truncation of the signal in this way has side effects, primarily the introduction of ripples in the frequency domain. These ripples show up as high frequency noise and adversely affect integrity of the datasets. The data with which we are provided is a finite subset of an infinite signal, and consequently, this process has been performed implicitly prior to any post processing. Application of a filtering function can attenuate these frequencies, resulting in a more faithful representation of the original signal.

Theory

As previously stated, truncation of the signal in the time domain leads to unwanted ripples being produced in the frequency domain. The basic idea behind windowing the data prior to performing a Fourier transformation is that we can apply a weighting function to the sample in order to reduce the introduction of unwanted frequencies. This reduction is a trade off, and the cost is an increase in the width of the spectral peak and leakage of energy away from the true frequency to the side lobe skirts. This adversely effects the resolution with which we can observe the signal (for a more in depth discussion of this topic see the section on Frequency Domain Analysis theory below). We must, therefore, strike a balance between spectral peak width and the degree of high frequency attenuation. Ideally we would like to have no skirts, but in practise the best we can do is try to control them. The available windowing algorithms vary in their ability to control this while trying to retain a fine spectral peak. The attributes of each algorithm can be summarised as follows:

- *Hanning:*

  This algorithm provides good resolution of spectral peaks and acceptable rejection of side lobe skirts. It is a suitable choice for most applications. The Hanning window is a simple cosine offset by sufficient amount to set the minima to zero. The coefficients are calculated using the equation:

  Equation 2-27. Hanning Window Coefficients

$$w(i) = \alpha - (1-\alpha)\cos(\frac{2.\pi.i}{size-1}), for \alpha = 0.5$$

- *Hamming*

  Hamming provides finer resolution of spectral peaks than Hanning windows, however, side lobe skirts are not controlled as well as with this method. The Hamming window is a slightly modified Hanning window, with the difference being that the DC bias is marginally different and the wave is compressed. The coefficients are calculated as follows:

  Equation 2-28. Hamming Window Coefficients

$$w(i) = \alpha - (1-\alpha)\cos(\frac{2.\pi.i}{size-1}), for \alpha = 0.54$$

- *Blackman*

  Blackman peak resolution is not as fine as that produced with the Hanning method. The advantage with the Blackman window is, however, that the response shape flares out less at lower levels and rejection of side lobe skirts is better. A Blackman window is composed of two cosine waves, causing the skirts to fall away faster than the previous two and resulting in a narrower peak. the equation is:

  Equation 2-29. Blackman Window Coefficients

  $$w(i) = \alpha - \beta \cos(\frac{2\pi i}{size - 1}) + \gamma . \cos(\frac{4\pi . i}{size - 1}), for\, \alpha = 0.42, \beta = 0.5, \gamma = 0.08$$

- *Dolph-Chebyshev*

  Dolph-Chebyshev windows are very similar to those produced by Hamming, with improved high frequency attenuation. Where Hamming can provide around 50dB attenuation for frequencies above the sampling frequency, Dolph-Chebyshev can provide approximately 63dB attenuation for the same frequencies. This window is vastly different to the other windowing implementations. The coefficients are generated by calculating the inverse Fourier transform of the Chebyshev function evaluated at **n** equally spaced points on the unit circle. The Chebyshev function is defined as:

  Equation 2-30. Chebyshev Function

  $$cheb(n,i) = \cos(n.a\cos(i)), for\ -1 \le i \le 1$$
  $$cheb(n,i) = \cosh(n.a\cosh(i)), otherwise$$

Conclusion

Windowing can be an effective tool in removing the undesirable effects of the Discrete Fourier Transform (DFT). Prudent use of a suitable algorithm removes the high frequencies introduced via aliasing when the the signal is truncated. Knowledge of the attributes and limitations of the individual windowing algorithms, and the judicious selection of the correct one, will assist the user in attaining optimal results from the application of the DFT.

Frequency Domain Analysis

Introduction

Fourier transforms enable us to view the frequency representation of data that exists in the time domain. This provides a means of detecting erroneous frequencies in the given data set such as noise. Whilst in the frequency domain, we can also produce Power Spectral Density (PSD) plots. Once erroneous data has been removed in the frequency domain, an inverse transform can be applied to return the data to the time domain. Here we can ascertain the effects that removing such data may have on the original system.

Theory

The concepts of frequency domain analysis should commence with the Fourier transform. Any waveform can be constructed purely from a set of sine and cosine waves. An example of this is a pure square wave, which can be constructed using only the odd harmonics of a sin wave as follows:

Equation 2-31. Square Wave

$$x(n) = \sin(f) + \frac{1}{3}\sin(3f) + \frac{1}{5}\sin(5f) + \ldots$$

for all odd harmonics. Where **f** is the fundamental frequency. In addition, just as any signal can be modelled by a set of sine and cosine waves, the converse is also true and thus any signal can be decomposed into a set of sine and cosine waves. This is achieved by using the Fourier Transform, which preforms this signal decomposition and transforms it into the frequency domain. The Fourier Transform is calculated as

Equation 2-32. Fourier Transform (Rabiner and Schafer, 1978)

$$X(e^{jw}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}$$

where $\epsilon^{j\omega}$ is equal to ω, or the angular frequency. We can also computer the inverse Fourier transform, returning a function of time

Equation 2-33. Inverse Fourier Transform (Rabiner and Schafer, 1978)

$$x(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}d\omega$$

The above transforms are not, however, available for all sequences **x**(**n**). This is due to the fact that the basic Fourier transform is not reliable for signals sampled at discrete time interval as it is designed for use with with continuous signals (Lepple 2000). In addition, this set of equations is not guaranteed to converge for all **x**(**n**). To remedy this, the Discrete Fourier transform (DFT) is often used, thus enabling the transformation of a discrete signal and guaranteeing convergence. This is achieved restricting the sequence **x**(**n**) to the following conditions:

- The **N** values **x**(0)..**x**(**N**-1).
- Equidistant values around the unit circle.

The formula for the DFT is

Equation 2-34. Forward Discrete Fourier Transform (Rabiner and Schafer, 1978)

$$X(k) = X(e^{\frac{j2\pi k}{N}}) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi kn}{N}}$$

and the inverse DFT is given by

Equation 2-35. Inverse Discrete Fourier Transform (Rabiner and Schafer, 1978)

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{\frac{j2\pi k}{N}}$$

The discrete frequency **k** above is given by

Equation 2-36. Discrete Frequency

$$f(k) = k\frac{f_s}{N}$$

where $\mathbf{f}_s$ is the sampling frequency of the signal, and **k** is the number of points.

The problem with the DFT is that it is computationally expensive, with O($\mathbf{N}^2$). In recognition of this limitation the Fast Fourier Transform (FFT) was developed. This function takes into account the fact that in calculating a DFT much time is wasted with unnecessary multiplications. For example, it is unnecessary to multiply any number by zero, as the answer will always be zero. Given that multiplication is one of the slowest operations on most types of computers, reducing the number that need to be calculated will minimise the time taken to calculate a solution. The FFT also decreases the computational load of a DFT by taking into account the fact that a sinusoid is periodic, and its values are mirrored around the x-axis at **T**/2. For example, every 90 degrees, the value one reappears in a sine or cosine signal, changing only its sign. Many results, therefore, may determined by taking into account these factors. Through this increased efficiency, the FFT reduces the computation time to O(**N**log**N**).

The power spectrum is very useful in signal processing. It is the most common frequency measurement, allowing you to determine the amount of energy that exists at a given frequency. Derived from the power spectrum is the power spectral density (PSD) measurement. A PSD plot provides information on the magnitude of energy that exists in a given band of frequencies. This technique is particularly useful for measuring the noise content in a signal. A PSD plot is created by plotting the magnitude of the real and imaginary components of the FFT results. A good illustration of this concept is that of a sine wave with its corresponding PSD plot.

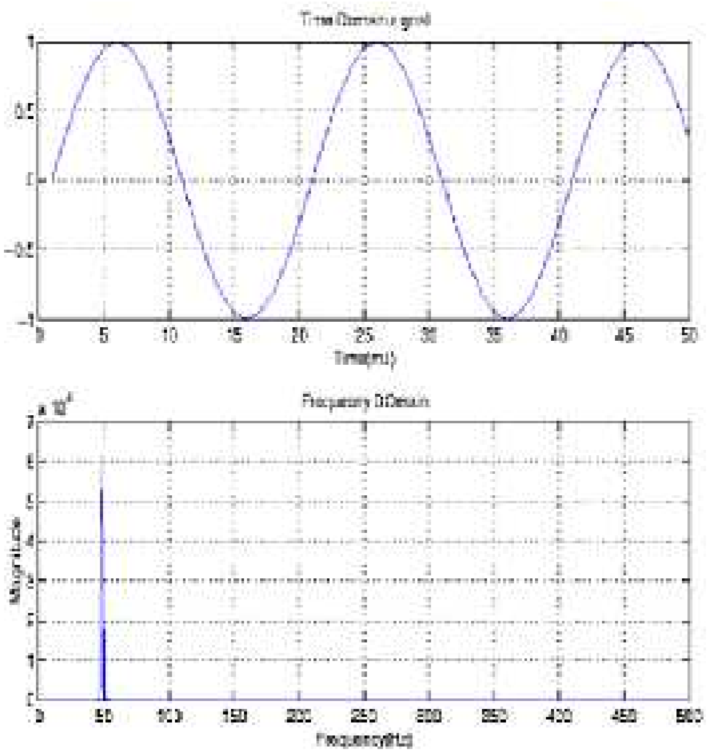

Figure 2-5. 50Hz sin wave and corresponding PSD plot.

In the above figure, we see a pure sin wave with a frequency of 50Hz. The PSD plot shows the energy content of the sine wave being at 50Hz.

There are some constraints which are implied when carrying frequency analysis. These are:-

- Maximum frequency that can be analysed.
- Frequency resolution.

The maximum frequency analysed relates to the sampling theorem, also know as the Nyquist theorem. The theorem states; that given a signal $\mathbf{x}(\mathbf{n})$ sampled at a sampling frequency of $\mathbf{f}_s$, the maximum frequency that can be analysed, $\mathbf{f}_{max}$ is equal to 1/2 of $\mathbf{f}_s$. In other words, to accurately represent and analyse the signal $\mathbf{x}(\mathbf{n})$, it must be sampled at a rate greater the $2\mathbf{f}_s$, where $2\mathbf{f}_s = \mathbf{f}_N$ and $\mathbf{f}_N$ is known as the Nyquist frequency. The simplest example to illustrate this is the sampling of a 20kHz audio signal. To accurately represent this signal, it must be sampled at greater than 40kHz. This value is typically 44100Hz, for example CD audio. Generally signals are sampled at several times $\mathbf{f}_N$ to ensure accurate results. In the case of time series analysis, where datasets are typically sampled at a rate of once per day, the maximum frequency that can be analysed is 0.5 cycles/day.

As mentioned earlier, noise is an inherent property in geodetic datasets. One common example is Random Walk. This type of noise changes with time and is therefore very difficult to model. For further details on Random Walk, please refer to the time domain analysis chapter of this thesis. A convenient method for viewing Random Walk, in particular the frequencies at which it is most prominent, is in the frequency domain. The conversion to the frequency domain is commonly done from the residual space of a dataset, i.e. the dataset has already undergone least squares regression analysis, including calculation of the residuals. Many of the datasets contain what appear to be Random Walk. One such example is the dataset from Ceduna. In the residual space(see figure below), the periodicity of the signal is clear.

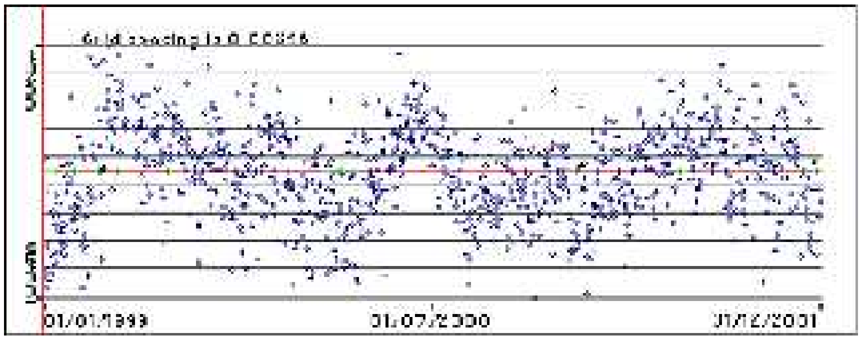

Figure 2-6. CEDUNA_GPS(UP) - Residuals

In the frequency domain, we can now expect to see the frequency components that make up the above signal. In particular, we want to identify the noise component of the signal. The following figure is the result of an FFT on the Residuals of CEDUNA (UP).

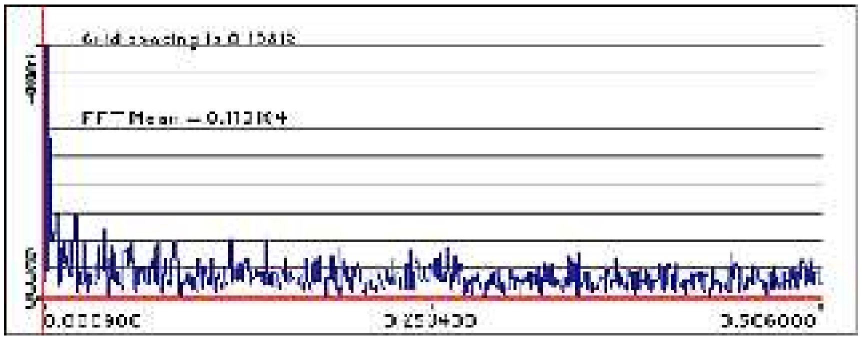

Figure 2-7. CEDUNA_GPS(UP) - PSD

From the PSD above, we can see energy at the low end of the frequency spectrum. This first major portion of energy is Random Walk. In fact, closer inspection of the PSD plot reveals a periodicity along the spectrum. This is also believed to be Random Walk. In addition, white noise can also be detected in the frequency domain, appearing as spikes.

The frequency resolution of a signal is also constrained by the fact that it is inversely proportional to the length of the waveform. Subsequently, if the length of the analysed waveform is doubled, the resolution is halved. This is clearer if we consider the relationship $\mathbf{f_{res} = f_s/N}$, where **N** is number of sample points. If we double the number of points while maintain the sample rate, our frequency resolution is halved.

Conclusion

Frequency domain analysis is a very useful tool for time series analysis. It enables us, for instance to see information not easily detectable in the time domain. Frequency trends and noise can also be identified in the data. This includes the detection of Random Walk though periodicity, and white noise that are represented as frequency spikes. It addition, we have seen that it is also possible to reduce noise by removing certain frequencies that are believed to be noise.

Chapter 3. Implementation

Design Philosophy

Introduction

The first design requirement for the **GDMS** project, was that it be licensed under the GNU General Public License. The reason for this is to allow access to the software by people who are not necessarily in a position to use proprietary software. The **GDMS** package is complex and relies on a number of external libraries for mathematical and graphical functionality. Similar proprietary libraries can be very expensive.

Another issue influencing our design methodology was the the need to have a package which is as platform independent as possible. This contributed significantly to the choice of programming language. In using ANSI ISO compliant C++, the **GDMS** package is intended to run on most Unix based operating systems. It should also be noted that most modern windowing toolkits require C++.

The choice of user interface toolkit also maximised the cross platform capabilities of the application. **WxWindows** was therefore chosen as it has been specifically built to be easily ported, and currently runs on the following platforms:-

- Microsoft Windows
- Unix (including BSD and SystemV)/GTK+
- Unix (including BSD and SystemV)/Motif and X11
- MacOS
- OS/2

The next important design decision involves the methodologies used to design implement the **GDMS** package. Given the complexity of the system, the Object Oriented approach was deemed the most suitable. Using OO, the package lends itself well to future enhancements, maintenance and extensibility. In fact, throughout the implementation process, changes in design were required to suit the requirements of the growing system. These changes, although not always trivial, were facilitate by the use of OO methodology.

The following figure is a deployment diagram for the **GDMS**. Note that the Error handling has not been connected in the diagram in order to preserve clarity. With few exceptions, all of the **GDMS** modules use the error handling sub-system. There are a number of reasons for this. The first is our desire to ensure that maximum possible stability of this system. This can only be achieved through efficient and appropriate error handling. For example, errors that are not fatal to the application will not cause termination, instead provide feedback on the error, before returning to a stable state. In addition, most operations use the matrix class which often invokes dynamic memory allocation and it is vital to handle any errors that may arise as a result for stability reasons. Therefore, there needs to be constant tracking to ensure an error has not occurred.

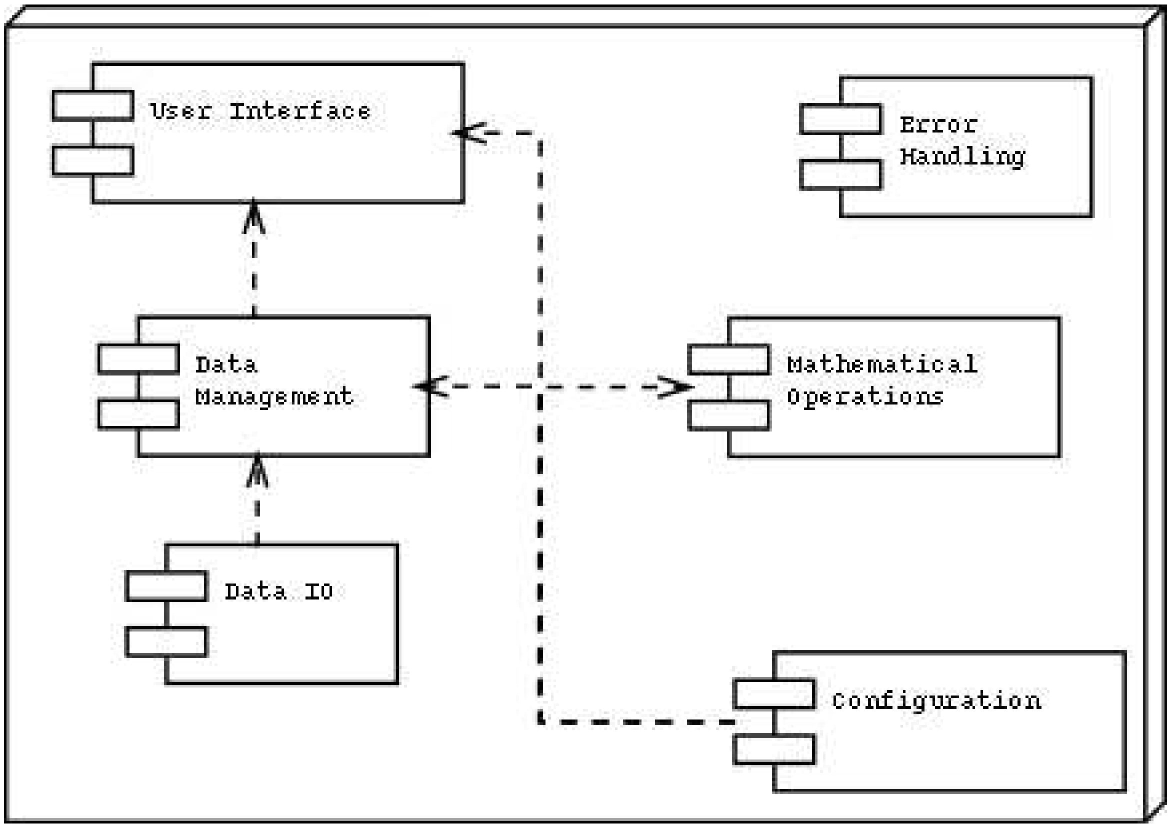

Figure 3-1. Deployment diagram for the GDMS.

The next digram is that of the error handling sub-system described above. In addition, each error is logged to file in order to facilitate debugging.

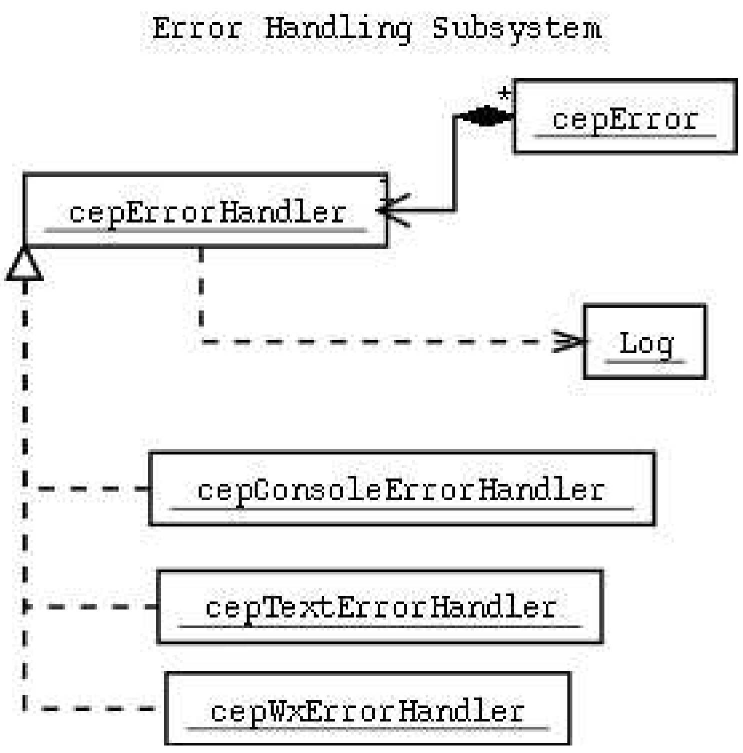

Figure 3-2. Class diagram for the Error Handling sub-system of the GDMS.

Future Enhancements

The modular design of the **GDMS** increases the potential for future improvements to the application, such as facilitating the addition of new mathematical functionality. The OO design can be built on by employing techniques such as inheritance. This technique has already been incorporated into the windowing sub-system of the **GDMS** and could used again in similar situations, where broad definitions can be defined for functionality, then specific classes can be derived from these higher level modules.

Prototype

Introduction

The MATLAB prototype was the minimum requirement of the project. It was developed not only to carry out all of the essential mathematical functionality thus providing a working model, but to also act as a test bed for the final product. Most of the features implemented in C++ were tested in this way to ensure correct and consistent results. Limited time was spent on the prototype user interface (UI) so that it could be completed early enough in the development cycle for group to be able concentrate on the full implementation of the **GMDS** in C++.

Theory

Prototyping can be an effective technique for identifying requirements in projects. We were able to use prototyping to prove our algorithms and also determine subtle functionalities that were required. One such example was recognising the needs for a generic data structure to handle various types that would be used in the mathematical functionality. This lead to the design of the Matrix class. The prototype was originally intended to be a product comparable to TSVIEW, with additional functionality, such as interpolation, windowing and frequency domain analysis to be added. Once the the mathematical functionality of the prototype was complete, work began on a UI. Not long into the its development, however, it was decided that it would not be of much benefit and that our attention should be focused on the main implementation of the application. The **m-files** from the MATLAB prototype have been included on the CD accompanying this thesis.

Conclusion

The prototype provided a useful method of testing design ideas and mathematical algorithms before the implementation of the C++ code base. This allowed us to generate an initial draft of the projects design. MATLAB, however, was a lot more flexible than C++ in implementing mathematical functions and thus some major design issues were overlooked.

Implementation Issues

Time Domain Analysis

Introduction

The **GDMS** offers three different types of Time Domain Analysis; VCV Least Squares Regression modelling, VCV Least Squares Regression modelling with automated re-estimation and Random Walk Least Squares. VCV regression modelling allows a linear model to be fitted to the data in a fast and precise manner, enabling any general data trends to be seen. It also allows the user to detect and remove any erroneous data, thus allowing the most optimal model to be calculated. In addition, an automated re-estimation model offers the capability of automatically detecting and removing erroneous data points, thereby computing the line of best fit that most accurately models the data. The third method of Time Domain Analysis is the Random Walk Least Squares model, which allows a weighting matrix to be specified that detects this underlying trend that may be occurring and affecting the accuracy of the data.

Implementation

Time Domain Analysis is implemented as a separate class inside the **GDMS** called **cepLs** and all calculations are encapsulated therein. A given solution is essentially reached by solving the Least Squares equation equation 2-7 and a set of residuals are calculated by equation 2-13. Each calculation produces three matrices, **X** which is the solution to the Least Squares matrix equation and contains the unknown co-efficients **m** and **c** of the line equation (see equation 2-1). The second matrix indicates whether or not the given data point was weighted out of the calculations and finally, the matrix of residuals is also produced.

The first method of Time Domain Analysis offered by the **GDMS** is known as VCV Least Squares Regression modelling. This method allows the user to specify a VCV weighting matrix to be used in the calculations, thus giving the user the maximum amount of flexibility in determining to what degree a given point in the data set will influence the line of best fit. Another method of VCV analysis offered is that of automatic re-weighting. In this type of analysis, the most accurate regression model is calculated by the process of automated re-weighing using the residual space which is discussed in depth in chapter 2, Time Domain analysis. The implementation of this process can also be seen in the following diagram

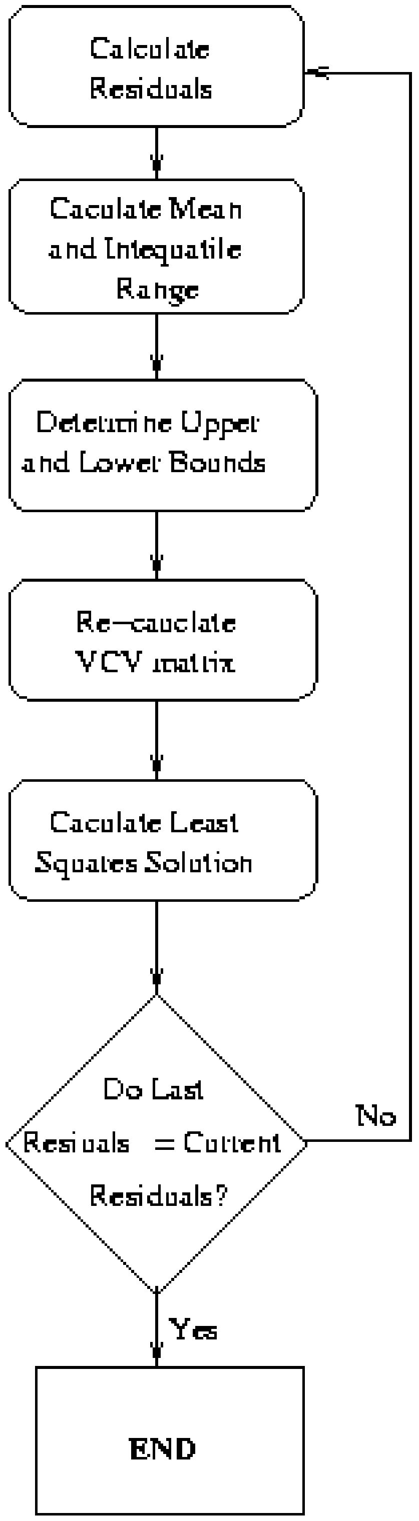
Calculate Residuals
Caculate Mean and Intequatile Range
Determine Upper and Lower Bounds
Re—cauclate VCV matrix
Caculate Least Squares Solution
Do Last Resiuals = Current Residuals?
No
Yes
END

Figure 3-3. Flow Diagram of the Automated Re-Weighting Process of VCV Least Squares Analysis

One important implementation problem that was discovered when applying this method was that in some cases the solution did not converge, and thus the program was remained in an infinite loop because the exit condition was never met. Another problem occurred where it appeared that the re-weighting process continued until every point in the data set had been weighted out. Both of these problems stemmed from the fact that in our implementation of this re-weighting algorithm, for over all design reasons, no point from data was ever entirely removed from the system but set it to 0 inside the weighting matrix, which achieved the same effect in the data domain. In the residual domain, however, the weighted out points were still being included due to the fact that the calculation of residuals does not require the use of a weighting matrix (see equation 2-13). The solution to this problem was to pre-multiply the residuals with the weighting matrix and removing any residual equal to zero before calculating the thresholds for determining whether a data point was to be weighted out or not.

The third method of Time Domain Analysis offered by the **GDMS** is the calculation of Random Walk Least Squares solutions. This method differers from the first two, in that a Random Walk weighting matrix is not a strictly diagonal matrix. At this time there is limited support for Random Walk Time Domain Analysis and although the **GDMS** is capable of producing a solution, the user must specify the Random Walk weighting matrix to be used. This is largely due to the fact that at this time there is still no general consensus as to how a Random Walk matrix is to be produced.

Optimisation

As one of the goals of **GDMS** speed of operations, great care has been taken to optimise the calculation speed of solutions in the Time Domain. This includes employing the use of passing values by reference to and from functions where ever possible, thus saving the time that it takes to replicate the parameters being passed in memory. The main optimisation, however, takes place in the area of matrix multiplication. It can be seen from equations 2-7 and 2-13 that this is the most often used operation in calculating a Least Squares solution and is by far the slowest of all the matrix operations. This is due to the fact that in a normal matrix multiplication if, for example, we were multiplying matrix A by matrix B where A is **m** x **n** and B is **n** x **o** it would take **m** x **n** x **o** additions plus **m** x **n** x **o** multiplications to reach as solution. Conversely, if certain properties are known about the matrices that are being used, the number of computations can be greatly reduced. For instance, in VCV Time Domain analysis we know that the weighting matrix P is always strictly diagonal, that is, every value of $\mathbf{P}_{ij} = 0$ where $i \neq j$. This means that when we calculate $\mathbf{A}^T\mathbf{P}$, for instance, where **A** is **n** x 2 and **P** is **n** x **n** we need only calculate $\mathbf{A}_{ij}\mathbf{P}_{ii}$ for each element in the resulting matrix. Thus the problem has been reduced from being 2 x $\mathbf{n}^2$ additions and 2 x $\mathbf{n}^2$ multiplications to just 2 x **n** multiplications.

Similar calculation reduction can also be achieved by using the properties of the design, or **A** matrix. By using the property that the second row the the design matrix is always one, and the fact that anything multiplied by one is itself, we can further reduce the the number of calculations made in our previous example to just **n** multiplications. To put this into perspective, if we were carry out this matrix multiplication on an Intel® Pentium 4® desktop processor at 2.0 GHz where each double precision multiplication operation takes 8 instruction cycles and each double precision addition takes 6 instruction cycles(http://developer.intel.com/design/pentium4/manuals/248966.htm, 2002). If we assume that matrix **A** is a 1000 x 2 matrix and **P** is 1000 x 1000, it would take 2 x 1000^2 multiplications 2 x 1000^2 additions and 14 ms to complete this operation. Conversely, by using the optimised method the number of operations required to compute this matrix equation reduces to 1000 multiplications or 50 μs.

Future Enhancements

The **GDMS** offerers much functionality in the way of Time Domain Analysis, especially when VCV weighting matrices are employed. It does currently, however, only have limited support of Random Walk Time Domain Analysis. It is hoped that once an agreement has been reached as to how to specify a Random Walk Matrix it will be fully integrated into this product.

Interpolation

Introduction

Within GDMS several different interpolation algorithms have been implemented. These methods include nearest neighbour, linear, natural spline, cubic spline, and Newton divided differences interpolation. These methods were build to remove gaps in the dataset without damaging the original integrity of the data.

Research

At the start of this project we were given the requirement that the package support a number of different interpolation algorithms. There are hundreds, if not thousands, of different interpolation algorithms available, each with their own strengths and weaknesses. As it was not possible to implement them all we needed to pick a set of algorithms that would work with discrete datasets and provide sufficient results with given dataset. In order to narrow the complexity of our task, we chose to implement the interpolation methods available within MATLAB as it is a commercial package designed to work with a large range of datasets.

Nearest neighbour and linear interpolation are the two basic interpolation methods included in the package. They were included in case the more elaborate methods failed on a particular dataset.

Spline interpolation is very popular method as the smooth curve it produces is usually aesthetically pleasing. Many different types of spline are available, although most are unsuitable for this type of data analysis. B splines and their derivatives, for example, are undesirable as the resulting dataset may not pass through the initial set of points. Cubic splines were chosen because the resulting curve is always guaranteed to pass through each point in the original datasets.

As cubic spline are known to be inaccurate near the end points two different types of splines have been included. The first is natural splines which tend to be too straight near to the end points. Conversely, the other cubic spline implementation trends too curved near the end points. On large data sets (i.e. more than 1000 points), however, both method should converge to the same result.

Newton divided difference interpolation uses a divided difference table to provide a polynomial estimate of a dataset. Divided difference can produce extremely accurate results when approximating curves derived from a polynomial.

Implementation

The interpolation methods for this program were implemented within a class called **cepInterp** and share a common accesor method, **doInterp**. The required information require to be passed to **doInterp** is the original dataset, the sample rate and the type of interpolation. **DoInterp** then converts the given data set to Julian days and generates a new time scale before calling the individual interpolation methods.

Nearest Neighbour Interpolation

New points added to the dataset are set equal to the nearest point in the original dataset.

The algorithm used for nearest neighbour is:

```
Imports:

   Input Dataset: Original set of points

   Output Dataset: New dataset containing only the time locations of points

Exports:
   Output Dataset: Now full of data

For each point in the new dataset
   if within the bounds of original time line
      if a point exists in the input dataset at same time
         Output dataset point = input dataset point
      else
         set point equal to nearest point
      end inner if statement
   else
      generate extrapolation error and exit function
   end outer if statement
end for
```

Linear Interpolation

The new that points added to the dataset in this method are positioned on straight lines directly between the nearest two points.

The algorithm for this is:

```
Imports:
   Input: Original set of points
   Output: New dataset containing only the time locations of points
Exports:
   Output: Now full of data

For each point in the new dataset(i)
   if within the bound of original time line
      if a point exists in the input dataset at same time
         Output dataset point = input dataset point
      else
         Output(i,value) = (input(pos+1,value)-input(pos,value)/
                           (input(pos+1,time)-input(pos,time)
      end inner if
   else
      generate extrapolation error and exit function
   end out if
end for
```

Natural Spline Interpolation

This method determines the value of new points by running a natural spline through the dataset. Adjacent points are joined by cubics and the the first and second derivatives of adjacent cubics are matched to get a smooth curve.

The algorithm for this is:

```
Imports:
   Input: Original set of points (obs, 4)
   Output: New dataset containing only the time
           locations of points (newobs, 4)
Exports:
   Output: Now full of data

let n = size(input) -1

// Have to Fit equation y = $a(x-x_0)^3+b(x-x_0)^2+c(x-x_0)+d$

create arrays of length n for a,b,c and d

create array of length n called h // for holding differences
for i = 0 to n-1
  h(i) = input(i+1,date)-input(i,date)
  d(i) = input(i,value)
end for

// Create array to hold second derivatives
create an array of length n + 1 called S

Set the first and last values of S equal to 0
// initialise the middle value of S
For i = 1 to n - 1
  S(i) = 6*((input(i+1,value)-input(i,value))/h(i) -
            (input(i,value)-input(i-1,value))/h(i-1))
end for

// create matrix for transitional values
create a n-1 by n-1 matrix called t

// initialise t
// Starting from all values = zero
//first row of t
t(0,0) = 2*(h(0)+h(1))
```

```
t(0,1) = h(1)
//middle of t
For i = 1 to n -3
  h(i,i-1) = h(i)
  h(i,i) = 2*(h(i)+h(i+1)
  h(i,i+1) = h(i+1)
end for
t(n-2,n-3) = h(n-2)
t(n-2,n-2) = 2*(h(n-2)+h(n-1))

Augment t with the middle n-1 rows and then row reduce
this gives correct value for the middle n-1 values of s

// get values of a,b,c
For i = 0 to n-1
  a(i) = (S(i+1) - S(i))/(6*h(i))
  b(i) = S(i)/2
  c(i) = (input(i+1,value)-input(i,value))/h(i) -
         (2*h(i)*S(i)+h(i)*S(i+1))/6
end for

// fill output dataset
For each point in output(i)
   if within the bound of original time line
      if a point exists in the input dataset at same time
         Output dataset point = input dataset point
      else
         Output(i,value) = a(i)*(output(i,date)-input(pos,date))^3 +
                           b(i)*(output(i,date)-input(pos,date))^2 +
                           c(i)*(output(i,date)-input(pos,date)) +
                           d(i)
      end inner if
   else
      generate extrapolation error and exit function
   end out if
end for
```

Optimisation note: To increase the speed and efficiency of this algorithm the **n-1** by **n-1** tridiagonal matrix **t** has been represented by a **n-1** by 3 dimensional matrix in the program. In addition, some commands have been grouped differently for similar reasons.

Divided difference Interpolation

Newton divided differences builds a table of divided differences and then uses this table to generate a polynomial representation of the data. To increase the accuracy of the results, this implementation also tracks the error of the equation limits them to achieve best results. Error limitation is calculated by limiting the order of the divided difference table when the error term starts growing.

The algorithm used for Newton divided differences is:

```
Imports:
   Input: Original set of points (obs, 4)
   Output: New dataset containing only the time
           locations of points (newobs, 4)
Exports:
   Output: Now full of data

n = size(input)-1

//construct first level of difference table
for i = 0 to n-1
  diff(0,i) = (input(i+1,value)-input(i,value))/
              (input(i+1,date)-input(i,date))
end for

// calculate first error term
errormod = (input(1,date)-input(0,date))/2
errorstart = input(1,date)+input(0,date))/2
error(0) = errormod*diff(0,0)
order = 1;

//Construct rest of divided differences table
For i = 2 to n
  For j = 0 to n-i
    diff(i-1,j) = (diff(i-2,j+1)-diff(i-2,j))/
                  (input(j+i,date)-input(j,date))
  end for

  errormod = errormod * (errorstart-input(i-1,date))
  error(i-1) = errorMod * diff(i-1,0)
  if error (i-1) > error (i-2)
```

```
    exit build table for loop
  end if

  order = order+1
end for

For each point in output(i)
   if within the bound of original time line
      if a point exists in the input dataset at same time
         Output dataset point = input dataset point
      else
         Output(i,value) = input(pos,value)
         mod = 1
         for j = 0 to order -1
          mod = mod * (output(i,date)-input(pos,date))
          Output(i,value) = Output(i,value) + mod*diff(j)
         end for
      end inner if
   else
      generate extrapolation error and exit function
   end out if
end for
```

Future Enhancements

In hindsight implementing the MATLAB set of methods was not an optimal choice, as the interpolation method that they used were too susceptible to very high frequency noise. To overcome this problem we would recommend the implementation of a number of new, noise resistant, interpolation methods. Noise resistant interpolation will include methods that work on weighted averaging of local points, and least squares based methods. By integrating these method you will get results that will more accurately represent the dataset as a whole.

Windowing

Introduction

Research

Locating algorithms for the implementation of the initial windowing algorithms was trivial. The majority of these are well documented and are easily implemented. Dolph-Chebyshev window algorithm, however, was a little more elusive. Initial attempts to implement this algorithm were unsuccessful. Two different approaches were taken; initially an attempt was made to implement this purely in the time domain, based on an algorithm sourced from the speech recognition research at the Institute for Signal and Information Processing, ISIP, located at Mississippi State University. After many hours spent trying to validate the algorithm, it was decided that this could not be used. A second attempt was made using a window generated in the frequency domain, using code based on the original 1947 paper by Dolph. This algorithm, however, only supported odd window sizes, and the implementation of the IFFT library contained in the **GDMS** required that its window size be 2^n and thus this algorithm had to be discarded. The final algorithm that was implemented is based on FORTRAN code published in 1977 by IEEE Acoustics, Speech, and Signal Processing Society,

Implementation

The windowing subsystem is designed to be extensible. Using OO techniques, a general algorithm for generating the window is combined with a specialised class which generates each individual coefficient. This means that to add another algorithm costs as little 10 lines of code, including user interface integration.

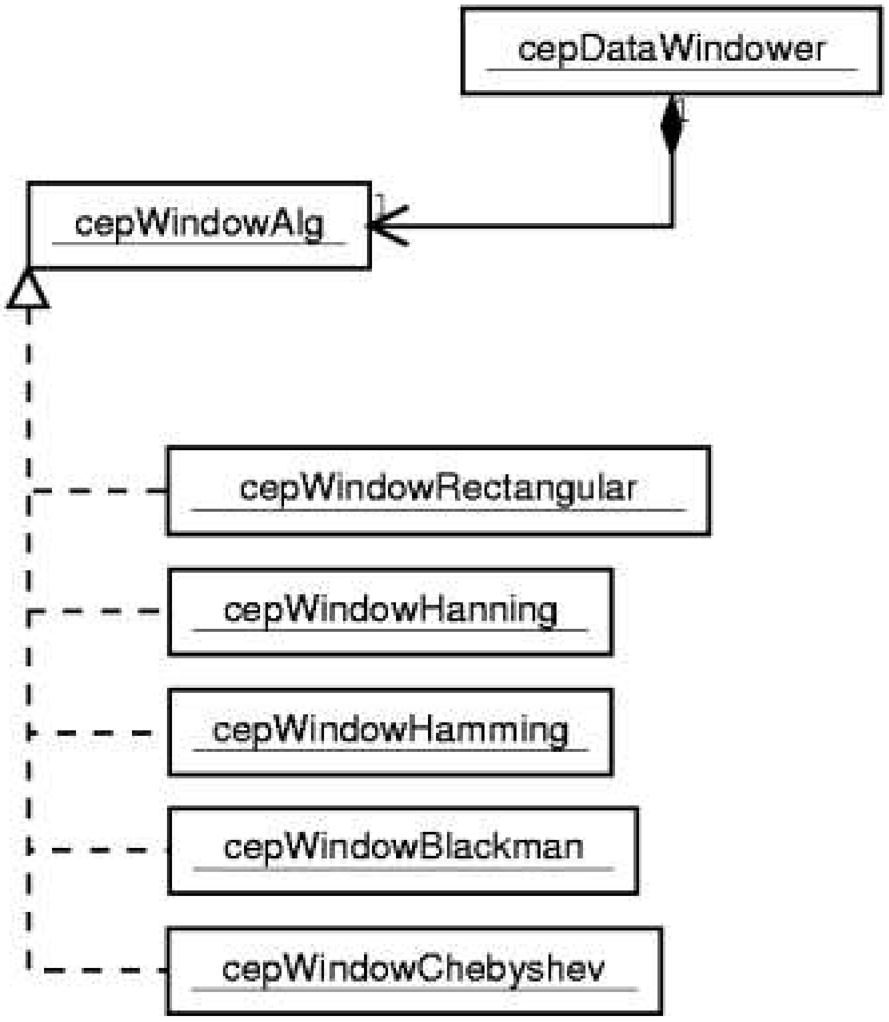

Figure 3-4. Windowing Class Structure

For all windows, coefficients can be generated for an arbitrary window size. The requirement that the windowed data be transformed to the Fourier domain imposes further limitation in that the Fourier libraries we are using will only operate on a data set of size 2^n, $0 < n < \infty$. This restriction has been implemented at the User interface level, maintaining the flexibility of the windowing subsystem.

Optimisation Issues

Due to the simple nature of the majority of the algorithms in this module, speed is not generally an issue. With the exception of Chebyshev, the most expensive calculation involves two cosine and four multiplication operations. Since the size of the windows used tend to be small this is a relatively insignificant overhead. Dolph-Chebyshev is a little different, as this window is calculated in the Fourier domain, then transformed into the time domain via an IFFT it is computationally more expensive. In order to support the flexibility of the windowing subsystem, this algorithm implements its own IFFT. Due to the small size of the window used, however, this cost is minimal. In fact, it is more than offset by the increase in flexibility and confidence in the integrity of the algorithm. To address performance concerns and to speed up the process as a whole, the number of calculations performed in this subsystem has been carefully considered and recalculation of the coefficients is only performed when it is unavoidable.

Future Enhancements

The set of algorithms supplied in this release is by no means exhaustive. There are some notable absences, such as the Kaiser window, which would be a valuable addition to the **GDMS**. Due to the flexible design of this subsystem, the addition of another algorithm can be achieved at minimal expense and is encouraged. Another avenue which might be explored with regard to enhancements would be the simplification and optimisation of the Dolph-Chebyshev algorithm. This was a port of a FORTRAN function and was implemented faithfully as maintenance of data integrity the primary concern. This algorithm uses an inverse Fourier transform internally and for windows of significant size this could prove to be expensive. Optimisation of this might prove to be beneficial if window sizes are becoming large.

Frequency Domain Analysis

Introduction

Being a common data analysis tool, there are a number of FFT libraries available in various languages. We therefore decided that it may be more efficient to use a third party library for a number of reasons and given that there were many available, we assumed that at least one package could be found that would meet our requirements. Some considerations affecting our choice of FFT packages is the format of the FFT routine, the language they were written in and whether it was a stand alone library, or simple class or a template class.

Research

The first library found in our research was the Fastest Fourier Transform in the West (FFTW) library from MIT. This library is widely referenced and documented, and is reputed to be the most efficient FFT library in existence. The FFTW library, however, was written in C, and was therefore discarded an attempt to maintain language consistency through the project. In addition, is the FFTW is quite large, and with the number of dependent libraries for **GDMS** increasing, it was preferable to find a smaller library.

Another candidate was discovered after testing had already commenced with the FFTW library. This was a complex template class to compute a FFT written in C++ (Arndt, 2002), and was considered a suitable choice as it was less complex than FFTW, and also suited our loose Object Oriented design. A further benefit of using this implementation of an FFT was due to the fact that it was a Template Class object was, therefore, not restricted by type. This gave us greater flexibility in the type of data that could be FFT and what format the data would be returned in. For example, we might desire simple **double** values for PSD plotting, or perhaps **complex** values for further processing after the FFT had been preformed. The FFT class also provides the ability to perform inverse FFTs on a given set of data. As far as the issue of speed was concerned, no evidence was found during the testing of the complex template FFT class to suggest that it would be much slower than FFTW. We therefore decided to integrate this implementation into the project.

Implementation

The FFT class could not be integrated without modifications. Firstly, there was an issue with the sign of the Fourier transform. This refers to the FFT theory section of this document where the standard Fourier formulae were discussed. The original FFT algorithm operated contrary to this standard and it produced incorrect results. Specifically, the algorithm implemented used the opposite signs for forward and inverse transforms. To overcome this problem, the algorithm needed to be modified such that it return the conjugate of what it originally computed, thereby producing the correct results. This problem was overlooked in the initial algorithm selection process and appeared only later in the unit testing. Further information specific to testing can be found in the Testing chapter later in this document.

Another integration problem occurred because of the way in which the original template class deals with data structures. Firstly, the class computes FFTs on and returns an array of data, where as **GDMS** uses matrices to pass data from class to class and thus it needed to be modified to accept this data type. Another change that was required was that the FFT library needed to modified to calculate a frequency scale in order to enable PSD plots to be graphed. To calculate this scale, the FFT class first determine the sampling rate of the data. This is achieved by examining the distances between consecutive sample dates, the sampling rate simply being the distance between them multiplied by the number of days in a year. This implementation, however, assumes that the data is regular and in fact, functionality in the FFT class has been implemented to prevent the transformation of irregular data. The period used for the sample rate calculation is one day. The frequency is then calculated by taking the inverse of the sample rate and is measured in cycles/day.

Future Enhancements

In the frequency domain analysis, the **GDMS** package currently only carries out PSD plotting and inverse IFFTs. Additional functionality such as the FFT class returning the raw complex FFT data is a reasonably trivial task to complete. This would allow operations on the complex data resulting from the FFT if required. Its omission from the final specifications was solely due to time constraints. This functionality is already supported within the FFT framework, so adding this in future releases should be trivial. A further enhancement to the frequency domain analysis of the **GDMS**, would be to allow the removal of erroneous data in the frequency domain, and then allowing the user to transform the data back into the time domain via an Inverse FFT. This would be an extremely useful function, as the detection and removal of erroneous data, such as signal noise is not easily achieved in the time domain but is reasonably trivial in the frequency domain. By allowing the transform of the remaining data back into the time domain, the effects of removing the erroneous signals can be determined.

CepMatrix and Template Classes

Introduction

At a very early stage in the design of the **GDMS** it became clear that some form of matrix object would be required, capable of fulfilling following requirements:-

- Storage of a given data set and the results of any transformation preformed on it.
- Capability of preforming matrix operations including addition, subtraction, with either another matrix or by a scalar value, division and transposition. In addition, support for many other operations was required in order to preform all the functionality of this system.
- Support for both two and three dimensional matrices.

In addition, the matrix object itself had to be fast, flexible and reliable as it lies a the heart of the **GDMS**. In order to meet these requirements, therefore, the **cepMatrix** class template was created.

Storage of data sets

The first problem encountered in creating the **cepMatrix** was the need to support multiple data types. This was due to the fact that while the original data sets themselves as well as the transformations in the time domain were all of type **double**, once a transformation into the frequency domain was conducted the data became of type **complex**. This problem was overcome, however, by using one of the most powerful C++ functions, namely Template Classes. A Template Class works by creating a class which is unrestricted by data type. The data type of the object itself is only specified when the object is instantiated, thereby allowing the same segment of code inside the Template Class to work with multiple types of data such as **int**, **float**, **double** and **complex**.

cepMatrix operations

Another key requirement of **cepMatrix** was the ability to carry out a whole range of matrix operations such as addition, subtraction, division and transposition. Much of this functionality was achieved by the use of operator overloading. The method of operation overloading is another feature of C++ that allows the simplification of function calls, especially when relating to mathematical operations. For example, in order to add matrices **A** and **B** together and save the results in **A** without operator overloading may look like this:-

```
copy(A, multiply(A,B));
```

Conversely, if the += operator were overloaded the problem would simplify to

```
A += B;
```

which is much easier to understand and actually removes a function call. In **cepMatrix** the following operations are overloaded:-

- A += B :- adds matrix B to matrix A and stores the result in matrix A
- A -= B :- subtracts matrix B from matrix A and stores the result in matrix A
- A *= B :- multiplies matrix A by matrix B and stores the result in matrix A
- A *= c :- multiplies matrix A by the scalar value c and stores the result in matrix A
- A /= B :- divides matrix A by matrix B stores the result in matrix A
- A = B :- copies matrix B to matrix A
- A == B :- returns true if matrix A is equal to matrix B
- A != B :- returns true is matrix A is not equal to matrix B

The **cepMatrix** Class Template also supports other operations built to facilitate implementation inside the **GDMS** application. This includes an operation to query the matrix and determine if it is a strictly diagonal and operations to determine the maximum and minimum values of a given column in a matrix. There is also an operation to resize a matrix by adding a given number of rows to it. In addition, there are several get and set accesor methods provided for accessing individual elements of a given matrix.

Two Dimensional and Three Dimensional Matrices

A third, key requirement of the **cepMatrix** Template Class was support for both two and three dimensional matrices. This is due to the fact that when data is windowed it requires a new matrix for each frame that is returned. It was decided that the preferred method of dealing with this problem was to keep all the data inside one **cepMatrix** object rather than having to use another container such as an array or vector. This, in turn, enabled the passing of data between objects in a uniform way and simplified the function calls required to access the given data.

Due to the fact that it was not strictly necessary to add any extra functionality and time constraints, three dimensional matrices are implemented purely as storage devices. As such, there is limited support for these matrices inside this object, that is, they can be created, accessed and copied but they can not be used in any other matrix operations such as multiplication, addition, subtraction and division.

Both two dimensional and three dimensional matrices are stored as arrays and are treated as separate member variables within the **cepMatrix** Template Classes. This approach was taken for several reasons, firstly due to the fact that **cepMatrix** is a Template class all member variables had to be of a primitive type, that is, other templates can not be instantiated inside these classes. This restriction on Template Classes is actually a compiler dependent and is not a limitation of the language specification, per se. It does, however, arise when using **g++** which is the most widely use compiler on the operating systems in which the **GDMS** is required to be used. This limitation, was therefore, applied the to **cepMatrix** class and thus each matrix was stored, internally as an array. In addition, the decision to treat two dimensional and three dimensional matrices as separate member variables was made for reasons speed in preforming matrix operations. This is due to the fact that if all matrices were stored using the same member variable, that is, the three dimensional member variable it would mean that an extra memory reference would be incurred each time a matrix element is accessed. Consequently, some matrix operations would take almost twice as long to complete.

Future Enhancements

Whilst the **cepMatrix** Template Class includes all the functionality required to of it to be a powerful tool inside the **GDMS** system there are a few enhancements that could be made. As mentioned, three dimensional matrices, unlike two dimensional matrices have very limited functionality inside this class. In future releases, it would be anticipated that three dimensional matrices have all the functionality associated with two dimensional matrices. Other future enhancements that could be made, would be to allow greater flexibility in re-sizing matrices, the implementation of iterators and the overloading of the [] operators to make the **cepMatrix** Template Class a full C++ Template container.

User Interface

Introduction

The **GDMS** user interface is the most prominent part of the application. It is the main way in which the user experiences the application. **GDMS** currently supports two user interface variants:

- The X windows user interface (the program called **GDMS**)
- The batch user interface (the program called **gdms-batch**)

This section will deal primarily with the research undertaken for implementing the X windows user interface. This is not only due to the fact that the scale of implementation required was much greater for this variant, and because there were many more alternatives available.

Graphical user interface

There are a plethora graphical user interface toolkits available today. As each has it's advantages, a small survey of the options considered, and then a discussion of why *wxWindows* was selected is appropriate.

CDE

The Common Desktop Environment (CDE) was jointly developed by Hewlett-Packard, IBM, Novell and Sun Microsystems. These companies have subsequently adopted CDE as the standard window manager as shipped on a variety of their various operating systems (Chapman 2002). CDE uses the Motif windowing toolkit.

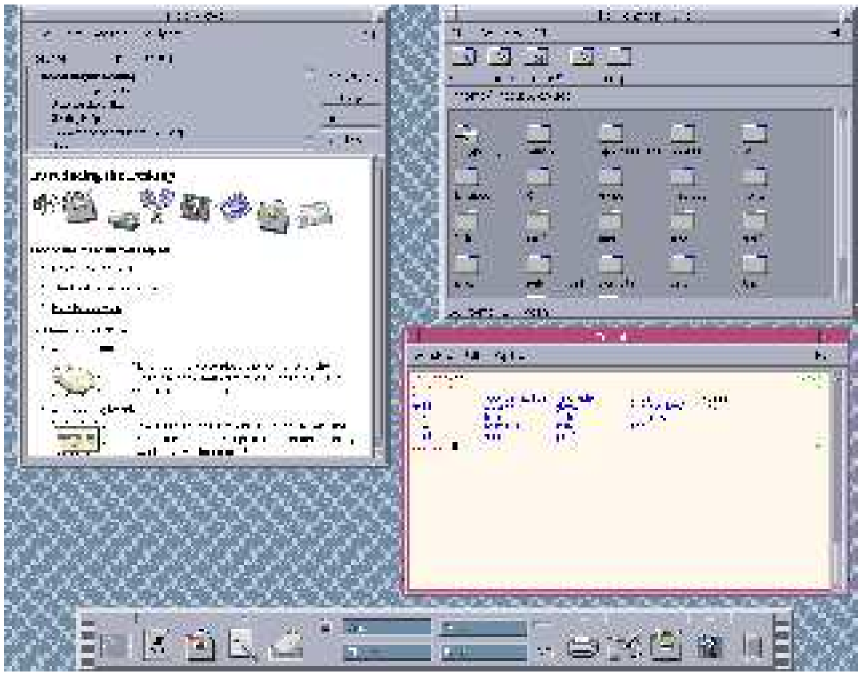

Figure 3-5. Sun Microsystem's default CDE configuration (Chapman 2002)

CDE

KDE development started in October 1996 (KDE 2002), as a replacement for Sun's CDE. KDE is based on Trolltech's qt widget set (Trolltech 2002). KDE is available as an installation option on the majority of the Linux distributions available.

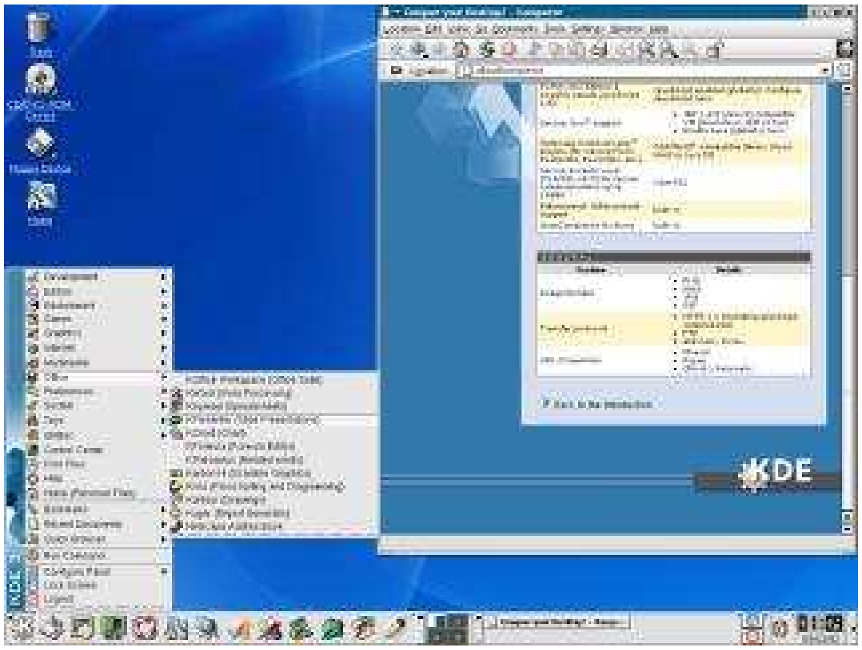

Figure 3-6. The default KDE theme, Konquerer (KDE 2002)

Gnome

Gnome is the default window manager and windowing toolkit set for many Linux distributions, including Red Hat Linux, which the Survey Lab at the University of Canberra makes extensive use of. Gnome is implemented using the GTK windowing toolkit (GTK 2002).

Sun Microsystems have recently announced that from Solaris 8, Gnome will be available as a window manager. They have also announce that CDE will be replaced with Gnome in future releases (Sun Microsystems 2002).

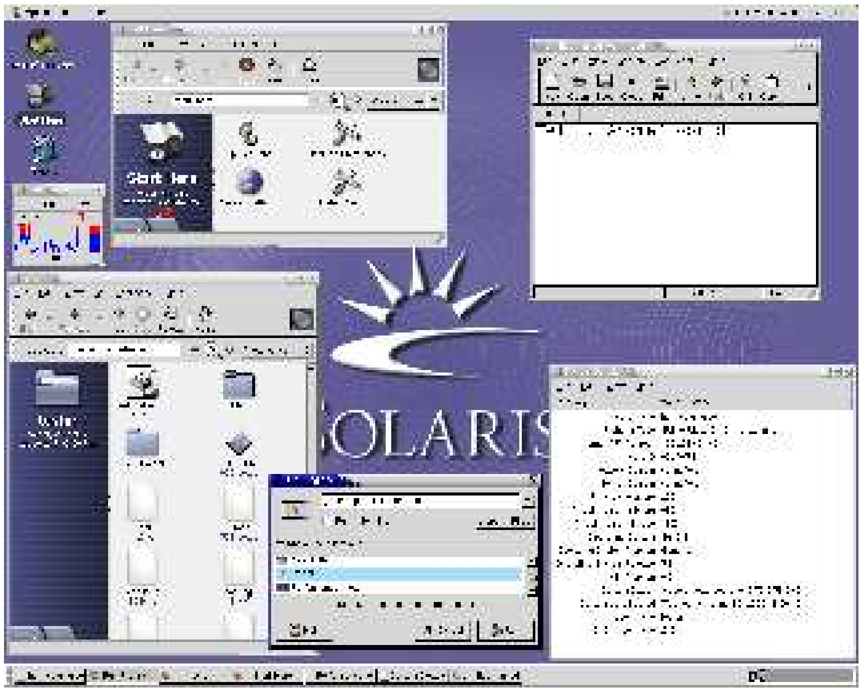

Figure 3-7. The Solaris version of the GNOME desktop (Gnome 2002)

Microsoft Windows

Whilst not a X windows toolkit, Window's MFC toolkit was considered as a development alternative for **GDMS**. However, the survey lab does not currently contain any Microsoft Windows machines, so this option was not pursued any further.

Figure 3-8. The Windows XP desktop (Microsoft 2002)

wxWindows

Whilst wxWindows is not a window manager in the same sense as the others discussed in this section, it has a variety of advantages relevant to **GDMS**. These include:

- *The ability to use a variety of windowing toolkits*. wxWindows supports GTK and Motif for Unix and Unix-like operating systems. Windows is also supported.
- *Cross platform functionality*. This allows **GDMS** to run on a variety of Unix and Unix-like operating systems.

Windowing toolkit selection

Based on the factors outlined above, the group selected wxWindows for the user interface implementation of **GDMS**. As previously mentioned, all of these toolkits require developers to write in C++, which was one of the determining factors in the language choice for the development of the application.

Batch user interface

There are two common implementation methodologies for scripting languages. Both of these were considered for the **GDMS** implementation. The most common manner for implementing the parser and grammar for a scripting language is by using the compiler construction tools **yacc**, and **lex** (or their free versions, **bison** and **flex**).

Compiler construction tools

Lex and **flex** are parser generation tools. They take a lexical specification file, and generate a LALR finite state machine for matching input strings (Aho, Sethi &

Ullman 1986). A sample of a lexical specification is shown below (in this case for the PDF file format):

```
<INITIAL>\%PDF-[0-9]+\.[0-9]+    {

  debuglex(yytext, -1, "version");

  yylval.sval.data = (char *) returnStr(yytext, -1);

  yylval.sval.len = yyleng;

  return VERSION;

  }


<INITIAL>[+-]?[0-9]+\.[0-9]+     {

  debuglex(yytext, -1, "floating point");

  yylval.sval.data = (char *) returnStr(yytext, yyleng);

  yylval.sval.len = yyleng;

  return FP;

  }


<INITIAL>[+-]?[0-9]+\.[0-9]+\>\> {

  debuglex(yytext, -1, "floating point");

  yyless(yyleng - 2);

  yylval.sval.data = (char *) returnStr(yytext, yyleng);

  yylval.sval.len = yyleng;

  return FP;

  }
```

In this example, two tokens are defined, VERSION, and FP. The syntax for these specifications is relatively simple. The first field, which is wrapped in angled brackets, is the name of the state that the specification should be used in. All of the examples shown here are for the default state, known as INITIAL (Levin, Mason & Brown 1990). What follows this is a posix compliant regular expression to be matched. Finally, within the braces lies C code to execute when a match occurs. In each of these examples, the C code writes a log entry for the match, sets up the data structure symbolising the match, and then returns the name of the lexical token which was matched.

Yacc and **bison** define the grammar which uses the tokens defined by **lex** or **flex**. In effect, this defines valid orders for the tokens to appear in, and what operations should be performed when a set of tokens is matched. Error handling for undefined token combinations is also supplied by the grammar. This is done through the use of hooks for reporting to the user in a manner which is appropriate to that particular application.

A sample grammar specification, in this case for SQL (Connolly & Begg 1998) is as follows:

```
SQL      : create SQL | insert SQL | select SQL

         |

         ;
```

```
create   : CREATE TABLE STRING '(' colvalspec ')' ';' {}

         ;


insert   : INSERT INTO STRING '(' colvalspec ')' VALUES

           '(' colvalspec ')' ';' {}

         ;


select   : SELECT cvsaster FROM STRING {} wsel ';' {}
```

A grammar consists of three building blocks. Each of the tokens which can be returned by the lexer is termed a *terminal*. Examples from above include *CREATE*, *TABLE*, *STRING*, *INSERT*, *INTO*, *VALUES*, and *SELECT*. These terminals are used by *non-terminals*, such as *SQL*, *create*, *insert*, and *select* from the example above. The rule which groups a non-terminal with a series of other non-terminals, and terminals, is called a *production* (Aho, Sethi & Ullman 1986).

This grammar defines a language in which a SQL statement can consist of either a *create*, an *insert*, or a *select* statement, as shown by the SQL production. The empty non-terminal on the SQL production ensures that an end of file or empty line is also matched. It should also be noted that the SQL production is recursive, except for this case which matches the empty line or end of file.

Hand coded parsers

The alternative to using the compiler construction tools is to develop a language parser by hand. Depending on the complexity of the grammar to be implemented, this can be quite a realistic alternative. On the other hand, if the grammar is complex, then it can rapidly escalate into task of much larger scale, comparable to that of the rest of the application. This option was found to be the most suitable for the **GDMS**. The grammar for the scripting language implemented by the batch version of **GDMS** is relatively simple, in that all commands are parsed based on only one line, and no recursion within productions is required.

Implementation

As mentioned earlier in this section, the graphical user interface to **GDMS** was implemented using the wxWindows windowing toolkit. The batch user interface was developed using a hand coded parser written in c++. This has maximised the amount of design flexibility enjoyed by the development team, whilst minimising the amount of unnecessary code development.

Integration

A significant portion of the **GDMS** user interface development was integrating the various classes and features with the graphical user interface. This integration has been facilitated by a standard process for integrating mathematical functionality. The process is as follows:

- *All functionality is either accessed from the mouse, or a menu*. This leads to a consistent user interface, and minimises user confusion.
- *The output of all mathematical operations is a new dataset*, and hence a new tab in the main application window. This has several advantages, including:-
  - Facilitating the implementation of the batch interface to the application.
  - Allowing ease of comparison of the dataset before and after the application of the mathematical operation

Further documentation of the actual functionality available in the **GDMS** application is available in the user manual.

Future Enhancements

Whilst the user interface for **GDMS** is fully functional, there are a variety of improvements which could be made in the future. These include:

- *Online help*: the application does not currently provide any help in the user interface. wxWindows supports the display of HTML files within windows, which would an efficient and standard method for the implementation of online help within the application.
- *Greater resilience to errors*: if an error is discovered, the application currently abandons all further processing on that dataset. This should be improved in the future, so that a failure will not stop all calculations.
- *Greater flexibility in the batch user interface*: Further research into what other commands should be implemented needs to be undertaken.
- *The ability to determine the value of the nearest data point by hovering over it*: Currently the application checks the location of the mouse and looks up the relevant graph values. This is misleading as the display resolution means that is almost impossible to determine with any accuracy, the value of the data point at which you are pointing.
- *The ability to edit out points from the dataset*: once accurate mouse pointing is integrated this will be trivial.
- *Meaningful graticules*: currently the graphs are displayed with a fixed number of graticules, and the scale is modified accordingly. This can be deceptive and must be revisited in a future release. Revision of this code was attempted late in the process, but due to time constraints and an uncooperative plotting library this was abandoned.
- *Analysis of plotting precision*: currently the precision of the graphs is such that using high sampling frequencies and cause them to lose precision. Analysis of the plotting functions needs to be considered to minimise this.

Chapter 4. Testing

Testing

Introduction

From the outset, it was decided that a testing regime was required to support debugging of subsystems and validation of mathematical routines. The basic requirements for the testing infrastructure were:

- *Standardised:* for all developers.
- *Automated:* to allow test execution with no user input.
- *Simple:* so as to minimise load placed on developers.
- *GPL:* The code must be released under the GPL.

Implementation

Unit testing, as adopted by the extreme programming community, was chosen as the ideal framework to support our system. A number of different libraries were examined as possible candidates and CPPUnit was selected as the most appropriate.

CPPUnit is a C++ framework based closely on the very mature JUnit testing framework. It allows the developers to write simple, focused tests, and provides a number of assertions with which to validate results. Additionally, the tests have special methods which are designed to support test independence. The first is setUp(), which is executed immediately preceding each test, and the second is tearDown() which is run following after each test. These allow the developers define dependencies for each test and deallocate resources that the test required.

JUnit makes extensive use of the reflection Application Programmers Interface (API) to locate test methods. Methods are deemed to be tests if their name begins with the word "test", they take no parameters and their return type is void. Due to limitations in the C++ language, there is no reflection API as such, thus ruling out reflection as an alternative. Consequently CPPUnit requires the user to register tests with the TestCaller via a static method. The Test class is then registered with the TestRunner and all registered tests are run. CPPUnit provides a number of macros which ease this registration process.

Errors and failures in CPPUnit are reported via a number of macros. These macros provide the ability to test for equality and truth and allow the user to supply meaningful error messages, thereby facilitating the debugging process. Failures, with the exception of segmentation violations, do not cause the tests to exit. If a failure occurs, the current test reports the error and returns, allowing the test suite to continue.

Early in the development of the project, a template was developed to provide a simple means by which developers could start their testing. A sample test, based on this template follows.

```
namespace {

class Test : public CppUnit::TestFixture {


public:

  /* default constructor */

  Test() : CppUnit::TestFixture() {}
```

```
        /** setup - run prior to each test */

        void setUp ()

        { /* initialise any resources*/ }


        /** teardown - run after each test */

        void tearDown ()

        { /* free any allocated resources */ }


       /**

        * constructs a test suite.

        * Add your tests to the suite by copying and editing the ad-
  dTest call

        */

        static CppUnit::Test *suite()

        {

          CppUnit::TestSuite *suiteOfTests = new CppUnit::TestSuite( "Test" );


          /* REGISTER YOUR TEST HERE */

          suiteOfTests->addTest(

            new CppUnit::TestCaller<Test>( "sampleTest", &Test::sampleTest ) );

          return suiteOfTests;

        }


      protected:



        /**

        * DEFINE YOUR TESTS HERE:

        * make your tests protected since you do not need to expose them

        */


        /** simple test 1. uses a generic assert true macro */

        void sampleTest ()

        {

          char* foo = (char*)malloc(32);

          CPPUNIT_ASSERT_MESSAGE ("foo failed", foo != 0);

          free(foo);

        }


      }; // end Test
```

```
/**
 * Register the test immediately after definition. This should probably
 * be done in the class header file for larger projects
 */
CPPUNIT_TEST_SUITE_REGISTRATION( Test );

} // end namespace
```

Testing has been used extensively to support the validation of the mathematical subsystem. There are currently 84 tests spread across the 10 mathematical modules.

User Interface Testing

The user interface is exhaustively tested across all datasets. The tests are generated automatically and seek to locate datasets with peculiarities that might cause the failure of the plotting library. There are currently 1131 tests dedicated to exercising this body of code.

Conclusion

The testing framework proved invaluable in the process of bug isolation and resolution. An additional benefit has been the early recognition of design problems, such as tight coupling between the subsystems and the user interface at an early stage in the project. In summary, without the support of a system such as CPPUnit, the **GDMS** would not have the level of robustness that it does at this time and mathematical validation would have been difficult.

Chapter 5. Documentation

Introduction

Documentation is of significant importance in any project, particularly with those that are not confined to a finite scope, and the **GDMS** is one such project. In a time series analysis system, there are an enormous number of possible features, therefore a careful balance must be drawn between the functionality and the documentation. We needed to maximise our coding time whilst still providing sufficient documentation to both meet the requirements and include all of the necessary information that be required by any future developers and users. This led to our decision to use a number of tools to assist with the documentation.

Docbook

Docbook is system for writing documents using SGML or XML. It facilitates structure and is well suited for writing books or papers of this kind. Unlike formal word processors, docbook greatly reduces the time spent on standardising the format of a document by providing a number of features. These include:-

- Ensuring title fonts are consistent in type and size
- Tables, equations and figures are listed
- Table of contents is included
- Document layout and text justification are automatic.

The syntax used for docbook is SGML, which is very similar to HTML. As most of the group members were already familiar with this type of syntax, it greatly reduced the learning curve to use docbook, which might have otherwise rendered the process counter productive.

Docbook Tools

Docbook is easy to automate, which was another determining factor in using this tool. Automation was achieved by using Docbook tools, a suite of programs used to generate the docbook output. The process is similar to compiling source code, utilising a makefile to target the desired files and build them. These files can then be structured in a hierarchical manner to ensure consistent and easy organisation. A typical document might contain a top level file, for example, thesis.sgml, which links the other files, such as timedomain.sgml and theory.sgml. This file also contains the top level information such as whether the document is a book or an article, in which case various pages are generated automatically by docbook accordingly. Thesis.sgml might also contains a preface and glossary and other such general information sections.

The target SGML files are linked from the top level file by issuing a command specifying the target file. The following example illustrates a top level file for a simple book:

```
<book>

  <bookinfo><title>Geodetic Date Modelling System: Thesis</title>

     <execute><cmd>builddb</cmd><input>authorgroup.sgml</input></execute>

     <execute><cmd>builddb</cmd><input>abstract.sgml</input></execute>

  </bookinfo>

  <chapter id="ch01"><title> Chapter 1 - Introduction and Theory</title>

    <execute><cmd>builddb</cmd><input>Introduction.sgml</input></execute>

    <execute><cmd>builddb</cmd><input>Theory.sgml</input></execute>
  </chapter>

  <chapter id="ch02"><title> Chapter 2 - Implementation</title>

    <execute><cmd>builddb</cmd><input>Implementation.sgml</input></execute>

    <execute><cmd>builddb</cmd><input>Conclusion.sgml</input></execute>

  </chapter>

  <chapter id="ch03"><title> Chapter 3 - Conclusion</title>

    <execute><cmd>builddb</cmd><input>Conclusion.sgml</input></execute>

  </chapter>

</book>
```

The above example describes a simple book with three chapters. Each of the target SGML files would in turn be divided into section within the chapter. This method of documentation has its benefits, some of these being:-

- Documentation changes need only be made in the one place, in this case a single file. This is advantageous in case where text is repeated, much like the re-use concept in programming.
- Formatting and layout issues are dealt with by docbook.
- The layout conforms to a general standard adhered to by a large majority of publishers.
- Table of Contents and indexing of equations and figures are dealt with automatically by docbook.

Our use of docbook, however, was not altogether seamless. We found version inconsistencies in different Linux distributions, and even after updating all of the development machines to use the same version of docbook, we ran into further trouble. It would seem that different distributions are not consistent when it came to building the documentation. An example of this, is that on some of the machines, although the docbook scripts would build the document, it was incomplete. This was due to the fact that SGML syntax errors were not being caught by the scripts and as a result, later in the build process when the build finally broke, it was extremely difficult to determine exactly where the problems originated. The same docbook version used on a different distribution caught the SGML errors allowing us to debug and fix any problems. Furthermore, the templates on the different distributions varied such that page numbering was not always consistent. This was even found to be the case on different versions of the same distribution. To overcome these issues, we simply generated the final submission of the documentation using the most suitable template.

Autodocbook

Autodocbook is tool employed for the documentation of source code. It is a simple Perl script designed to parse C or C++ source files extracting comment blocks within specific tags and turning them into docbook SGML files. These can then be used to create man pages, info pages and HTML documentation. This again serves to minimise the need to make changes in several places, for example, if comments need to be changed for a specific class or file, it is done in the class file and the Autodocbook script is run again. The second situation is in the creation of the other possible documents, such as man pages, info pages and HTML files. The appropriate conversion script is run on the target files and the task is complete.

Chapter 6. Conclusion

Conclusion

The GDMS packages offers a suite of time series analysis tools intended for the processing of data from various geodetic sources. The package was designed and implemented to be as flexible and platform independent as possible given the time constraints under which the group were working. We aimed to provide a package which would complement and enhance currently available analysis tools. This involved providing both time and frequency domain analysis, windowing and interpolation functionality. The loose object oriented design provides the **GDMS** with a great deal of flexibility and extensibility. The documentation provides both users and developers with an understanding of how the system was constructed facilitating any changes that may be desired.

The success of this project can be measured by a number of factors. Perhaps the first and foremost concern in a software application is whether or not it achieves its stated objectives. In the case of the **GDMS**, we started with a broad requirements description, which were refined as much as possible in the early stages of the project, however, many were undiscovered until much later in the development process, leading to the delay in completion of many of the mathematical modules due to their complexity. In order for the application to be considered *useful*, it was essential to implement as much functionality as possible yet as the deadline drew near, we were required to assess and implemented the most important functionality for the application. This was largely a role that Professor Peter Morgan played, being the key consultant in project.

A number of valuable lessons have been learnt from the **GDMS** development process, not all of which relate directly to application development itself. That which we consider to be the most critical lesson here, is planning. As previously mentioned, the original requirements were broad and did not cover many of the finer details required. It was during our initial requirements definition process, that we should have sought to discover as much as possible about exactly what was involved with various features in the broad specification. Our mathematical body of knowledge before the project began, was considered at least enough to proceed, yet perhaps not quite enough to foresee these finer specification details. A more comprehensive requirements definition phase in this project would quite likely have reduced the number of problems that arose later, with missing or incorrect functionality. Programming in a Linux environment, for the duration of the project, has also provided us with an opportunity to expand our knowledge base. We have also furthered our experience in C++ object orient design. Furthermore, our knowledge of geodetic data analysis and processing has benefited, not only in terms of the theory and processes involved, but also the background, applications and its significance to geodetic community.

From the very beginning, the **GDMS** was always intended to be an evolving system and it was not our goal to provide a definitive time series analysis package. What we have provided, however, is both a fully functional system that complements and expands upon existing data analysis application, as well as one which can be further built upon. The project has also served to enhance our knowledge of system software design and implementation. In addition, we have gained valuable experience in the area of time series analysis. As we have achieved our stated objectives, we feel that the project has been a success.

Glossary of Terms

API

Application Programmers Interface.

CPPUnit

The C++ unit testing framework.

CVIEW

C File View

DFT

Discrete Fourier Transform

GAMIT

Global Positioning System At Massachusetts Institute of Technology

GDMS

Geodetic Data Modelling System

GPS

Global Positioning System

FFT

Fast Fourier Transform

FFTW

Fastest Fourier Transform in the West, a C library to perform Fast Fourier Transforms.

IDFT

Inverse Discrete Fourier Transform

IFFT

Inverse Fast Fourier Transform

ISIP

Institute for Signal and Information Processing, Mississippi State University

JUnit

The Java unit testing framework.

MIT

Massachusetts Institute of Technology

NTB

Normalised Transition Bandwidth, used for tuning the Dolph-Chebyshev window

OO

Object Oriented.

PSD

Power Spectral Density

SLR

Satellite Laser Ranging

SQL

Structured Query Language

TSVIEW

Time Series View

VCV

Variance Co-variance

VLBI

Very Long Baseline Interferometry

Chapter 7. References

References

Aho, A. V, Sethi, R., Ullman, J.D. 1986, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley Publishing, Massachusetts.

Arndt, J., 11 November 2002 [last update], *cplxfft.h, Complex Fast Fourier Transform Template Class (Part of FXT)* [Online] Available: http://www.jjj.de/fxt/, July-2002.

Bellanger, M., 1984, *Digital Processing of Signals*, John Wiley and Sons, New York.

Carezia, A., 18 June 2002 [last update], *Dolph-Chebyshev Window*, [Online], Octave Sources, Available: http://www.octave.org/octave-lists/archive/octave-sources.2002/msg00019.html, Jul-2002.

Chapman, M., 2002, *Window managers for X* [Online] Available: http://www.plig.org/xwinman/cde.html, Nov-2002

Connolly, T., Begg, C. 1998, *Database Systems: A practical approach to design, implementation, and management*, Addison-Wesley Publishing, Massachusetts.

Cppunit Project, 11 April 2002 [last update], *CPP Unit, Unit Testing Library*, [Online], Sourceforge.net, Available: http://cppunit.sourceforge.net, May-2002.

Deitel, H.M., Deitel, P.J., 1998, *C++: How to Program*, Prentice-Hall, New Jersey.

Flannery, B., Press, W., Teulosky, S., Vettering, W., 1988, *Numerical Recipes in C*, Cambridge University Press, Cambridge.

Fowler, M., 1999, *UML Distilled: A Brief Guide to the Standard Object Modelling Language*, Addison-Wesley, New York.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, New York.

Gerald, C., Wheatley, P., 1999, *Applied Numerical Analysis*, Addison Wesley Longman, USA

Geo-science Australia, unknown [last update], *National Mapping - Geodesy, Very Long Baseline Interferometry*, [Online], Geo-science Australia, Available: http://www.auslig.gov.au/geodesy/sgc/vlbi/vlbitech.htm, 15 November 2002.

Geo-science Australia, unknown [last update], *Satellite Laser Ranging*, [Online], Geo-science Australia, Available: http://www.auslig.gov.au/geodesy/slr/whatis.htm, 17 November 2002.

Gnome 2002 *Gnome: Computing made easy* [Online] Available: http://www.gnome.org, Nov-2002

GTK 2002 *GTK+ FAQ* [Online] Available: http://www.gtk.org/faq/, Oct-2002

Holzner S., 2001, *C++ Black Book*, The Coriolis Group, Scottsdale.

Institute for Information and Signal Processing, 30 March 2002 [last update], *ISIP Software Documentation*, [Online], Mississippi State University, Available: http://www.isip.msstate.edu/projects/speech/software/documentation/class /algo/Window/, Jun 2002.

Intel.com, 2002 [copyright], *Intel® Pentium 4® Optimisation Reference Manual* [Online], Intel Corporation, Available: http://developer.intel.com/design/pentium4/manuals/248966.htm, 18 September 2002.

Interstellar Research, 1 August 2001 [last update], *Daqarta: FFT Windowing*, [Online], Daqarta.com, Available: http://www.daqarta.com/ww00wndo.htm, Nov-2002.

Jeffereis, R.E., 14 January 2002 [last update], *XProgramming.com, an extreme programming resource*, [Online] Jeffereis, R.E., Available: http://www.xprogramming.com, May-2002.

Junit Project, 1 August 2002 [last update], *JUnit, Testing Resources for Extreme Programming*, [Online], objectmentor.com, Available: http://www.junit.org/, May-2002.

KDE 2002, *KDE Home Page* [Online] Available: http://www.kde.org, Nov-2002

Korner, T., March 2002[last update], *Divided Differences*, [Online],mathematics nrichment,http://www.nrich.maths.org.uk/mar02/art1/index_printable.html

Lay, D.C., 1997, *Linear Algebra and its Applications (2nd edition)*, Addison Wesley Longman Inc., Massachusetts

Lepple, C., 23 July 2000 [last Update], *Fast Fourier Transforms Demystified* [Online] Available: http://www.foo.tho.org/charles/fft.html, November-2002.

Levine, J.R., Mason, T., Brown, D., 1990, *lex & yacc*, O'Reilly, Farnham.

Lynch, P., 27 June 1996, *The Dolph-Chebyshev Window: A Simple Optimal Filter*, [Online] Available: http://www.maths.tcd.ie/~plynch/Publications/Dolph.pdf, Jul-2002.

Mathews, J.H., Fink, K.D., 1999, *Numerical Methods Using MATLAB (3rd edition)*, Prentice Hall, New Jersey.

Microsoft 2002 *Windows XP screenshot* [Online] Available: http://www.microsoft.com/presspass/newsroom/winxp/images/img016.jpg, Nov-2002

Rabiner L.R, Schafer R.W., 1978, *Digital Processing of Speech Signals*, Prentice Hall, New Jersey.

Royster, D., 15 February 2000 [last updated], *Box Plots and Box & Whiskers Plots* [Online], University of North Carolina at Charlotte, Avalible: http://www.math.uncc.edu/~droyster/courses/spring00/maed3103/Box_Plots.htm, July 2002.

Scott, B., Taylor, R., 29 July 1999 [last update], *What is Radio Astronomy*, [Online], University of Calgary, Available: http://www.ras.ucalgary.ca/Spacevlbi1.html, 17 November 2002.

Stroustrup, B., 2000, *The C++ Programming Language, Special Ed*, Addison-Wesley, New York.

Sun Microsystems 2002, *GNOME 2.0 Desktop for the Solaris™ Operating Environment* [Online] Available: http://wwws.sun.com/software/star/gnome/, Nov-2002

Trolltech 2002, *Trolltech Home Page* [Online] Available: http://www.trolltech.com, Nov-2002

Unit++ Project, 4 May 2002 [last update], *The Unit++ Testing Framework*, [Online], Sourceforge.net, Available: http://unitpp.sourceforge.net, May 2002.

Unknown, 5 November 2002 [last update], *Goddard Geodetic VLBI Group*, [Online], Space Geodesy Program, NASA, Available: http://lupus.gsfc.nasa.gov/vlbi.html, 17 November 2002.

Wittke, J., 29 October 2001 [last updated], *Electron Microprobe Notes - Statistics* [Online], Northern Arizona University, Avalible: http://jan.ucc.nau.edu/~wittke/Microprobe/Statistics.html, 12 November 2002.