

SEVER INSTITUTE OF TECHNOLOGY
Master of Science Degree

THESIS ACCEPTANCE
(To be the first page of each copy of the thesis)

DATE: 1994 May 4

STUDENT'S NAME: Reece Kimball Hart

This student's thesis, entitled ***AN OBJECT-ORIENTED SYSTEM FOR THE ANALYSIS OF AUTOMATED DNA SEQUENCING DATA*** has been examined by the undersigned committee of three faculty members and has received full approval for acceptance in partial fulfillment of the requirements for the degree Master of Science.

APPROVAL: _____ Chairman

Distribution:

- 4 - One for each copy of the thesis
- 1 - Candidate
- 1 - Department
- 1 - Dean's Office
- 7 - Total

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY

ABSTRACT

***AN OBJECT-ORIENTED SYSTEM FOR THE ANALYSIS OF
AUTOMATED DNA SEQUENCING DATA***

by Reece Kimball Hart

ADVISOR: Professor David J. States

1994 August

Saint Louis, Missouri

A C++ Class Library for the manipulation and analysis of automated sequencing data was designed and implemented. The Library was used to implement an analytical model which uses Bayes' Theorem to assign probabilities to peaks in the data. The performance of the library and statistical model was evaluated for a set of eleven data files by comparing the results to those of the proprietary methods provided by the manufacturer. The average agreement is 92.2% over 367 bases.

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY

***AN OBJECT-ORIENTED SYSTEM FOR THE ANALYSIS OF
AUTOMATED DNA SEQUENCING DATA***

by

Reece Kimball Hart

Prepared under the direction of Professor David J. States

A thesis presented to the Sever Institute of Washington University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

1994 August

Saint Louis, Missouri

CONTENTS

Figures	iv
Tables	v
Chapters	
1. Introduction	1
2. DNA Structure and Sequencing	5
3. Peak Model and Heuristic Analysis	14
4. C++ Class Library & Implementation of the Peak Model. . . .	28
CTrace	29
CTraceFile	34
CPeakList.	36
5. Performance of the Peak Model	42
6. Application to DNA Sequencing	47
7. Improvements and Extensions	51
8. Conclusion	56

Appendices

A. Relevant Statistical Methods	58
B. Validity of Peak Model Assumptions	61
C. autoseq Users' Guide.	67
D. Availability.	72
E. Class Library Declarations	73
Bibliography	89
Vita	90
Acknowledgments.	91

FIGURES

<u>Figure</u>	<u>Page</u>
2-1 Overview of dideoxynucleotide DNA sequencing.	6
2-2 Detection of sequencing fragments by fluorescence	8
2-3 Details of chain elongation in DNA sequencing	9
2-4 Sample chromatograms from fluorescent sequencing.	13
3-1 Diagram of peak shape and residual	24
4-1 Calculation of the derivative in a trace	30
4-2 Determination of peak location by first derivative test.	31
4-3 Pseudocode for determining peak location.	32
4-4 Sample peaks chosen from real data	33
4-5 Determination of statistics for a set of peaks	39
4-6 Gaussian model of peak fluorescence	40
5-1 Distribution of mismatches as a function of base position	44
5-2 Excessive peak pruning occurs early in chromatograms	45
6-1 Diagram of base-position-delta (bpd) terminology	48
6-2 Histograms of base-pair deltas for terminal (3') mononucleotides.	50
A-1 Standard Gaussian probability density function.	58
B-1 Gaussian model of peak fluorescence	62
B-2 Gaussian model of peak amplitudes.	63
B-3 Gaussian distribution of noise	64
B-4 Exponential decay of mean peak amplitude	66

TABLES

<u>Table</u>	<u>Page</u>
3-1 Peak model notation	15
5-1 Summary of results	43
6-1 Summary of mono- and dinucleotide populations	49
C-1 <i>autoseq</i> command line flags and arguments	68

AN OBJECT-ORIENTED SYSTEM FOR THE ANALYSIS OF AUTOMATED DNA SEQUENCING DATA

1. INTRODUCTION

In an effort to understand the genetic origins of our species, scientists have undertaken a monumental task: to determine the entire chemical sequence of our inherited blueprints. These blueprints are made of a sequence of building blocks and assembled into a long polymeric structure of deoxyribonucleic acid (DNA). Miraculously, there are only four types of building blocks and the entirety of an organisms physical structure can be reduced to sequences of this material.

Conventional methods are sufficient — albeit laborious — when the sequence of relatively short, specific regions of DNA is sought. An experienced technician can read 300-400 bases (b) of DNA in a single experiment, and will read in overlapping frames of this size in order to achieve larger sequences of

many kilobases (kb; 1kb = 1000b). However, as biologists attempt to determine the sequence of larger segments of DNA, automating the process of data acquisition and analysis becomes increasingly important.

Many disciplines of engineering have cooperated to meet the challenge of reading large sequences of DNA by automating the process. The result is a new automated sequencing technology which uses colored dyes, lasers, photodetectors, and computers to monitor sequencing experiment results (9). The current state of the art automated sequencer generally outperforms manual methods in both total sequence throughput and reliability.

Despite this impressive technology, improvements are still required. The human genome consists of approximately 3×10^9 basepairs (bp), and sequencing DNA of this size is intractable by current automated methods. Improving the reliability and efficiency of automated sequence determination is the underlying motivation for this thesis.

Automated sequencing of DNA using fluorescent detection of DNA fragments results in a set of four chromatographic traces, one for each of the four bases of DNA. It is the job of a computer to analyze the data, determine the sequence of the DNA, and identify regions of ambiguity. Although proprietary and confidential methods for the analysis of these data exist, it is generally believed that improvements to both the quantity and quality of the

sequence inferred from the data are possible. Unfortunately, the complexity of the file format and data access is a barrier to more extensive experimentation with analytical methods.

There were three primary objectives for this Master's Thesis. First, we sought to develop a library of C++ classes (*CTrace* and *CTraceFile*) which would be useful in manipulating chromatographic data from DNA sequencing experiments. It was intended that these classes would provide a simple interface to the chromatographic traces and associated data which result from automated DNA sequencing. These classes will facilitate experimentation with methods for the interpretation of the data by allowing researchers to focus efforts on analytical methods without concern for issues such as file format, data orthogonalization, smoothing, translation, and so forth. In addition, these classes may enable researchers to explore a new range of applications whose analytical methods are currently unsupported by the software provide by the manufacturers of automated DNA sequencers.

Second, we used these classes to implement a statistical model for the automated inference of DNA sequence from peaks picked on chromatographic traces (*CPeakList*). An advantage of this model over others is that it offers a probabilistic assessment of sequence in a chromatogram.

Finally, the C++ representations of traces and tracefiles were used in conjunction with the proposed statistical model to study the mobilities of individual nucleotides and dinucleotide combinations at the 3' and 3'

penultimate positions. Knowledge of the individual mobility contributions may provide additional methods for discrimination of peaks in chromatograms, thus improving both the quality and quantity of the inferred sequence.

2. *DNA STRUCTURE AND SEQUENCING*

This chapter summarily describes the structure of deoxyribonucleic acid and a method commonly used to determine its sequence. It is intended to provide background information sufficient to comprehend the remainder of this thesis. More detailed information may be found in most general biology or biochemistry textbooks (1), or by consulting the references in the bibliography.

Many current approaches to automated DNA sequencing use the dideoxynucleotide termination method developed by Sanger for use with radiolabelled manual sequencing (8). This chapter briefly describes the Sanger method and its application to fluorescent-labelled automated DNA sequencing.

An overview of DNA sequencing is shown in Figure 2-1. Sequencing is dependent upon the ability to randomly terminate fragments of DNA at bases of a specified type, resulting in a distribution of fragments of varying size, all of which terminate at the given base. When this process is repeated for all base types (A,C,G,T) and sorted by size, the sequence can be inferred by reading from the smallest to the largest fragment size.

In the most widely used form of automated DNA sequencing, the sorted fragments are observed by uniquely coloring strands which terminate in a particular base with a fluorescent dye or label (9). In effect, this creates a correspondence between the color of a strand of DNA and the base in which it

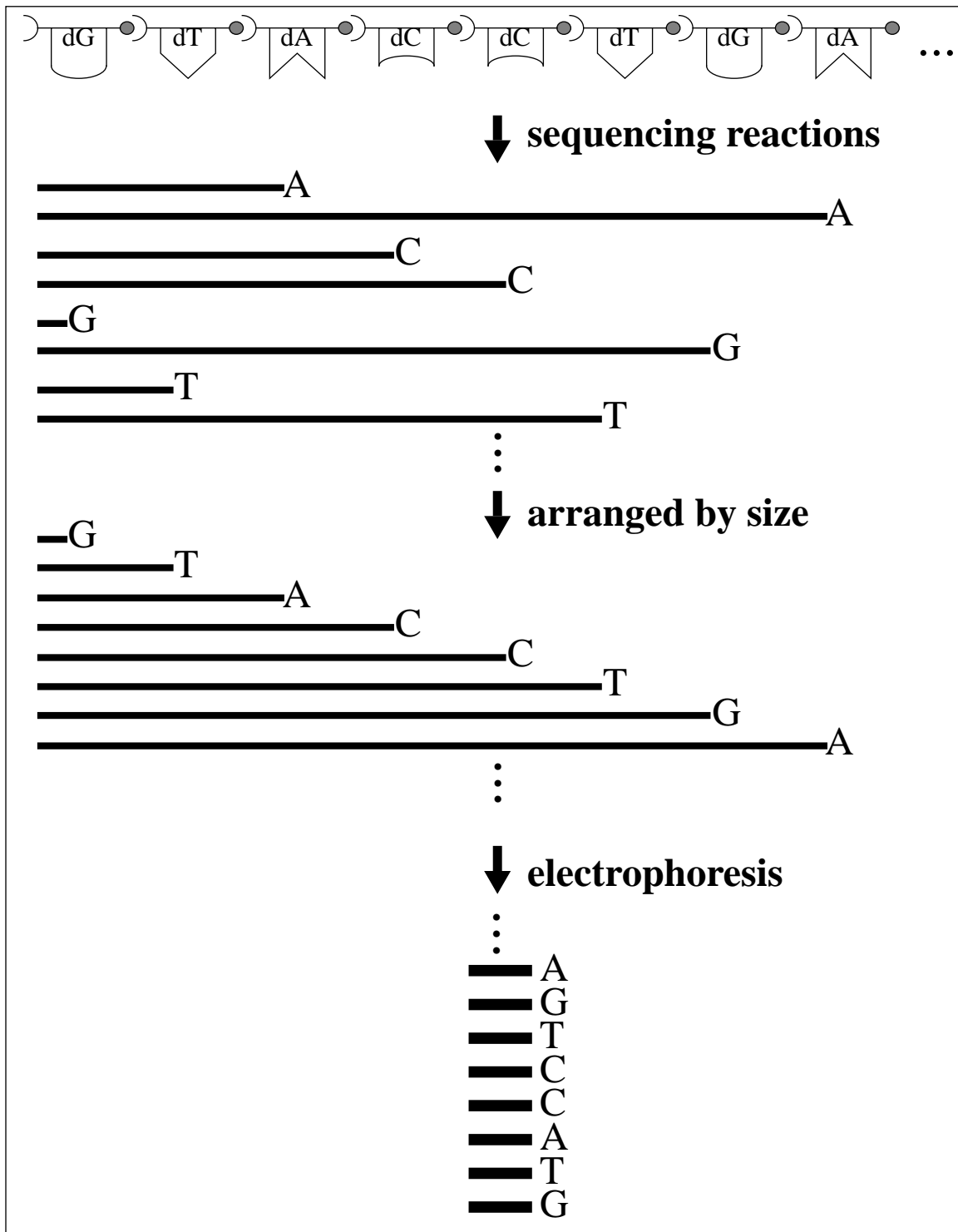
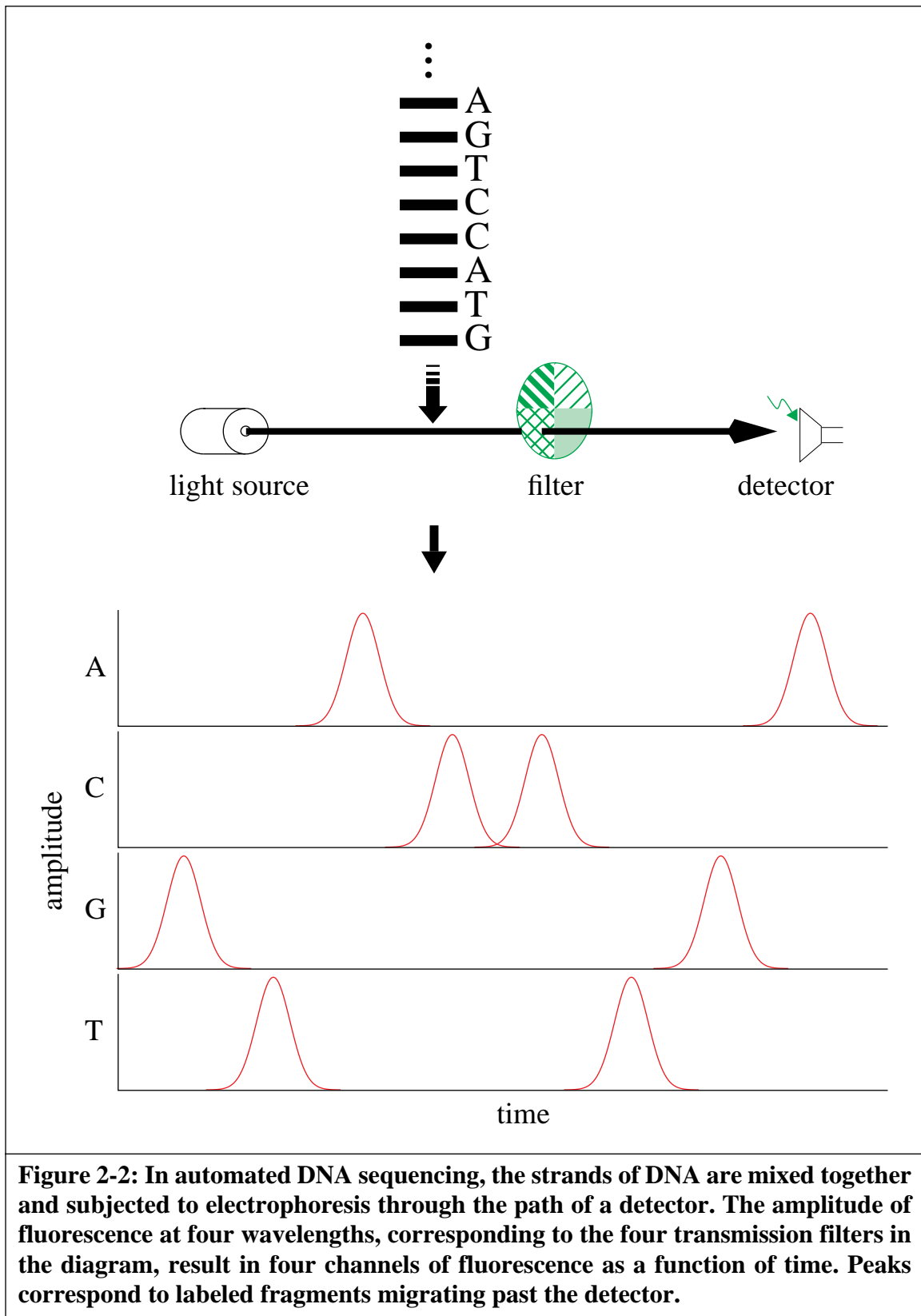


Figure 2-1: An overview of Sanger method DNA sequencing. DNA sequencing essentially involves the mapping of the positions of each base type (A, C, G, T). Electrophoresis sorts these fragments by size and is typically shown as migrating from top to bottom (that is, with the smallest/fastest fragments at the bottom).

terminates. The migration of the DNA is monitored by passing the sorted fragments through a detector which consists of an excitation light and filters which correspond to the colors of the four dyes as shown in Figure 2-2. The result is a set of digitized fluorescence intensities as a function of time. In practice, the peaks are separated by approximately 8-15 sample points.

The Sanger method makes use of the intrinsic ability of cells to replicate their own DNA. In the cells of most organisms, DNA exists as two intertwined and complementary strands in a structure referred to as the double helix (see Figure 2-3a). During cell division, DNA replication is effected by the piecewise unraveling of the double helix and simultaneous duplication of the separated “template” strands. Each template strand is copied by the consecutive addition of bases of DNA, each one complementary to the base in the corresponding position on the template strand. In this way, the two pairs of resulting DNA strands are exactly identical to the parental strands (disregarding duplication errors).

Each base or nucleotide of DNA has two chemically reactive junctions — referred to as the 3' and 5' (“three-prime” and “five-prime”) ends — which are capable of bonding to adjacent bases. 3' junctions may bond only with 5' ends and vice versa. Thus, the DNA strands are directional (see Figure 2-3b & c). The 3' ends of deoxynucleotides of DNA (dA, dC, dG, dT) terminate in -OH (a hydroxyl group); *dideoxynucleotide* bases (ddA, ddC, ddG, ddT) terminate in a -H and are unable to bond chemically with another base at the 3' end, although



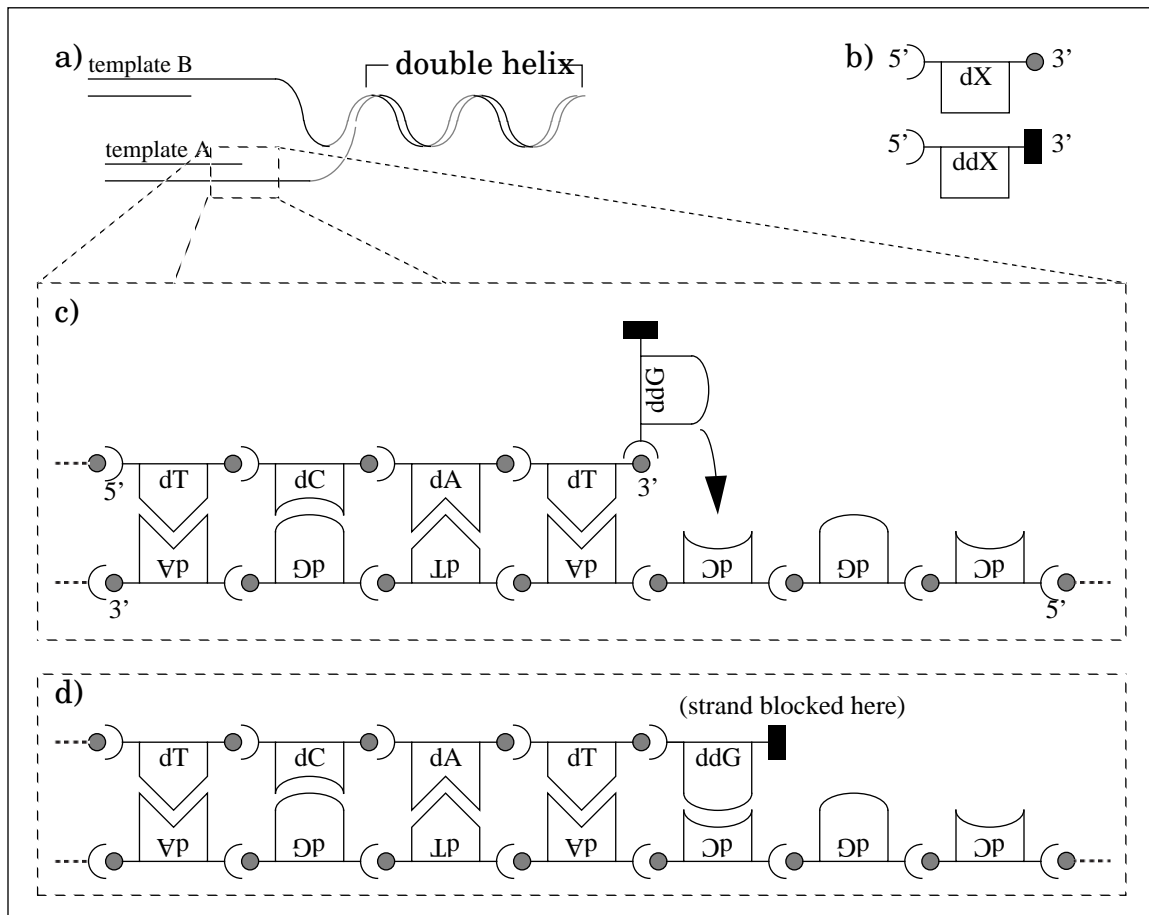


Figure 2-3: Schematic diagrams of DNA showing the double helix and complementarity of bases (A-T and C-G pairing). (a) The parental strands in the double helix unwind and are replicated individually. Each resulting double-stranded pair is identical to the parental pair. (b) Schematic diagrams of a deoxynucleotide (dX) and a dideoxynucleotide (ddX) are shown. 3'-OH groups are indicated with ●, and 3'-H groups by ■. (c) The addition of the dideoxynucleotides dT, dC, dA, and dT continue chain elongation. ddG is shown as the next base to be added. (d) The addition of ddG permanently blocks chain elongation.

bonding at the 5' end is unaffected. DNA replication and sequencing is a directional process: nucleotides are added only at the 3' end of the duplicate strand, and for this reason is said to occur in the 5'-to-3' direction.

In a typical sequencing experiment, a large number of identical template strands are incubated with deoxynucleotides (which are able to bond at both the 5' and 3' ends) as well as a small fraction of *dideoxynucleotides* (which cannot bond at the 3' end and thereby cause termination). Because nucleotides are incorporated randomly, a distribution of incompletely replicated chains of varying length is obtained, each with a *dideoxynucleotide* at its 3' end.

So far, we have discussed the elongation of a strand of DNA which is already paired to its complementary strand (as shown in Figure 2-3). But, how does the process begin? Cells provide short sequences of DNA called primers to begin the process of DNA replication. Because sequences of DNA have moderate affinity and high specificity for a region of complementary bases, primers are able to find their appropriate binding site and form a region for replication initiation.

Two methods are commonly employed to incorporate fluorescent label into the elongating chain: dye primer and dye terminator. With the dye primer method, four primers are synthesized, each of which contains one of four fluorescent labels. In the sequencing reaction mixtures, the A primer is incubated with 4 deoxynucleotides (dA, dC, dG, dT), and the *dideoxynucleotide* ddA; thus, only chains which terminate with ddA will be observed because they are the only strands which have a primer with the A color. Similar reactions are used for the remaining primers.

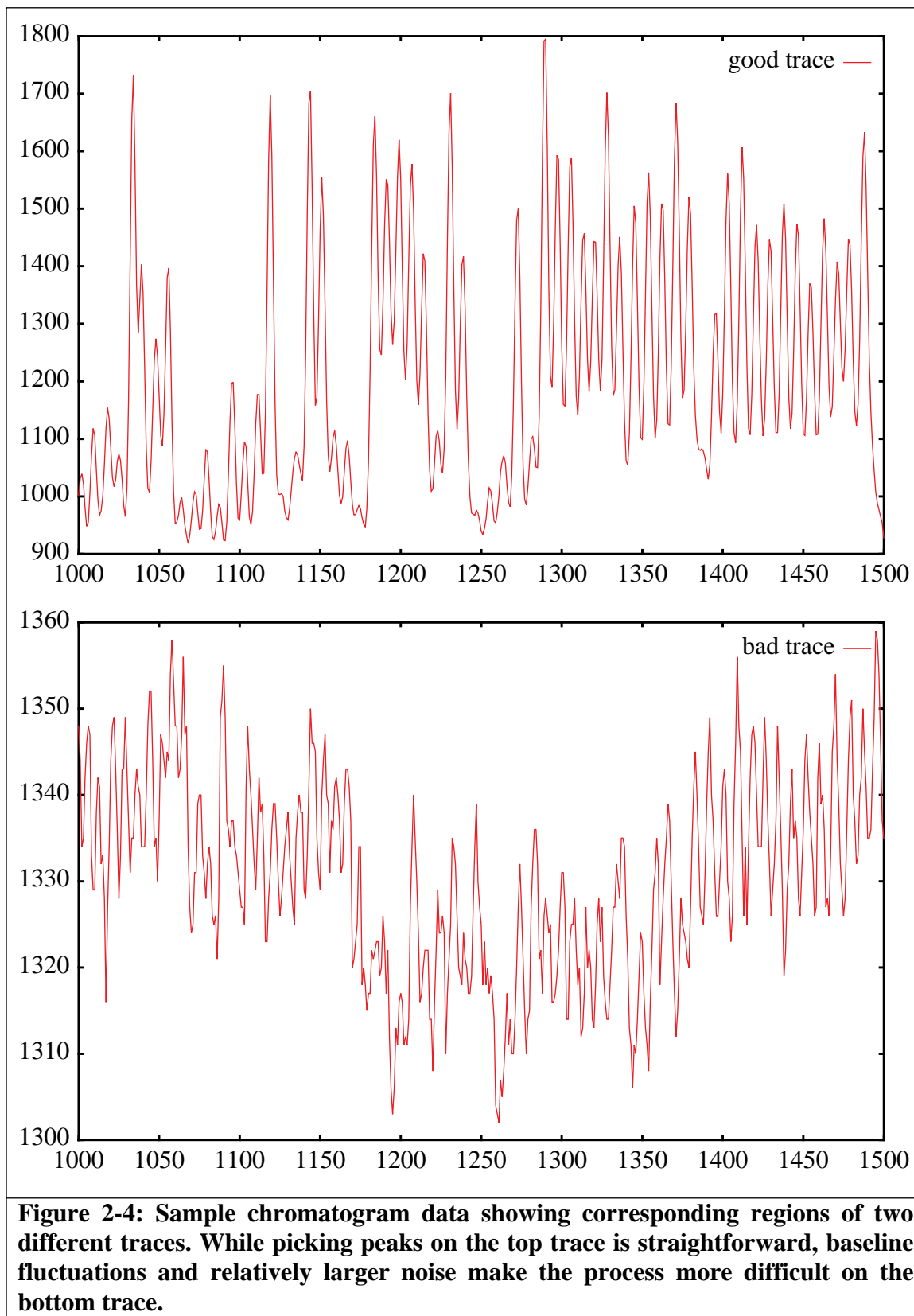
The dye terminator method uses a single primer and fluorescent *dideoxynucleotide* terminators. There is only one reaction mixture which contains template, primer, four deoxynucleotides (dX), and four fluorescence-labeled *dideoxynucleotide* terminators (ddX). An important advantage to this method is that there is only one reaction mixture. A disadvantage of this method is that the fluorescent labels tend to be large relative to the nucleotide and cause significant mobility perturbation.

A frequently-used technique in molecular biology is size separation of DNA by gel electrophoresis. Samples of DNA are loaded onto a bed of gelatin (a porous medium of agarose or polymerized resin lattices) and exposed to an electric field. The electric field causes the DNA to migrate through a porous medium at a rate that is a function of its length. In this way, DNA fragments may be sorted by size.

In practice, an automated sequencer is loaded with the sequencing reaction solution. The samples are subjected to an electric field which causes them to migrate at a length-dependent rate. At a fixed point along the electrophoresis path, the sequencer excites the chromophore with a low-power laser and monitors fluorescence with a photomultiplier and appropriate filters for each base's characteristic frequency. The fluorescence at each wavelength is recorded as a function of time. Because there is overlap between the emission spectra of the different dyes, the observed data are a convolved sum of the component chromatograms. To obtain the data for each base as a distinct

chromatogram, it is necessary to deconvolve this data (see **4. C++ CLASS LIBRARY & IMPLEMENTATION OF THE PEAK MODEL** on page 28). With appropriate calibration, time may be transformed into the length of the sequence (5).

The analysis of the data which results — essentially the fluorescence at four wavelengths as a function of time — is the subject of this thesis. Figure 2-4 shows samples of raw data (called chromatograms or traces) which result from automated DNA sequencers. The aim of a DNA sequencing experiment is to infer the well-defined and unique sequence of nucleotides in a strand of DNA from local maxima (‘peaks’) observed in the sequencing chromatograms. However, noise and mechanistic anomalies induce ambiguity in the inferred sequence; that is, it is impossible to determine the sequence from the data with absolute certainty. Methods which provide a quantitative estimation of the most likely sequence are desired. **3. PEAK MODEL AND HEURISTIC ANALYSIS** presents a model of the data and heuristics which provide a probabilistic assessment of competing sequence hypotheses.



3. PEAK MODEL AND HEURISTIC ANALYSIS

Introduction

The previous chapter suggested that the peaks in a chromatogram represent the individual bases of DNA in the sequence being analyzed. A primary objective of this thesis was to develop a mathematically valid model of fluorescence-labeled DNA sequencing which would provide quantitative estimation of the probability that a specific sequence which represents the physical template being analyzed would result in the observed data. In this chapter it will be shown that a rigorously correct approach to this calculation is computationally intractable; this fact necessitates the invocation of several simplifying assumptions which enable the development of a reasonable model for the data and heuristics which approximate the results of a rigorously correct analysis.

Throughout this chapter, the segment of DNA being sequenced is referred to as the “template”. The template has a well defined and unique sequence which consists of a series of “bases” or nucleotides of DNA (i.e., {A,C,G,T}). Performing automated DNA sequencing on a template results in four channels of fluorescence which are referred to individually as “observed data” or simply “data” and collectively as a “set of observed data”. (See Figure 2-1 and

Figure 2-2 for diagrams of these terms.) The term “peak” refers to a region in observed data around a local maximum. The event that a base of a particular type exists at a particular location in the data is the “base hypothesis”, and a collection of many such hypotheses is termed a “sequence hypothesis” and implies a specific prediction of DNA sequence content for the entire length of the DNA template being analyzed.

The development of the probabilities which follows assumes that a set of peaks in the chromatogram, picked by some reasonable method (i.e., concave down), already exists. Each peak in this set of peaks will be used as an initial set of base hypotheses for the nucleotides within the sequence. Except where stated otherwise, the data being analyzed originates from any *one* of the four channels from an automated sequencer. The notational conventions used to derive base hypothesis probabilities are given in Table 3-1.

Table 3-1: Peak Model Symbols and Notation

symbol	meaning
i	index (time) of sample point in a chromatogram
p	arbitrary peak; a record of a peak consisting of the 4-tuple $\langle h_p, i_p, \mu_p, w_p \rangle$
i_p	index of the center of peak p
h_p	amplitude of peak p
\hat{h}_{obs}	mean of peak amplitude over entire trace
σ_h	standard deviation of peak amplitude about expected mean amplitude
$\hat{h}_{ed}(i_p)$	peak amplitude at index i expected by exponential decay of observed data
μ_p	mobility of peak p
w_p	width of peak p
$\hat{w}(i)$	expected width of a peak at i , corrected for linear increase in peak width

Table 3-1: Peak Model Symbols and Notation

symbol	meaning
$F(i, h_p)$	expected fluorescence at index i as a result of peak with amplitude h_p
$F(i, p)$	expected fluorescence at index i as a result of peak p (centered at i_p with amplitude h_p , width w_p , and mobility μ_p)
$F'_{exp}(i)$	expected fluorescence intensity at time i , assuming noise and baseline are zero
$F'_{obs}(i)$	observed fluorescence intensity at time i
$F'_{baseline}(i)$	fluorescence baseline incorporating both background fluorescence and detector baseline effects
$F'_{obs}(i)$	observed fluorescence intensity at time i , baseline corrected
$F'_{noise}(i)$	noise in fluorescence
σ_n	standard deviation of noise
$F'_{residual}(i)$	residual fluorescence
H	hypothesis that a base exists
H_s	hypothesis that a sequence of bases describes the physical template of DNA i.e. $H_s = \langle b_1, b_2, b_3, \dots, b_L \rangle$
\emptyset	the null hypothesis is that a particular peak in question does not represent a base in the template sequence
D	the data (a single channel of fluorescence from the chromatogram), representing the baseline-corrected values $F'_{obs}(i)$.

The starting point for this analysis is a probabilistic assessment of possible sequence hypotheses given a set of observed data derived from a template of DNA. The goal of the probabilistic approach is to estimate the probability of a particular sequence hypothesis given the observed data and is denoted $P(H_s/D)$, thus allowing sequence hypotheses to be ranked and the most likely sequence to be determined.

A Rigorous Approach is Computationally Intractable

A rigorous investigation of all possible sequence hypotheses is computationally intractable for several reasons. For a sequence of length L , there are $|\{A, C, G, T\}|^L = 4^L$ possible sequence hypothesis sequences (H_s) which must be considered. The combinatorial explosion of this computation makes explicit calculation of each sequence hypothesis very unattractive. To make matters worse, this measure of complexity underestimates the actual complexity because the length of sequence is generally unknown at the time of sequencing. Thus the search space spans a number of sequence lengths and all possible sequences of a given length.

In addition to the combinatorial issue, there is a more complicated problem. Each observed fluorescence value consists of a signal and noise components. (It is assumed that the data have already been corrected for baseline fluorescence.) Because the noise contribution is unknown, the probability of a given observed data value is the integral over all possible partitions of signal and noise at that point in the chromatogram, where the signal term is itself a complicated summation of the fluorescence contributions of each base in the sequence hypothesis. Even if knowledge of the mechanisms of DNA sequencing provided methods which allowed determination of individual base contributions from first principles, this integral would be prohibitively time consuming.

Simplifying Assumptions Enable Computationally Tractable Heuristics

The impossibility of evaluating $P(H_s/D)$ over all possible sequence hypotheses in a rigorously correct manner dictates the necessity for heuristics which reliably predict DNA sequence from a set of observations, and the feasibility of large scale DNA sequencing relies on the development of such algorithms.

A model for the observed data based on a specific sequence hypothesis was developed incorporating five assumptions about the data:

Assumption 1: The probability of the existence of any base is independent of all other bases of the same or a different type.

Assumption 2: Peak amplitudes conform to a Gaussian distribution.

Assumption 3: Peak amplitudes are subject to an exponential decay with increasing index in the chromatogram.

Assumption 4: The amplitude of noise in the signal is Gaussian.

Assumption 5: Peaks have Gaussian shape.

The purpose of these assumptions is to enable the development of computationally tractable heuristics which approximate the computationally intractable methods described earlier. The appropriateness of these assumptions is examined in ***APPENDIX B: VALIDITY OF PEAK MODEL ASSUMPTIONS***, but it should be noted now that the form of the model was based on an analysis of the biochemical and physical processes used in the sequencing experiment and that the parameterization of the model is

empirical. They are used to provide prior knowledge which will be used in conjunction with Bayes' Theorem, shown in Equation 3-1, to estimate the probability of a particular base hypothesis, H , given the observed data, D .

$$P(H|D) = \frac{P(D|H)}{P(D)} P(H) \quad (3-1)$$

The only requirement for any model which uses Bayes' Theorem is that the terms on the right hand side of Equation 3-1 must be probabilities.

Assumption 1 allows each maximum in the data to be considered independently. This means that the conditional probability of a sequence hypothesis is the product of the *independent* conditional probabilities of the each base in the sequence hypothesis. That is,

$$P(H_s|D) = P(H_1|D) P(H_2|D) P(H_3|D) \dots P(H_n|D) \quad (3-2)$$

where H_i is some base hypothesis corresponding to p_i in the data. Thus, the problem of computing the most probable sequence is reduced to that of finding the most probable base assignment at every peak in the chromatogram.

A peak in the data either results from a base of DNA in the template or it does not. That is, $P(H) + P(\emptyset) = 1$ and $P(D) = P(D|H) + P(D|\emptyset)$. Bayes' Theorem (Equation 3-1) for individual peaks becomes

$$P(H|D) = \frac{P(D|H)}{P(D|\emptyset) + P(D|H)} P(H) . \quad (3-3)$$

In order for Bayes' Theorem to be useful, a probabilistic basis for the terms on the right hand side of Equation 3-1 must be developed.

$P(H)$: The prior probability of a particular base hypothesis is independent of the data. Without additional information, this term must be assumed to be equal for each of the four bases of DNA. If the base composition of a sequence is known, this data may be used to improve the estimate of $P(H)$. Throughout this chapter, the $P(H)$ term will refer to the estimate we have chosen according to either of these methods. It should be noted that because chromatograms are treated independently, context-sensitive probabilities are not supported by this model.

For every peak in the chromatogram, two possible hypotheses are considered. One hypothesis is the absence of a base. $P(D|\emptyset)$ is the probability that this observation results exclusively from noise in the system. If one assumes that the distribution of noise is Gaussian (Assumption 4), then

$$P(D|\emptyset) = \frac{1}{\sigma_n \sqrt{2\pi}} e^{-\frac{(F'_{obs}(i))^2}{2\sigma_n^2}}. \quad (3-4)$$

The other possible hypothesis is that there is a base in this channel at this point, and the probability of this event is denoted by $P(D/H)$. Because the contribution of noise is unknown, computing $P(D/H)$ requires an integral over all possible partitions of signal and noise. More specifically, this term is equal to the integral over all possible fluorescence values of the joint probability of a hypothetical peak of height h_p and a noise contribution equal to the residual $(F'_{obs}(i_p) - h_p)$.

A histogram of a set of peaks selected by a combination of amplitude and local maximum criteria revealed that the peak amplitudes approximated a Gaussian distribution (see Figure B-2), and this observation forms the basis for assessing the probability of a peak with a particular height. Assuming that this approximation is valid (Assumption 2), the probability density function shown in Equation 3-5 gives the probability density of a peak with height h_p assuming a *constant* mean peak height of \hat{h}_{obs} .

$$f(h_p) = \frac{1}{\sigma_h \sqrt{2\pi}} e^{-\left(\frac{(h_p - \hat{h}_{obs})^2}{2\sigma_h^2}\right)} \quad (3-5)$$

An analysis of the amplitudes of the peaks selected by this method revealed a systematic decrease in mean peak amplitude with increasing index which was modeled with an exponential decay as shown in Figure B-4 (Assumption 3). Upon correction for peak amplitude decay, Equation 3-5 becomes

$$f(h_p) = \frac{1}{\sigma_h \sqrt{2\pi}} e^{-\left(\frac{(h_p - \hat{h}_{ed}(i_p))^2}{2\sigma_h^2}\right)} \quad (3-6)$$

where $\hat{h}_{ed}(i_p)$ is the mean peak amplitude as a function of trace position corrected for exponential decay. The probability that a peak has a height h_p in the range $[h, h+dh]$ is given by:

$$P(h \leq h_p \leq h + dh) = \frac{1}{\sigma_h \sqrt{2\pi}} e^{-\left(\frac{(h - \hat{h}_{ed}(i_p))^2}{2\sigma_h^2}\right)} dh \quad (3-7)$$

Equation 3-7 is valid only for a continuous distribution of peak heights. However, peak heights obtained in practice have discrete integer values. The prior probability of a peak with height h_p in the range $[h_p-0.5, h_p+0.5]$ may be approximated by assuming that the integrand is constant over this narrow region and evaluated as shown in Equation 3-8.

$$\begin{aligned} P(h = h_p) &= \frac{1}{\sigma_h \sqrt{2\pi}} \int_{h_p-0.5}^{h_p+0.5} e^{-\left(\frac{(h - \hat{h}_{ed}(i_p))^2}{2\sigma_h^2}\right)} dh \\ &= \frac{1}{\sigma_h \sqrt{2\pi}} e^{-\left(\frac{(h_p - \hat{h}_{ed}(i_p))^2}{2\sigma_h^2}\right)} \end{aligned} \quad (3-8)$$

With the assumption that the distribution of noise is Gaussian (Assumption 4), the probability of the data in the presence of the base is the probability of the residual under this model for noise.

A Gaussian distribution was employed to model the expected fluorescence, $F(i,p)$, at index i attributable to peak p (Assumption 5). The peak amplitude and width of a band are model parameters determined from the data. $F(i,p)$ is calculated as shown in Equation 3-9.

$$F(i, p) = h_p e^{-\left(\frac{(i-i_p)^2}{2w_p^2}\right)} \quad (3-9)$$

The fluorescence expected by the model can be calculated using Equation 3-9 summed over each peak in the set as shown in Equation 3-10.

$$F_{exp}(i) = \sum_p F(i, p) \quad (3-10)$$

The total observed fluorescence may be decomposed into components representing noise, baseline, and signal as shown in Equation 3-11.

$$F_{obs}(i) = F_{obs}(i) + F_{baseline} + F_{noise}(i) \quad (3-11)$$

Assuming that the model accurately predicts the signal fluorescence at index i , $F_{obs}(i)$ is equal to the expected fluorescence at index i , $F_{exp}(i)$. Making this substitution into Equation 3-11, invoking the definition of $F_{exp}(i)$ given by Equation 3-10, and solving for $F_{noise}(i)$ yields Equation 3-12. Note that any errors in the model appear in $F_{noise}(i)$. See Figure 3-1.

$$\begin{aligned} F_{noise}(i) &= F_{obs}(i) - F_{baseline} - F_{exp}(i) \\ &= F'_{obs}(i) - F'_{exp}(i) \\ &= F_{residual}(i) \end{aligned} \quad (3-12)$$

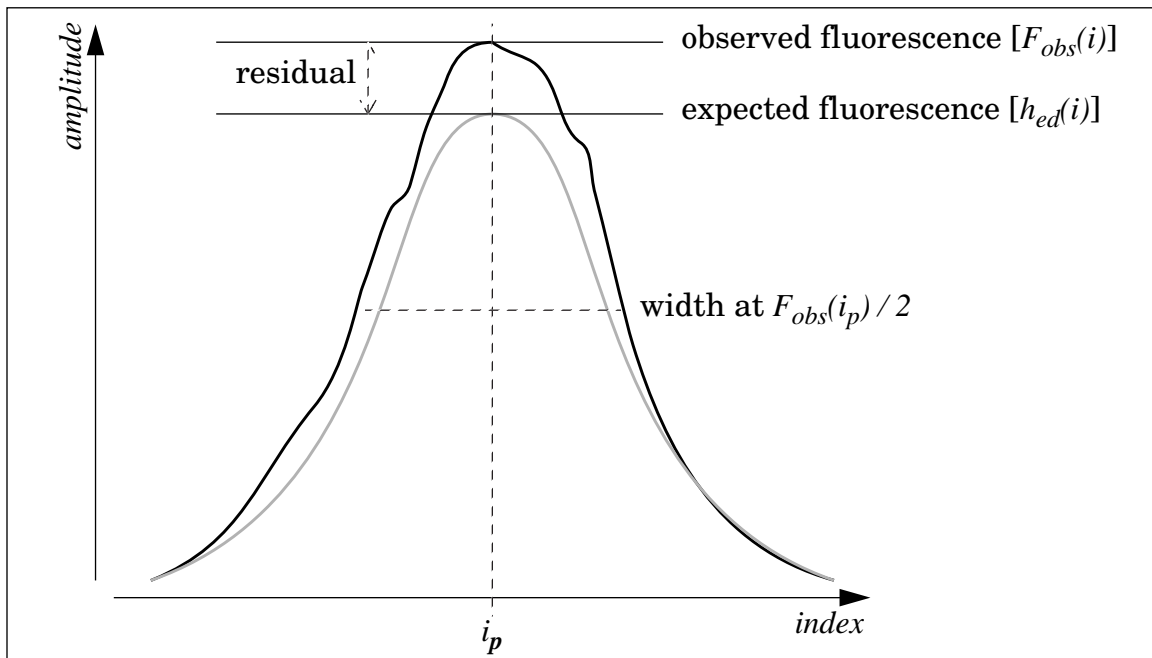


Figure 3-1: Diagram of peak shape. The residual is calculated by subtracting expected fluorescence (with Gaussian peakshape) from observed fluorescence. The residual is assumed to be attributable to normally-distributed noise.

Equation 3-13 shows the derivation of $P(D|H)$ by combining the product of Gaussian distributions integrated over the range of all possible amplitudes, completing the square, and invoking the definition of the complementary error function, erfc .

$$\begin{aligned}
P(D|H) &= \int_0^{\infty} \frac{e^{-\left(\frac{(h_p - \hat{h}_{ed}(i_p))^2}{2\sigma_h^2(i_p)}\right)} e^{-\left(\frac{(F(i_p, h_p) - F'_{obs}(i_p))^2}{2\sigma_n^2}\right)}}{\sigma_h(i_p) \sqrt{2\pi} \sigma_n \sqrt{2\pi}} dh_p \\
&= \frac{1}{2\pi\sigma_h\sigma_n} \int_0^{\infty} e^{-\left(\frac{(h_p - \hat{h}_{ed}(i_p))^2}{2\sigma_h(i_p)^2} + \frac{(h_p - F'_{obs}(i_p))^2}{2\sigma_n^2}\right)} dh_p \\
&= \frac{1}{2\pi\sigma_h\sigma_n} \int_0^{\infty} e^{-(Ah_p^2 + Bh_p + C)} dh_p \\
&= \frac{1}{2\pi\sigma_h\sigma_n} \int_0^{\infty} e^{-(Dh_p + E)^2 - F} dh_p \\
&= \frac{e^{-F}}{2D\pi\sigma_h\sigma_n E} \int_0^{\infty} e^{-G^2} dG \\
&= \frac{e^{-F}}{4D\sigma_h\sigma_n\sqrt{\pi}} \operatorname{erfc}(E)
\end{aligned}$$

(3-13)

where:

$$\begin{aligned}
A &= \frac{1}{2} \frac{\sigma_h^2 + \sigma_n^2}{(\sigma_h\sigma_n)^2} & D &= \sqrt{A} \\
B &= \frac{\sigma_h^2 F'_{obs}(i_p) + \sigma_n^2 \hat{h}_{ed}(i_p)}{(\sigma_h\sigma_n)^2} & E &= \frac{B}{2D} \\
C &= -\frac{1}{2} \frac{(\sigma_h F'_{obs}(i_p))^2 + (\sigma_n \hat{h}_{ed}(i_p))^2}{(\sigma_h\sigma_n)^2} & F &= C - \frac{B^2}{4A} \\
& & G &= Dh_p + E
\end{aligned}$$

and using the definitions of completion of the square and erfc:

$$Ax^2 + Bx + C = (Dx + E)^2 + F$$

$$erfc(z) \equiv 1 - \frac{2}{\sqrt{\pi}} \int_0^z e^{-u^2} du$$

Bayes' Theorem can now be applied using the heuristic definitions given in Equations 3-4, 3-13 and 3-8.

$$P(H|D) = \frac{P(D|H)}{P(D|\emptyset) + P(D|H)} P(H) \quad (3-3)$$

$$P(H) = \text{constant}$$

$$P(D|\emptyset) = \frac{1}{\sigma_n \sqrt{2\pi}} e^{-\left(\frac{F_{obs}(i)}{2\sigma_n}\right)^2} \quad (3-4)$$

$$P(D|H) = \frac{e^{-F}}{4D\sigma_h\sigma_n\sqrt{\pi}} erfc(E) \quad (3-13)$$

These equations permit the heuristic calculation of the probability that a particular base corresponds to (results from) a particular base given the observed data.

Thus far, the channels of fluorescence have been considered independently. It is impossible for two or more bases to occupy a given position in a DNA sequence, although this fact has not been specifically accounted for in the heuristics described above. In order to choose among these mutually-exclusive decisions of base at a given position, the most probable base is

chosen. The process of selecting the most probable base, called peak pruning, as well as the implementation of the heuristics are described in the next chapter.

4. C++ CLASS LIBRARY & IMPLEMENTATION OF THE PEAK MODEL

The second goal of this thesis was the development of a library of C++ classes which facilitate the manipulation of chromatogram data. Three classes were designed and constructed to represent three common elements of chromatogram analysis: traces, tracefiles, and lists of peaks. In addition, a doubly-linked list class template, *CSequence*, was developed to provide list manipulation methods and operators for data of an arbitrary type. Although it is used by two of the classes in this library, the implementation of linked lists is straightforward and will not be discussed.

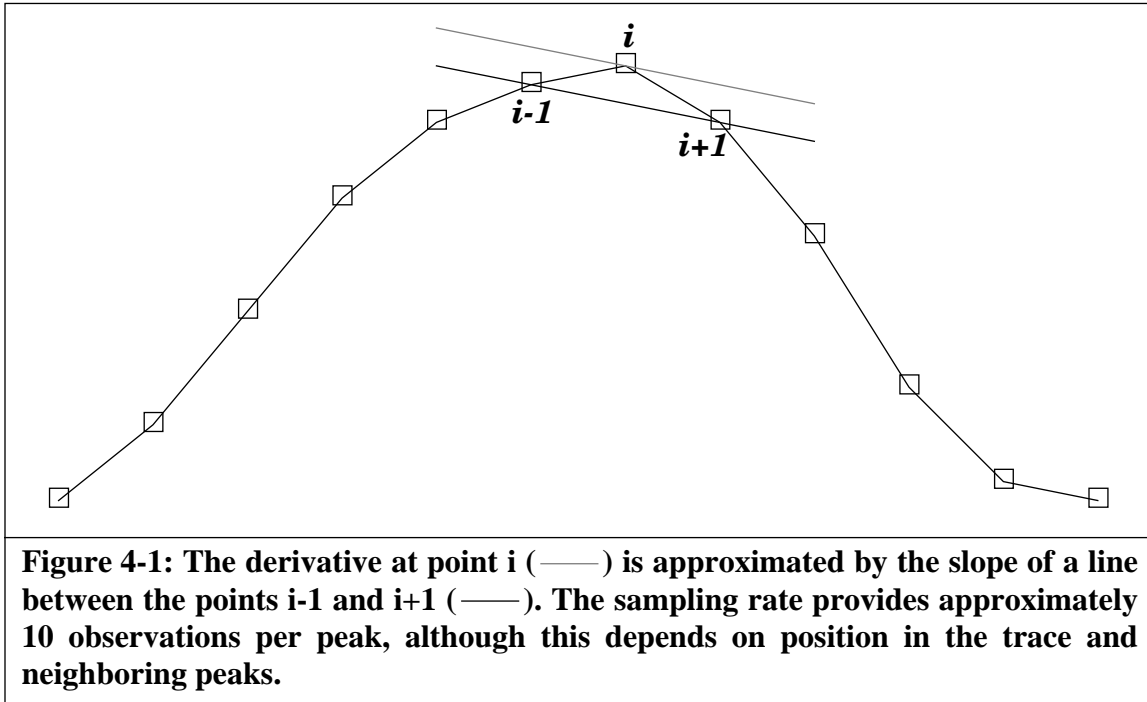
A noteworthy advantage of object-oriented programming and these classes is the ability to override class methods for the purpose of experimentation. The classes are designed to provide a complete workbench from which a researcher may choose the provided tools or create her own on a per-method basis. For instance, if a new algorithm for peak pruning is desired, it is a simple matter to override the provided method without requiring modification of any other method or function. (All of the classes developed for this thesis are prefixed with the letter C to remind the reader that the type definition is a Class, as opposed to some other method-less type.)

CTrace

CTrace is a template class which stores an array of numbers of an arbitrary type (i.e., integer, float, double, etc.) and length, and provides many methods which are common for an array of numbers which represent sampled data. These methods include functions which calculate minimum, maximum, mean, and variance, formatted input and output with C++ iostreams, scaling, translation, and array selection operator [] for array-like access to trace elements. In addition, there are several *CTrace* methods which merit more extensive discussion: *Derivative*, *PeakBounds*, *PickPeakIndices*, *PickPeaks*, and *Smooth*.

Derivative computes the derivative of a trace and returns the result in a new *CTrace<double>*. Because the result is returned as a *CTrace*, the *Derivative* method may be easily applied to the result to obtain the second and

successive derivatives. For a trace with s elements, numbered $0..s-1$, the derivative at point i is set equal to the slope of the line between points $i-1$ and $i+1$ for $1 \leq i \leq s-2$ as depicted in Figure 4-1.



PickPeakIndices returns a *CSequence* of indices of the centers of peaks using a combination of threshold and local maximum criteria. If the amplitude at a particular index exceeds a user-specifiable threshold, then the first derivative is consulted to determine whether the slope crosses zero (or, optionally, approaches zero within an arbitrary tolerance) and the second derivative is negative. Figure 4-2 shows data obtained from an Applied Biosystems, Incorporated (ABI) sequencer and the corresponding first derivative. Pseudocode for PickPeaks is given in Figure 4-3.

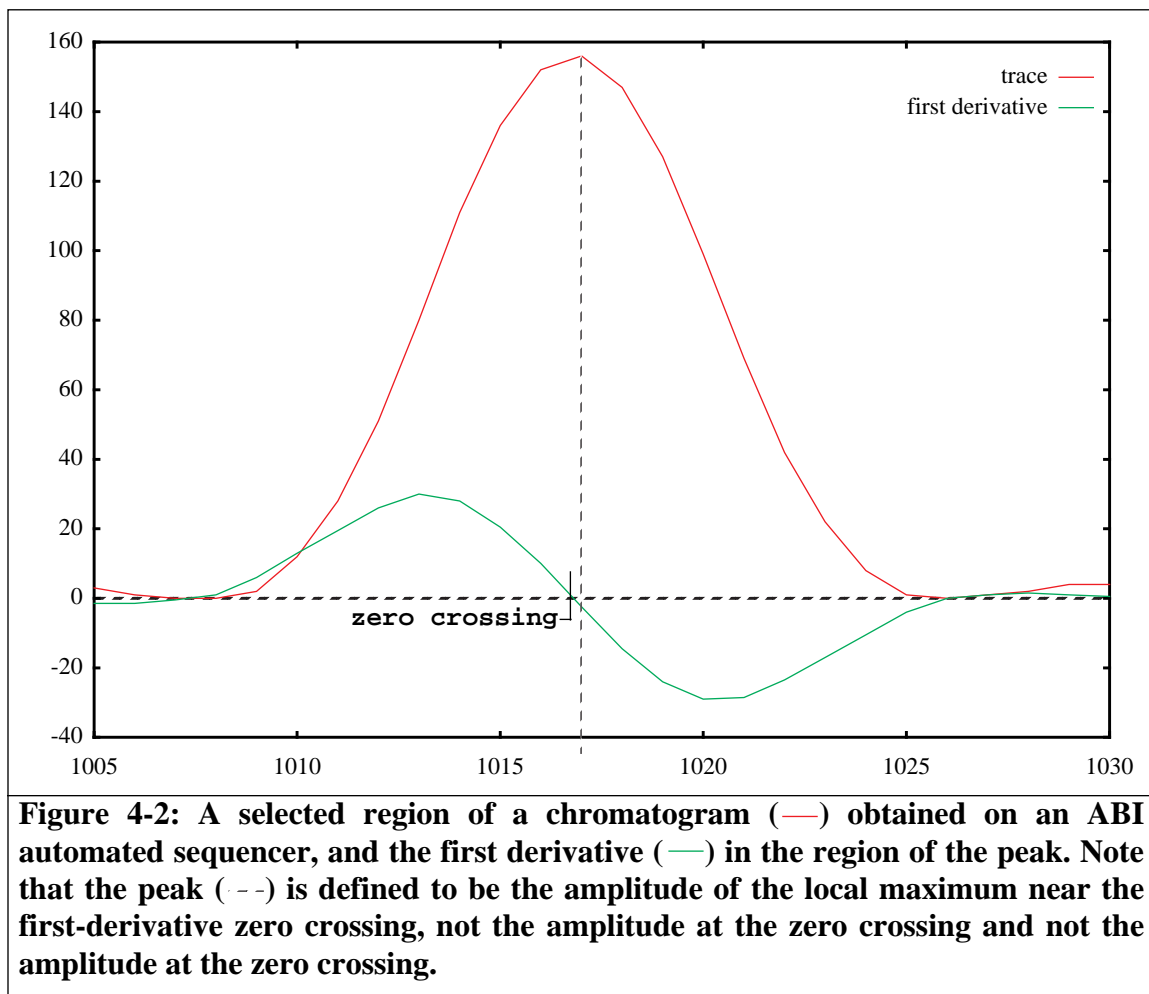


Figure 4-2: A selected region of a chromatogram (—) obtained on an ABI automated sequencer, and the first derivative (—) in the region of the peak. Note that the peak (- -) is defined to be the amplitude of the local maximum near the first-derivative zero crossing, not the amplitude at the zero crossing and not the amplitude at the zero crossing.

PickPeaks uses PickPeakIndices to generate a *CPeakList* (described below) which contains detailed information about peak position, amplitude, lateral bounds, and width at one-half of peak amplitude. A discussion of how these data are used for statistical analysis appears in the description of the *CPeakList* class on page 36.

PeakBounds searches within a user-specified window for the lateral bounds of a peak at a given amplitude, interpolating to a fractional index if necessary. It correctly recognizes cases of overlapping peaks (that is, peaks

```

Algorithm: PickPeakIndices
Input:      MinPeakAmplitude - a minimum amplitude threshold
           ZeroThreshold - the neighborhood about zero in which the derivative
                is considered equal to zero
           trace[] - an array of sampled data in the range [0,size-1]
Output:     a list of peak indices

begin
  let dtrace[] = the derivative of trace (using the Derivative method)

  for i in [1,size-2]
    if trace[i] >= MinPeakAmplitude then
      // peak is above a user-defined amplitude threshold
      if abs(dtrace[i-1]) < ZeroThreshold then
        // the derivative is within a user-defined limit 'near'
        // zero... call it a peak and append i to the list of
        // peak indices
        add i to peak list
      else
        if (i<=size-3) and (dtrace[i-1] > 0 and dtrace[i] < 0) then
          // ensure that i is within [0,size-3] and
          // derivative crossed 0... take index of max(trace[i],trace[i+1])
          // note that crossing 0 in this way is an indirect use
          // of the second derivative.
          if trace[i] > trace[i+1] then
            add i to peak list
          else
            add i+1 to peak list (trace[i+1]>=trace[i])
          fi
        fi
      fi
    fi
  rof
nigeb

```

Figure 4-3: Pseudocode for picking peaks by a combination of threshold and local maximum criteria.

whose bounds intersect) and limits the bounds to the midpoint between the two peak centers, and can be constrained to search within a user-definable window centered around the peak index. Figure 4-4 shows a region of a chromatogram and the framework of peaks picked in the region.

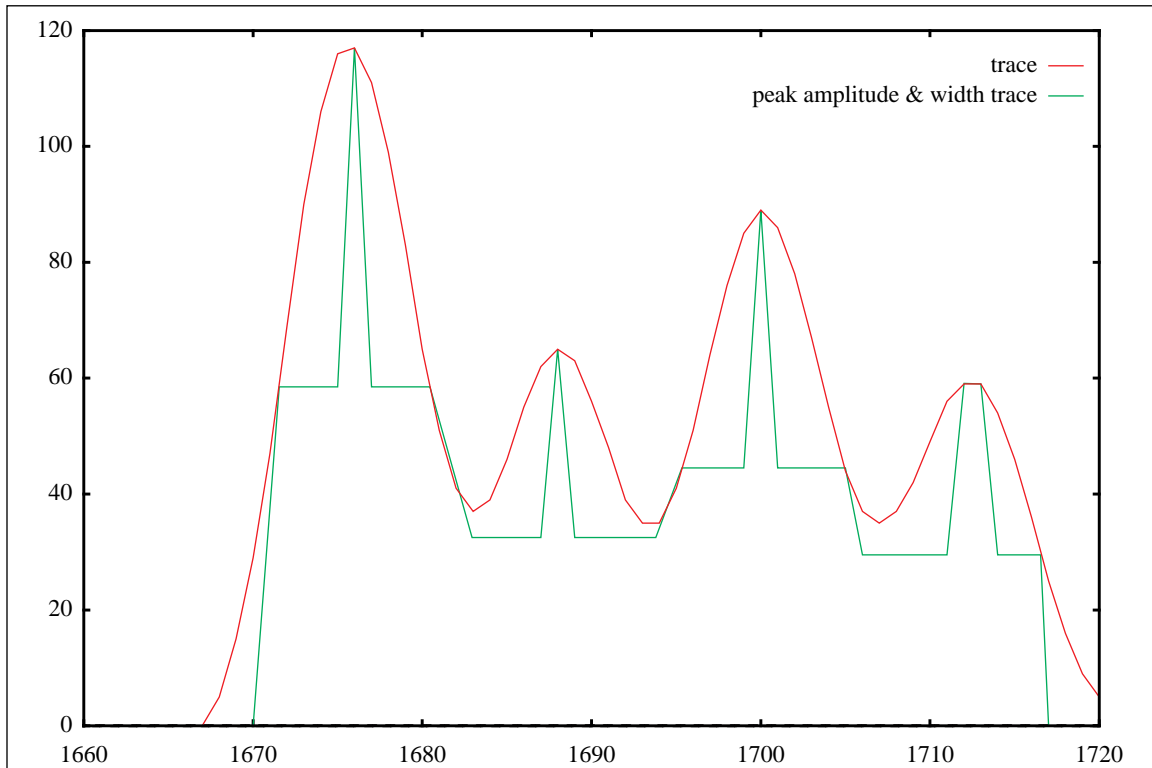


Figure 4-4: A trace and peaks from a single channel of fluorescence. The peak framework depicts both amplitude and width of the peak determined from the observed data. Peak width is determined by searching laterally for the intersection of the peak with a line at one-half of peak amplitude. For closely-spaced peaks, this may require arbitrary truncation midway between the two peaks.

Smooth implements a simple smoothing algorithm for removing small local fluctuations in data. At every trace index, i , a programmer-specifiable vector of weights is applied to data indices $i-1$, i , and $i+1$. Endpoints in the trace are handled by the normalization of the weights for the two remaining points. All figures in this section were smoothed using this method with the matrix $[0.25, 0.50, 0.25]$.

CTraceFile

CTraceFile represents the collection of data (traces, experimental conditions, etc.) from automated DNA sequencing projects. It allocates one *CTrace* class for each of the four fluorescence channels, the sequence of DNA stored with the trace (predicted by the equipment manufacturer's proprietary method, for instance) and numerous other data members. It also provides methods for picking peaks in the set of four traces (using the peak picking methods described for *CTrace*) and merging these selections into a preliminary list of peaks in the four traces.

Perhaps the most important feature of *CTraceFile* is its ability to read ABI and Standard Chromatogram Format (SCF) files. (SCF is a non-proprietary interchange file format for chromatographic data.) A primary goal of this thesis was to facilitate user experimentation with methods for peak picking. Because *CTraceFile* can read and write these formats — and convert between them — the complicated task of tracefile input and output is simplified. An infrastructure exists for the implementation of I/O in other formats, although time constraints and other thesis priorities necessitated limiting input and output capabilities to ABI and SCF files. In the meantime, utilities are available for the conversion of other formats to SCF.

CTraceFile provides methods for the transformation of the set of four chromatogram channels with a programmer-specifiable transformation matrix. Orthogonalization of chromatographic information is a standard

processing technique used to minimize the effects of correlations between data. In the case of automated DNA sequencing with fluorescent deoxynucleotides, one often observes an overlap in the emission spectra of the fluorescent labels and orthogonalization provides a mechanism to compensate for the effects of this overlap. The vector notation for this transformation is:

$$R = MO$$

or, equivalently:

$$R_T(i) = \sum_{S \in \{A, C, G, T\}} m_{TS} O_S(i)$$

where R is the resulting vector of 4 traces (4 x *length of trace*)

O is the original vector of 4 traces (4 x *length of trace*)

S & T are trace identifiers (Source & Target) in {A,C,G,T}

M is the 4x4 matrix whose elements m_{TS} are the cross-term contributions of channel S to channel T

$$\text{i.e., } M = (m_{TS}) = \begin{bmatrix} m_{AA} & m_{AC} & m_{AG} & m_{AT} \\ m_{CA} & m_{CC} & m_{CG} & m_{CT} \\ m_{GA} & m_{GC} & m_{GG} & m_{GT} \\ m_{TA} & m_{TC} & m_{TG} & m_{TT} \end{bmatrix}$$

i loops over the indices of the trace ($1 \leq i \leq \text{length}$)

AssimilatePeaks returns a merge-sorted list of peaks in a *CPeakList*, but provides no peak conflict resolution mechanisms; that is, it contains the union of the lists of peaks from individual traces without regard for the overlap of two peaks from different traces. It is important to note that the peak probabilities are normalized with respect to other peaks selected from the same trace and that this allows the comparison of probabilities of two peaks from *different* traces.

A principal component of *CTraceFile* is PrunePeaks, which resolves conflicts in the assimilated peak list. Two peaks separated by less than an arbitrary minimum separation threshold are deemed to conflict. That is, in the sequencing of homozygotic DNA, exactly one base occupies any given position; therefore, two peaks within this threshold — representing two bases — are assumed to contradict this requirement. While PrunePeaks works well in most cases, there are circumstances in which it fails to choose the most desirable peak from a series of conflicting peaks. This condition and suggested remedies, as well as potential applications to heterozygotic DNA sequencing, are discussed in **7. IMPROVEMENTS AND EXTENSIONS**.

CPeakList

CPeakList defines a peak record structure, *PeakRec*, which represents the parameters of individual peaks selected by *CTrace::PickPeaks*. The record contains information about peak center, amplitude, lateral bounds, width, and the probabilities $P(D)$, $P(H)$, $P(D/H)$ and $P(H/D)$ which were discussed in **3. PEAK**

MODEL AND HEURISTIC ANALYSIS. The structure is a C++ class and provides C++ output stream (ostream) operator << and manipulators which greatly facilitate the output of human-readable lists of peak records.

The *CPeakList* class itself is derived by inheriting a base class derived from *CSequence* (*CSequence<PeakRec>*). Therefore, *CPeakList* inherits the linked-list functionality of *CSequence* and adds methods specific to the analysis of a list of peaks. This list may be pruned according to a user's experimental heuristic using insert and delete methods provided by *CSequence*.

CPeakList is responsible for the majority of the statistical analysis in this thesis and calculates probabilities essentially as described in **3. PEAK MODEL AND HEURISTIC ANALYSIS**. However, the calculation of several functions and parameters merit further consideration and a discussion of these methods will consume the remainder of this section. Recalling the equations for $P(D)$, $P(H)$, $P(D|H)$ and $P(H|D)$ (Equations 3-3, 3-4, and 3-13, reprinted below) derived in **3. PEAK MODEL AND HEURISTIC ANALYSIS** will provide the motivation for the data computed by the methods of *CPeakList*.

$$P(H|D) = \frac{P(D|H)}{P(D|\emptyset) + P(D|H)} P(H) \quad (3-3)$$

$$P(H) = \text{constant}$$

$$P(D|\emptyset) = \frac{1}{\sigma_n \sqrt{2\pi}} e^{-\left(\frac{F_{obs}(i)}{2\sigma_n}\right)^2} \quad (3-4)$$

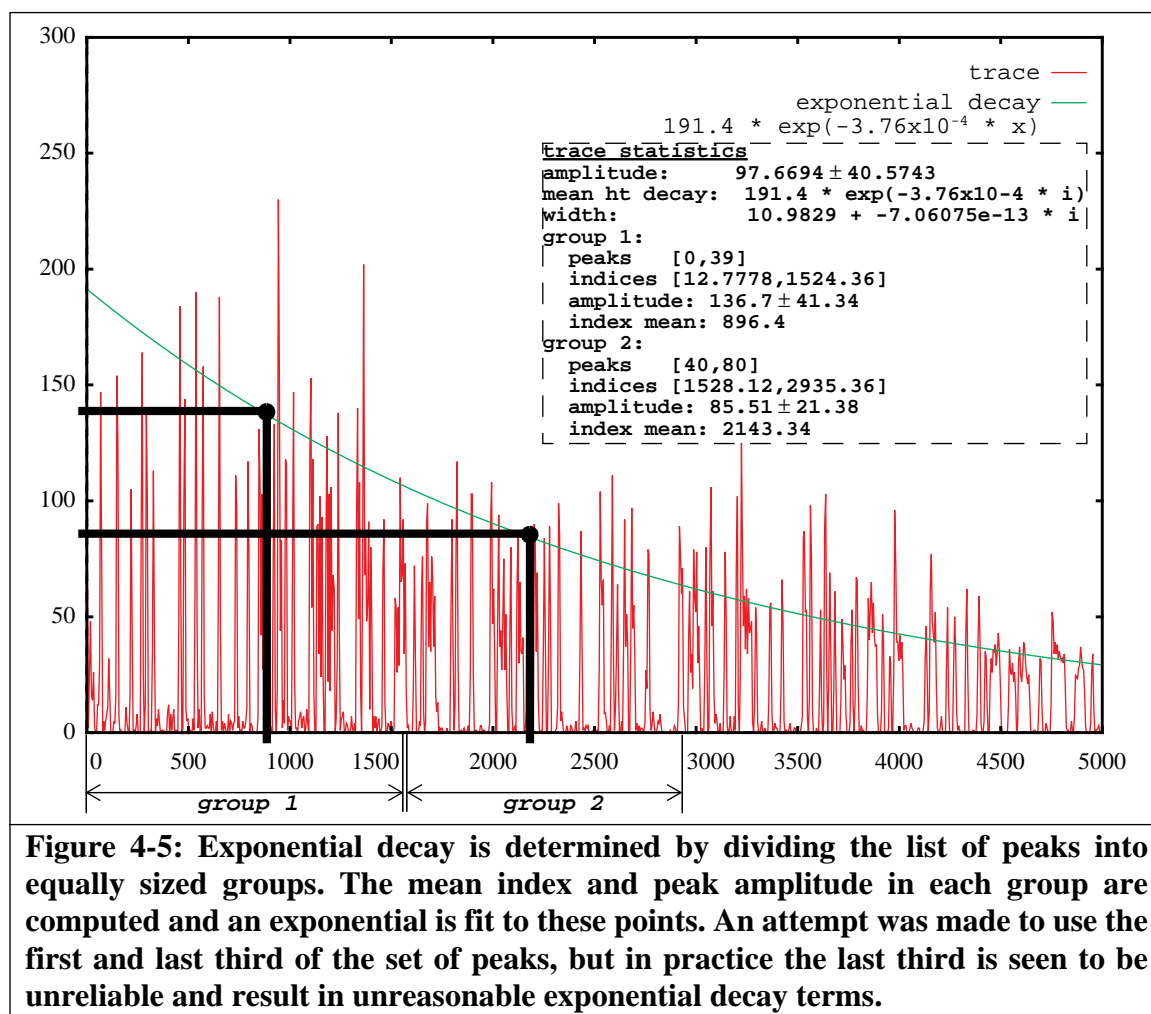
$$P(D|H) = \frac{e^{-F}}{4D\sigma_h\sigma_n\sqrt{\pi}} \operatorname{erfc}(E) \quad (3-13)$$

A survey of the terms on the right hand sides of these equations reveals that $\hat{h}_{obs}(i_p)$, σ_h , σ_n , and $h_{exp}(i)$ must be determined. $h_{obs}(i)$ is simply the amplitude of the peak which has already been determined and can be easily found in the peak record. \bar{h}_{obs} is the baseline of the data which is assumed to have been provided or determined elsewhere.

CPeakList provides the *CalculateStats* method to determine a variety of statistics for a population of peaks selected by *CTrace*. Mean and variance are determined for the entire population, although these terms are not directly relevant to this analysis. In addition, the method of least squares is used to generate an equation for the linear increase in peak width with index position.

CPeakList also fits equations for the exponential decay of mean peak amplitude, $\hat{h}_{obs}(i_p)$, and variance. The population of peaks is divided into two (nearly) equally sized contiguous groups. For two of the partitions, it determines the mean and variance of peak amplitude and mean peak index in each group. (Mean index of each partition is computed in order to compensate for skewed distributions of peaks along the trace. Figure 4-5 demonstrates the division of peaks into two groups and the determination of the exponential decay of mean peak amplitude.)

ComputeFTrace generates a *CTrace* of expected fluorescence, $h_{exp}(i)$, by modeling each peak as a Gaussian in a window of user-specifiable size. A sample of the expected fluorescence is shown in Figure 4-6 on page 40. ComputeRTrace generates the residual trace that results from subtracting expected fluorescence from observed fluorescence and can be used to estimate the mean and variance of noise. ComputePTrace constructs a trace intended to overlay observed traces to show peak amplitude and widths (see Figure 4-4 on



page 33). Because each of these methods generates a *CTrace*, the methods already mentioned may be used to write these data to streams, compute mean and variance, and so forth.

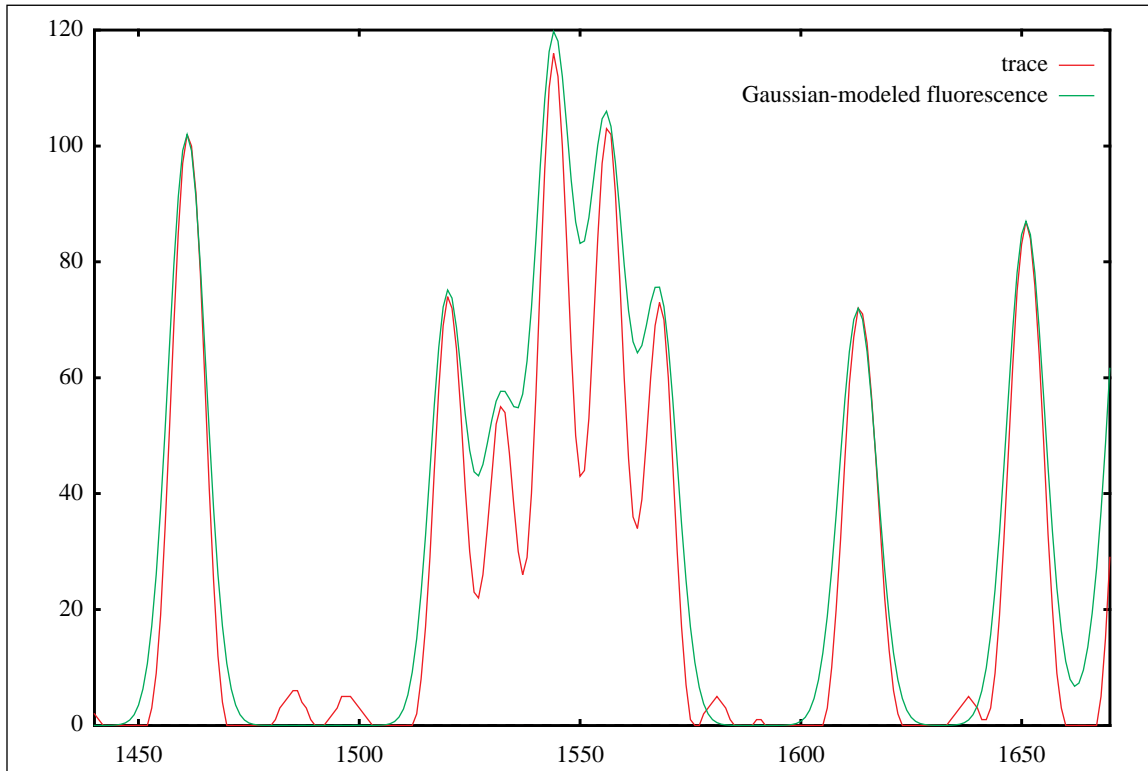


Figure 4-6: Sample peaks from data (—) obtained on an ABI sequencer. The expected fluorescence (—) was calculated as described in the text using a Gaussian model of the peak shape (model width = 50) without noise. The peaks were initially identified by a combination of threshold and local maximum criteria, although the choice of criteria does not affect the peak shape model. Deviations of expected versus observed fluorescence are believed to be partially attributable to errors in peak width determination which result from inaccurate baseline estimation, and consequently inaccurate one-half peak-amplitude estimation. Noise contributions are calculated from the residuals obtained by subtracting expected fluorescence from observed fluorescence (not shown).

The last remaining term to compute is the variance of noise, σ_n . If the model is correct and the set of peaks is a good approximation of the actual sequence, then the residuals should be composed entirely of noise (i.e., noise \approx

residual). As was just mentioned, the residual values are stored in a *CTrace*, and thus σ_n is readily obtained from the statistical methods provided by that class.

`CalculatePeakStats` uses the statistics computed above by `CalculateStats` to iterate through all peaks and determine $P(D)$, $P(H)$, $P(D/H)$, and $P(H/D)$. The determination of these values is straightforward from Equations 3-3, 3-4, & 3-13 and the parameters described in this section.

5. PERFORMANCE OF THE PEAK MODEL

A program called *autoseq* was developed to test the functionality of *CTrace*, *CTraceFile*, and *CPeakList* and to assess the performance of the peak model relative to current methods. Eleven ABI tracefiles were provided by the *Caenorhabditis elegans* mapping and sequencing group headed by Bob Waterston and Rick Wilson at the Washington University School of Medicine. All sample files were sequenced using the dye-primer methodology (see **2. DNA STRUCTURE AND SEQUENCING** on page 5).

Data for the comparison was generated using *autoseq* (see **APPENDIX C:AUTOSEQ USERS' GUIDE** on page 67). The command-line invocation used was:

```
% autoseq -fmt ABI0 -l p -r -100 -sm 2 -bl -x ortho -p 4 -ps -fs {file}
```

This command directs *CTraceFile* to load raw ABI data, clipped to a region from the primer to 100 sample points from the end of the trace. Each trace was smoothed twice, baseline corrected by translating each trace by the minimum value for that trace over the specified region, and orthogonalized with the Transform method provided by *CTraceFile* using a custom orthogonalization matrix (shown below) which was developed empirically using the Splus statistical package (10). Peaks were pruned with the PrunePeaks method as described in **4. C++ CLASS LIBRARY &**

IMPLEMENTATION OF THE PEAK MODEL using a minimum separation of 4. The sequences predicted by ABI and by the model presented in this thesis were written to output files.

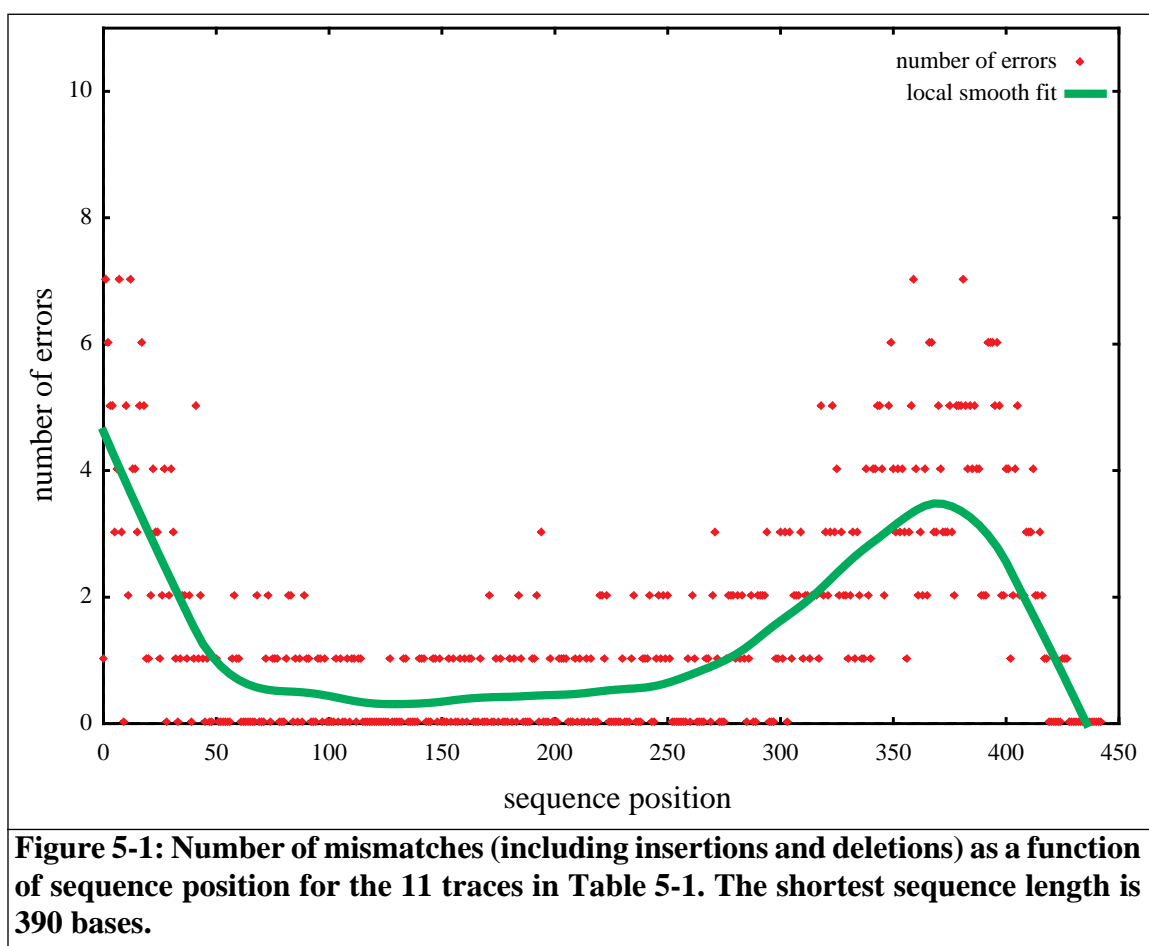
$$M = \begin{bmatrix} 1.1574327 & -0.4084339 & -0.2466887 & 0 \\ -0.4051014 & 0.4538154 & 0.0308361 & 0 \\ -0.6365880 & 0 & 0.9250816 & -0.2856967 \\ 0 & 0 & -0.7400653 & 1.9998770 \end{bmatrix}$$

To verify that the predicted sequence was similar to that chosen by ABI, the two sequences were compared with FASTA, a sequence alignment tool (6,7). The alignment results for the 11 traces are indicated in Table 5-1.

Table 5-1: Alignment scores and overlaps for fasta comparisons of predicted sequence versus sequence predicted by ABI. Rows marked with ❖ represent tracefiles in which fasta comparisons of the sequence predictions by *CPeakList* and ABI had similarity scores in excess of 80%.

file	score (%)	overlap (bases)
ba16d3.s1	89.6	347
ba16d4.s1	93.4	364
ba16d5.s1	91.7	362
ba16d6.s1	92.5	399
ba16d7.s1	91.4	420
ba16d8.s1	92.3	405
ba16d9.s1	92.5	385
ba16d12.s1	94.5	384
<i>Average for scores ≥ 85% (N=8):</i>	92.2	383.2
ba16d1.s1	78.0	277
ba16d2.s1	82.4	335
ba16d11.s1	73.0	355
<i>Average for all entries (N=11):</i>	88.3	366.6

To investigate the possibility of position-dependent bias in the errors, the alignments were scored for mismatches at each base in the alignments. Figure 5-1 shows the total number of mismatches in the set of 11 tracefiles as a function of sequence position, and reveals a propensity of mismatches at the extremities of the traces.



An investigation of the causes for high mismatch rate below 100 bases revealed that a large number of peaks are pruned early in the trace, as shown in Figure 5-2. Grossman *et al.* (5) have previously shown that the effect of sequence length on mobility is non-linear, with short sequences being having

the greatest deviation from a linear approximation. This is in agreement with the observation that the sequences predicted by the C++ library are consistently shorter than those predicted by ABI by 20-30 bases. This observation implies that pruning mechanisms which rely on fixed peak spacing (i.e., a constant window size) will excessively prune peaks at short sequence lengths.

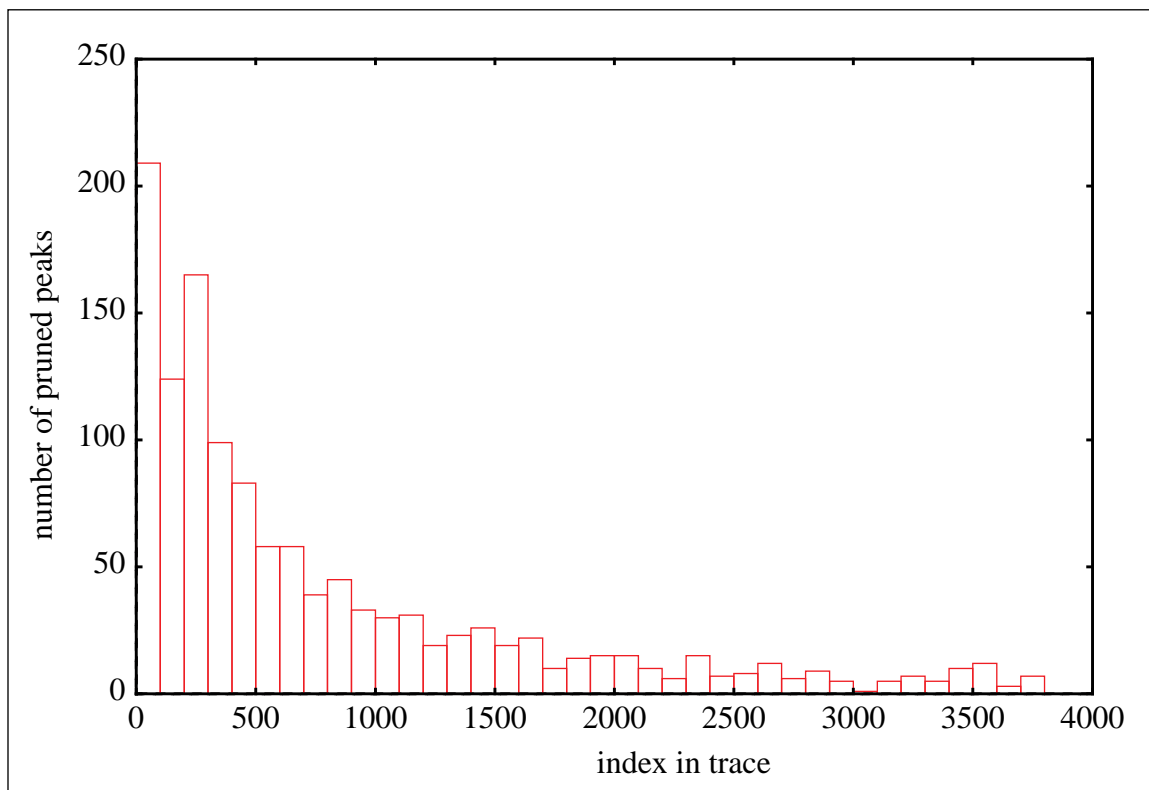


Figure 5-2: The number of peaks pruned early in the trace is significantly larger than the number pruned in later regions. The histogram is the total number of peaks pruned in 100-base intervals in the 11 traces from Table 5-1. (100 bases is arbitrary)

The high mismatch rate beyond 300 bases seen in Figure 5-1 suggests sensitivity of the model to trace quality degradation near the end of traces. Although the breakdown of analytical methods is expected in regions of poor trace quality, the 300-base limit is less than that obtained by current methods which currently achieve reliable reading up to about 400 bases. A portion of these errors might also be attributable to fixed-window pruning. In contrast to the above case, the window in this region of the trace might be too narrow and result in the preservation of false peaks, and results in insertions in the sequence.

6. APPLICATION TO DNA SEQUENCING

The feasibility of automated DNA sequencing depends on the correspondence between the mobility of fragments of DNA and their length. In ideal cases, the mobility of DNA is independent of its sequence. In practice, however, DNA migration *does* depend on sequence. An understanding of the influence of sequence on DNA migration will provide clues regarding the mobility contribution of individual bases and fluorescent labels to electrophoretic migration. This, in turn, might significantly improve the interpretation of chromatograms by offering position-dependent identification of peaks.

Although a detailed study of these effects are beyond the scope of this thesis and would have required more time than was available, the preliminary results are in general agreement with Grossman *et al.* (5) who showed non-linear mobility dependence on sequence length for short (< ~150 bases) strands of DNA. This chapter demonstrates the use of *autoseq* in researching the mobility contributions of bases and dye labels.

To study this effect, the peak-to-peak spacing of the 3'-most mononucleotide and 3'-most dinucleotide combinations were obtained using *autoseq*. The peak-to-peak spacing was named base-position-deltas (bpds, or

simply deltas) because they may be interpreted as the increase in migration time which results by adding the mononucleotide or dinucleotide. A diagram of the delta is shown in Figure 6-1.

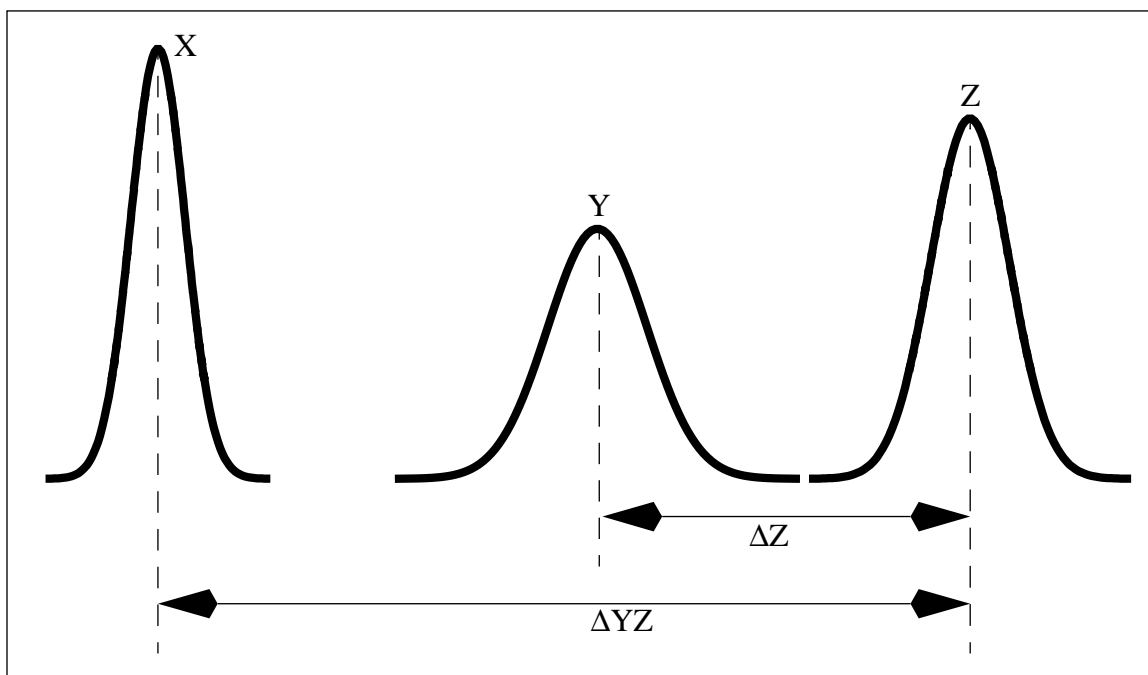


Figure 6-1: Diagram of 1 and 2 base-position-deltas (ΔZ and ΔYZ). X, Y, and Z represent arbitrary bases in the chromatogram. Note that these bases are not necessarily on the same chromatogram, and therefore the peaks are disconnected in the trace.

Data were generated using *autoseq* with the command-line invocation:

```
% autoseq -fmt ABI0 -l p -r -100 -sm -bl -x ortho -p 4 -bpd 2 {file}
```

This command performs data processing as described in **5. PERFORMANCE OF THE PEAK MODEL**, and the base-position-deltas (bpd's) were computed for the 1 and 2 base termination combinations. The data set was limited to the eight files with fasta scores greater than 80% (see Figure 5-1 on page 43).

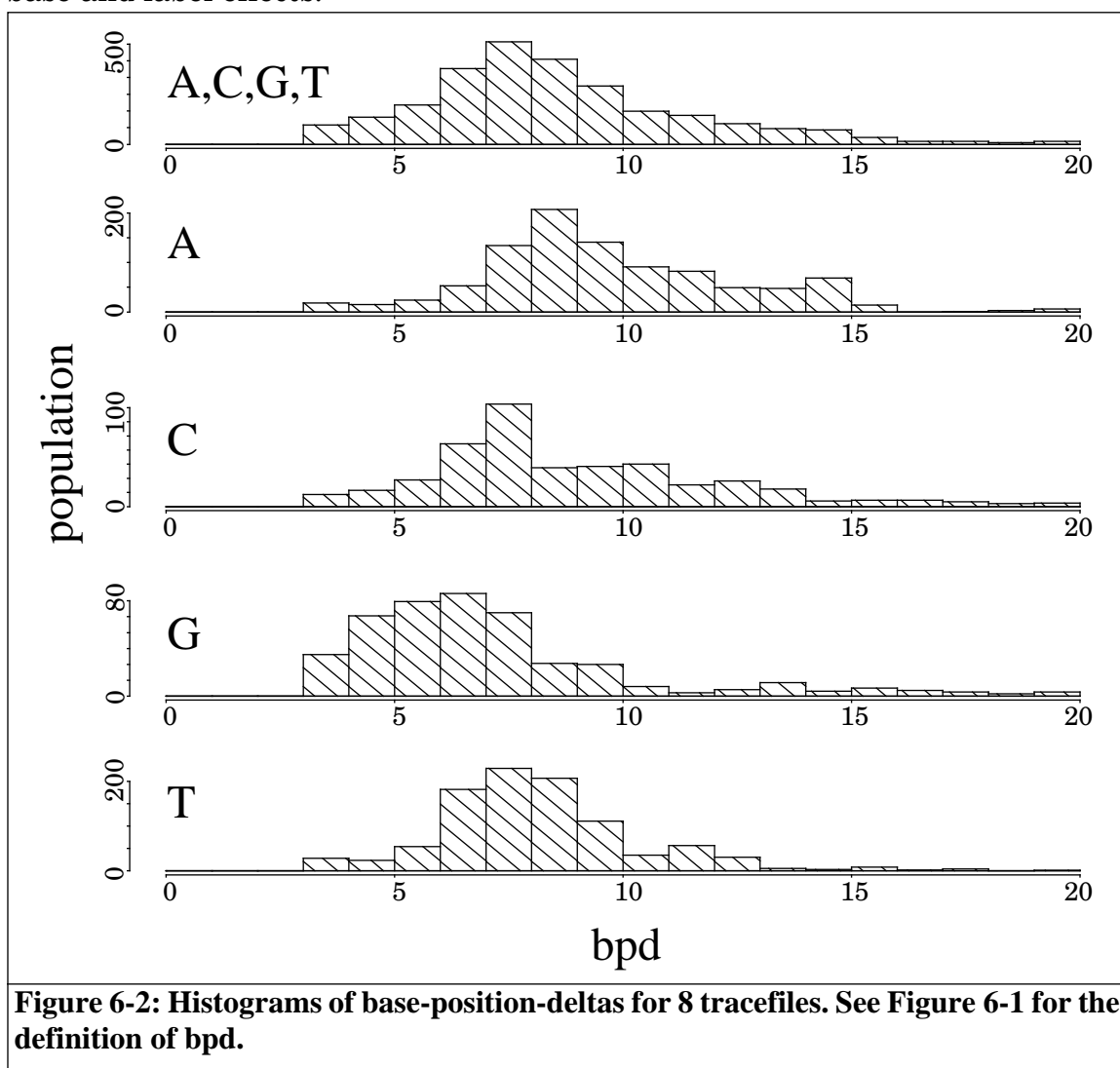
The *autoseq* analysis of each input file resulted in a file of all nucleotides and dinucleotides in the sequence, the position of the 3' base (of the mono- or di-nucleotide) in the traces, and the mono- or di-nucleotide delta. These were merge-sorted and the lists parsed by sequence content to produce four mononucleotide and sixteen dinucleotide bpd tables. The number of entries in each file is shown in Table 6-1.

Table 6-1: The number of each mono- and di-nucleotides after aggregation and parsing from the bpd files as described in the text.

3' mononucleotide	Population	3' dinucleotide	Population
A	1171	AA	354
		CA	237
		GA	321
		TA	256
C	818	AC	220
		CC	150
		GC	204
		TC	244
G	1008	AG	250
		CG	178
		GG	242
		TG	333
T	1329	AT	343
		CT	252
		GT	238
		TT	493

A histogram of the aggregate (A, C, G, & T combined) and individual mobilities is shown in Figure 6-2 and reveals biased migration of different DNA bases during gel electrophoresis. Although the standard deviations are relatively large, there is a migration tendency which appears to be correlated

with the 3' nucleotide. However, it is not evident from the 3' mononucleotides whether migration differences are caused by the chemical structure base or the fluorescent label. We believe that *autoseq* would be useful in separating the base and label effects.



7. IMPROVEMENTS AND EXTENSIONS

There are many possibilities for improvements to the implementation of the C++ class library and the model, and extensions to the application of the classes developed for this thesis. Several of these are discussed in this section.

Peak Spacing and Conflict Resolution Heuristics

The current method of combining lists of peaks from individual traces uses a simple merge sort method based upon index which is subsequently pruned by removing the less probable peak when the distance between two adjacent peaks conflict by occupying the same window of user-specifiable size. For a set of peaks with monotonically increasing probabilities and whose pairwise separations are less than the window size, the result is that only one peak from the set — the last one — will remain. In addition, it was seen in **6. APPLICATION TO DNA SEQUENCING** that pruning with a fixed window size ignores the mobility effects of short sequences and results in a tendency to excessively prune peaks early in the trace.

A solution is to incorporate peak spacing directly into the probability of a peak. As was discussed in **3. PEAK MODEL AND HEURISTIC ANALYSIS**, the probability of a peak was modeled on both its amplitude and position in the trace. For simplicity, the spacing term was set to unity (1), implying no bias for

trace position. In reality, chromatographic data shows remarkably consistent spacing for the first approximately 70% of the trace, and a simple adaptation to the implementation described herein is the development of a periodic function for peak spacing. (In the last 30% of most traces, data are too distorted to discriminate peaks using methods developed for this thesis.) Incorporation of this function to both peak picking and peak resolution will undoubtedly improve predicted sequence reliability. This method could incorporate the mobilities of individual bases to better estimate spacing.

Applications to Heterozygotic DNA Sequencing

The sequencing discussed throughout the majority of this thesis has assumed that a unique region of DNA is being sequenced. One application of automated DNA sequencing — in fact, a technique that requires this technology — is the simultaneous analysis of two nearly-identical (heterozygotic) DNA sequences.

Most organisms contain two copies of the same region of DNA, one inherited from each parent. This implies that individuals have two copies of every gene (this is not strictly true, but suffices for our discussion). During the process of evolution and mating, genes are shuffled and constantly undergo small mutations. A particular class of mutation, point mutations, involves the change of one base of DNA to another, resulting in no change in sequence length. The result is that the two copies of a gene in a cell are rarely exactly identical.

When such DNA is sequenced, one expects and observes similar traces throughout the majority of the data, with an occasional overlap of two peaks on different traces corresponding to the point at which the sequences differ.

Because the bases at these point mutations are real (as opposed to being caused by noise), one might expect relatively high probabilities for the corresponding peaks. Identification of these peaks could be implemented by a slight modification to the peak conflict resolution heuristics in which overlapping peaks with similar probabilities [$P(H/D)$] are flagged as possible locations for point mutations. This differs from the current implementation of PrunePeaks in that there is currently no qualitative estimate of the similarity of two peaks; the less probable peak is discarded even if it is only insignificantly less than the one to which it is compared.

Analysis of the Residuals

A common technique in exploratory data analysis is the interpretation of outliers — results which cannot be explained by the current model. As has already been mentioned, a trace of residuals is generated during the process of calculating peak probabilities. If the model perfectly explained the data, one would expect that the residual trace consisted entirely of noise. With this assumption, the residual trace is an ideal indicator of regions in which the model differs from the observation; such regions are apparent by large deviations from the mean of the residual.

Because the residual traces are *CTrace* classes, peak picking on the residuals is a trivial task. Peaks in the residual traces isolate areas of the trace which require further analysis.

Iteration

Much of the process of examining experimental data by robust statistical methods requires 'bootstrapping'. That is, the data to be analyzed are examined for trends and these trends are used to identify data which can be explained. The remaining data are grouped into a set of outliers or residuals and a second phase of searching for trends begins and may use information obtained from the first phase to predict the behavior of the residuals.

This paradigm might be effective in the peak selection. For instance, one might use a first pass of selecting peaks using the local maximum/threshold method used in this thesis. From this initial set of peaks, the mean, variance, and exponential decay could be computed. If this exponential decay is used to pick peaks in lieu of a constant threshold, one might expect more precise.

Likewise, iteration could be used to reduce the mean and variance in the residual trace. If the model is correct, then the residual trace should reflect noise alone. We expect that the mean of the noise will be 0 and that iterative minimization of σ_n implies an optimal description of the data by the proposed set of peaks.

I/O (machine-independent and file formats)

Although file and device I/O is provided by all of the classes developed for this thesis, they do so in a machine-specific format. Specifically, none of the methods are endian-independent, and this is an important concern for the portability of these classes. (Endian refers to the byte order used for the internal representation of multi-byte data. Machines which are endian compatible may freely exchange files without; machines which are not endian compatible may exchange files, but the contents may be useless because the byte order is different.)

Several modifications have already been considered. A particularly convenient improvement to accommodate endian-independent I/O is to override the operators `<`, `>`, `<=`, and `>=` for `iostreams`. These can be defined for an array of types (just as `<<` and `>>` are) and would provide flexible, type-safe formatting. The choice of these operator tokens is important because of the operator evaluation order and binding is not specifiable by the programmer in C++; thus, existing operators with operator precedence and binding similar to `<<` and `>>` are desired.

8. CONCLUSION

The thesis fulfilled its primary objectives of developing C++ classes which facilitate experimentation with analytical methods for the inference of sequence from automated DNA sequencers and the implementation of a peak model which provides a statistical basis for the assignment of probabilities to bases in a sequence.

The class library consists of three classes which correspond to chromatograms, tracefiles, and lists of peaks. Because each is a C++ class, the methods may be overridden to allow selective experimentation with aspects of chromatogram analysis.

The model presented has promise for extending both the quantity and quality of data which can be inferred from automated DNA sequencing, although it currently falls short in both the length and reliability of the sequence. Several recommendations for improvements were presented.

The three C++ classes were incorporated into a *autoseq*, which was used to analyze a set of dye-primer automated sequencing data for the mobilities of individual bases of DNA. It was shown that dye-label effects are at least partially responsible for the migration differences of DNA during electrophoresis.

APPENDICES

APPENDIX A: RELEVANT STATISTICAL METHODS

Gaussian distributions (also called normal distributions) are commonly used to model a variety of natural phenomena, especially those which involve the random sampling of a large data set. The distribution of examination scores, the number of heads in a large number of tosses of a coin (when Gaussian distributions approximate binomial distributions), and life-span of an organism can all be accurately modeled by Gaussian distributions.

Figure A-1 shows a standard Gaussian distribution. The intuitive interpretation of a probability density function is a function which gives the probability, $P(z)$, of a particular event whose outcome can be summarized by the random variable z .

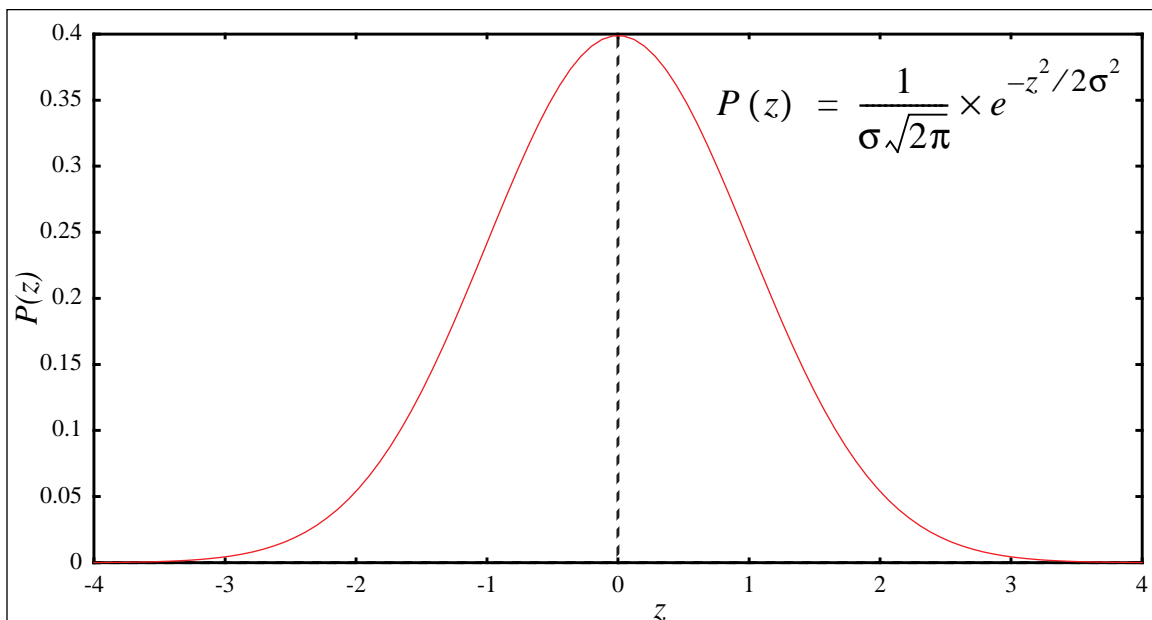


Figure A-1: Standard Gaussian distribution about $z=0$ for $\sigma = 1$. The curve is symmetric about $z=0$ and the integral of the Gaussian density function $[P(z)]$ in the limits of $[-\infty, +\infty]$ is equal to unity.

Most experimental results are not initially expressed with a mean (average) value of zero and a standard deviation of 1. Fortunately, these values are easily computed from a set of sampled data.

Suppose that we observe N samples of a continuous random variable, X , whose elements are $x_1 \dots x_N$. The mean, μ_X , and standard deviation, σ_X , of X are given by:

$$\mu_X = \frac{\sum_{i=1}^N x_i}{N} \quad \text{and} \quad \sigma_X = \sqrt{\frac{\sum_{i=1}^N (x_i - \mu_X)^2}{N}}$$

These two values may then be used to transform all data into z-scores using the formula:

$$z_i = \frac{x_i - \mu_X}{\sigma_X}$$

More simply, the probabilities of each x_i may be computed with:

$$P(x_i) = \frac{1}{\sigma_X \sqrt{2\pi}} e^{-(x_i - \mu_X)^2 / 2\sigma_X^2}$$

X was assumed to be a continuous variable. For a discrete distribution Y , the probability of a particular observation y_i is obtained by evaluating the integral of the density function in $[y_i - \Delta/2, y_i + \Delta/2]$, where Δ is the separation between observable values in Y .

In the case of peak amplitudes, for example, μ_h and σ_h are calculated from the set of peaks chosen initially by threshold/local-maximum criteria and $\sigma_h = 1$.

The probability of a peak with a height h_p is given by:

$$P(h = h_p) = \frac{1}{\sigma_h \sqrt{2\pi}} \int_{h_p - 0.5}^{h_p + 0.5} e^{-\left(\frac{(h - \mu_h)^2}{2\sigma_h^2}\right)} dh$$

APPENDIX B: VALIDITY OF PEAK MODEL ASSUMPTIONS

In **3. PEAK MODEL AND HEURISTIC ANALYSIS**, four assumptions about the data were stated but not validated by the empirical observations which support them. These assumptions are required for the peak model statistics, but the implementation of the class library is model-independent. The four assumptions are:

1. Gaussian peak shape
2. Gaussian distribution of peak amplitudes
3. Gaussian distribution of sample noise
4. Exponential decay of peak amplitude

Figure B-1 demonstrates Gaussian peak shape for a sample region of a trace after processing and peak picking with the modeled fluorescence of 4 peaks. The qualitative interpretation is that the Gaussian model provides good fit with raw observed data.

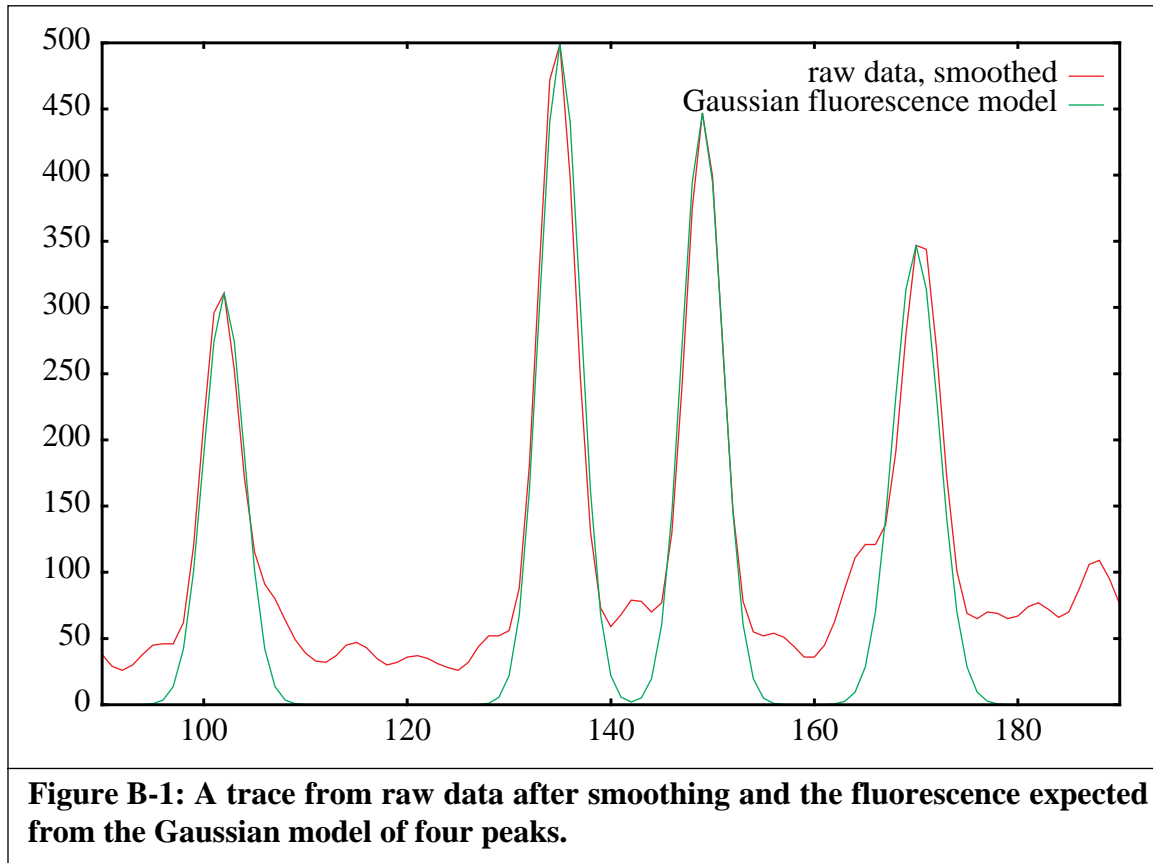
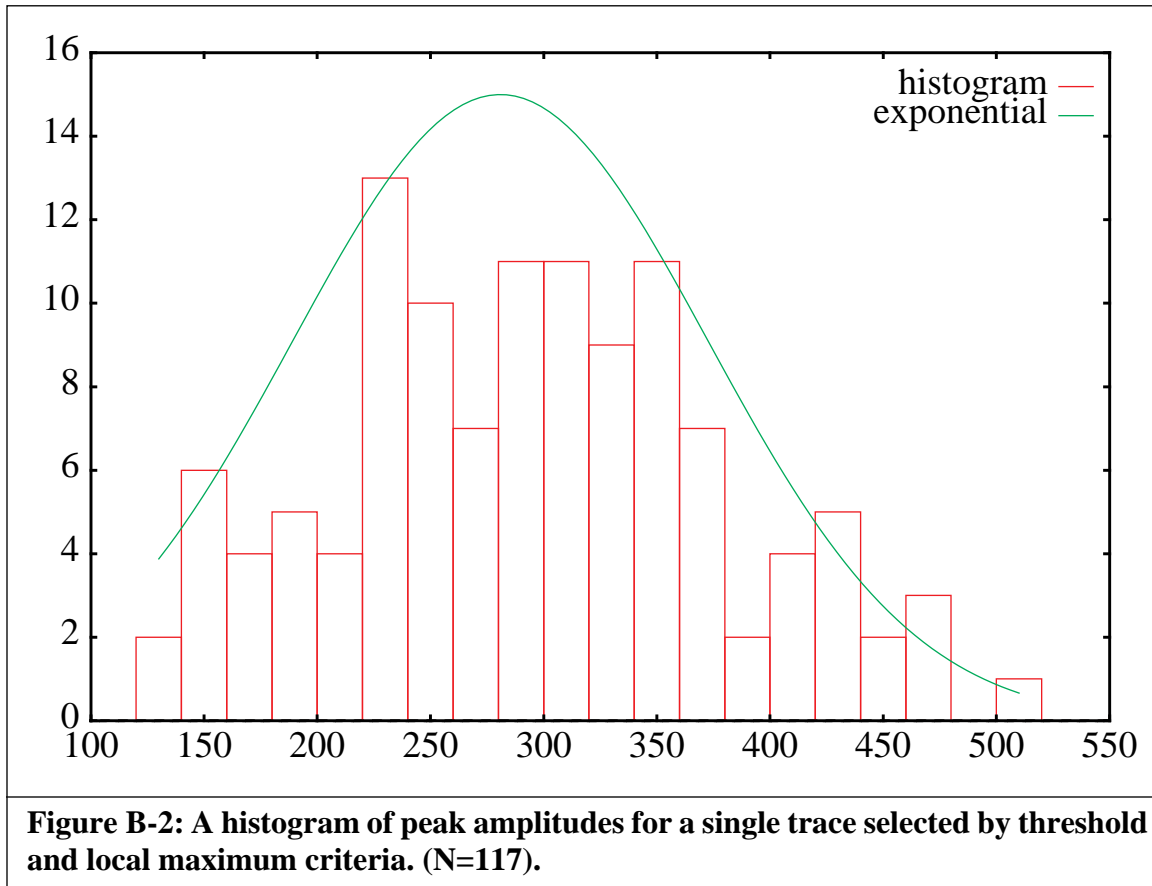


Figure B-2 shows a histogram of peak amplitudes after pruning. Although the fit of a Gaussian distribution to the data are marginal for the small sample set, the model is believed to be an accurate approximation until other models can be supported by an understanding of the biological mechanisms upon which peak amplitude depends.



The fluorescence signal from the sequencer during the void time before DNA reaches the detector is shown in Figure B-3a. A histogram of one channel of fluorescence and a Gaussian distribution with identical mean and variance is shown in Figure B-3b and demonstrates that the assumption of Gaussian noise distribution is reasonable.

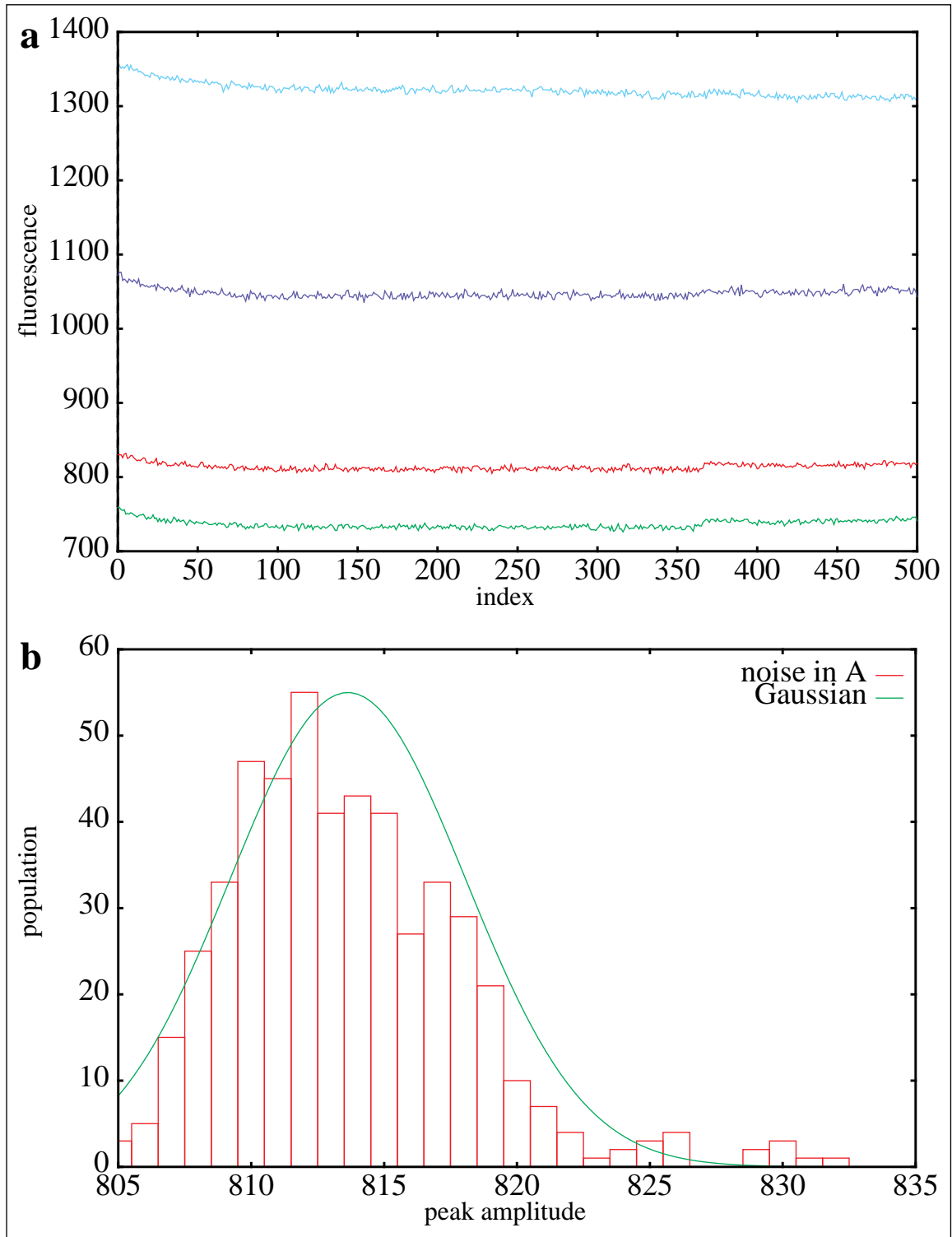


Figure B-3: (a) Four channels of raw fluorescence in the void region before valid fluorescence. (b) A histogram of the noise from channel A in the top panel.

There are several factors which contribute to the amplitude of fluorescence. As fragments migrate through a gel, the effects of increasing peak width and decreasing velocity vary linearly with length (5) and approximately negate one another.

The multiplicative effects of the probability of the addition of a nucleotide to a replicating strand. That is, if the probability of nucleotide incorporation is x , then the probability of the incorporation of i nucleotides is x^i . The fluorescence amplitude is dependent upon the number of size i bases, and is proportional to x^i . Although the probability of incorporation of dideoxynucleotides may be different than that for deoxynucleotides, only one is incorporated in any case and the effects will still be proportional to x^i . The net effect of these three factors is a decrease of peak amplitude which is expected approximate exponential decay as shown in Figure B-4.

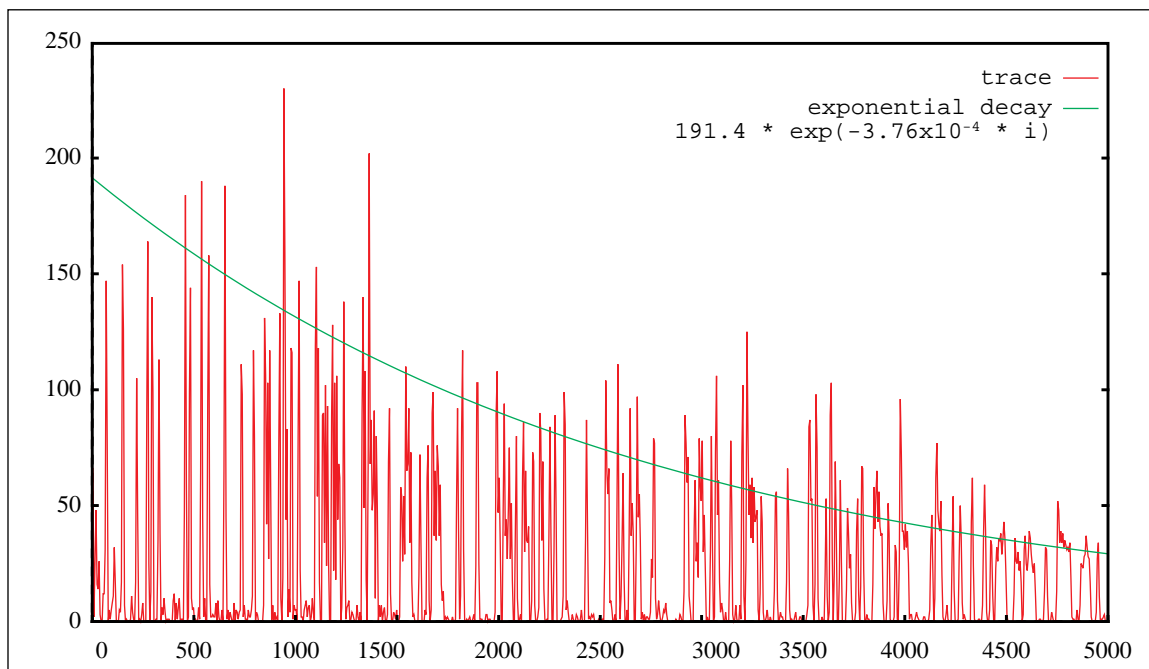


Figure B-4: A chromatogram of unprocessed data collected on an ABI automated DNA sequencer. The decay of mean peak amplitude was modeled with an exponential decay fit to peaks which had been selected by a combination of threshold and local maximum criteria. The method used to fit the exponential decay is discussed in 4. C++ CLASS LIBRARY & IMPLEMENTATION OF THE PEAK MODEL.

APPENDIX C: AUTOSEQ USERS' GUIDE

autoseq was written to act as interface to the classes created as a part of this thesis. It performs no data analysis or manipulation itself; instead, it directs the appropriate classes to perform the calculations described in this thesis. In addition to this appendix, a UNIX manual page is available with the source code. (See **APPENDIX D:AVAILABILITY**.)

The general form for the command line invocation is:

```
% autoseq [flags] [filenames]
```

The flags, associated arguments, and default parameters recognized by *autoseq* are enumerated in Table C-1. All flags and arguments must be separated by a space. The hyphen preceding a flag may be omitted in the case where the previous flag expects no argument or the optional argument is provided (i.e., bl). *autoseq* will parse flags until the first unrecognized flag, which is then assumed to be the first filename in *filenames*.

All results from *autoseq* operations are written to output files whose names are generated by appending a suffix to the input filename. The user may specify an alternate prefix with the -o flag. If *filenames* is omitted, then 'tracefile' is assumed. Peaks will be picked only if necessary for the requested operations. If any operation which generates output is specified then all default operations are reset and only those operations explicitly requested are performed.

Table C-1: *autoseq* command-line arguments, syntax, and description

flag	argument type	Description [output filename]
b	{A,C,G,T}	Specifies to which bases individual actions should be performed. More than one base may be specified. The default is -bACGT (all bases).
bl	short	Set the baseline of the selected traces to bl. If bl is omitted then each trace is translated downward by the minimum value for that trace.
bpd	positive integer	For every subsequence of length nbhd in the predicted sequence, write the index of the 3' most peak center for that subsequence, the delta for the subsequence, and the subsequence itself. The delta between bases m and n is defined to be the distance between the centers of the peaks corresponding to bases m and n.
bp		Output the base-to-trace-position mapping stored in file [prefix.bp]
d1		Compute 1st derivative trace [prefix.#.d1]
d2		Compute 2nd derivative trace [prefix.#.d2]
fmt	{ABI0,ABI1,ABI,SCF}	Read in specified format; guess if not specified. ABI0 - ABI (raw data) ABI1 - ABI (2nd trace set) ABI - ABI (processed data) SCF - Standard Chromatogram Format
fs		Write the sequence predicted stored in the file. [prefix.fseq]
ft		Writes the trace of expected fluorescence from the list of selected peaks. Expected fluorescence is calculated by fitting a Gaussian at the peak center using the peak amplitude & width; see fw. [prefix.#.ft]
fw	positive short	Specifies the breadth of the Gaussian used to model the fluorescence of each peak (i.e., Gaussian limits are [peakcenter - fw, peakcenter + fw]).
h		Include descriptive headers in output files
help		This help display
i		Write the individual traces after translation (bl), scaling (s), smoothing (sm), and transformation (x). [prefix.#]
l	positive integer or 'p'	Specifies the left cutoff index. Peak positions are reported relative to the left cutoff. If 'p' is passed as the argument, the left cutoff is set to the primer position.

Table C-1: *autoseq* command-line arguments, syntax, and description

flag	argument type	Description [output filename]
o	string	Specifies file prefixes for output filenames. If the prefix is a file, suffixes will be added as appropriate. If the prefix is a directory (that is, ends in a '/'), files will be redirected to that directory and the input filename will be used as the filename prefix. The hash symbol ('#') may be used a placeholder for the base identifier; if it is omitted, .# will be appended to the prefix specified with this flag.
pmc	positive double	Specifies the minimum peak amplitude as a product of the mean and the Peak Mean Coefficient (PMC). For instance, if the mean trace value is 20, then -a 1.2 will only pick peaks above $24 = (1.2 * 20)$.
p	integer	Prunes peaks by doing a pairwise comparison of adjacent peaks and discarding the less-probably peak of each pair which is separated by less than separation. If os is specified, a list of pairwise comparisons which resulted in the removal of a peak is output. [prefix.pruned]
ps		Write the predicted sequence (the sequence chosen by <i>CPeakList</i>)
pt		Generates traces which represent peaks and their widths & amplitudes. Peak traces are especially informative when overlaid with the individual (i) trace. [prefix.#.pt]
q		Quiet mode. Only errors will be displayed.
r	integer	Specifies the right cutoff index. If index <0, the right cutoff will be set to +index from the end of the trace. See l.
rt		Output the residual trace computed by subtracting the expected fluorescence trace from the observed data [prefix.#.rt]
s	string	Scale traces using scales from filename (in ACGT order), or default scales if not specified.
sm	short	Smooths by using a weighted average of the 3 points about a particular index, with the special case of the endpoints handled by throwing out the third point and normalizing the weighting coefficients. The process is repeated iterations times.
tfs		Output a summary of tracefile statistics and peaks [prefix.tfs]
ts		Output a summary of individual trace statistics and peaks [prefix.#.ts]
version		Print version info
v		(extremely) Verbose mode

Table C-1: *autoseq* command-line arguments, syntax, and description

flag	argument type	Description [output filename]
x	string	<p>Applies a 4x4 transformation/orthogonalization matrix to the 4 traces, producing a new set of traces which replaces the existing set. The general equation of the transformation is:</p> $R = MO$ <p>Transformation matrix are expected to be 4x4 matrix of the form: (file consists of mTS values only)</p> $M = (m_{TS}) = \begin{bmatrix} m_{AA} & m_{AC} & m_{AG} & m_{AT} \\ m_{CA} & m_{CC} & m_{CG} & m_{CT} \\ m_{GA} & m_{GC} & m_{GG} & m_{GT} \\ m_{TA} & m_{TC} & m_{TG} & m_{TT} \end{bmatrix}$ <p>or, equivalently:</p> $R_T(i) = \sum_{S \in \{A, C, G, T\}} m_{TS} O_S(i)$ <p>where R is the resulting vector of 4 traces O is the original vector of 4 traces M is the 4x4 ($\{ACGT\} \times \{ACGT\}$) matrix whose elements m_{TS} are the cross-term contributions of channel S to channel T S & T are trace identifiers (Source & Target) in $\{A, C, G, T\}$ i loops over the indices of the trace</p>
z	positive double	Specifies the epsilon about zero in which derivatives are considered to be exactly zero, and thus the crest (trough) of a local maxima (minima).

Examples of *autoseq* Command Line Invocation

1. `% autoseq -i tracefile`

Writes the individual traces stored in *tracefile* to the files *tracefile.A*,

tracefile.C, *tracefile.G*, and *tracefile.T*. The format of *tracefile* is guessed.

```
2. % autoseq -i -o '#' -bl -sm 1 -x my_matrix
```

Translates each trace in *tracefile* by the minimum value for that trace, smooths the data once, transforms it by the matrix specified in the file *my_matrix* (see *-x* flag for format), and writes the traces to files *A*, *C*, *G*, and *T*.

Note that *#* may require quoting to prevent interpretation by the shell.

```
3. % autoseq fmt ABI0 1 p r -100 sm -bl -p 4 x mtx i pt ps fs tfs o '#x'
tracefile
```

autoseq will attempt to read the first data set of the ABI file *tracefile*. If successful, each trace is clipped to the range [primer,100 from end], translated by the minimum value in that trace for that range, smoothed the default number of times (2), and transformed by the matrix mTS. Peaks are picked automatically (as required for options *pt*, *ps*, *fs*, *bpd*, and *tfs*) and pruned to remove peaks separated by less than 4 sample points. Finally, the output files are written: individual traces to *Ax*, *Cx*, *Gx*, *Tx*; peak traces to *Ax.pt*, *Cx.pt*, *Gx.pt*, *Tx.pt*; predicted sequence to *x.pseq*; file sequence to *x.fseq*; base position deltas for $n \leq 2$ to *x.bpdn*; and the tracefile summary to *x.tfs*. Note that the hyphen is required with the *bl* and *p* options because the preceding flags take optional arguments.

APPENDIX D: AVAILABILITY

The source code described herein is public domain. The source code, sample data, orthogonalization matrices, execution scripts, this thesis, and PGP signatures for these files are available by anonymous ftp to `ibc.wustl.edu` in the directory `/pub/c++-tools`. World Wide Web (i.e., NCSA Mosaic) users may use the Uniform Resource Locator *<http://ibc.wustl.edu:70/1/c++-tools/>*.

APPENDIX E: CLASS LIBRARY DECLARATIONS

```

// =====
// CTrace.H                                     80 columns
// Reece Hart (reece@ibc.wustl.edu)             tab=4 spaces
// Washington University School of Medicine, St. Louis, Missouri
// This source is hereby released to the public domain. Bug reports, code
// contributions, and suggestions are appreciated (to email address above).
// - - - - -
// CTrace is a C++ template designed to store a sequence of numbers and perform
// simple operations on it. It was originally designed for chromatogram data
// from automated DNA sequencing projects.
//
// It was designed as a template for several reasons, but the most practical
// advantage is that we can generate a CTrace of doubles for the derivatives,
// even in cases where the original traces are shorts (for example). One may
// then take the derivative of that (ie. the second derivative) by an identical
// call. The behavior of using CTrace to store anything other than numbers is
// undefined at best, and at worst may subject you to immense ridicule from
// your peers (unless it works to solve the solution to the universe.)
//
// CTrace maintains a pointer to an array of the parameterized type. There are
// three ways to get data into this class:
// 1. read the data yourself and pass the pointer to CTrace; don't forget
//    to use the Size(ulong) method to specify the size. (This is space
//    that you have allocated; do not pass temporary space to CTrace.)
//    You should then call CalculateStats to complete the installation.
// 2. open a file which contains a block dump of the specified type and
//    pass the FILE* and size to read to ReadTrace. (CTrace allocates the
//    space.) By block dump, I mean something written with the equivalent
//    of fwrite(YourDataPtr,ElementSize,#ofElements,FILE*).
// 3. open an ifstream and pass it and the size to ReadAsText. (CTrace
//    will allocate the space.)
// Data can also be written using the WriteTrace and WriteAsText methods.
//
// NOTES
// ! The destructor /always/ attempts to free space pointed to by the trace
// data member. Set the trace pointer to NULL with Trace(NULL) if you
// wish to assume responsibility for disposal of the space.
// + Statistics are automatically recalculated when necessary (ie. after
// setting the left clipping).
// * Derivatives of an entire trace contain n-2 points; however, derivatives
// of clipped traces contain the full trace size.
//
// SLATED IMPROVEMENTS
// * better error mechanism (current method dirties namespace).
//   - exceptions?
//   - conserve global namespace by hiding error codes
// * should ensure that peak list is empty before picking
// * stats dirty flag in lieu of automatic recal? (maybe faster...)
// * derivative size inconsistency should be fixed. If uncalculatable deriv
// values are filled, then it's wrong; but, we're forced to unnecessarily
// toss deriv values in the clipped trace just to maintain consistency.
// peakpicking may have to be modified if deriv numbering is changed.
//
// MODIFICATION HISTORY
// 93.06.30 Reece Original coding
// 93.07.29-31 Reece Added Derivative, ReadAsText, WriteAsText methods
// Converted to template, *AsText methods now use fstreams,
// [] operator for trace access (returns lvalue).
// 93.11.11 Reece First release
// =====|=====

#ifndef _H_CTrace // include this file only once
#define _H_CTrace

#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include "CSequence.H"
#include "CPeakList.H"
#include "DNA.H"
#include "RInclude.H" // homegrown definitions
#include "RInlines.H"

```

```

#include    "Definitions.H"

enum CError
{
    noError,                // no error occurred
    traceEmpty,            // no trace to do that on
    traceNotEmpty,        // can't overwrite trace
    badFile,               // bogus file
    memError,              // memory error occurred
    ioError,                // i/o error occurred
    rangeError,           // something's out of bounds
    analysisError         // couldn't analyze peaks
};

template<class T>
class CTrace
{
//private:                // see note with Version(), below
//static vrsn    version;

//
// Instance variables
//
//
CTError    error_flag;    // error flag
T*         trace;        // pointer to trace
size_t     size;         // number of trace points
double     scale;        // scale of trace
double     mean;         // mean trace value
T          max;          // max trace value
T          min;          // min trace value
double     variance;    // variance of trace values
T          baseline;    // baseline correction
CPeakList  peaks;        // list of peaks
tracepos   leftCutoff;   // ignore indices before leftCutoff
tracepos   rightCutoff;  // and after rightCutoff (0 based)

//
// Methods
//
public:
//vrsn&         Version();    // return class version (broken)
//              // there's a prize for figuring out
//              // why this works for other classes
//              // but not here...
CTError    Error(void);     // get and ...
// inline void Error(CTError new_error=noError); // set/clear error
// inline void Error(CTError new_error); // set/clear error

//              CTrace(size_t sz=0);    // constructor, allocate space
//              CTrace(size_t sz);    // constructor, allocate space
//              ~CTrace(void);        // destructor

CTError    AllocateTrace(size_t);    // allocate trace, update size, etc
inline void DeallocateTrace(void);    // deallocate & update size, etc.

inline size_t Size(void);             // get and ...
inline void Size(size_t new_size);    // set trace size
inline T* Trace(void);               // get and ...
inline void Trace(T* tp);            // set trace pointer (see NOTES)
inline CPeakList& Peaks(void);       // return reference to peak list

inline T Min(void);                  // access functions for min
inline T Max(void);                  // max,
inline double Mean(void);            // mean, and
inline double Variance(void);        // variance of trace values
CTError CalculateStats(void);        // compute min,max,mean,variance

inline tracepos LeftCutoff(void);    // get and ...
inline void LeftCutoff(tracepos t);  // set left cutoff

inline tracepos RightCutoff(void);   // get and ...
inline void RightCutoff(tracepos t); // set right cutoff

inline T& operator[](tracepos index); // trace value by index

inline void Baseline(T bl);          // get and ...

```

```

inline T      Baseline(void);          // set baseline
inline double Scale(void);            // get trace scale
void         Scale(double scale);     // scale trace; ie. 0.5 = scale 50%
void         Translate(T bl);         // add bl to each element

CTError      Smooth(void);
// Smooths by using a weighted average of the 3 points about a
// particular index, with the special case of the endpoints
// handled by throwing out the third point and normalizing the
// weighting coefficients.

CTError      Derivative(CTrace<double>** deriv);
// Compute the derivative of the trace and return it in a
// CTrace<double>. deriv will be allocated automatically.
// If deriv can be allocated, the derivative will be generated
// and noError returned; otherwise, an appropriate error is
// returned.

CTError      PickPeakIndices(
    T          MinPeakHeight,
    double     ZeroThreshold,
    CSequence<tracepos>** peaks);
// Picks local maxima>MinPeakHeight by a concave down method.
// Returns in a CSequence<tracepos> all indices, i, for which
// 1) the derivative at point i is within +/- ZeroThreshold or
// 2) derivative at the point i-1 is positive and the derivative
// at i+1 is negative (note that this implies the second
// derivative is negative).

CTError      PickPeaks(
    T          MinPeakHeight,
    double     ZeroThreshold);
// Calls PickPeakIndices to get a sequence of tracepos. This
// sequence is converted into a sequence of peak records and
// the bounds and width of the peak are computed. The arguments
// have the same meaning as those for PickPeaks.

CTError      PeakBounds(
    tracepos   CenterOfSearch,
    T          Elevation,
    double&    LeftIntersection,
    double&    RightIntersection,
    tracepos   MaxWidth = 0);
// tracepos   MaxWidth);
// Determines the left and right bounds of a concave-down peak
// at Elevation by searching laterally no more than MaxWidth on
// either side of CenterOfSearch (ie. [COS-MW,COS+MW]). If no
// data point exactly equals Elevation, the approximate bound is
// determined by linear interpolation. The resulting left and
// right bounds are returned in Left~ & RightIntersection.
// MaxWidth=0 (default) specifies no limits (ie. search is
// exhaustive, limited only by limits of trace). If the search
// bounds [COS-MW,COS+MW] exceed the bounds of the trace, they
// are automatically truncated to the nearest limit.

// I/O
CTError      WriteAsText(ofstream& os);
// Writes trace from a stream as a return-delimited list of
// points. All numerical data types are supported.
CTError      ReadAsText(ifstream& is, size_t);
// Read analog of ReadAsText.

CTError      WriteTrace(FILE*);
// Writes trace as a block of numbers. The good news is that
// it's extremely fast; the bad news is that the file is
// essentially a dump of the internal representation of the
// numerical array and is difficult (for humans) to interpret.
CTError      ReadTrace(FILE*, size_t);
// The read analog to the above.

friend ostream& operator<<(ostream& os, CTrace<T>& t);
// Writes a user-friendly summary of CTrace data members and
// the list of picked peaks if there are any.
};

//

```

```

//  INLINE FUNCTION DEFINITIONS
//  =====
//

//template<class T>
//inline
//vrsn
//CTrace<T>::Version()
// {
//   return version;
// }

template<class T>
inline
void
CTrace<T>::DeallocateTrace(void)
{
    delete trace; size = 0; min = max = 0; mean = 0;
}

template<class T>
inline
CTError
CTrace<T>::Error(void)
{
    return error_flag;
}

template<class T>
inline
void
CTrace<T>::Error(CTError new_error)
{
    error_flag = new_error;
}

template<class T>
inline
double
CTrace<T>::Mean(void)
{
    return mean;
}

template<class T>
inline
double
CTrace<T>::Variance(void)
{
    return variance;
}

template<class T>
inline
T
CTrace<T>:: Max(void)
{
    return max;
}

template<class T>
inline
T
CTrace<T>:: Min(void)
{
    return min;
}

template<class T>
inline
T&
CTrace<T>::operator[](tracepos index)
{
    return trace[index];
}

template<class T>

```

```

inline
void
CTrace<T>::LeftCutoff(tracepos t)
{
    if ((t>=0) && (t<=size-1))
    {
        leftCutoff = t;
        CalculateStats();
    }
}

template<class T>
inline
void
CTrace<T>::RightCutoff(tracepos t)
{
    if ((t>=0) && (t<=size-1))
    {
        rightCutoff = t;
        CalculateStats();
    }
}

template<class T>
inline
CPeakList&
CTrace<T>::Peaks(void)
{
    return peaks;
}

template<class T>
inline
size_t
CTrace<T>::Size(void)
{
    return size;
}

template<class T>
inline
void
CTrace<T>::Size(size_t new_size)
{
    size = new_size;
}

template<class T>
inline
T*
CTrace<T>::Trace(void)
{
    return trace;
}

template<class T>
inline
void
CTrace<T>::Trace(T* tp)
{
    trace = tp;
}

template<class T>
inline
double
CTrace<T>::Scale(void)
{
    return scale;
}

template<class T>
inline
void
CTrace<T>::Baseline(T bl)
{
    baseline = bl;
}

```

```

    }

template<class T>
inline
T
CTrace<T>:: Baseline(void)
{
    return baseline;
}

template<class T>
inline
tracepos
CTrace<T>::LeftCutoff(void)
{
    return leftCutoff;
}

template<class T>
inline
tracepos
CTrace<T>::RightCutoff(void)
{
    return rightCutoff;
}

template<class T>
ostream&
operator<<(ostream& os, CTrace<T>& t)
{
    tracepos length = t.rightCutoff-t.leftCutoff+1;
    os << "trace size:\t"
      << t.Size() << endl
      << "Window:\t\t"
      << "[" << t.leftCutoff << "," << t.rightCutoff << "]"
      << " (" << length << " point" << Plural(length) << ")" << endl
      << "min/max/mean/var: "
      << t.Min() << "/"
      << t.Max() << "/"
      << t.Mean() << "/"
      << t.Variance() << endl;
    return os;
}

#endif // conditional inclusion

```

```

// =====
// CTraceFile.H                                80 columns
// Reece Hart (reece@ibc.wustl.edu)           tab=4 spaces
// Washington University School of Medicine, St. Louis, Missouri
// This source is hereby released to the public domain. Bug reports, code
// contributions, and suggestions are appreciated (to email address above).
// - - - - -
// CTraceFile represents chromatogram data from automated DNA sequencers. It
// currently supports reading and writing files in ABI and SCF formats.
// Thanks to LaDeana Hillier for providing the code from which these file
// formats were inferred.
//
// NOTES
// ! This source makes no attempt to accommodate machine independent I/O
// because it was more important to tackle other issues first. Nonetheless
// I have tried to implement most I/O functions in a way that will make
// migration to machine-independent I/O.
// ? bottom flag is unimplemented. the seqIO implementation of reading
// each trace in reverse for bottoms is, ahem, interesting. I'd suggest
// using the template function Invert in RInlines to invert both sequences
// and traces.
// * CTraceFile is generic in that it can represent several file formats.
// Some fields of the abi format have been omitted and are copied into the
// comments field during reading. Before this class can correctly write
// abi files, the field integrity must be maintained. This may be done
// by (1) making a derived class which declares these members and
// appropriate read & write methods, or (2) make this class a superset of
// all file formats.
// * SCF doesn't currently read or write comments
//
// MODIFICATION HISTORY
// 93.11.11 Reece First release
// =====|=====

#ifndef _H_CTraceFile // include this file only once
#define _H_CTraceFile

#include <stddef.h>
#include "CTrace.H"
#include "CPeakList.H"
#include "DNA.H"
#include "FileFormat.H"
#include "RInclude.H"

typedef double xform_mtx[NUM_BASES][NUM_BASES]; // See Transform(), below

static const double MAX_SCF_TRACE_VALUE = 255.0;

class CTraceFile
{
public:
    enum strand_t
    { top, bottom };

    enum error_t
    {
        noError, // no error occurred
        memError, // memory couldn't be allocated
        ioError, // file/strm access error
        fileExistsError, // file already exists
        fileDoesntExistError, // no such file
        emptyError, // I'm empty and can't do that
        fmtError, // I don't understand data format
        unkFmtError, // can't figure out the format
        fmtNotSuppError, // format not yet supported
        peakPickError // error in peakpicking
    };

    //
    // Instance variables
    //
private:
    static vrsn version;
    error_t error; // error flag

    char* filename; // filename
    format_t nativeFormat; // format used to read this data

```

```

strand_t    whichStrand;           // which strand are we looking at?

CTrace<trace_t> trace[4];          // the traces
size_t      numPoints;            // # of points in traces
tracepos    leftCutoff;           // left cutoff
tracepos    rightCutoff;          // right cutoff
tracepos    primerPosition;       // start of primer
CPeakList   peaks;                // assimilated (ACGT) list of peaks
trace_t     min;                  // min,
trace_t     max;                  // max,
trace_t     mean;                 // & mean for the 4 traces

size_t      numBases;             // # of bases
char*       sequence;             // the bases as called by mfr
tracepos*   basePositions;        // base # <-> trace point # rel'n

char*       comments;            // miscellaneous comments

//
// Methods
//
public:
vrsn&       Version();            // return class version
error_t     Error(void);          //
void        Error(error_t new_error); // set/clear error

          CTraceFile(            // constructor
            const char* fn=NULL,
            format_t fmt=unknown);
          ~CTraceFile(void);      // destructor

error_t     Allocate(             // allocate traces,
                size_t points,    // base positions,
                size_t bases,     // and comment string
                size_t comment);
void        Deallocate(void);     // deallocate space

void        NumPoints(size_t np); // set and...
size_t      NumPoints(void);      // get # of points in traces
trace_t     Mean(void);           // get mean,
trace_t     Min(void);            // min, &
trace_t     Max(void);            // max values over all 4 traces
void        CalculateStats(void); // calc. min/max/mean for all traces

CPeakList&  Peaks(void);          // get the list of peaks
size_t      NumPeaks(void);       // # of peaks picked
void        NumBases(size_t nb);  // set and...
size_t      NumBases(void);       // get the number of bases
void        Sequence(char* seq);  // set and...
char*       Sequence(void);       // get the sequence
void        Comments(char* newComment); // set and...
char*       Comments(void);       // get the comment

CTrace<trace_t>*
          operator[](enum_t whichTrace); // trace selector

tracepos    LeftCutoff(void);     // get and...
void        LeftCutoff(tracepos t); // set left cutoff
tracepos    RightCutoff(void);    // get and...
void        RightCutoff(tracepos t); // set right cutoff

tracepos    PrimerPosition(void); // return the position of the
// primer (if stored in file)

tracepos*   BasePositions(void);  // return base positions array

error_t     Transform(xform_mtx& matrix);
// Applies a 4x4 transformation/orthogonalization matrix to the
// 4 traces, producing a new set of traces which replaces the
// existing set.
// Transformation matrices are expected to be 4x4 matrix of the
// form:
//      M =
//      [A,] [A] [C] [G] [T]
//      [A,] mAA mAC mAG mAT
//      [C,] mCA mCC mCG mCT
//      [G,] mGA mGC mGG mGT
//      [T,] mTA mTC mTG mTT

```

```

// The general equation for the resulting traces is:
//  $R = M O \Leftrightarrow R(T,i) = \sum_{S \in \{ACGT\}} [ mTS \times O(S,i) ]$ 
// where R is the resulting vector of 4 traces
// O is the original vector of 4 traces
// M is the 4x4 ({ACGT}x{ACGT}) matrix whose elements
// m(i,j) are the cross-term contributions of channel j
// to channel i
// S & T are trace identifiers (Source & Target)
// in {A,C,G,T}
// i loops over the indices of the trace

error_t PickPeaks(
    double PeakMeanCoefficient = 1.5,
    double ZeroThreshold = 0.0);
// Calls PickPeaks on each trace in the tracefile, then
// assimilates the peaks into a list of peaks for the entire
// set sorted by index in trace. No pruning is performed.

error_t AssimilatePeaks(void);
// Performs a merge sort of the list of peaks stored in each
// trace's peak list and puts the result in this tracefile's
// own (assimilated) peak list.

error_t PrunePeaks(
    tracepos minSeparation,
    ostream* os=NULL);
// Prunes peaks by doing a pairwise comparison of adjacent peaks
// and discarding the less-probably peak of each pair which is
// separated by less than minSeparation. If os is specified,
// a list of pairwise comparisons which resulted in the
// removal of a peak is output.

//
// I/O methods
//
error_t Read(
    const char* filename,
    format_t fmt=unknown);
// Top-level read routine. Attempts to read the named file
// in the specified format, or in the format guessed by
// FileFormat() (see FileFormat.H).

error_t Write(
    const char* filename,
    format_t fmt=unknown);
// Top-level write routine. Attempts to write the named file
// in the specified format, or in the native format if
// unspecified.

private:
// The following are private I/O methods. Users should not
// need access to any of these.
error_t Read(
    FILE* fp,
    format_t fmt);
error_t Write(
    FILE* fp,
    format_t fmt);
error_t ReadSCF(FILE* fp); // Read as SCF file
error_t WriteSCF(FILE* fp); // Write as SCF file
error_t ReadABI(FILE* fp, short whichSet=0); // Read as ABI file
error_t WriteABI(FILE* fp); // Write as ABI file
error_t ReadALF(FILE* fp); // Read as ALF file
error_t WriteALF(FILE* fp); // Write as ALF file

public:
friend ostream& operator<<(ostream& os, CTraceFile& ctf);
// Places an easy-to-read summary of the tracefile contents
// on the specified output stream.
};

//
// INLINE FUNCTION DEFINITIONS
// =====

```

```

//

inline
vrsn&
CTraceFile::Version()
{
    return version;
}

inline
void
CTraceFile::Error(error_t new_error)
{
    error = new_error;
}

inline
CTraceFile::error_t
CTraceFile::Error(void)
{
    return error;
}

inline
void
CTraceFile::NumPoints(size_t np)
{
    numPoints = np;
}

inline
size_t
CTraceFile::NumPoints(void)
{
    return numPoints;
}

inline
trace_t
CTraceFile::Mean(void)
{
    return mean;
}

inline
trace_t
CTraceFile::Min(void)
{
    return max;
}

inline
trace_t
CTraceFile::Max(void)
{
    return max;
}

inline
tracepos
CTraceFile::LeftCutoff(void)
{
    return leftCutoff;
}

inline
void
CTraceFile::LeftCutoff(tracepos t)
{
    leftCutoff = t;
    for(uint i=A;i<NUM_BASES;i++) trace[i]->LeftCutoff(t);
    CalculateStats();
}

inline
tracepos
CTraceFile::RightCutoff(void)
{
    return rightCutoff;
}

```

```

inline
void
CTraceFile::RightCutoff(tracepos t)
{
    rightCutoff = t;
    for(uint i=A;i<NUM_BASES;i++) trace[i]->RightCutoff(t);
    CalculateStats();
}

inline
tracepos
CTraceFile::PrimerPosition(void)
{
    return primerPosition;
}

inline
CPeakList&
CTraceFile::Peaks(void)
{
    return peaks;
}

inline
size_t
CTraceFile::NumPeaks(void)
{
    return peaks.Size();
}

inline
CTrace<trace_t>*
CTraceFile::operator[](enum_t whichTrace)
{
    return trace[whichTrace];
}

inline
void
CTraceFile::NumBases(size_t nb)
{
    numBases = nb;
}

inline
size_t
CTraceFile::NumBases(void)
{
    return numBases;
}

inline
void
CTraceFile::Sequence(char* seq)
{
    sequence = seq;
}

inline
char*
CTraceFile::Sequence(void)
{
    return sequence;
}

inline
void
CTraceFile::Comments(char* newComment)
{
    comments = newComment;
}

inline
char*
CTraceFile::Comments(void)
{
    return comments;
}

```

```
    }  
    inline  
    tracepos*  
    CTraceFile::BasePositions(void)  
    {  
        return basePositions;  
    }  
#endif
```

```

// =====
// CPeakList.H                                     80 columns
// Reece Hart (reece@ibc.wustl.edu)                tab=4 spaces
// Washington University School of Medicine, St. Louis, Missouri
// This source is hereby released to the public domain. Bug reports, code
// contributions, and suggestions are appreciated (to email address above).
// - - - - -
// This class represents a sequence of peaks picked by CTrace and is used by
// both CTrace and CTraceFile. It uses CSequence as a base class, but adds
// many application-specific variables and methods which accommodate
// statistical analysis of the peaks.
//
// Specifically, this source collects a list of peaks and computes the
// following:
// 1. the equation for the exponential decay of peak heights with
//    increasing index.
// 2. the equation for the linear increase in peak widths with increasing
//    index.
// 3. the expected fluorescence trace by modeling the fluorescence of each
//    peak as a Gaussian
// 4. the peak trace, which qualitatively shows the peak height and width
//    and is useful for superimposing over the observed trace.
// 5. the residual trace, obtained by subtracting the expected fluorescence
//    from the observed fluorescence at each sample.
// 6. P(H|D) from P(H), P(D|NULL), P(D|H) (See ComputePeakStats)
//
// NOTES
// * This is the most volatile portion of this project. For that reason,
//   error handling is rudimentary. I haven't bothered to implement access
//   functions.
//
// MODIFICATION HISTORY
// 93.11.11 Reece First release
// =====|=====

#ifndef _H_CPeakList
#define _H_CPeakList                                // include this file only once

#include <iostream.h>
#include <iomanip.h>
#include "CSequence.H"
#include "RInclude.H"
#include "DNA.H"
#include "Definitions.H"

template<class T>
class CTrace;                                     // forward declaration

struct PeakRec
{
    base_t      whichTrace;                       // which trace (A,C,G,T)
    tracepos    offset;                          // leftCutoff offset
    tracepos    center;                          // index of peak
    double      height;                          // value at index
    double      width;                           // width at 1/2 height
    double      leftBound;                       // left bound at 1/2 height
    double      rightBound;                      // right bound at 1/2 height
    double      PH;                              // P(H)
    double      PDN;                             // P(D|NULL)
    double      PDH;                             // P(D|H)
    double      PHD;                             // P(H|D) = P(D|H)*P(H)/P(D|NULL)

    PeakRec(void);                               // constructor

    friend
    ostream& operator<<(ostream& os, const PeakRec& pr);
    // Writes the instance of the peak object in a single line, 100
    // columns across. See PeakRecHeader just below class.

    inline friend
    int operator==(const PeakRec& r1, const PeakRec& r2);
    // Operator specifies conditions for two peaks to be considered
    // identical.
};

ostream& PeakRecHeader(ostream& os);
// ostream manipulator which writes a line of column headings

```

```

// for the << output operator.
// usage: myostream << PeakRecHeader;

class CPeakList : public CSequence<PeakRec>
{
private:
static vrsn version;

// inherit CSequence variables
public:
double      hmean;           // peak height mean
double      hvariance;      // and variance

double      hm0;            // height mean @ i=0
double      hmdecay;        // hm exp. decay

double      hv0;            // height variance @ i=0
double      hvdecay;        // hv exp. decay

double      w0;             // width modeled by least sq. fit
double      wconst;        // w(i)=w0 + wconst*i

private:
uint        group1_left;    // group 1 = first third
uint        group1_right;
tracepos    group1_leftidx;
tracepos    group1_rightidx;
double      group1_hmean;
double      group1_hvariance;
double      group1_wd_mean;
double      group1_index_mean;

uint        group2_left;    // group 2 = second third
uint        group2_right;
tracepos    group2_leftidx;
tracepos    group2_rightidx;
double      group2_hmean;
double      group2_wd_mean;
double      group2_hvariance;
double      group2_index_mean;

CTrace<double>* FTrace;     // computed fluorescence trace
CTrace<double>* PTrace;     // trace of peak widths/heights
CTrace<double>* RTrace;     // trace of residuals

public:
vrsn&       Version();      // return version # of class

CPeakList(void);           // constructor
~CPeakList(void);         // destructor

char*       Sequence(char* buf=NULL); // return sequence

inline
tracepos    Delta(ulong n, ushort nbhd=1);
// The delta between bases m and n is defined to be the distance
// between the centers of their corresponding peaks m and n in
// the trace. By default, it returns the delta for base n;
// specifying nbhd returns the delta for the nbhd 3'-most bases.

inline
void        RemovePeak(tracepos peakCenter);
// Removes a peak from a trace. For the time being, it does
// so only on the basis of peak center.

bool        Analyze(CTrace<trace_t>& ct); // Automates the following:
// Automates the process of computing peak probabilities from a
// peak list. It generates the fluorescence and residual traces
// and (permanently) stores the results in the FTrace and RTrace
// members.

void        CalculateStats(void); // compute peak statistics
// Computes statistics for the set of peaks in the list.
// Specifically, it computes:
// 1. the overall mean and variance of peak heights
// 2. the linear increase of peak width with index

```

```

// 3. the exponential equation decay of mean peak height decay

void CalculatePeakStats(double h_baseline);
// Computes the individual peak probabilities using a Gaussian
// model for the peak fluorescence, height distribution, and
// noise estimate, and Bayes' Theorem to compute the probability
// of a particular peak given the observations.

void WriteDeltas(ostream& os, uint nbhd, bool header=FALSE);
// For every subsequence of length nbhd, write the index of the
// 3' most peak center for that subsequence, the subsequence
// itself, and the delta for the subsequence. See Delta(...)
// If header is TRUE, a simple column header is written first.

void Offset(tracepos offset);
// Horizontally translates all references to trace indices by
// offset (offset added to each index). Useful for defining the
// zero point (ie. the primer position).

CTrace<double>*
ComputeFTrace(size_t sz, tracepos extent=50);
// Computes a trace which represents the expected fluorescence
// from the list of peaks by assuming a Gaussian distribution at
// each peak (using the center, height, and width fields of the
// PeakRec). The extent parameter specifies the horizontal
// extent of the Gaussian on each side of center. The size of
// the original trace must be passed so that the original and
// fluorescence traces will be the same size.

CTrace<double>*
ComputePTrace(CTrace<trace_t>& src_trace);
// Generates a trace which shows peak height and width for every
// peak in the peak list. The argument passed to ComputePTrace
// is the size of the original trace, which will also be the
// size of the peak trace. Peak traces are especially
// informative when overlaid with the trace from which the peaks
// were picked.

CTrace<double>*
ComputeRTrace(CTrace<trace_t>& src_trace);
// Generates a new trace of residuals (data that cannot be
// explained by peaks) by subtracting the computed FTrace from
// the original trace. A pointer to the result is stored in
// RTrace and returned (or NULL if error).

friend
ostream& operator<<(ostream& os, CPeakList& cpl);
// Display a summary of the peak list and it's members
};

inline
vrsn&
CPeakList::Version()
{
return version;
}

inline
void
CPeakList::RemovePeak(
tracepos peakCenter)
{
for(ulong index=0; index<size; index++)
if ((*this)[index].center == peakCenter)
{ Remove(index); break; }
}

inline
tracepos
CPeakList::Delta(ulong n, ushort nbhd)
{
if ((n-nbhd<0) || (n>size-1))
return 0;
else

```

```
        return (*this)[n].center-(*this)[n-nbhd].center;
    }

inline
int
operator==(
    const PeakRec& r1,
    const PeakRec& r2)
{ return (r1.center == r2.center); }

#endif // conditional inclusion
```

BIBLIOGRAPHY

1. Biochemistry, Third Edition, Stryer, L., Freeman & Company, p. 71.
2. Understanding Robust and Exploratory Data Analysis, David C. Hoaglin, Frederick Mosteller, & John Tukey, Editors, John Wiley & Sons, Inc. 1986
3. Bowling, J. M., Bruner, K. L., Cmarik, J. L., and Tibbets, C. (1991) *Nucleic Acids Res.* **19**, 3089-3097.
4. Gleeson, T. and Hillier, L. (1991) *Nuc. Acids Res.* **19**, 6481-6483.
5. Grossman, P.D., Mechen. S., and Hershey, D. (1992) *GATA* **9(1)**, 9-16.
6. Lipman, D.J. and Pearson, W.R.(1985) *Science* **227**,1435.
7. Pearson, W.R. and Lipman, W.J. (1988) *Proc. Natl. Acad. Sci. USA* **85**, 2444.
8. Sanger, F., and Coulson, A.R. (1975) *J. Mol. Bio.*, **94**, 441-448.
9. Smith, L. M., Sanders, J. Z., Kaiser, R. J., Hughes, P., Dodd, C., Connell, C. R., Heiner, C., Kent, S. B., and Hood, L. E. (1986) Fluorescence detection in automated DNA sequence analysis. *Nature* 321(6071):674-9.
10. Splus is a “programming environment for data analysis”. It is available from Statistical Sciences, Inc., 1700 Westlake Ave. N., Suit 500, Seattle, WA 98109 USA. Telephone: 206/283-8802. Fax: 206/283-8691. Internet: mktg@statsci.com

VITA

Name: Reece Kimball Hart

Date of Birth: 1968 November 22

Undergraduate Study: University of California, San Diego
B.A., Molecular Biology, 1990

Graduate Study: Washington University,
St. Louis, Missouri, 1991-present
M.S. (Computer Science) 1994
Ph.D. (Biophysics) expected 1997

Scholastic and Professional Experience: Teaching Assistant, 1993
Washington University
School of Medicine (St. Louis, MO)
Research Assistant, 1988-1991
The Salk Institute (La Jolla, CA)

Publications:
Eubanks JH. Selleri L. Hart R. Rosette C. Evans GA.; "Isolation, localization, and physical mapping of a highly polymorphic locus on human chromosome 11q13", *Genomics*, 11(3):720-9, 1991 Nov.

ACKNOWLEDGMENTS

I am grateful for the unwavering support from Larry and Roianne, my parents, and Brooke, my sister, in this and other endeavors. I also appreciate the support I've received from my good friends. Not a day passes in which I fail to recognize the importance of these people in my life.

I am indebted to Mark Frisse whose encouragement and friendship have been invaluable during my graduate training. This project and my graduate training were supported in part by grant number LM07049 awarded to Mark from the National Library of Medicine.

I appreciate the generous efforts of Pankaj Agarwal in proofreading and providing thoughtful criticism of this thesis. Thanks also to Kirsten Anderson for proofreading the section on DNA sequencing.

Thanks to LaDeana Hillier, Bob Waterston, and Rick Wilson and the *Caenorhabditis elegans* sequencing group for their provisions of raw sequencing data and source code from which I derived file I/O for ABI and SCF files.

Finally, thank you to Will Gillett, Philip Green, and David States for their guidance and patience throughout this project.

AUTOMATED DNA SEQUENCE ANALYSIS

HART, M.S., 1994