Optimization of the Speed Dependent Line Shape Calculation

A thesis submitted in partial fulfillment of the requirement for

the degree of Bachelor of Science with Honors in Physics

from the College of William and Mary in Virginia.

by

Kendra Letchworth

Accepted for: _____
(Honors)

_____
Advisor: Dr. D. Chris Benner

_____
Dr. Marc Sher

_____
Dr. Gina Hoatson

_____
Dr. Sarah Day (Mathematics)

Williamsburg, VA
May 2, 2007

Abstract

A new algorithm for computation of a speed dependent lineshape function is presented. The lineshape that is calculated is based upon a quadratic model for the Lorentz width as a function of velocity. The calculation includes both real and imaginary parts of the lineshape for applications to line mixing and Dicke narrowing. This algorithm sacrifices a small amount of memory space for a considerable gain in speed and accuracy and employs methods similar to the techniques used to calculate the Voigt Profile as described by Letchworth and Benner[1]. For Lorentz widths greater than about five times the Doppler width and for points more than about five Doppler widths from the center of the spectral line, Gauss-Hermite quadrature is employed. For most other cases, a Taylor series expansion about the nearest point in a precomputed table is used. In some cases where the Doppler width is more than an order of magnitude larger than the Lorentz width, Lagrange interpolating polynomials are used with a table of precomputed points. The accuracy is less than one part in 10^{-5} of the value of the function itself for the real part and most cases of the imaginary part. In a few rarely used instances, the relative accuracy of the imaginary part calculation is less than one part in 10^{-4} but the absolute accuracy remains less than one part in 10^{-5}. The algorithm provides a good approximation for speed dependent lineshapes and requires approximately the same calculation time as the Voigt Profile calculation using the Drayson[2], Humlíček[3], and Letchworth and Benner[1] routines.

Acknowledgements

First, I would like to acknowledge Dr. D. Chris Benner for his involvement as my research advisor for the past two years; his input and advice have been invaluable. In addition, I would like to thank Adriana Predoi-Cross from Lethbridge University for sharing with us the speed dependent model upon which this thesis is based and the code which provided a starting point for our algorithm. I am also grateful to Alan Pine of Alpine Technologies for his input regarding a variety of speed dependent models and for the internal documentation and code he provided. This material was based in part on work supported by the National Science Foundation under Grant No. ATM-0338475.

Table of Contents

I. Introduction

Molecular spectroscopy is used by scientists from virtually every discipline, for applications ranging from monitoring carbon dioxide levels in the atmosphere to detecting planets circling faraway stars. Technological improvements have created opportunities to map spectra with unprecedented resolution and signal to noise ratios. However, these recent advances have also increased the need for more accurate models to analyze the data. For many years, the Voigt profile, a convolution of the Lorentz profile, which models collisional broadening, and the Doppler profile, which models the broadening caused by the Doppler shift due to molecular motion, has been adequate for describing line shapes. Now, as data fits are reaching the level of tenths of a percent in accuracy, the Voigt profile needs to be modified. The Lorentz portion of the Voigt profile assumes that all collisions occur at the statistical average velocity; however, molecules actually have a distribution of speeds. The speed dependent model presented in this report addresses this discrepancy using an efficient approximation which does not require significantly more calculation time. In addition, the algorithm presented is valid for all physical values of the input parameters and is more useful in a broad context than those routines designed for a certain application.

II. Applications of Lineshape Calculations

Spectral lineshape functions have many applications within molecular spectroscopy as well as in many other subfields. Since the Voigt profile, the most well known spectral lineshape function, is merely a convolution of Gaussian and Lorentzian probability distributions, it is applicable to many problems. In optical spectroscopy and spectroscopy of stellar atmospheres, a Gaussian distribution describes the Doppler shifts due to thermal velocity and Lorentzians arise from atomic absorption and emission rates. Outside of spectroscopy, when determining the reaction rates of thermal neutrons in nuclear-fission reactors, the Gaussian distribution of neutron velocities is folded into a Lorentzian distribution describing neutron resonance reactions with reactor materials to create a Voigt profile[4]. In plasma physics, the Voigt function can be used to describe plasma temperature through the broadening of electromagnetic emission lines from hydrogenlike atoms in low density plasmas since it is essentially the plasma dispersion function [4]. Scientists in many of these disciplines would benefit from including speed dependent effects in their calculations.

Within the discipline of atmospheric science, in order to understand the effects of air pollution, ozone depletion, and global warming, accurate and efficient spectral analysis techniques are required. Satellites such as the Orbiting Carbon Observatory (OCO) will be providing new data which could be used to determine the levels of carbon dioxide in the atmosphere. Lineshape calculation routines must be able to extract all of the available information and spectral parameters from this data by including higher order effects such as line mixing and speed dependence but without sacrificing calculation speed [5]. Since data analysis routines should provide an accuracy level of a few times

less than the inverse of the signal to noise ratio, the accuracy of 10^{-5} provided by the routine in this paper is required in order to fully analyze spectra with signal to noise ratios of a few thousand to one. Signal to noise ratios of this quality are produced by tunable diode laser spectrometers and by Fourier transform spectrometers, such as the instrument at Kitt Peak National Observatory used by our laboratory to take carbon dioxide spectra under controlled laboratory conditions.

The increasing use of multispectrum least-squares fitting techniques [6] has also required that these analysis techniques must also be efficient. In a multispectrum fit, spectra taken under different experimental conditions are fit simultaneously; this technique allows the solutions to be more physical, since spectra under a variety of temperatures, pressures, and path lengths are fit with the same spectral parameters. However, a single multispectrum fit requires millions of lineshape calculations. To perform ten million calculations using a routine which takes one second per calculation, like some numerical integration routines, would take more than three months. If one calculation can be performed in a millisecond, ten million calculations would take about three hours. However, if one calculation can be performed in a microsecond, ten million calculations would take only 10 seconds. This example demonstrates why efficiency is absolutely essential in lineshape calculation routines.

III. Lineshape Functions

In molecular spectroscopy, optical transmission through a gas is defined as

$$T = \exp(SA\kappa(\nu)) \tag{1}$$

where S is the line strength, A is the abundance of the gas, and $\kappa(\nu)$ is the normalized lineshape function. The intensities of the spectral lines are given completely by the factor SA so the lineshape merely describes the shape of the line regardless of intensity. Thus, the lineshape $\kappa(\nu)$ obeys the normalization condition in Equation (2).

$$\int_{-\infty}^{\infty} \kappa(\nu) d\nu = 1 \tag{2}$$

The most basic line shape is the natural lineshape, arising because of uncertainty principle requirements on the frequency spread which can be observed due to the lifetime of the transition. The natural lineshape is not considered in this paper because the effects of Doppler and pressure broadening are so prounounced that the very narrow natural linewidth is unimportant for most applications.

The simplest line shape which is readily observable is the Doppler profile, which is due to thermal motion of the molecules, as shown in Figure 1. The Doppler shift is dependent upon the line of sight thermal velocity of the molecules v_T and the wavenumber of the line center ν_0. The number of molecules with velocity between v_T and $v_T + dv_T$ is provided by the Maxwell-Boltzmann distribution in Equation (3), which is a basic Gaussian lineshape dependent upon temperature and mass of the molecules. By combining the formula for Doppler shift with the Maxwell-Boltzmann distribution, one obtains the normalized Doppler profile in terms of the distance from line

center $\nu - \nu_0$, the temperature T, the mass m, Boltzmann's constant k_B, and the speed of light c [7].

$$G(\nu) = \frac{c}{\nu_0}\sqrt{\frac{m}{2\pi k_B T}}\exp\left(-\frac{mc^2(\nu-\nu_0)^2}{2k_B T\nu_0^2}\right) \tag{3}$$

The Doppler width α_D is the half width at half maximum of this profile and is defined by

$$\alpha_D = \frac{\nu_0}{c}\sqrt{\frac{2k_B T}{m}} \tag{4}$$

Therefore, the normalized Doppler profile in terms of the Doppler width is given by

$$\kappa_D(\nu) = \frac{\sqrt{\ln(2)}}{\alpha_D\sqrt{\pi}}\exp\left(-\ln(2)\left(\frac{\nu-\nu_0}{\alpha_D}\right)^2\right) \tag{5}$$

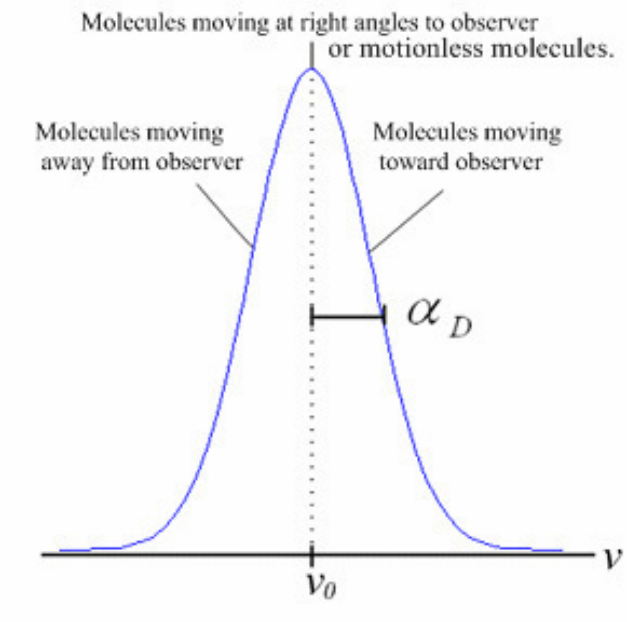

Figure 1: Contributions of molecular motion to the Doppler profile

The Lorentz profile is due to distortions of the energy levels during molecular collisions and the Lorentz halfwidth is dependent upon the time between molecular

collisions [7]. Unless a molecule absorbs or emits light, it remains at a given energy level until it collides with another molecule. The lifetime of that state is dependent upon the time between collisions, so if the time between collisions is short, the frequency spread must be broad by the uncertainty principle. Thus, Lorentz broadening is called pressure broadening because the time between collisions becomes short as the pressure increases. The Lorentz half-width at half-maximum, α_L, is inversely proportional to the time between collisions and can be modeled as [6]

$$\alpha_L = {\alpha_L}^0 P\left(\frac{T_0}{T}\right)^n \tag{6}$$

where ${\alpha_L}^0$ is the line width at temperature T_0, P is the gas pressure, and n is the temperature dependence of the half width. The lineshape for pressure broadening is given by [7].

$$\kappa_L(\alpha_L, \nu) = \frac{\alpha_L}{\pi} \frac{1}{(\nu - \nu_0)^2 + {\alpha_L}^2} \tag{7}$$

The Doppler profile and the Lorentz profile are shown together in Figure 2, along with the Voigt profile. In Figure 2 a), the Lorentz width is about half the Doppler width and in Figure 2 b), the Lorentz width is about twice the Doppler width. Figure 2 b), with the broader Lorentz profile could correspond to a higher pressure, but could also correspond to an intrinsically broader line or a lower temperature. If the Doppler width remains constant, the Doppler profile does not change shape due to variation in the Lorentz width since it is only dependent on the distance from line center.

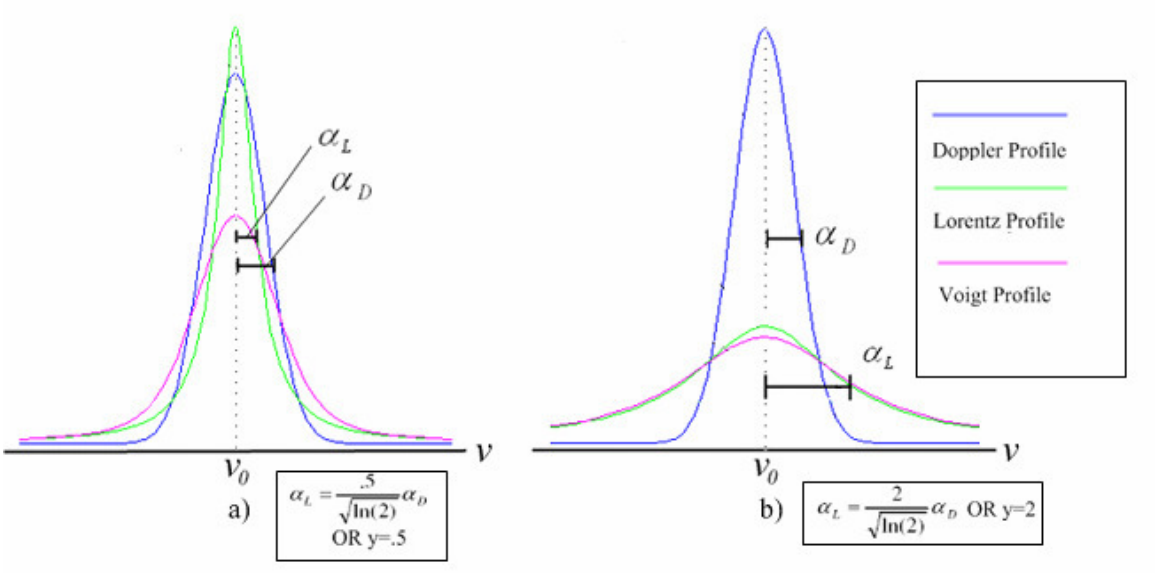

Figure 2: Effect of changing the Lorentz width on Voigt, Lorentz and Doppler profiles

This figure also shows that when both collisional broadening and thermal motion must be considered, both the Lorentz and Doppler profiles are inadequate; the Voigt profile is necessary.

To derive the Voigt profile, the Lorentz profile in Equation (7) and the Doppler profile in Equation (5) are convolved as shown in Equation (8). [4]

$$\kappa_V(\nu,\alpha_L) = \int_{-\infty}^{\infty} \kappa_D(\alpha_D,t)\kappa_L(\alpha_L,(\nu-\nu_0)-t)dt \tag{8}$$

The normalized Voigt profile [8] can then be expressed as

$$\kappa_V(x,y) = \frac{\sqrt{\ln(2)}}{\alpha_D\sqrt{\pi}}\mathrm{Re}(V(x,y)) = \frac{\sqrt{\ln(2)}}{\alpha_D\sqrt{\pi}}K(x,y) \tag{9}$$

where

$$x = \sqrt{\ln 2}(\nu-\nu_0)/\alpha_D \tag{10a}$$

$$y = \sqrt{\ln 2}(\alpha_L/\alpha_D) \tag{10b}$$

The parameter α_L is the Lorentz half-width of the line due to collisional broadening, $\nu - \nu_0$ is the distance from the line center, and α_D is the Doppler width of the line. Equation (9) is in terms of the real part of the complex Voigt function *V(x,y)* defined in Equation (11).

$$V(x, y) = K(x, y) + iL(x, y) \tag{11}$$

where the real and imaginary parts of the Voigt function are given by

$$K(x, y) = \frac{y}{\pi} \int_{-\infty}^{\infty} \frac{\exp(-t^2)}{y^2 + (x-t)^2} dt \tag{12}$$

$$L(x, y) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{(x-t)\exp(-t^2)}{y^2 + (x-t)^2} dt \tag{13}$$

It can be shown that as *x* and *y* approach infinity, *V(x,y)* approaches the function

$$V(x, y)_{x\to\infty, y\to\infty} = \frac{1}{\sqrt{\pi}} \left(\frac{y}{x^2 + y^2} + \frac{ix}{x^2 + y^2} \right) \tag{14}$$

The real part of equation (14) is equivalent to the normalized Lorentz profile. Conversely, as the parameter *y* approaches zero, the real part of the Voigt function approaches the Doppler profile. By applying a change of variables *u=x-t*, the complex Voigt function becomes

$$V(x, y) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{\exp(-x^2 - u^2)(y + iu)}{y^2 + u^2} \sinh(2xu) du \tag{15}$$

At the small *y* limit, the imaginary part of this function approaches

$$L(x, y)_{y\to 0} = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{\exp(-x^2 - u^2)}{u} \sinh(2xu) du \tag{16}$$

It requires a bit more equation manipulation to show that the real part of the function, as described in Equation 17, approaches the Doppler profile.

$$K(x,y) = \frac{\exp(-x^2)}{\pi} \int_{-\infty}^{\infty} \frac{y \exp(-u^2)}{y^2 + u^2} \sinh(2xu) du \tag{17}$$

where $\lim_{y \to 0, y \neq 0} \int_{-\infty}^{\infty} \frac{y \exp(-u^2)}{y^2 + u^2} \sinh(2xu) = \pi$. Therefore,

$$K(x,y)_{y \to 0} = \exp(-x^2) \tag{18}$$

provides a simple formula for the normalized Doppler profile in terms of the parameter x.

The complex Voigt function is useful because it is the basis for the approximation of several additional line shapes; other higher order effects which can be modeled include line mixing and Dicke narrowing [6]. Line mixing results from high pressure spectral lines which are close together; these transitions affect each other creating asymmetrical line shapes. The line shapes considered so far have assumed that the spectral line being calculated is sufficiently isolated. If two strong lines $i \to f$ and $i' \to f'$ are close to each other in frequency, as shown in Figure 3, there are several possible paths for the molecule to make the transition from state *i* to state *f*. The line could absorb a photon of energy $E_f - E_i$, or it could collide with another molecule to receive energy $E_{i'} - E_i$, absorb a photon of energy $E_{f'} - E_{i'}$, then release energy $E_{f'} - E_f$ in another collision [5]. The effect of line mixing causes lineshapes to drop off more gradually on the side where another strong line is present and more quickly on the side where there is no coupling, also shown in Figure 3.

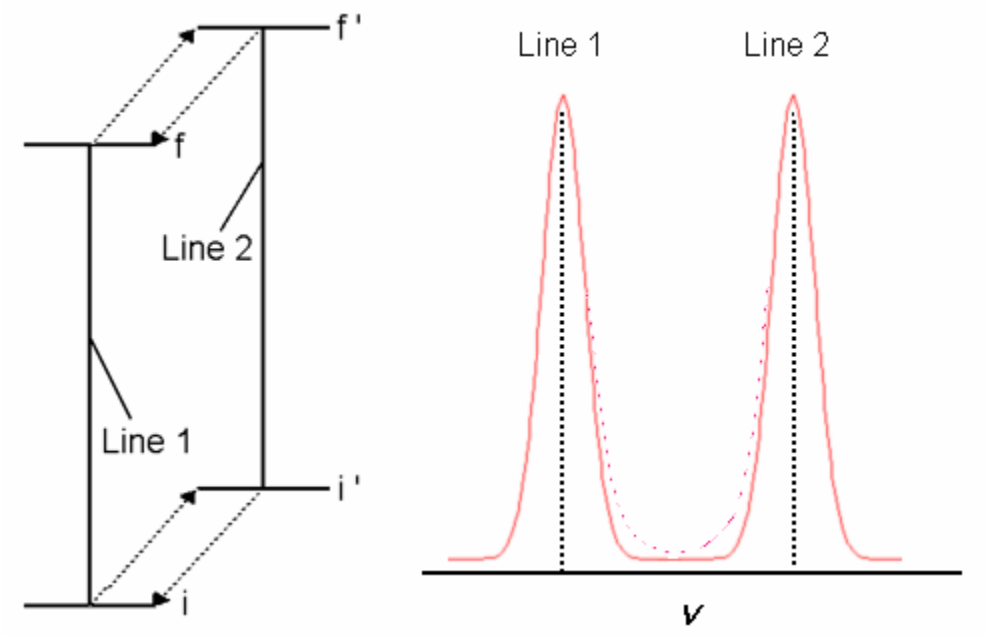

Figure 3: Line mixing in a two line system. The solid arrows respresent optical transitions while the dotted arrows show collisional transitions [5]. The solid lineshapes represent two typical Voigt profiles while the dotted lineshapes are the lineshapes with line mixing included.

Line mixing can be effectively modeled with the real and imaginary parts of the Voigt profile by the Rosenkranz approximation [6].

$$M(x,y) = \frac{\sqrt{\ln 2}}{\alpha_D \sqrt{\pi}} \left[K(x,y) + PYL(x,y) \right] \tag{19}$$

where P is the gas pressure and Y is the Rosenkranz parameter. Since the imaginary part of the Voigt function is anti-symmetric about the line center, it is a useful tool in modeling the asymmetry introduced by line coupling. Since the effect is dependent upon molecular collisions, line mixing is a modification of the Lorentz or collisional broadening portion of the Voigt function, so it becomes more important for lines with large values of the parameter *y*.

The phenomenon of Dicke narrowing or pressure narrowing must be considered when collisions do not change the internal state of a molecule; in these cases the line is

substantially narrowed rather than broadened by pressure [9]. Dicke narrowing becomes important for small values of the parameter *y* because it is a modification of the Doppler portion of the Voigt Profile and is not noticeable when the pressure broadening effect dominates. Normally, the Doppler profile assumes that the time of a collision is negligible and that all recoil momentum due to a transition is transferred to the molecule that is absorbing the photon. The effect of Dicke narrowing occurs when this assumption is no longer valid because a substantial amount of the recoil momentum is instead transferred to the surrounding molecules through collisions. The hard-collision model assumes that the collisions with the surrounding molecules are elastic rather than inelastic. Thus, Dicke narrowing is a pressure dependent effect and can be modeled in the hard collision limit by the Rautian Profile [10],

$$R(x,y)=\frac{\sqrt{\ln 2}}{\alpha_D\sqrt{\pi}}\,\mathrm{Re}\left[\frac{K(x,y+H)+iL(x,y+H)}{1-\sqrt{\pi}H\{K(x,y)+iL(x,y+H)\}}\right] \tag{20}$$

which also includes Lorentz broadening and Doppler broadening. In this equation, the parameter H describes Dicke narrowing and is proportional to pressure.

IV. Previous Research and Background

Our research group has previously developed an algorithm to calculate the complex Voigt function to a relative accuracy of 10^{-6} and the derivatives of the Voigt function with respect to the parameters *x* and *y* to a relative accuracy of 0.5% [1]. This algorithm employs the mathematical methods of Gauss-Hermite quadrature, Taylor series expansion about pre-computed points, and Lagrange interpolating polynomials to create a routine which is a factor of 4 to 18 times faster than common Voigt function algorithms as published [2],[3] and two orders of magnitude more accurate. These same methods

have been used in an attempt to create an efficient algorithm to calculate a speed dependent line shape.

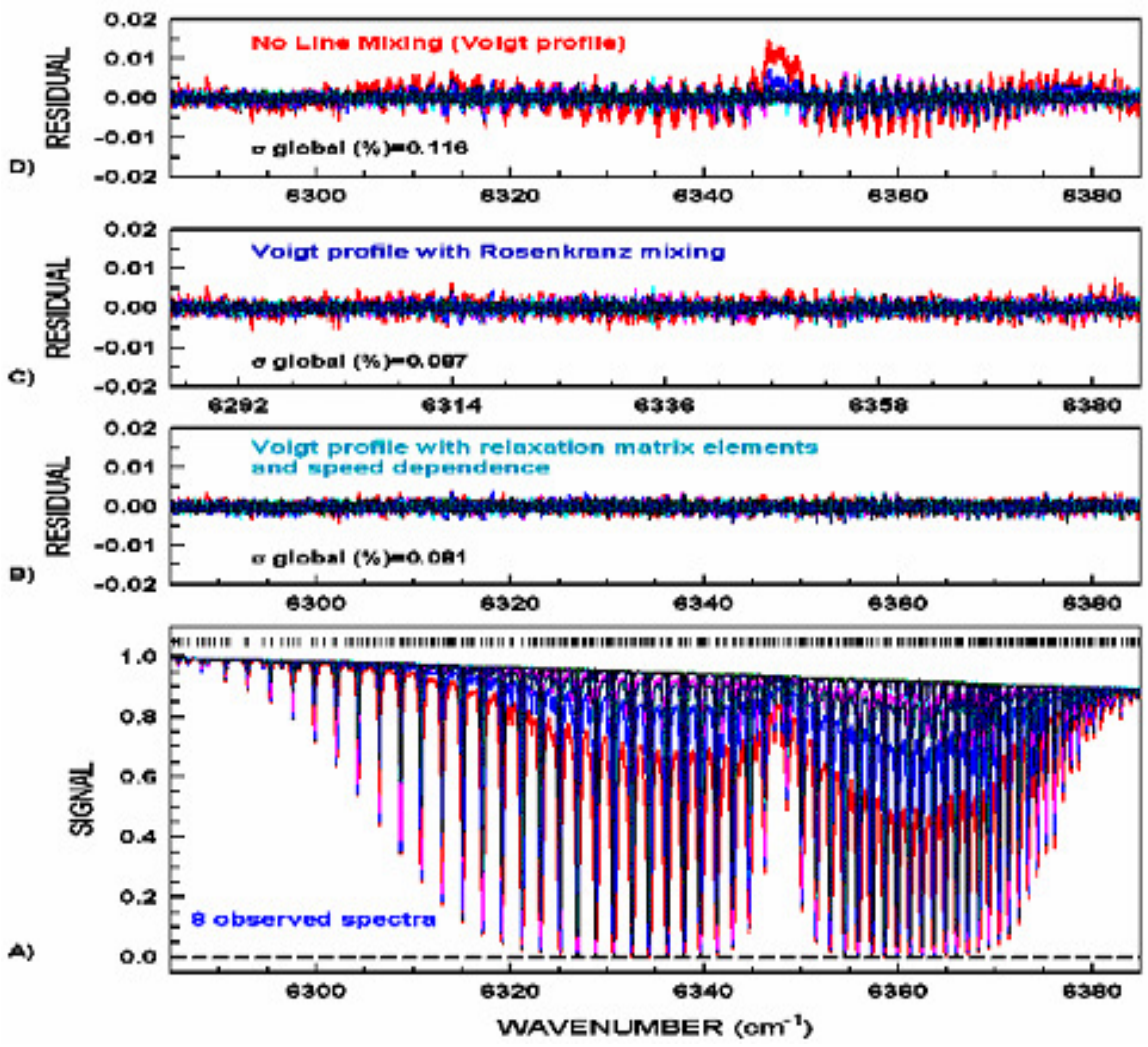

Figure 4: Multispectrum fit of 8 observed carbon dioxide spectra in the $2\nu_1 + 2\nu_2^0 + \nu_3$ band of $^{12}C^{16}O_2$ showing a decrease in residuals due to inclusion of line mixing and speed dependence.

Though the speed dependent effect is only beginning to be detected in observed spectra, as technology and data analysis techniques advance, the need for accurate, efficient methods of calculating speed dependent profiles will also increase. In a recent study by our laboratory, the spectra of carbon dioxide were measured with unprecedented accuracy. Including the speed dependent effect in the fits decreased the global standard deviation

from 0.087% to 0.081%, as seen in Figure 4. Though including line mixing is responsible for most of the improvement in the fit, the decrease in residuals due to speed dependence is definitely noticeable.

V. Choice of Speed Dependent Lineshape Model

Since line mixing is prominent in the spectra observed by our laboratory, a speed dependent model which could also include line mixing would be useful. Thus, a speed dependent modification of the complex Voigt function, including both real and imaginary parts, would be ideal since it could be employed in the Rosenkranz approximation, which reduces to the full implementation of line mixing for a given temperature and pressure, and also in the Rautian profile to model Dicke narrowing.

In the Voigt profile, the Lorentz width assumes all collisions occur at the average speed of the molecules. To modify the Voigt profile into a speed dependent profile, the Lorentz width must become a function of *v*, where *v* shall be defined as the velocity of a single molecule over the average thermal velocity of all the molecules. Since the parameter *y* is dependent on the Lorentz width, *y* becomes *y(v)*, and by integrating Equations (11) and (12) over angular momentum[10], one obtains a complex speed-dependent function[12]:

$$K_S(x,y) = \frac{2}{\pi}\int_{-\infty}^{\infty} \exp(-v^2)\cdot v \cdot \arctan\left(\frac{x+v}{y(v)+H}\right)dv \tag{21}$$

$$L_S(x,y) = \frac{1}{\pi}\int_{-\infty}^{\infty} \exp(-v^2)\cdot v \cdot \ln\left(1+\left(\frac{x+v}{y(v)+H}\right)^2\right)dv \tag{22}$$

The hard collision parameter *H* is also included in this profile to model Dicke narrowing. There are several possible equations for speed dependence of the Lorentz width, but a

quadratic model was chosen for this approximation. Since speed dependence is a small effect, a quadratic model for *y(v)* should be sufficient when the goal is efficiency; one model proposed by Pine and Ciurylo [13] and used by the University of Toronto[14] is $y(v) = y(\bar{v})(1 + S(v^2 - c))$, where *S* is a parameter to be fitted by a data analysis program, $y(\bar{v})$ is the Lorentz width at the average velocity, and *c*=1.5.

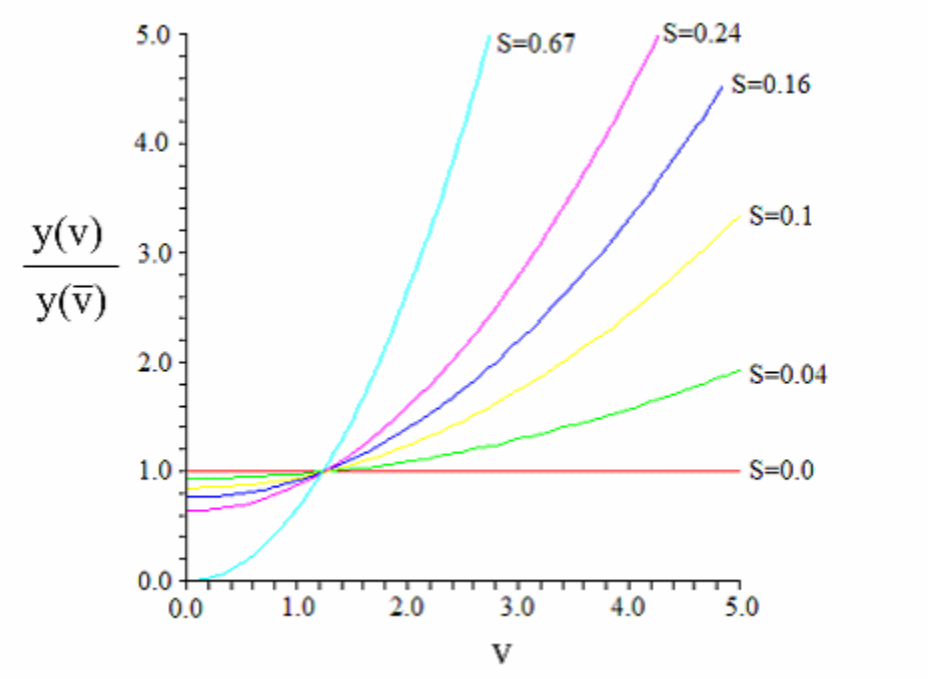

Figure 5: Dependence of the Lorentz halfwidth upon velocity for various values for the speed dependent parameter S. Here, the parameter v is the velocity of a single molecule over the statistical average velocity.

Figure 5 displays the quadratic model for the Lorentz halfwidth as a function of velocity for different values of S. This graph also displays the limitations on S; if $S \geq c^{-1}$, then the Lorentz width becomes zero or negative for some values of *v*. Since a negative or nonexistent width is unphysical, $S < c^{-1}$, or in our case, *S*<0.67. The graph shows that as *S* becomes larger, the slope of the parabola becomes greater; a Lorentz width which is so highly sensitive to velocity changes is unlikely since speed dependence is a small effect.

Therefore, values for *S* should remain significantly less than the upper bound, a concept that is reinforced by Figure 6, a graph of the real part of the speed dependent line shape at *y*=1.0, *H*=0.0, but for different values of *S*. It can be noted that when *S* is 0.1, the function is very close to the Voigt profile, as should be expected. However, when *S* is 0.5, the function diverges greatly from the Voigt profile. If speed dependence were such a large effect, this divergence would have been noticed before now.

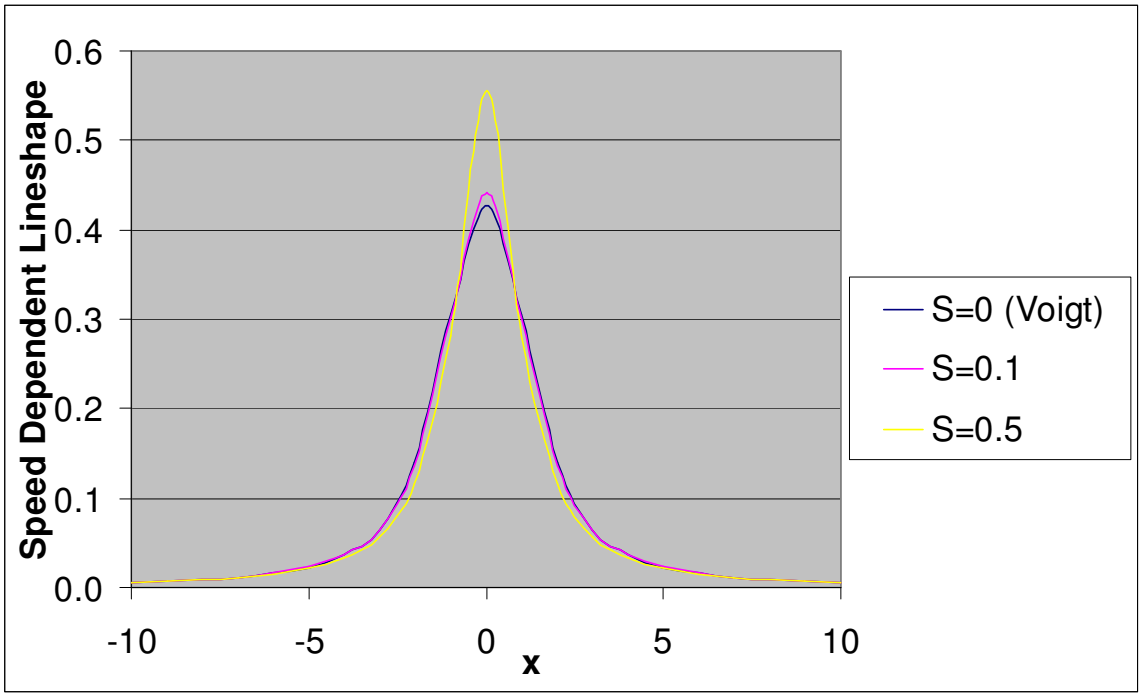

Figure 6: Line shapes with y=1.0, H=0 for Voigt profile and speed dependent models with S=0.1, 0.5, shows how speed dependent model diverges unrealistically from Voigt profile for large S values.

Ciurylo and Pine [13] also indicate that the pressure shift could be modeled using the same speed dependent parameter S; though there is no physical reason for this choice, the speed dependence of the pressure shift appears to be highly correlated with the speed dependence of the linewidth. The pressure shift changes the center of the line from ν_0 to $\nu_{line} = \nu_0 + [\delta + \delta'(T - T_0)]P$ [6] where δ is the pressure shift coefficient at temperature T_0, T is the temperature and P is the gas pressure and δ' is the temperature dependence

coefficient for the pressure shift. Adding this effect changes the parameter x in the lineshape model. If speed dependent pressure shift were to be included, x would also become dependent upon the velocity v in a similar manner as y as shown in Equation (23).

$$x(v) = \sqrt{\ln(2)}\frac{\nu - \nu_0 - [\delta(1 + S(v^2 - c)) + \delta'(T - T_0)]P}{\alpha_D} = x - \frac{\sqrt{\ln(2)}}{\alpha_D}(\delta PS(v^2 - c)) \quad (23)$$

In this equation, x and δ are the values of the distance from line center and the pressure shift at the average statistical velocity. This new value of x results in a new speed dependent profile, where the real part is given by equation (24) and the imaginary part changes in a similar fashion.

$$K_S(x, y) = \frac{2}{\pi}\int_{-\infty}^{\infty}\exp(-v^2)\cdot v \cdot \arctan\left(\frac{x + v - pS(v^2 - 1.5)}{y(1 + S(v^2 - 1.5)) + H}\right)dv \quad (24)$$

In this equation, $p = \frac{\sqrt{\ln(2)}\delta P}{\alpha_D}$ depends on the pressure, the value of the pressure shift at T_0 and the Doppler width. In order to calculate this profile, the parameter p would have to become another input parameter into the function since it changes for any given pressure, temperature, or intrinsic pressure shift coefficient. Though it was hoped that adding speed-dependent pressure shift would be trivial after the speed dependence of the Lorentz width was calculated, this additional parameter, as well as the increased complexity of the profile itself, make adding speed dependent pressure shift mathematically difficult.

VI. University of Toronto Algorithm

The method used by the University of Toronto [14] to calculate the complex speed dependent function is a numerical integration technique involving equally spaced

points. This method is described by Equations (25) and (26) where the parameters used by the Toronto group are $v_0 = -4.0$, $\Delta = 0.5$, and $n = 16$.

$$K_T(x, y, S, H) = \frac{2\Delta}{\pi} \sum_{i=0}^{n} \exp(-(v_0 + i\Delta)^2) \cdot (v_0 + i\Delta) \cdot \arctan\left(\frac{x + (v_0 + i\Delta)}{y(1 + S((v_0 + i\Delta)^2 - 1.5)) + H} \right)$$

(25)

$$L_T(x, y, S, H) = \frac{\Delta}{\pi} \sum_{i=0}^{n} \exp(-(v_0 + i\Delta)^2) \cdot (v_0 + i\Delta) \cdot \ln\left(1 + \left(\frac{x + (v_0 + i\Delta)}{y(1 + S((v_0 + i\Delta)^2 - 1.5)) + H} \right)^2 \right)$$

(26)

These equations were used in our laboratory to calculate the speed dependent line shape for the CO_2 fitting in Figure 4, providing an improvement in fit and proving the model is at least a valid first approximation.

However, Toronto's calculation technique is highly inefficient, as shown by the time trials in Table 2, as well as inaccurate in several regions. When *x* and *y* are small, the behavior of the speed dependent function becomes highly nonlinear and this method, along with many other types of numerical integration, is no longer acceptable. Also, Toronto's numerical integration routine requires many divisions, and is not time-efficient; it uses equal point spacing rather than trying to customize to the function and also must calculate the exponential, arctangent, and natural logarithm functions at each evaluation. Their technique wastes calculation time by employing a 17^{th} order approximation in some regions of the *x*-*y* plane where a much lower order approximation would suffice.

VII. Mathematical Approximation Techniques

As a Voigt calculation routine [1] has recently been optimized by our laboratory, some different methods for calculation are proposed in this paper. Just as with the Voigt function, programs in Maple™ [15] and Fortran 90 have been written to analyze the accuracy of various mathematical techniques. The goal was to develop an algorithm which will calculate this speed-dependent model to a relative accuracy of 1 part in 10^5 and will be no more than a factor of a few slower than a Voigt calculation. Some of these mathematical methods should work for most speed-dependent models $y(v)$, but they are first being optimized for a quadratic model. In regions where x or y is large, an optimized integration routine involving Gauss-Hermite quadrature has been created. In regions where both x and y are small, an interpolation scheme is required to meet the relative error goal because other forms of numerical integration become inaccurate.

A. Gauss-Hermite Quadrature

Gauss-Hermite quadrature is a rapid way to evaluate an integral of the form of Equations (21) and (22) by assuming that the integrand can be well approximated by $\exp(-v^2)$ times a polynomial, $f(v)$. Since $\exp(-v^2)$ appears in the integrand on the right side of the equations, one would expect this approximation, shown in Equation (27), to be good as long as the rest of the integrand behaves as a low order polynomial. This integration is exact if $f(v)$ is a polynomial of order $2n-1$ or less.

$$\int_{-\infty}^{\infty} \exp(-v^2) f(v) dv = \sum_{i=1}^{n} w_i f(v_i) \tag{27}$$

The weights, w_i, and the zero points of the Hermite polynomials, v_i, are tabulated as a function of the order of quadrature, n [16]. Odd order quadrature is advantageous due to the fact that one of the values of v_i is always zero and $f(0)$ is calculable with fewer arithmetic operations. Further simplification is possible given that the remaining evaluation points come in pairs which have identical weights and whose values of v_i are the negatives of each other. The improvements to Gauss-Hermite quadrature for a general function $f(v)$ are summarized in Equation (28).

$$\int_0^\infty \exp(-v^2)(f(v)+f(-v))dv = w_0 f(0) + \sum_{i=1}^{(n-1)/2} w_i (f(v_i)+f(-v_i)) \tag{28}$$

The first approach taken when attempting to approximate Equations (21) and (22) with Gauss-Hermite quadrature was to convert both integrals into the form shown in Equation (28); resulting in Equation (29) for $f_R(v)+f_R(-v)$ of the real part and Equation (30) for $f_I(v)+f_I(-v)$ of the imaginary part.

$$f_R(v)+f_R(-v) = \frac{2v}{\pi}\arctan\left(\frac{x+v}{y(v)+H}\right) - \frac{2v}{\pi}\arctan\left(\frac{x-v}{y(v)+H}\right) = \frac{2v}{\pi}\arctan\left(\frac{2v}{y(v)+x^2-v^2+H}\right) \tag{29}$$

$$f_I(v)+f_I(-v) = \frac{v}{\pi}\ln\left(1+\left(\frac{x+v}{y(v)+H}\right)^2\right) - \frac{v}{\pi}\ln\left(1+\left(\frac{x-v}{y(v)+H}\right)^2\right) = \frac{v}{\pi}\ln\left(1+\frac{4xv}{(y(v)+H)^2+(x-v)^2}\right) \tag{30}$$

To remove the extra calculation time required for the transcendental functions, efficient approximation routines for the arctangent in Equation (29) and natural logarithm in Equation (30) were created to be implemented with Gauss-Hermite quadrature. These Taylor series expansions meet an accuracy requirement of 10^{-6} and dispense with the extra calculation time required for calculating the transcendental functions using internal functions built into the programming language. Each approximation subroutine covers the entire domain of the

transcendental function so Taylor series about different points are employed for different regions of the domain of the arctangent and natural logarithm functions. However, many common cases are successfully calculated using two well-known Taylor expansions. The arctangent is approximated to varying degrees for $|z| < 1$ by

$$\arctan(z) = z - \frac{z^3}{3} + \frac{z^5}{5} - \frac{z^7}{7} + \ldots (-1)^n \frac{z^{2n+1}}{2n+1}, \text{ where } z = \frac{2v}{y(v) + x^2 - v^2 + H} \qquad (31)$$

and the natural logarithm is approximated for $|q| << 1$ by

$$\ln(1+q) = q - \frac{q^2}{2} + \frac{q^3}{3} - \frac{q^4}{4} + \ldots + (-1)^{n+1} \frac{q^n}{n}, q = \frac{4xv}{(y(v) + H)^2 + (x - v)^2} \qquad (32)$$

As a first attempt, this method of calculating Gauss- Hermite quadrature has merit; however is it not as efficient as desired compared to Voigt in the time trials shown in Table 7. For each Taylor series expansion, as many as seven terms are required, in addition to a complex logical structure to determine which approximation to use. Though this method is faster than just using the internal transcendental functions in Fortran 90, it is still not optimal. In addition, though the relative error requirement of 10^{-5} is met, the behavior of errors is difficult to predict and varies with the errors in the elementary function approximation. This variation makes it difficult to recognize a simple pattern which determines the required order of quadrature as a function of *x, y*, *S*, and *H*.

As an alternative method to Taylor series approximation of the arctangent and natural logarithm, Equations (21) and (22) were integrated by parts, yielding Equations (33) and (34). Thus, the elementary functions can be eliminated before quadrature is implemented.

$$K_S(x,y) = \exp(-v^2)\cdot\arctan\left(\frac{x+v}{y(v)+H}\right)\Big|_{-\infty}^{\infty} + \int_{-\infty}^{\infty}\exp(-v^2)\cdot\frac{\partial}{\partial v}\left(\arctan\left(\frac{x+v}{y(v)+H}\right)\right)dv =$$

$$\int_{-\infty}^{\infty}\exp(-v^2)\cdot\frac{y+H\text{-}yS(v^2+c+2xv)}{y^2S^2v^4\text{-}2y^2S^2v^2c\text{+}2y^2Sv^2\text{+}2Sv^2yH\text{+}y^2c^2S^2\text{-}2y^2cS\text{-}2cSyH\text{+}y^2\text{+}2yH\text{+}H^2\text{+}x^2\text{+}2xv\text{+}v^2)} \tag{33}$$

$$L_S(x,y) = \exp(-v^2)\cdot\ln\left(1+\left(\frac{x+v}{y(v)+H}\right)^2\right)\Big|_{-\infty}^{\infty} + \int_{-\infty}^{\infty}\exp(-v^2)\cdot\frac{\partial}{\partial v}\left(\ln\left(1+\left(\frac{x+v}{y(v)+H}\right)^2\right)\right)dv =$$

$$\int_{-\infty}^{\infty}\exp(-v^2)\cdot\frac{(x+v)(y+H\text{-}yS(v^2+c+2xv))}{y^2S^2v^4\text{-}2y^2S^2v^2c\text{+}2y^2Sv^2\text{+}2Sv^2yH\text{+}y^2c^2S^2\text{-}2y^2cS\text{-}2cSyH\text{+}y^2\text{+}2yH\text{+}H^2\text{+}x^2\text{+}2xv\text{+}v^2)} \tag{34}$$

Integration by parts is a helpful technique in this case since the first term involving $\exp(-v^2)$, evaluated from negative infinity to infinity, vanishes in both cases, leaving only one integral for each equation. These integrals are further simplified after the fashion of Equation (28) before being implemented in a subroutine. Though the resulting equations appear complicated, calculation time is saved by the elimination of the arctangent and natural logarithm. In addition, the final form may be simplified considerably by the use of clever programming; for example, the denominator is the same for both the real and imaginary parts and thus must only be calculated once.

As compared to the unpredictable errors generated by the Taylor series expansion of the arctangent and logarithm in the previous method, this technique yields a relatively simple pattern describing the choice of quadrature order. These patterns are summarized by Figure 7 a) and Figure 7 b), which display an approximate diagram of the quadrature orders used over a region of the *x*-*y* plane for various values of *S*. Tables A.1-A.4 in Appendix A also display in more detail the exact boundaries of the various regions. These graphs and tables assume that H=0, which is the most difficult case to model; these quadrature regions thus also meet the error requirement for other values of *H*. Additional

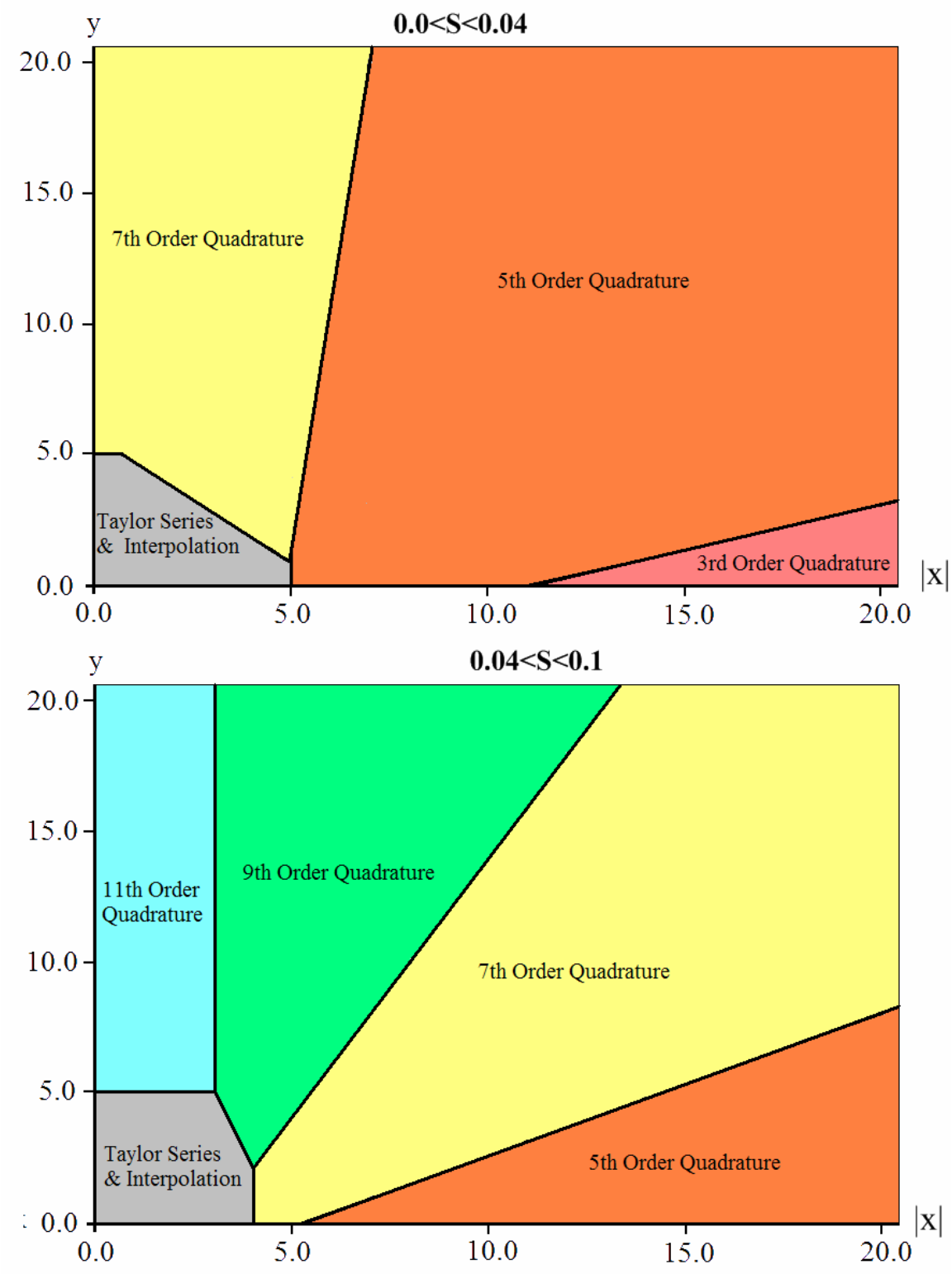

Figure 7 a): Displays a schematic of the quadrature approximation regions employed for certain values of the speed dependent parameter: 0<S<0.1

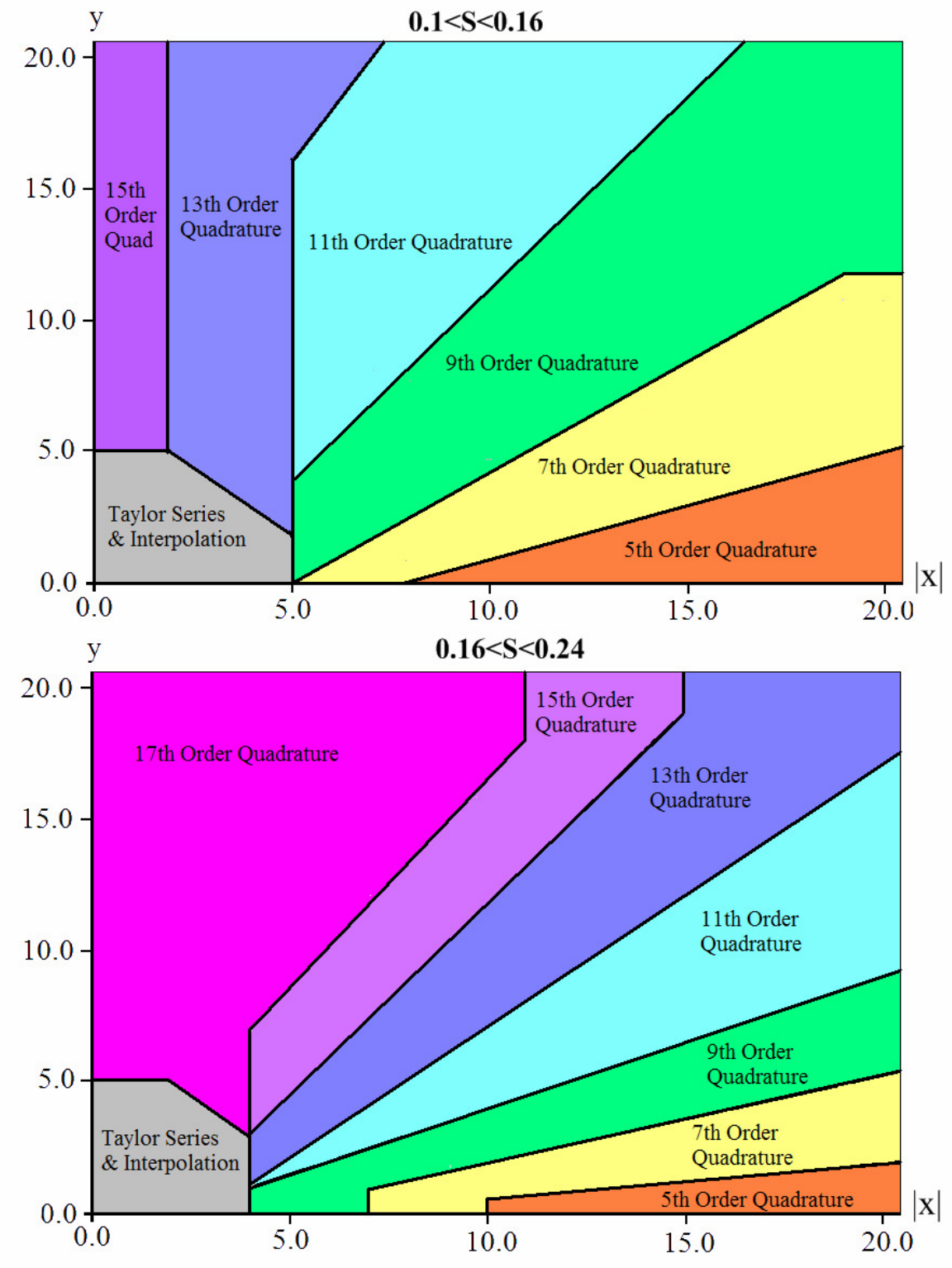

Figure 7 b): Displays a schematic of the quadrature approximation regions employed for certain values of the speed dependent parameter: 0.16<S<0.24.

regions employing lower order quadrature for larger values of H were not created because the increase in the effectiveness of the approximations at larger H values was not dramatic enough to warrant the increase in complexity of the program. Determining the accuracy of each order of quadrature was done by comparison with a numerical integration program [18] set to provide a relative error of less than 10^{-12} and with Maple™ [15]. Since a given order of quadrature becomes less effective as S increases, these graphs were created by determining the boundaries for quadrature regions adequate on the x-y plane for a certain value of S, then assigning those boundaries to a range of S values beneath the chosen value. The grey area labeled as Taylor series and interpolation for each graph signifies the region where quadrature is no longer effective and alternative methods must be employed; as expected, quadrature begins to fail for small x and y, specifically for $x, y < 5.0$. For $0.1<S<0.16$, graphs of the exact regions used by the algorithm, developed from the output of the function itself, are displayed in Figures A.1 and A.2 in Appendix A. The regions of calculation differ slightly from the regions shown in the schematic diagrams in Figure 7, but the overall pattern remains the same. It also must be noted that for $S > 0.24$, all calculation is performed using 17^{th} order quadrature and the relative error requirement may not be met in a few cases. In fact, S values this large are only included to ensure that the fits converge because they are not likely to be measured. Figure 8 displays the fitted values of the speed dependence parameter S for a carbon dioxide spectrum; it seems to reach a maximum value of about 0.15, a result which probably also typical of other gases. As shown by Figure 4, larger S values become increasingly unphysical since they imply a high sensitivity of the Lorentz width to very small speed changes.

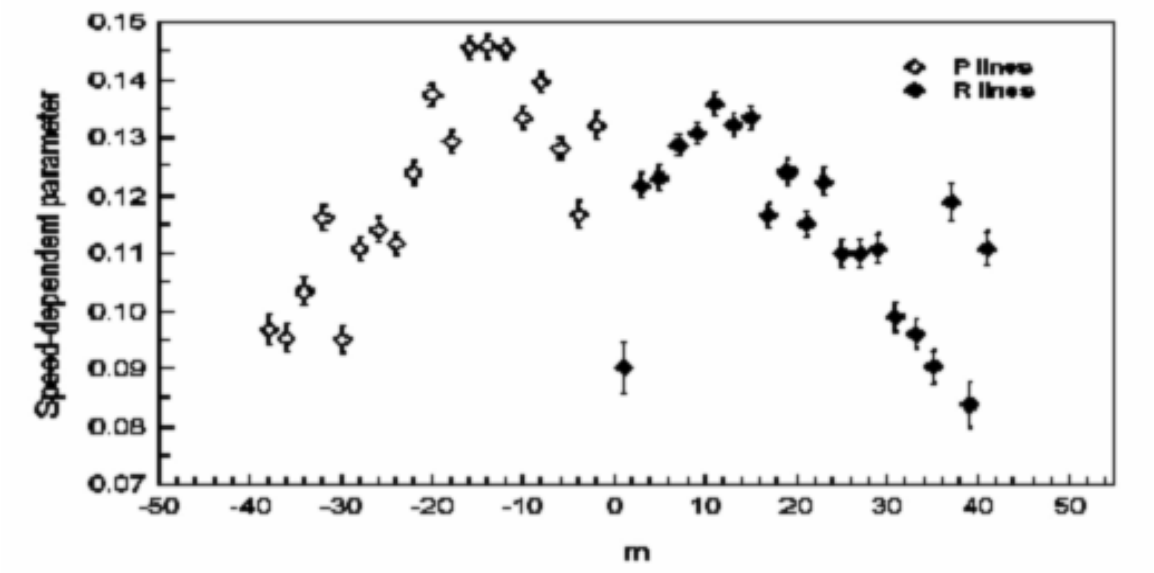

Figure 8: Displays the values of the speed dependent parameter for the $2\nu_1 + 2\nu_2^0 + \nu_3$ band of $^{12}C^{16}O_2$ as a function of the parameter m, where m=-J" for the P branch and m=J"+1 for the R branch. J" is the rotational quantum number of the lower state.

B. Interpolation Methods

The next segment of this research project was to test and create a routine for the regions in which Gauss-Hermite quadrature is not sufficient. In these regions, the function *f(v)* for the speed dependent function no longer behaves like a polynomial, thus making many forms of quadrature and numerical integration ineffective. For the area where *x* and *y* are small, evaluation of the Voigt function has traditionally relied upon mathematical methods developed for the complex error function, which is closely related to the Voigt function. For the speed dependent profile, these shortcuts no longer work since *y* becomes dependent on the variable of integration, *v*. Our recently developed Voigt subroutine takes advantage of the memory space in a modern computer by creating a routine dependent upon 1.5 MB of data stored on the hard drive. The same method works for the speed-dependent subroutine, though the addition of the parameters *S* and *H*

adds more dimensions to the interpolation table, increasing the required memory by several orders of magnitude. In order to incorporate S only, about 24 MB of data was required. To incorporate various values of *H*, the amount of data would increase by another order of magnitude, perhaps signaling that interpolation is no longer a good option when it is required to fit 4 parameters. For this project, it was decided to choose *H=0* for the interpolation tables since observing Dicke narrowing requires very high resolution spectra and therefore the effect is seen infrequently. As an additional option, it is still possible to calculate the profile including a nonzero Dicke narrowing parameter using 17th order quadrature in the interpolation region. Though this method does not always meet the 10^{-5} error requirement, the improvements in fit shown in Figure 4 for a first approximation speed dependent model prove that often a first approximation can create valid results.

Several different options are available for interpolation; one technique involves storing the values of the profile and the values of its N^{th} order derivatives at points in a grid, enabling evaluation of the profile by Taylor expansion about the nearest grid point. For second order derivatives, the real part of the speed dependent function becomes:

$$K_s = K_0 + \frac{\partial K_0}{\partial x}\Delta x + \frac{\partial K_0}{\partial y}\Delta y + \frac{\partial K_0}{\partial S}\Delta S + \frac{1}{2}\cdot\left(\frac{\partial^2 K_0}{\partial x^2}\Delta x^2 + \frac{\partial^2 K_0}{\partial y^2}\Delta y^2 + \frac{\partial^2 K_0}{\partial S^2}\Delta S^2\right) + \frac{\partial^2 K_0}{\partial x \partial S}\Delta x \Delta S + \frac{\partial^2 K_0}{\partial y \partial x}\Delta y \Delta x + \frac{\partial^2 K_0}{\partial S \partial y}\Delta S \Delta y \qquad (35)$$

in which K_0, $\frac{\partial K_0}{\partial x}$, $\frac{\partial K_0}{\partial y}$, $\frac{\partial K_0}{\partial S}$ etc. are all stored in the table and $\Delta x, \Delta y, \Delta S$ are the displacements of each parameter from the closest point. The imaginary part would be represented by a similar function. Since the derivatives of the complex Voigt function have symmetric relationships which greatly simplify the Taylor series expansion, it was

hoped that the speed dependent function would display similar relationships. However, testing with Maple™ shows that no such symmetry exists for the speed dependent function. As with the Voigt function, it was found that a 3rd order Taylor expansion provided the best compromise between speed and accuracy, with spacing in x and y of 0.02, 0.05, and 0.1 in different regions of the x-plane and a spacing in S of 0.03 everywhere. Figure 10 displays the various regions for different spacing and types of interpolation and Table A.5 in Appendix A summarizes their exact locations as well as the sizes of the interpolation tables. To increase efficiency, the grid points in the interpolation tables are evenly spaced, simplifying calculation of array indices.

The second type of interpolation used is a spline interpolation, as displayed in Figure 9. For small y-values, or lines where the Doppler width is about 50 times larger than the Lorentz width, and for small *x*-values, close to the center of the line, spline interpolation is required. For small *x*-values, spline interpolation is only required for the imaginary part of the function since it changes rapidly with respect to *x*, approaching a zero value around $x=0$ while the real part smoothly approaches a maximum value. For small y-values, spline interpolation is required for both the real and imaginary parts of the speed dependent function since many of the derivatives of the profile begin to approach infinity or zero as y becomes small. Spline interpolation has an advantage because choosing the Doppler profile as the limit at $y=0$ ensures that the function will always converge to the Doppler limit. Specifically, Equation (18) is used to calculate the real part of the function and a numerical integration of Equation (16) is used for the imaginary part of the function at $y=0$. Since speed dependence is a modification of the Lorentz profile only, it is acceptable to use the Doppler limit of the Voigt function as the $y=0$

value for this algorithm. It is also possible to obtain Equations (16) and (18) using the speed dependent Equations (33) and (34), but the calculation is much more complicated Many traditional Voigt functions are off by orders of magnitude for very small values of *y*, as shown by Table 1 [19].

The Letchworth and Benner algorithm, which also interpolates smoothly to the Doppler limit, provides an accurate answer, even when most other routines are off by two orders of magnitude or more. Note that the Lether and Winston algorithm used as a standard of comparison in Table 1 requires seventeen complex polynomial evaluations for one Voigt calculation [4], as compared to a maximum of three simple polynomials used by Letchworth and Benner. Thus, their method is highly accurate but very inefficient.

Real Voigt Profile x=5.5, y=10^{-14}	Humliček [3], Drayson [2]	Lether and Winston [4] (regarded as standard)	Letchworth & Benner [1]
	1.966215×10^{-16}	7.307386×10^{-14}	7.307386×10^{-14}

Table 1: Displays the calculated values of the Voigt Profile for several different algorithms [17] and shows that interpolation to the Doppler limit [1] yields the correct result where other methods fail.

The spline interpolation is performed using Lagrange interpolating polynomials, which is calculation-intensive, but very accurate; Lagrange interpolating polynomials of degree *n*-1 are of the form:

$$P(\varphi) = \sum_{j=1}^{n} P(\varphi_j) \prod_{k=1, k\neq j}^{n} \frac{\varphi - \varphi_k}{\varphi_j - \varphi_k}. \tag{36}$$

For a *k* dimensional interpolation, the number of polynomials of degree *n*-1 which must be evaluated for one calculation is $\sum_{i=0}^{k-1} n^i$. As in the Taylor expansion method, evenly spaced interpolation points are be employed; this approach allows the denominators in Equation (36), $\varphi_j - \varphi_k$, to be replaced by pre-calculated integer multiples of the point

spacing. Polynomials of varying degrees could have been used to interpolate over $\varphi = x, y, S$, but a reasonable spline interpolation requires low-degree polynomials to prevent inefficiency. Thus, second order polynomials in the form of Equation (37) were chosen for *x,y,* and *S*.

$$P(\varphi) = P(\varphi_2) + \frac{1}{2}\Delta\varphi(P(\varphi_3) - P(\varphi_1)) + \frac{1}{2}(\Delta\varphi)^2(P(\varphi_1) + P(\varphi_3) - 2P(\varphi_2)) \qquad (37)$$

In this equation, $P(\varphi_2)$ is the stored values of the function at the closest gridpoint while $P(\varphi_1)$ and $P(\varphi_3)$ are the stored values of the function at the gridpoints above and below the closest gridpoint respectively. $\Delta\varphi$ is the signed distance from the closest gridpoint to the point where the function is to be evaluated, divided by the spacing between gridpoints.

$$\Delta\varphi = \frac{\varphi - \varphi_2}{\varphi_2 - \varphi_1} \qquad (38)$$

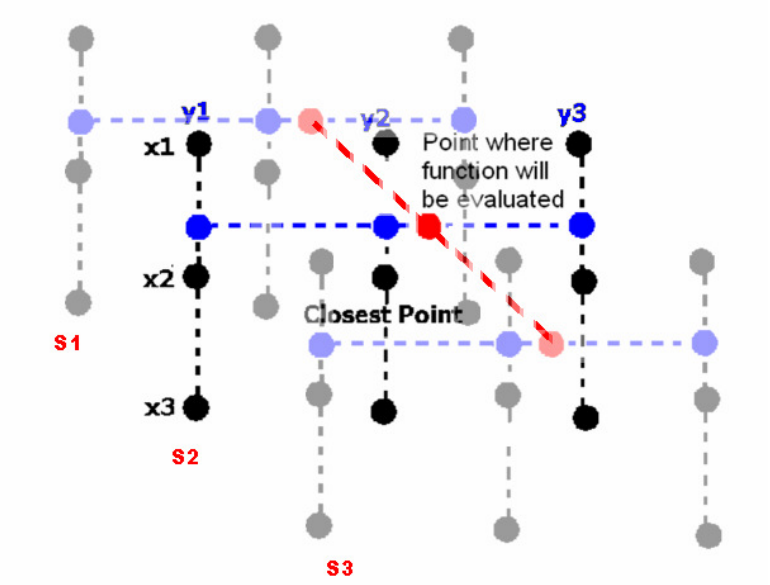

Figure 9: Schematic diagram of three dimensional spline interpolation. Each dotted line represents a polynomial evaluation. There are nine grey/black dotted lines, representing x-interpolations, three blue dotted lines representing y-interpolations, and one red dotted line representing S-interpolation.

Thirteen polynomial evaluations are required for one calculation of the real or imaginary part of the speed dependent function: nine in *x*, three in *y*, then one in *S*. Gridpoint spacing of 0.01 in S and 0.005 in x and y were employed for the interpolation tables in both the small x and small y regions. The exact regions for spline interpolation and the memory required by the tables are shown in Table A.5.

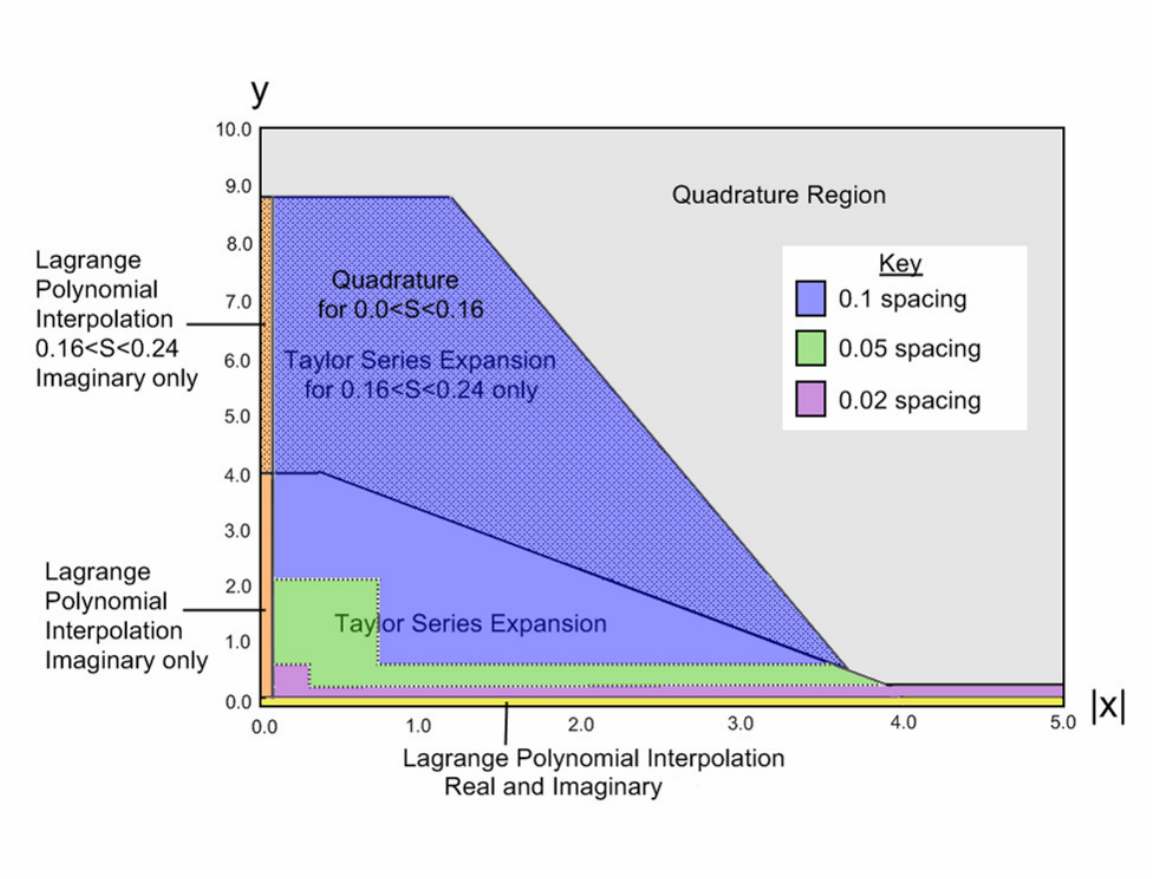

Figure 10: Displays regions in the x-y plane for which Taylor series expansion and Lagrange polynomial interpolation are required. Also shows regions of different grid point spacing in x and y.

VIII. Programming Techniques

Calculation of a speed dependent profile based upon the regions defined in Tables A.1-A.5 is made efficient through the use of sophisticated programming techniques. Since *y*, *S*, and *H* remain constant for one spectral line and only *x* changes based on wave

number, the subroutine may be called only once per line rather than once per (x,y,S,H) point. In addition, quantities involving y, S, and H may be pre-calculated outside the DO loop over the range of x values, saving additional calculation time. Throughout the program, an emphasis was placed on eliminating excess calculation of polynomials, array indices, and other quantities. Though these programming choices have decreased the calculation time in all cases, they have also increased the number of lines of code required. Though the code may look complicated, it is actually built for speed. A comprehensive example of the Fortran 90 code used to calculate the function for a range of x values in one spectral line for the case that H=0 is presented in Appendix C.

Many intermediate variables are also stored in the program, reducing the number of floating point calculations by eliminating redundant calculations. For example, the quantity yS is required ten times for one 3rd order quadrature calculation of the real and imaginary parts; calculating it only once saves time, not only for the one evaluation but it need not be recalculated for other values of x. The logical structure for choosing the order of quadrature to use, pictured in Appendix C, is also optimized to require as little calculation time as possible. Whenever the binary files used to store the interpolation tables and grids for the Taylor Series expansion are read into working memory, they remain there until the program which called the speed dependent terminates. Thus, the binary tables must only be read in once for an entire fit. In addition, the tables are read in only as they are required, so if the spectral lines in the spectra exhibit similar features, most likely only a fraction of the 24 MB needed to store all the tables would be needed at one time.

IX. Calculation Time Comparisons

These programming techniques allow the speed dependent routines created to become competitive in speed with traditional routines to calculate the Voigt function. Table 2 in illustrates the time taken for each routine to calculate 10^8 points in a variety of orders quadrature using a 1.53 GHz Pentium IV processor and a Compaq Visual Fortran 90 Compiler. Though the actual times signify very little since they will vary greatly based upon processor speed, the ratios of calculation times should remain fairly constant. From this table, one can see that the Voigt function, as calculated by our optimized routine, runs approximately 2 to 3 times faster than the speed dependent function calculated using quadrature after the equations are integrated by parts. Since our Voigt algorithm is at least twice as fast as most other Voigt routines [1],[19] which give a relative accuracy within two orders of magnitude, the speed dependent calculation is quite competitive. One can also see that using quadrature and integration by parts is clearly the most efficient means of calculating the speed dependent function in the regions where it meets the accuracy requirement; it runs a factor of 1.75 to 2.5 times faster than using quadrature with Taylor series expansion of the arctangent and natural log and 6 to 14 times faster than the Toronto method.

	Quadrature & Integration by Parts	Quadrature & Taylor series Expansion	Toronto Method [14]	Benner & Letchworth [1] Voigt Subroutine
3^{rd} order	9 s	16 s	n/a	3.6 s
5^{th} order	16 s	30 s	n/a	5.5 s
7^{th} order	22 s	46 s	n/a	7.7 s
17^{th} order	54 s	130 s	340 s	n/a

Table 2: Displays time trial information for various quadrature algorithms used to calculate 10^8 points on a 1.53 GHz Pentium IV processor.

The time trials for the entire speed dependent algorithm over a range $0<x,y<100$, including both interpolation and quadrature regions as well as the logical structures that decide which approximation to use, are located in Table 3. These time trials were performed on a 1.83 GHz Intel Centrino Duo Processor with a Compaq Visual Fortran compiler. Using each algorithm, 10^7 points over the range in x and y were calculated. The Toronto method calculation times do not vary appreciably for different values of S, so only one time is given which applies for all S values. It is clear from the table that the speed dependent algorithm presented in this paper is about 5 times as fast as the Toronto algorithm as it is presently written. However, it is also approximately 6 times slower than the Voigt calculation using Letchworth and Benner. Though these times means the routine is still comparable with Voigt routines such as Drayson and Humliček, there is room for improvement.

Routine	Letchworth & Benner (Voigt)	Speed dependence S<0.04	Speed dependence 0.04<S<0.1	Speed dependence 0.1<S<0.16	Speed dependence 0.16<S<0.24	Toronto Algorithm
Calculation time for 10^7 points 0<x<100 0<y<100	1s	6s	6s	7s	8s	29s

Table 3: Displays time trial information for calculation of 10^7 points for various values of S and $0<x,y<100$ on a 1.83 GHz Intel Centrino Duo processor.

Currently, in order to improve readability during the testing and debugging process, the code has been subdivided into several subroutines and modules. Each spectral line shape calculation requires 2-3 subroutine calls, passing many parameters through as arguments. There are also some duplicate calculations which are in place to improve ease of use during the development stage. It can be estimated that once these excess subroutine calls and logical structures are removed, the calculation time for the algorithm will decrease by approximately a factor of two, making the time trials in Table

3 more consistent with the results in Table 2. This improvement will make the speed dependent algorithm competitive indeed; not only will it be comparable in speed and relative accuracy to common Voigt routines, but it will also model a higher order effect.

X. Accuracy Comparisons

To test the accuracy of the speed dependent algorithm, extensive testing against data provided by Maple™ [15] and CADRE [18] has been performed. The error calculations in this paper are based on the relative error ε where

$$\varepsilon = \left| \frac{V_{calculated} - V_{exact}}{V_{exact}} \right| \tag{39}$$

The aforementioned numerical integration routines provide the "exact" value of the function V_{exact} to within 10-16 digits of precision and the speed dependent algorithm provides $V_{calculated}$. It has been determined that the real part of the speed dependent function is calculated to a relative accuracy of 10^{-5} for all cases while the imaginary part of the speed dependent function also meets this requirement for all but a few cases. These cases occur near the *x*-axis for S>0.16 and the relative error still remains less than 10^{-4}. The value of the imaginary part of the function approaches zero at this point, so the absolute error certainly remains less than 10^{-5}. In addition, values of S higher than 0.16 begin to be unphysical, as shown for carbon dioxide in Figure 8, so these values are comparatively unimportant. Figure 11 displays the relative error data for the speed dependent algorithm presented in this thesis for S=0.0999 and $0<x,y<10$. For each point, the largest of $\log_{10} \varepsilon_{RE}$ (the relative error in the real part) and $\log_{10} \varepsilon_{IM}$ (the relative error

in the imaginary part) is plotted on a color scale, showing that the error requirement of $\varepsilon < 10^{-5}$ is indeed met.

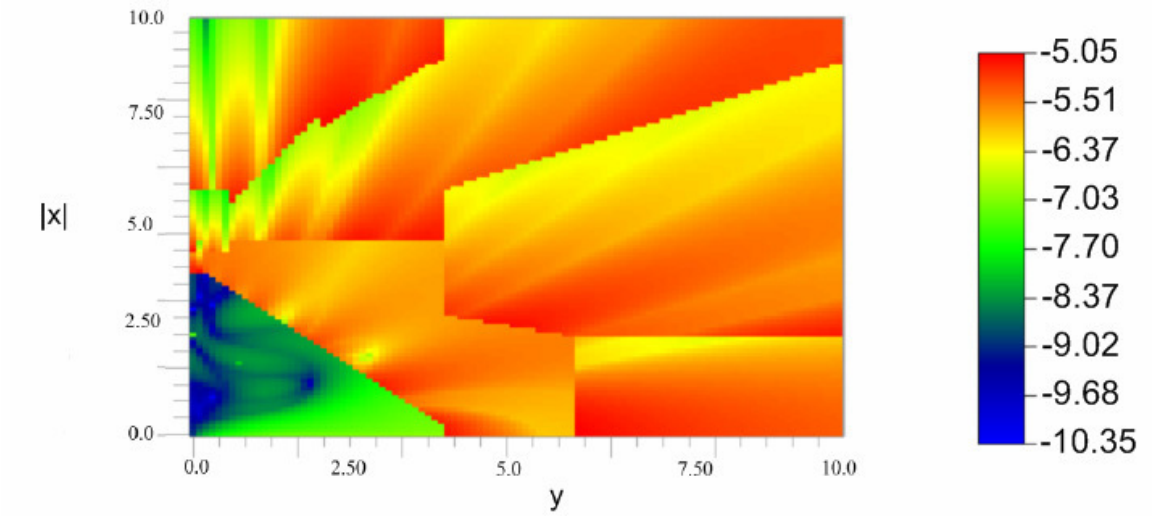

Figure 11: Displays the base ten logarithm of the relative error ($\log_{10}\varepsilon$) for the speed dependent algorithm over the x-y plane for S=0.0999. The scale shows that ε remains below 10^{-5}.

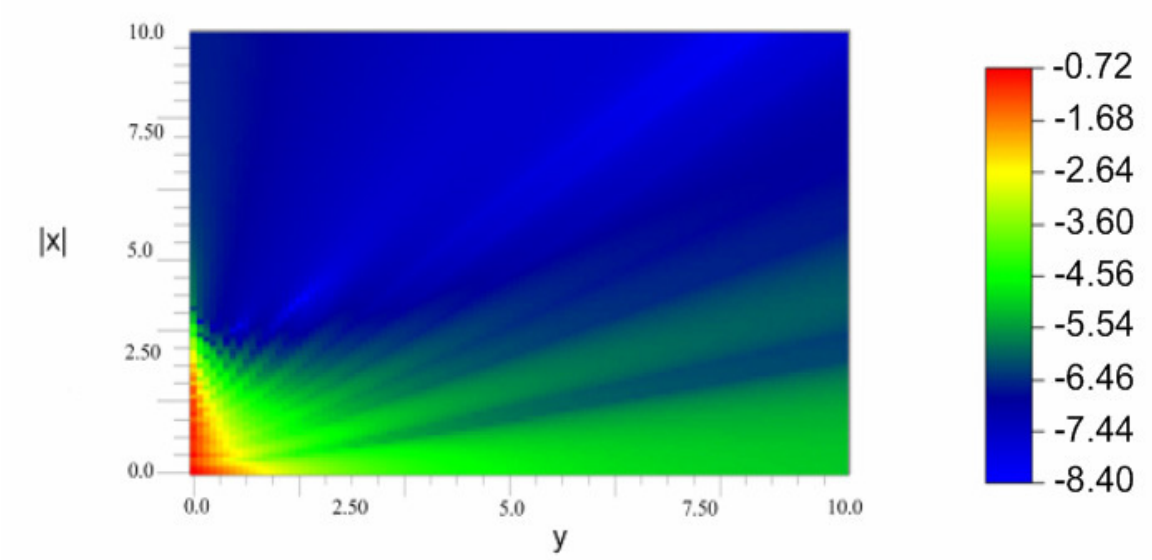

Figure 12: Displays the base ten logarithm of the relative error ($\log_{10}\varepsilon$) for the Toronto algorithm over the x-y plane for S=0.0999. The scale shows that the relative error can be as high as 20%.

Figure 12 displays the same data for the Toronto algorithm, showing that for values of y around 0.1, the relative error in the method can be up to 20%. Any area which displays a green, red, or yellow color on the graph is an area in which the accuracy requirement is not met. In addition, for larger values of x and y when $x \approx y$, there is a region in the Toronto approximation where a subtraction results in the cancellation of two nearly equal terms, causing the routine to lose several digits of accuracy. Combining the poor accuracy of the Toronto routine with its unsatisfactory performance in the time trials rule it out as a viable method for anything other than a first approximation attempt.

Additional accuracy comparison figures for the speed dependent algorithm in this paper and for the University of Toronto method are detailed in Appendix B in Figures B.1-B.5. Figures B.1-B.4 show that within the range displayed ($0<x,y<10$), the algorithm presented in this paper meets the error requirement of one part in 10^{-5} for all but a few isolated cases in the imaginary part. This discrepancy could be fixed easily if desired, at the cost of a few more megabytes of memory space to hold more interpolation tables. The error graphs in Figure B.5 for the University of Toronto method show that while the relative error in the algorithm for some values of x and y is lower than the accuracy requirement of 10^{-5}, the calculations at other values miss that requirement by several orders of magnitude, signifying that the Toronto algorithm is not generally applicable.

XI. Conclusions

In this paper, an algorithm has been presented to analyze speed dependent lineshapes; a quadratic model has been chosen and an efficient, accurate method of calculating the model has been found for all values of x and y. Extensive testing has taken place to ensure that the function meets the relative error requirement of one part in 10^5 in most locations and is bounded elsewhere. Once the routine is in use, some intriguing possibilities for additional research still remain. First, the speed dependent subroutine can be applied to spectra to determine how much improvement is seen in the fits. Also, it would be interesting to investigate how the speed dependent parameter S is related to the temperature dependence exponent of the halfwidth n. Since the Lorentz width has been assumed to be dependent upon the temperature, which is also dependent upon molecular speed, in theory, it would be possible to relate n and S. This relationship could decrease the number of parameters which need to be fit for a given set of spectra. In addition, this relationship would imply that it would reduce the need for temperature measurements since the temperature could then be deduced by measuring the value of the parameter S. Several additional effects could also be modeled more accurately, including Dicke narrowing and pressure shift. Overall, the algorithm we have developed seems likely to become a competitive alternative to traditional Voigt functions; it offers a more precise and physically accurate model without a decrease in efficiency. In addition, since accuracy requirements are met for all values of the parameters x and y, this routine will not be specific to any application but will be a general approach which any research group which currently uses the Voigt profile can use to efficiently include speed dependence in their measurement algorithm.

References

[1] Letchworth K., Benner, D.C. "Rapid and Accurate Calculation of the Voigt Profile." In *J. Quant. Spectrosc. Radiat. Transfer*. Under Review. (2006)

[2] Drayson, S.R. "Rapid Computation of the Voigt Profile." *J . Quant. Spectrosc. Radiat. Transfer*. Vol. 16. pp. 611-614 (1976)

[3] Humlíček J. "Optimized computation of the Voigt and complex probability functions." *J Quant Spectrosc Radiat Transfer*. Vol. 27. pp. 437-444. (1982).

[4] Thompson, W. J. "Numerous Neat Algorithms for the Voigt Profile Function." *Computers in Physics*. Vol. 7, No. 6. pp.627-631. (1993)

[5] Levy, Lacome, & Chackerian. "Collisional Line Mixing", Academic Press, Inc. pp. 261-337. (1992)

[6] Benner D.C., Rinsland C.P., Devi, V.M., Smith M.A.H., Atkins, D. "A multispectrum nonlinear least squares fitting technique." *J Quant Spectrosc Radiat Transfer* Vol. 53. pp.705-721. (1995)

[7] Fox, M. *Quantum Optics: An Introduction*. Oxford University Press, New York (2006).

[8] Armstrong, B. H. "Spectrum Line Profiles: The Voigt Function." *J. Quant. Spectrosc. Radiat. Transfer*. Vol. 7. pp.61-88. (1967)

[9] Dicke, R. H., "The effect of collisions upon the Doppler width of spectral lines", *Physical Review*. Vol. 89, No. 2. pp. 472-473 (1953)

[10] Pine A.S., Looney J.P. "N and air broadening in the fundamental bands of HF and HCl." *J Mol Spectrosc*. Vol.122. pp. 41-55. (1987)

[11] Pine, A. *J . Quant. Spectrosc. Radiat. Transfer*. Vol. 62. pp. 397-423 (1999)

[12] Ciurylo R. “Shapes of pressure- and Doppler- broadened spectral lines in the core and near wings.” *Physical Review A*. Vol. 58 No. 2. (1998)

[13] Pine, A. S., Ciurylo, R. “Multispectrum Fits of Ar-Broadened HF with a Generalized Asymmetric Lineshape: Effects of Correlation, Hardness, Speed Dependence and Collision Duration.” *J Mol Spectrosc*. Vol. 208. pp.180-187. (2001)

[14] Vitcu, A. Ph. D. Dissertation. University of Toronto. (2003)

[15] Maple 9. Waterloo, Ontario: Maplesoft, a division of Waterloo Maple Inc., 2003.

[16] Abramowitz and Stegun. *Handbook of Mathematical Functions*. Dover, New York (1965).

[17] R.J. Wells, “Rapid approximation to the Voigt/Faddeeva function and its derivatives”, *J. Quant. Spectrosc. Radiat. Transfer*. Vol. 62, pp. 29–48. (1999)

[18] de Boor C. “On Writing an Automatic Integration Algorithm.” In: Rice JR, editor. Mathematical Software, New York: Academic Press. pp. 201-209, 430-449. (1971).

[19] Schreier F. “The Voigt and complex error function: A comparison of computational methods.” *J . Quant. Spectrosc. Radiat. Transfer*. Vol. 48. pp. 743-762 (1992)

Appendix A: Tables

	S<0.04
3rd order quadrature	(y<0.02 AND x>13+4.72728y) OR (0.02<y<6.5 AND x>9.27275+4.72728y) OR (6.5<y<20.0 AND x>11.1111+4.444y) OR (20<y<200 AND x>5.26316+4.73684y) OR (200<y<2000) AND x>500+4.75y) OR (y>2000 AND x>5625+4.6875y)
5th order quadrature	(y<0.02 AND 5.5<x<13+4.72728y) OR (0.02<y<3.6 AND 5.5<x<9.27275+4.72728y) OR (3.6<y<6.5 AND 6.5-0.3333y<x<9.27275+4.72728y) OR (6.5<y<20.0 AND 2.5+0.25y<x<11.1111+4.444y) OR (20<y<200 AND 12.5+0.25y<x<5.26316+4.73684y) OR (200<y<2000) AND 125+0.25y<x<500+4.75y) OR (y>2000 AND 1250+0.25y<x<5625+4.6875y)
7th order quadrature	(y<0.02 AND 4.6<x<5.5) OR (0.02<y<0.3 AND 4.4<x<5.5) OR (0.3<y<1.0 AND 4<x<5.5) OR (1.0<y<3.6 AND 4.77778-0.77778y<x<5.5) OR (3.6<y<5.5 AND 4.77778-0.77778y<x<6.5-0.3333y) OR (5.5<y<6.5 AND x<6.5-0.3333y) OR (6.5<y<20.0 AND x<2.5+0.25y) OR (20<y<200 AND x<12.5+0.25y) OR (200<y<2000) AND x<125+0.25y) OR (y>2000 AND x<1250+0.25y)
9th order quadrature	(0.02<y<0.2 AND 4.2<x<4.4) OR (0.2<y<0.3 AND 4.0<x<4.4) OR (0.3<y<1.0 AND 4.25-y<x<4) OR (1.0<y<3.6 AND 4.25-y<x<4.77778-0.77778y) OR (3.6<y<5.5 AND x<4.77778-0.77778y)
Interpolation	(y<0.02 AND x<4.6) OR (0.02<y<0.2 AND x<4.2) OR (0.2<y<0.3 AND x<4.0) OR (0.3<y<3.6 AND x<4.25-y)

Table A.1: Describes regions of quadrature in *x* and *y* used by the routine presented in this paper for the speed dependent calculation when *S*<0.04.

	S<0.1
3rd order quadrature	(0<y<4.0 AND x>26.6668+23.3334y) OR (4.0<y<15 AND x>52.7277+21.8183y) OR (15<y<90 AND x>150+23.3334y) OR (90<y<200 AND x>450+22.5) OR (200<y<2000 AND x>200+24y) OR (y>2000 AND x>2000+24y)
5th order quadrature	y<0.65 AND 6<x<26.6668+23.3334y) OR (0.65<y<2.1 AND 4.59692+1.53846y<x<26.6668+23.3334y) OR (2.1<y<4.0 AND 5.6471+.8824y<x<26.6668+23.3334y) OR (4.0<y<15 AND 2.25+2.25y<x<52.7277+21.8183y) OR (15<y<90 AND 37.1428+2.14286y<x<150+23.3334y) OR (90<y<200 AND 37.1428+2.14286y<x<450+22.5) OR (200<y<2000 AND 386.667+2.1333y<x<200+24y) OR (y>2000 AND 3785.72+2.21429y<x<2000+24y)
7th order quadrature	(y<0.02 AND 4.6<x<6) OR (0.02<y<0.65 AND 4.5<x<6) OR (0.65<y<2.1 AND 4.75<x<4.59692+1.53846y) OR (2.1<y<4.0 AND 4.75<x<5.6471+.8824y) OR (4.0<y<15 AND 4.0+0.5y<x<2.25+2.25y) OR (15<y<200 AND 12.0+0.4y<x<37.1428+2.14286y) OR (200<y<2000 AND 28.5714+0.428572y<x<386.667+2.1333y) OR (y>2000 AND 1122.81+0.438596y<x<3785.72+2.21429y)
9th order quadrature	(0.02<y<0.3 AND 4<x<4.5) OR (4.0<y<6.0 AND 3.9375-0.25y<x<4.0+0.5y) OR (6<y<15 AND 2.5<x<4.0+0.5y) OR (15<y<90 AND 2.5<x<12.0+0.4y) OR (90<y<200 AND x<12.0+0.4y) OR (200<y<2000 AND x<28.5714+0.428572y) OR (y>2000 AND x<1122.81+0.438596y)
11th order quadrature	6<y<90 AND x<2.5
13th order quadrature	(0.3<y<0.65 AND 4.25-y<x<4.5) OR (0.65<y<4.0 AND 4.25-y<x<4.75) OR (4.0<y<6.0 AND x<3.9375-0.25y)
Interpolation	(y<0.02 AND x<4.6) OR (0.02<y<0.3 AND x<4) OR (0.3<y<4.0 AND x<4.25-y)

Table A.2: Describes regions of quadrature in *x* and *y* used by the routine presented in this paper for the speed dependent calculation when 0.04<*S*<0.1.

	S<0.16
3rd order quadrature	(y<0.75 AND x>18+24.6154y) OR (0.75<y<1.25 AND x>2.5+50y) OR (1.25<y<2.75 AND x>13.6664+46.6657y) OR (2.75<y<7.5 AND x>17.7774+48.8878y) OR (7.5<y<40 AND x>126.664+46.6657y)
5th order quadrature	(y<0.75 AND 6.0+1.75y<x<18+24.6154y) OR (0.75<y<1.25 AND 6.0+1.75y<x<2.5+50y) OR (1.25<y<2.75 AND 12<x<13.6664+46.6657y) OR (2.75<y<5.5 AND 4+3.2y<x<17.7774+48.8878y) OR (5.5<y<7.5 AND 4y-2<x<17.7774+48.8878y) OR (7.5<y<40 AND 12.4138+3.44828y<x<126.664+46.6657y) OR (40<y<200 AND x>59.3548+3.35484y) OR (200<y<2000 AND x>640.626+3.4375y) OR (y>2000 AND x>8125+3.4375y)
7th order quadrature	(y<0.2 AND 4.6+y<x<6.0+1.75y) OR (0.2<y<1.25 AND 4.0+y<x<6.0+1.75y) OR (1.25<y<2.1 AND 4.25+0.75y<x<12) OR (2.1<y<2.75 AND 4.875+0.75y<x<12) OR (2.75<y<4.0 AND 4.5+y<x<4+3.2y) OR (4.0<y<5.5 AND 4.5+y<x<4+3.2y) OR (5.5<y<7.5 AND 4.5+y<x<4y-2) OR (7.5<y<40 AND 27+y<x<12.4138+3.44828y) OR (40<y<200 AND 25.8065+1.54839y<x<59.3548+3.35484y) OR (200<y<2000 AND 200+1.6y<x<640.626+3.4375y) OR (y>2000 AND 2800+1.6y<x<8125+3.4375y)
9th order quadrature	(y<0.2 AND 4.6<x<4.6+y) OR (0.2<y<0.75 AND 4.0<x<4.0+y) OR (40<y<200 AND 12.9032+0.774192y<x<25.8065+1.54839y) OR (200<y<2000 AND 103.03+.787879y<x<200+1.6y)
11th order quadrature	(7.5<y<40 AND 3.41937+0.225807y<x<27+y) OR (40<y<200 AND 5.0+0.28571y<x<12.9032+0.774192y) OR (200<y<2000 AND 49.0423+0.28393y<x<103.03+.787879y) OR (y>2000 AND 642.857+0.285714y<x<2800+1.6y)
13th order quadrature	(0.75<y<1.25 AND 4.25-y<x<4+y) OR (4.0<y<4.8 AND 2.5<x<4.5+y AND S<0.15) OR (4.8<y<5.5 AND 2.5<x<4.5+y) OR (5.5<y<7.5 AND 1.75<x<4.5+y) OR (y>2000 AND 62.5+0.04375y<x<642.857+0.285714y)
15th order quadrature	(1.25<y<2.1 AND 4.25-y<x<4.25+0.75y) OR (2.1<y<2.75 AND 4.25-y<x<4.875+0.75y) OR (2.75<y<4.0 AND 2.5<x<4.5+y) OR (7.5<y<40 AND x<3.41937+0.225807y) OR (40<y<200 AND x<5+0.28571y) OR (200<y<2000 AND x<49.0423+0.28393y) OR (y>2000 AND x<62.5+0.04375y)

17^{th} order quadrature	(4.0<y<4.8 AND 0.6<x<4.5+y AND S>0.15) OR (4.0<y<4.8 AND 0<x<2.5 AND S<0.15) OR (4.8<y<5.5 AND x<2.5) OR (5.5<y<7.5 AND x<1.75)
Interpolation	(y<0.2 AND x<4.6) OR (0.2<y<0.75 AND x<4.0) OR (0.75<y<2.75 AND x<4.25-y) OR (2.75<y<4.0 AND x<2.5) OR (4.0<y<4.8 AND S>0.15 AND x<0.6)

Table A.3 Describes regions of quadrature in *x* and *y* used by the routine presented in this paper for the speed dependent calculation when 0.1<*S*<0.16.

	S<0.24
3rd order quadrature	(y<2.5 AND x>20+85y) OR (2.5<y<4 AND x>31.6667+83.3333y) OR (4<y<8.75 AND x>49.9982+87.4967y) OR (8.75<y<30 AND x>242.105+84.2105y) OR
5th order quadrature	(y<2.5 AND 6.5+5y<x<20+85y) OR (2.5<y<4 AND 8.8889+4.4444y<x<31.6667+83.3333y) OR (4<y<8.75 AND 8.52942+4.11765y<x<49.9982+87.4967y) OR (8.75<y<30 AND 12.6316+4.73684y<x<242.105+84.2105y) OR (30<y<200 AND x>59.2594+4.81484y)
7th order quadrature	(y<0.02 AND 4.6<x<6.5+5y) OR (0.02<y<2.5 AND 4.5+1.875y<x<6.5+5y) OR (30<y<200 AND 60.3447+3.10345y<x<59.2594+4.81484y) OR (200<y<2000 AND x>562.5+3.125y) OR (y>2000 AND x>5357.15+3.21429y)
9th order quadrature	(2.5<y<4 AND 2.45455+1.81818y<x<8.8889+4.4444y) OR (4<y<8.75 AND 3.66667+1.66667<x<8.52942+4.11765y) OR (8.75<y<30 AND 11.0588+1.41176y<x<12.6316+4.73684y) OR (30<y<200 AND 26.6667+1.62963y<x<60.3447+3.10345y) OR (200<y<2000 AND 275+1.625y<x<562.5+3.125y) OR (y>2000 AND 2750+1.625y<x<5357.15+3.21429y)
13th order quadrature	(0.02<y<0.2 AND 3.9<x<4.5+1.875y) OR (0.2<y<1.25 AND 3.75<x<4.5+1.875y) OR (8.75<y<30 AND 3.52941+0.705882y<x<11.0588+1.41176y) OR (30<y<200 AND 10.6667+0.73333y<x<26.6667+1.62963y) OR (200<y<2000 AND 118.182+.727273y<x<275+1.625y) OR (y>2000 AND 1363.64+.727273y<x<2750+1.625y)
15th order quadrature	(1.25<y<2.5 AND 3.82353-0.294118y<x<4.5+1.875y) OR (200<y<2000 AND 119.355+0.516129y<x<118.182+.727273y) OR (y>2000 AND 1500+0.5y<x<1363.64+.727273y)
17th order quadrature	(2.5<y<4 AND 3.82353-0.294118y<x<2.45455+1.81818y) OR (4<y<8.75 AND 3.82353-0.294118y<x<3.66667+1.66667) OR (8.75<y<30 AND x<3.52941+0.705882y) OR (30<y<200 AND x<10.6667+0.73333y) OR (200<y<2000 AND x<119.355+0.516129y) OR (y>2000 AND x<1500+0.5y)
Interpolation	(y<0.02 AND x<4.6) OR (0.02<y<0.2 AND x<3.9) OR (0.2<y<1.25 AND x<3.75) OR (1.25<y<8.75 AND x<3.82353-0.294118y) OR

Table A.4: Describes regions of quadrature in *x* and *y* used by the routine presented in this paper for the speed dependent calculation when 0.16<*S*<0.24.

	Number of tables	Regions 0.0<S<0.16	Regions 0.16<S<0.24	Total Space
Taylor Series 0.1 spacing in x, y 0.03 spacing in S	5	0.7<x<3.75, .3<y<2.1 0.1<x<3.2, 2.1<y<4	0.7<x<3.75, .3<y<2.1 0.1<x<3.2, 2.1<y<4 0.1<x<2.7, 4<y<8.75	5.81 MB
Taylor Series 0.05 spacing in x, y 0.03 spacing in S	4	0.2<x<4.0, 0.2<y<0.3 0.2<x<0.7, 0.3<y<0.65 0.05<x<0.7 0.65<y<2.1	0.2<x<4.0, 0.2<y<0.3 0.2<x<0.7, 0.3<y<0.65 0.05<x<0.7 0.65<y<2.1	2.52 MB
Taylor Series 0.02 spacing in x, y 0.03 spacing in S	4	0.02<x<5.0, 0.02<y<0.2 0.02<x<0.2, 0.2<y<0.64	0.02<x<5.0, 0.02<y<0.2 0.02<x<0.2, 0.2<y<0.64	9.45 MB
Lagrange Polynomials 0.005 spacing in x, y 0.01 spacing in S	3	Imaginary only: 0.0<x<0.02, 0.0<y<0.65 0.0<x<0.05, 0.65<y<2.1 0.0<x<0.1, 2.1<y<4.0 Real and Imaginary: 0.0<x<5.0, 0.0<y<0.02	Imaginary only: 0.0<x<0.02, 0.0<y<0.65 0.0<x<0.05, 0.65<y<2.1 0.0<x<0.1, 2.1<y<8.75 Real and Imaginary: 0.0<x<5.0, 0.0<y<0.02	4.29 MB 2 MB
ALL	**16**	**-**	**-**	**24.1 MB**

Table A.5: Details the regions in which Taylor series expansions and Lagrange polynomial interpolation are used, the memory space required by the binary files, and the gridpoint spacing within the interpolation tables.

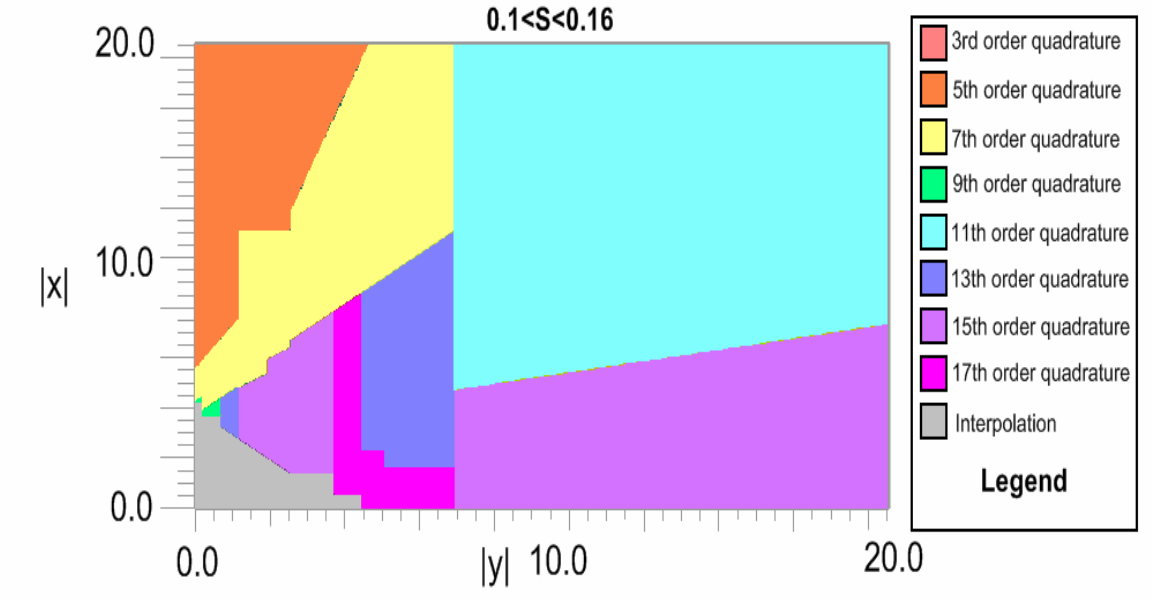

Figure A.1: Displays the exact regions of quadrature used for 0.1<S<0.16 and 0<x,y<20.

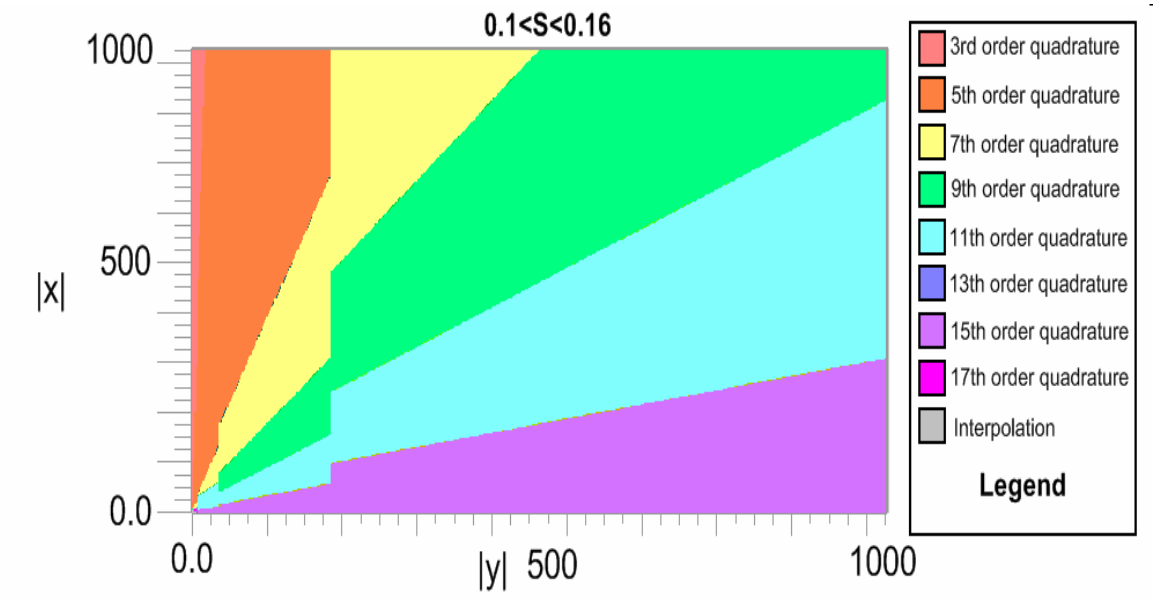

Figure A.2: Displays the exact regions of quadrature used for 0.1<S<0.16 and 0<x,y<1000.

Appendix B: Additional Accuracy Comparisons

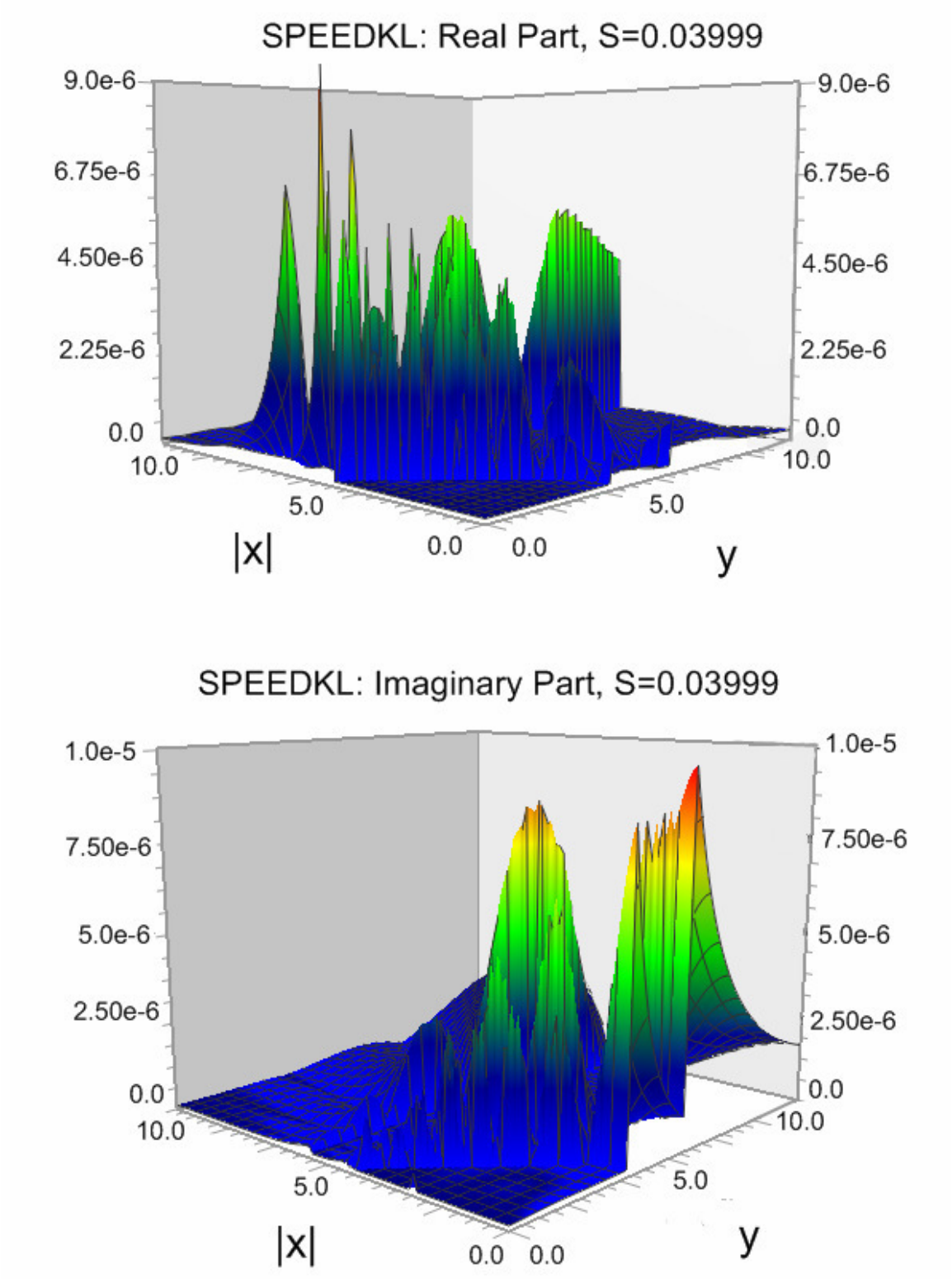

Figure B.1: Displays $\mathcal{E}$, the relative error for the real and imaginary functions calculated by the speed dependent routine presented in this paper for $0<x,y<10$ and S=0.03999.

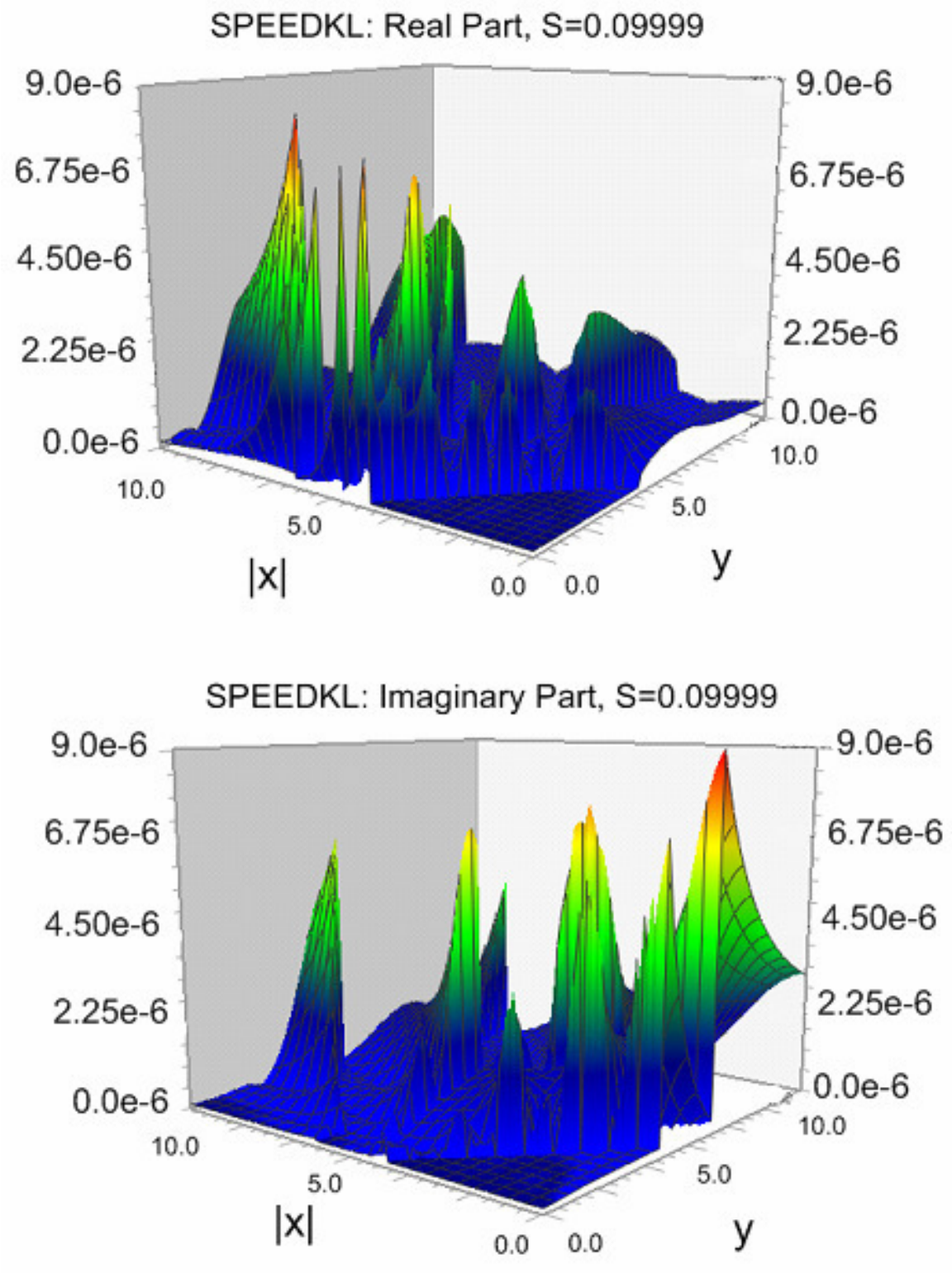

Figure B.2: Displays $\mathcal{E}$, the relative error for the real and imaginary functions calculated by the speed dependent routine presented in this paper for $0<x,y<10$ and S=0.09999.

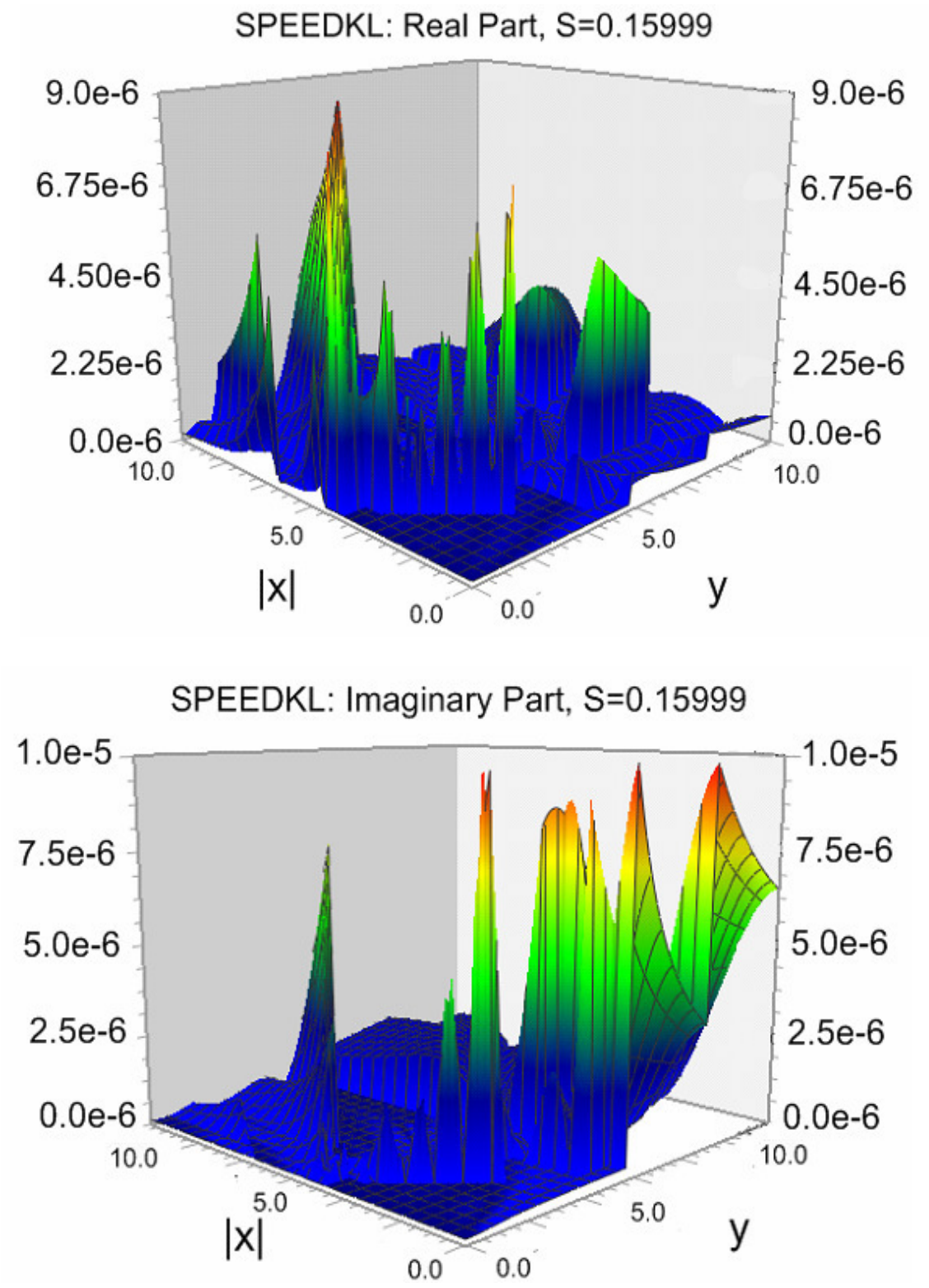

Figure B.3: Displays $\mathcal{E}$, the relative error for the real and imaginary functions calculated by the speed dependent routine presented in this paper for $0<x,y<10$ and S=0.15999.

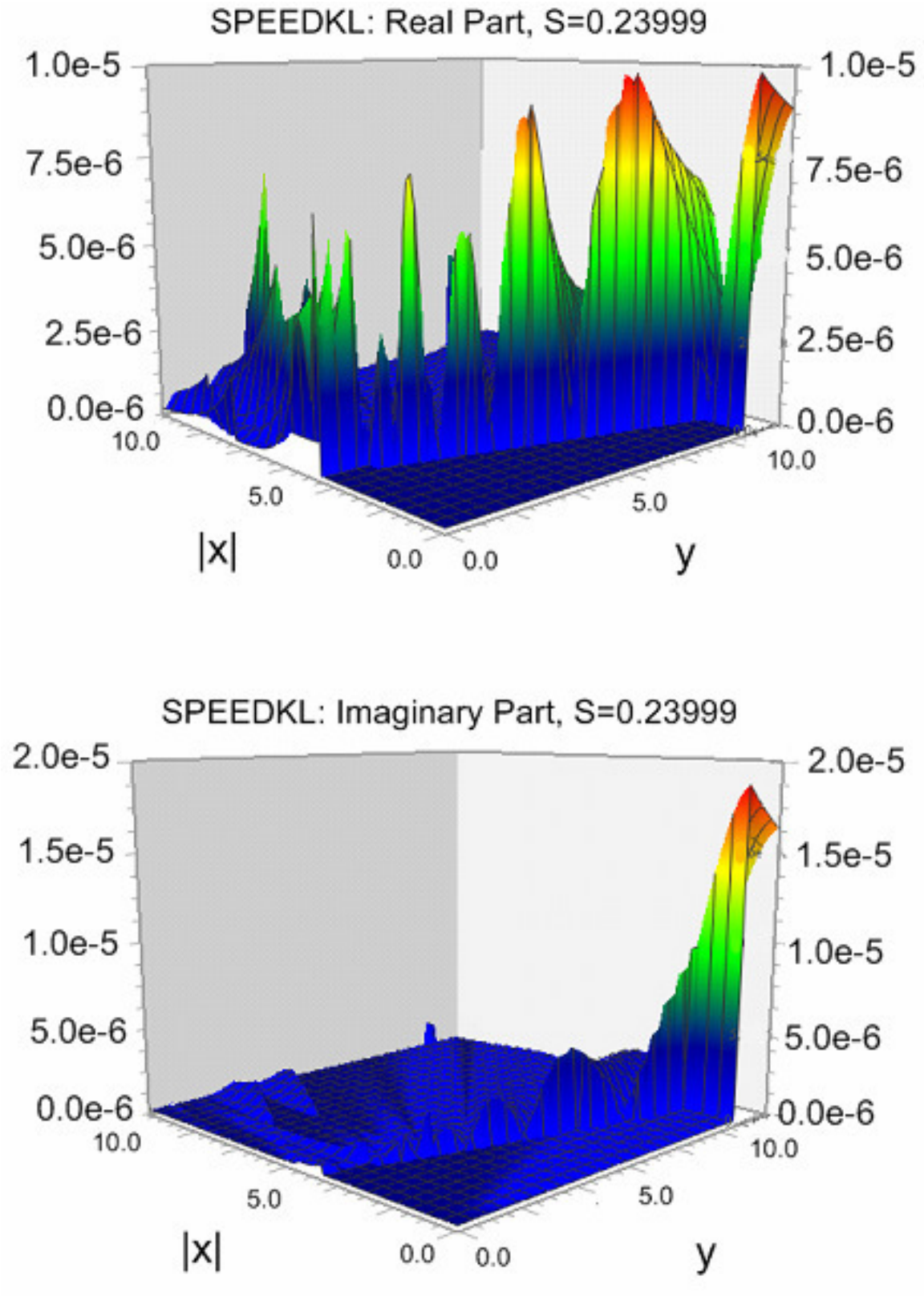

Figure B.4: Displays $\mathcal{E}$, the relative error for the real and imaginary functions calculated by the speed dependent routine presented in this paper for $0<x,y<10$ and S=0.03999.

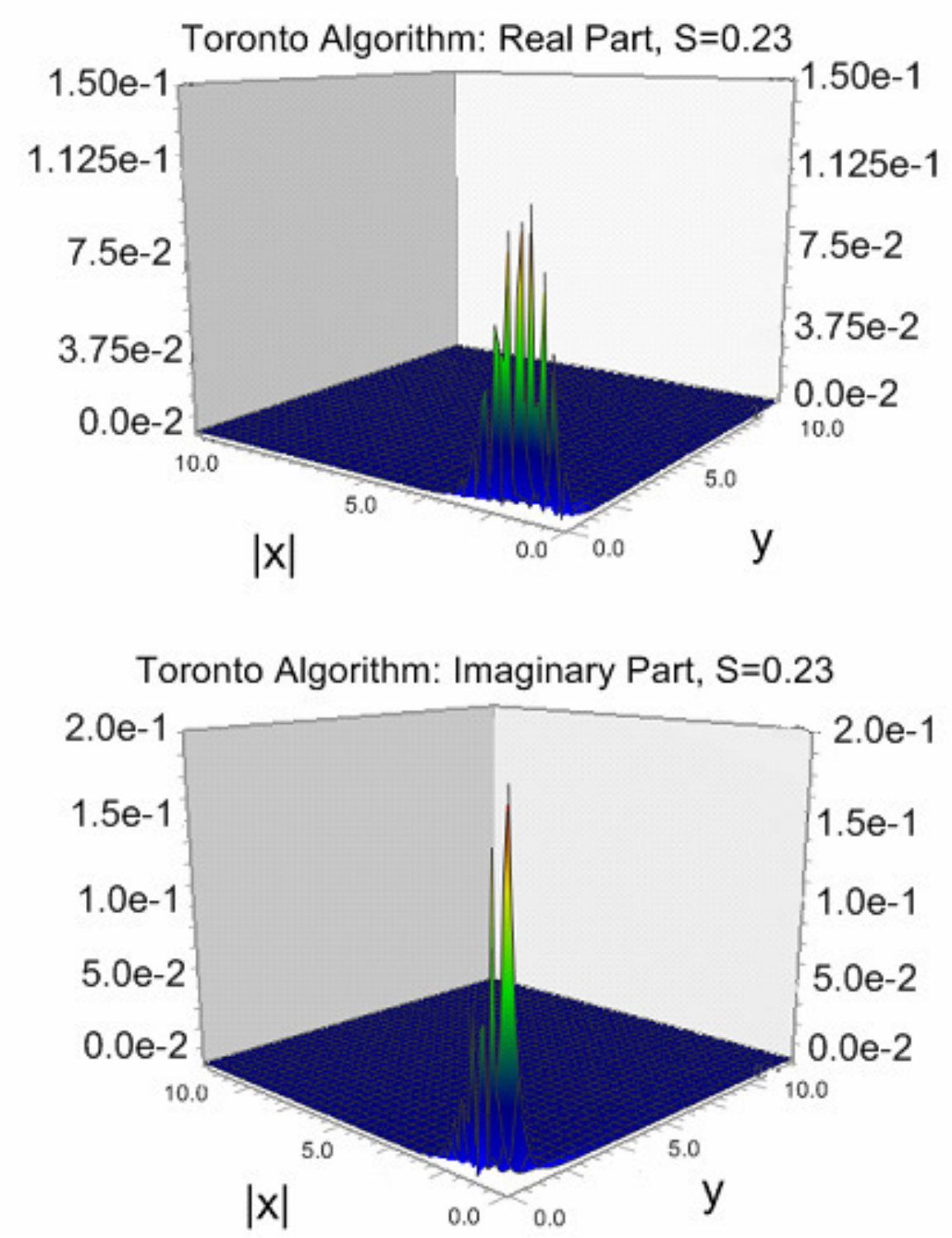

Figure B.5: Displays $\mathcal{E}$, the relative error for the real and imaginary functions calculated by the Toronto algorithm for $0<x,y<10$ and S=0.23.

Appendix C: Program Code

The actual program code contains a total of 14 subroutines and 2 modules. This excerpt contains 6 subroutines and 2 modules. Subroutines which are similar in structure to those present in the excerpt have been omitted, but are listed here underneath the subroutines which they resemble.*

*Certain subroutines and modules used for input/output tasks and which were not written specifically to calculate the speed dependent lineshape have been omitted from this list.

```
      SUBROUTINE SPEEDKL(XSTART1,XDELTA1,Y1,S1,SPEEDRE,SPEEDIM,NPTS1)

      USE CONSTANT
      USE SPEEDDEPVAR
!*********************************************************************
!
!     SPEEDKL - Program calculating the real part of the speed
!               dependent lineshape profile
!               (2/pi)*V*exp(-V^2)*arctan((X+V)/(Y(1+S(V^2-1.5)))
!               and the imaginary part
!               (1/pi)*V*exp(-V^2)*ln(1+((X+V)/(Y(1+S(V^2-1.5)))^2)
!               with an accuracy of 1E-5.  Uses interpolation for
!               small x, small y, and Gauss-Hermite quadrature
!               elsewhere.
!
!      Revised 8 April 2007 - Kendra L. Letchworth
!
!*********************************************************************
      IMPLICIT NONE

      INTEGER*4 NPTS1  ! Number of different equally spaced x values at
                               ! which the function is to be evaluated
      REAL*8 XDELTA1 !Spacing between x values for function evaluations
      REAL*8 XSTART1               ! First x value for the computations
      REAL*8 Y1     ! Voigt y parameter equal to the square root of the
                     ! natural log of 2 times the ratio of the Lorentz
                                          !width to the Doppler width
      REAL*8 S1                          ! Speed dependent parameter ~=.1
      REAL*8 SPEEDRE(NPTS1)                 !Output real function values
      REAL*8 SPEEDIM(NPTS1)              !Output imaginary function values

        NPTS=NPTS1
        XDELTA=XDELTA1
        XSTART=XSTART1
        Y=Y1
        S=S1
        IF (S.LT.0.24D0) THEN
        !Quadrature, Taylor Series expansion,
        !and Lagrange polynomial interpolation
          IF (S.LT.0.04D0) THEN
            IF (Y.LT.3.6D0) THEN
              IF (Y.LT.0.3D0) THEN
                IF (Y.LT.0.02D0) THEN
                !Interpolation reg. 1 x<4.6
                !7th order quadrature x<5.5,
                !5th order quadrature x-4.72728y<13,
                !3rd order quadrature elsewhere
                  CALL QUADINTERP(1,7,5,3,3,4.6D0,5.5D0,13.D0+&
                  4.72728D0*Y,2000.D0,SPEEDRE,SPEEDIM,NPTS1)
                ELSE IF (Y.LT.0.2D0) THEN
                !Interpolation reg. 2 x<4.2
                !9th order quadrature x<4.4
                !7th order quadrature x<5.5,
                !5th order quadrature x-4.72728y<9.65,
                !3rd order quadrature elsewhere
                  CALL QUADINTERP(2,9,7,5,3,4.2D0,4.4D0,5.5D0,9.65D0+&
                  4.72728D0*Y,SPEEDRE,SPEEDIM,NPTS1)
                ELSE
                !Interpolation reg. 3 x<4
                !9th order quadrature x<4.4
```

```
      !7th order quadrature x<5.5,
      !5th order quadrature 0.211538x-y<1.96154,
      !3rd order quadrature elsewhere
        CALL QUADINTERP(3,9,7,5,3,4.D0,4.4D0,5.5D0,&
        9.27275D0+4.72728D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      END IF
    ELSE
      IF (Y.LT.0.65D0) THEN
      !Interpolation reg. 4 x+y<4.25
      !9th order quadrature x<4
      !7th order quadrature x<5.5,
      !5th order quadrature 0.211538x-y<1.96154,
      !3rd order quadrature elsewhere
        CALL QUADINTERP(4,9,7,5,3,4.25D0-Y,4.D0,5.5D0,&
        9.27275D0+4.72728D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (Y.LT.1.D0) THEN
      !Interpolation reg. 5 x+y<4.25
      !9th order quadrature x<4
      !7th order quadrature x<5.5,
      !5th order quadrature 0.211538x-y<1.96154,
      !3rd order quadrature elsewhere
        CALL QUADINTERP(5,9,7,5,3,4.25D0-Y,4.D0,5.5D0,&
        9.27275D0+4.72728D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (Y.LT.2.1D0) THEN
      !Interpolation reg. 5 x+y<4.25
      !9th order quadrature 1.28571x+y<6.14286
      !7th order quadrature x<5.5,
      !5th order quadrature 0.211538x-y<1.96154,
      !3rd order quadrature elsewhere
        CALL QUADINTERP(5,9,7,5,3,4.25D0-Y,4.7778D0-&
        .77778D0*Y,5.5D0,9.27275D0+4.72728D0*Y,SPEEDRE,&
        SPEEDIM,NPTS1)
      ELSE
      !Interpolation reg. 6 x+y<4.25
      !9th order quadrature 1.28571x+y<6.14286
      !7th order quadrature x<5.5,
      !5th order quadrature 0.211538x-y<1.96154,
      !3rd order quadrature elsewhere
        CALL QUADINTERP(6,9,7,5,3,4.25D0-Y,4.7778D0-&
        .77778D0*Y,5.5D0,9.27275D0+4.72728D0*Y,SPEEDRE,&
        SPEEDIM,NPTS1)
      END IF
    END IF
  ELSE IF (Y.LT.5.5D0) THEN
  !9th order quadrature 1.28571x+y<6.14286,
  !7th order quadrature y+3x<19.5,
  !5th order quadrature 0.211538x-y<1.96154,
  !3rd order quadrature elsewhere
    CALL QUADRATURE4(9,7,5,3,4.7778D0-.77778D0*Y,6.5D0-&
    .33333D0*Y,9.27275D0+4.72728D0*Y,SPEEDRE,SPEEDIM,NPTS1)
  ELSE IF (Y.LT.6.5D0) THEN
  !7th order quadrature y+3x<19.5,
  !5th order quadrature 0.211538x-y<1.96154,
  !3rd order quadrature elsewhere
    CALL QUADRATURE3(7,5,3,6.5D0-.33333D0*Y,9.27275D0&
    +4.72728D0*Y,SPEEDRE,SPEEDIM,NPTS1)
  ELSE IF (Y.LT.20.D0) THEN
  !7th order quadrature 4x-y<10,
  !5th order quadrature 0.225x-y<2.5,
  !3rd order quadrature elsewhere
```

```
      CALL QUADRATURE3(7,5,3,2.5D0+.25D0*Y,11.1111D0+&
      4.4444D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    ELSE IF (Y.LT.200.D0) THEN
    !7th order quadrature 4x-y<50,
    !5th order quadrature 0.211111x-y<1.1111,
    !3rd order quadrature elsewhere
      CALL QUADRATURE3(7,5,3,12.5D0+.25D0*Y,5.26316D0+&
      4.73684D0*Y,SPEEDRE,SPEEDIM,NPTS1) !COULD BE 52.6
    ELSE IF (Y.LT.2000.D0) THEN
    !7th order quadrature 4x-y<500,
    !5th order quadrature 0.210526x-y<105.263,
    !3rd order quadrature elsewhere
      CALL QUADRATURE3(7,5,3,125.D0+.25D0*Y,500.D0+&
      4.75D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    ELSE
    !7th order quadrature 4x-y<5000,
    !5th order quadrature 0.213333x-y<1200,
    !3rd order quadrature elsewhere
      CALL QUADRATURE3(7,5,3,1250.D0+.25D0*Y,5625.D0+&
      4.6875D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    END IF
  ELSE IF (S.LT.0.1D0) THEN
    IF (Y.LT.4.0D0) THEN
      IF (Y.LT.0.3D0) THEN
        IF (Y.LT.0.02D0) THEN
        !Interpolation reg. 1 x<4.6
        !7th order quadrature x<6
        !5th order quadrature .042857x-y<1.14286
        !3rd order quadrature elsewhere
          CALL QUADINTERP(1,7,5,3,3,4.6D0,6.D0,26.6668D0+&
          23.3334D0*Y,2000.D0,SPEEDRE,SPEEDIM,NPTS1)
        ELSE IF (Y.LT.0.2D0) THEN
        !Interpolation reg. 2 x<4
        !9th order quadrature x<4.5
        !7th order quadrature x<6,
        !5th order quadrature .042857x-y<1.14286
        !3rd order quadrature elsewhere
          CALL QUADINTERP(2,9,7,5,3,4.D0,4.5D0,6.D0&
          ,26.6668D0+23.3334D0*Y,SPEEDRE,SPEEDIM,NPTS1)
        ELSE
        !Interpolation reg. 3 x<4
        !9th order quadrature x<4.5
        !7th order quadrature x<6,
        !5th order quadrature .042857x-y<1.14286
        !3rd order quadrature elsewhere
          CALL QUADINTERP(3,9,7,5,3,4.D0,4.75D0,6.D0&
          ,26.6668D0+23.3334D0*Y,SPEEDRE,SPEEDIM,NPTS1)
        END IF
      ELSE
        IF (Y.LT.0.65D0) THEN
        !Interpolation reg. 4 x+y<4.25
        !13th order quadrature x<4.5
        !7th order quadrature x<6,
        !5th order quadrature .042857x-y<1.14286
        !3rd order quadrature elsewhere
          CALL QUADINTERP(4,13,7,5,3,4.25D0-Y,4.5D0,6.D0&
          ,26.6668D0+23.3334D0*Y,SPEEDRE,SPEEDIM,NPTS1)
        ELSE IF (Y.LT.2.1D0) THEN
        !Interpolation reg. 5 x+y<4.25
        !13th order quadrature x<4.75
```

```
      !7th order quadrature .65x-y<2.975,
      !5th order quadrature .042857x-y<1.14286
      !3rd order quadrature elsewhere
        CALL QUADINTERP(5,13,7,5,3,4.25D0-Y,4.75D0,4.59692D0+&
        1.53846D0*Y,26.6668D0+23.3334D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      ELSE
      !Interpolation reg. 6 x+y<4.25
      !13th order quadrature x<4.75
      !7th order quadrature 1.133333x-y<6.4,
      !5th order quadrature .042857x-y<1.14286
      !3rd order quadrature elsewhere
        CALL QUADINTERP(6,13,7,5,3,4.25D0-Y,4.75D0,5.6471D0+&
        .8824D0*Y,26.6668D0+23.3334D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      END IF
    END IF
  ELSE IF (Y.LT.6.D0) THEN
  !13th order quadrature 4x+y<15.75
  !9th order quadrature 2x-y<8
  !7th order quadrature .4444444x-y<1,
  !5th order quadrature .045833x-y<2.41666667
  !3rd order quadrature elsewhere
    CALL QUADRATURE5(13,9,7,5,3,3.9375D0-.25D0*Y,4.D0+&
    .5D0*Y,2.25D0*(1.D0+Y),52.7277D0+21.8183D0*Y,SPEEDRE,&
    SPEEDIM,NPTS1)
  ELSE IF (Y.LT.15.D0) THEN
  !11th order quadrature x<2.5,
  !9th order quadrature 2x-y<8,
  !7th order quadrature .4444444x-y<1,
  !5th order quadrature .045833x-y<2.41666667
  !3rd order quadrature elsewhere
    CALL QUADRATURE5(11,9,7,5,3,2.5D0,4.D0+.5D0*Y,2.25D0*&
    (1.D0+Y),52.7277D0+21.8183D0*Y,SPEEDRE,SPEEDIM,NPTS1)
  ELSE IF (Y.LT.90.D0) THEN
  !11th order quadrature x<2.5,
  !9th order quadrature 2.5x-y<30
  !7th order quadrature .4666667x-y<17.3333,
  !5th order quadrature .042857x-y<6.42857
  !3rd order quadrature elsewhere
    CALL QUADRATURE5(11,9,7,5,3,2.5D0,12.D0+.4D0*Y&
    ,37.1428D0+2.14286D0*Y,150.D0+23.3334D0*Y,SPEEDRE&
    ,SPEEDIM,NPTS1)
  ELSE IF (Y.LT.200.D0) THEN
  !9th order quadrature 2.5x-y<30,
  !7th order quadrature .4666667x-y<17.3333,
  !5th order quadrature .04444444x-y<20,
  !3rd order quadrature elsewhere
    CALL QUADRATURE4(9,7,5,3,12.D0+.4D0*Y,37.1428D0+&
    2.14286D0*Y,450.D0+22.5D0*Y,SPEEDRE,SPEEDIM,NPTS1)
  ELSE IF (Y.LT.2000.D0) THEN
  !9th order quadrature 2.3333333x-y<66.66667,
  !7th order quadrature .46875x-y<181.25,
  !5th order quadrature .041666667x-y<8.3333,
  !3rd order quadrature elsewhere
    CALL QUADRATURE4(9,7,5,3,28.5714D0+.428572D0*Y,&
    386.667D0+2.133333D0*Y,200.D0+24.D0*Y,SPEEDRE,&
    SPEEDIM,NPTS1)
  ELSE
  !9th order quadrature 2.28x-y<2560,
  !7th order quadrature .451613x-y<1709.68,
  !5th order quadrature .041666667x-y<83.3333,
```

```
  !3rd order quadrature elsewhere
    CALL QUADRATURE4(9,7,5,3,1122.81D0+.438596D0*Y,&
    3785.72D0+2.21429D0*Y,2000.D0+24.D0*Y,SPEEDRE,&
    SPEEDIM,NPTS1)
  END IF
ELSE IF (S.LT.0.16D0) THEN
  IF (Y.LT.4.D0) THEN
    IF (Y.LT.0.75D0) THEN
      IF (Y.LT.0.02D0) THEN
      !Interpolation reg. 1 x<4.6,
      !9th order quadrature x-y<4.6,
      !7th order quadrature .571429x-y<3.42857,
      !5th order quadrature .040625x-y<.73125
      !3rd order quadrature elsewhere
        CALL QUADINTERP(1,9,7,5,3,4.6D0,4.6D0+Y,6.D0+&
        1.75D0*Y,18.D0+24.6154D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (Y.LT.0.2D0) THEN
      !Interpolation reg. 2 x<4.6,
      !9th order quadrature x-y<4.6,
      !7th order quadrature .571429x-y<3.42857,
      !5th order quadrature .040625x-y<.73125
      !3rd order quadrature elsewhere
        CALL QUADINTERP(2,9,7,5,3,4.6D0,4.6D0+Y,6.D0+&
        1.75D0*Y,18.D0+24.6154D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (Y.LT.0.3D0) THEN
      !Interpolation reg. 3 x<4,
      !9th order quadrature x-y<4,
      !7th order quadrature .571429x-y<3.42857,
      !5th order quadrature .040625x-y<.73125
      !3rd order quadrature elsewhere
        CALL QUADINTERP(3,9,7,5,3,4.D0,4.D0+Y,6.D0+&
        1.75D0*Y,18.D0+24.6154D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (Y.LT.0.65D0) THEN
      !Interpolation reg. 4 x<4,
      !9th order quadrature x-y<4,
      !7th order quadrature .571429x-y<3.42857,
      !5th order quadrature .040625x-y<.73125
      !3rd order quadrature elsewhere
        CALL QUADINTERP(4,9,7,5,3,4.D0,4.D0+Y,6.D0+&
        1.75D0*Y,18.D0+24.6154D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      ELSE
      !Interpolation reg. 5 x<4,
      !9th order quadrature x-y<4,
      !7th order quadrature .571429x-y<3.42857,
      !5th order quadrature .040625x-y<.73125
      !3rd order quadrature elsewhere
        CALL QUADINTERP(5,9,7,5,3,4.D0,4.D0+Y,6.D0+&
        1.75D0*Y,18.D0+24.6154D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      END IF
    ELSE
      IF (Y.LT.1.25D0) THEN
      !Interpolation reg. 5 x+y<4.25,
      !13th order quadrature x-y<4,
      !7th order quadrature .571429x-y<3.42857,
      !5th order quadrature .02x-y<0.05
      !3rd order quadrature elsewhere
        CALL QUADINTERP(5,13,7,5,3,4.25D0-Y,4.D0+Y,6.D0+&
        1.75D0*Y,2.5D0+50.D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (Y.LT.2.1D0) THEN
      !Interpolation reg. 5 x+y<4.25,
```

```
    !15th order quadrature 1.333x-y<5.66667,
    !7th order quadrature x<12,
    !5th order quadrature .021429x-y<0.292857
    !3rd order quadrature elsewhere
      CALL QUADINTERP(5,15,7,5,3,4.25D0-Y,4.25D0+.75D0*Y&
      ,12.D0,13.6664D0+46.6657D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    ELSE IF (Y.LT.2.75D0) THEN
    !Interpolation reg. 6 x+y<4.25,
    !15th order quadrature 1.333x-y<6.5,
    !7th order quadrature x<12,
    !5th order quadrature .021429x-y<0.292857
    !3rd order quadrature elsewhere
      CALL QUADINTERP(6,15,7,5,3,4.25D0-Y,4.875D0+.75D0*Y&
      ,12.D0,13.6664D0+46.6657D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    ELSE
    !Interpolation reg. 6 x<2.5,
    !13th order quadrature x-y<4.5,
    !7th order quadrature .3125x-y<1.25
    !5th order quadrature .020455x-y<.36363636
    !3rd order quadrature elsewhere
    CALL QUADINTERP(6,15,7,5,3,2.5D0,4.5D0+Y,4.D0+3.2D0*&
    Y,17.7774D0+48.8878D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    END IF
  END IF
ELSE IF ((S.GT.0.15D0).AND.(Y.LT.4.8D0)) THEN
  !Interpolation reg. 7 x<.6
  !17th order quadrature x-y<4.5,
  !7th order quadrature .3125x-y<1.25
  !5th order quadrature .020455x-y<.36363636
  !3rd order quadrature elsewhere
    CALL QUADINTERP(7,17,7,5,3,.6D0,4.5D0+Y,4.D0+3.2D0*&
    Y,17.7774D0+48.8878D0*Y,SPEEDRE,SPEEDIM,NPTS1)
ELSE IF (Y.LT.5.5D0) THEN
  !17th order quadrature x<2.5,
  !13th order quadrature x-y<4.5,
  !7th order quadrature .3125x-y<1.25
  !5th order quadrature .020455x-y<.36363636
  !3rd order quadrature elsewhere
    CALL QUADRATURE5(17,13,7,5,3,2.5D0,4.5D0+Y,4.D0+&
    3.2D0*Y,17.7774D0+48.8878D0*Y,SPEEDRE,SPEEDIM,NPTS1)
ELSE IF (Y.LT.7.5D0) THEN
  !17th order quadrature x<1.75,
  !13th order quadrature x-y<4.5,
  !7th order quadrature  y-.25x>.5
  !5th order quadrature .020455x-y<.36363636
  !3rd order quadrature elsewhere
    CALL QUADRATURE5(17,13,7,5,3,1.75D0,4.5D0+Y,4.D0*Y-&
    2.D0,17.7774D0+48.8878D0*Y,SPEEDRE,SPEEDIM,NPTS1)
ELSE IF (Y.LT.40.D0) THEN
  !15th order quadrature 4.42857x-y<15.1429,
  !11th order quadrature x-y<27,
  !7th order quadrature .29x-y<3.6
  !5th order quadrature .021429x-y<2.71429
  !3rd order quadrature elsewhere
    CALL QUADRATURE5(15,11,7,5,3,3.41937D0+.225807D0*Y,&
    27.D0+Y,12.4138D0+3.44828D0*Y,126.664D0+46.6657D0*Y,&
    SPEEDRE,SPEEDIM,NPTS1)
ELSE IF (Y.LT.200.D0) THEN
  !15th order quadrature  3.5x-y<17.5,
  !11th order quadrature 1.29167x-y<16.6667
```

```
      !9th order quadrature  .645833x-y<16.6667,
      !7th order quadrature .298077x-y<17.6923
      !5th order quadrature elsewhere
        CALL QUADRATURE5(15,11,9,7,5,5.D0+.285714D0*Y,&
        12.9032D0+.774192D0*Y,25.8065D0+1.54839D0*Y,&
        59.3548D0+3.35484D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    ELSE IF (Y.LT.2000.D0) THEN
      !15th order quadrature 3.522x-y<172.727,
      !11th order quadrature 1.26923x-y<130.769
      !9th order quadrature  .625x-y<125,
      !7th order quadrature .290909x-y<186.364
      !5th order quadrature elsewhere
        CALL QUADRATURE5(15,11,9,7,5,49.0423D0+.28393D0*Y,&
        103.03D0+.787879D0*Y,200.D0+1.6D0*Y,640.626D0+&
        3.4375D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    ELSE
    !15th order quadrature 22.8571x-y<1428.57,
    !13th order quadrature 3.5x-y<2250,
    !11th order quadrature .625x-y<1750,
    !7th order quadrature .290909x-y<2363.64
    !5th order quadrature elsewhere
      CALL QUADRATURE5(15,13,11,7,5,62.5001D0+.04375D0*Y,&
      642.857D0+.285714D0*Y,2800.D0+1.6D0*Y,8125.D0+&
      3.4375D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    END IF
  ELSE
    IF (Y.LT.8.75D0) THEN
      IF (Y.LT.0.65D0) THEN
        IF (Y.LT.0.02D0) THEN
        !Interpolation reg. 1 x<4.6,
        !7th order quadrature, .2x-y<1.3
        !5th order quadrature .011765x-y<.235294,
        !3rd order quadrature elsewhere
          CALL QUADINTERP(1,7,5,3,3,4.6D0,&
          6.5D0+5.D0*Y,20.D0+85.D0*Y,2000.D0,&
          SPEEDRE,SPEEDIM,NPTS1)
        ELSE IF (Y.LT.0.2D0) THEN
        !Interpolation reg. 2  x<3.9,
        !13th order quadrature .53333x-y<2.4,
        !7th order quadrature, .2x-y<1.3
        !5th order quadrature .011765x-y<.235294,
        !3rd order quadrature elsewhere
          CALL QUADINTERP(2,13,7,5,3,3.9D0,4.5D0+1.875D0*Y&
          ,6.5D0+5.D0*Y,20.D0+85.D0*Y,SPEEDRE,SPEEDIM,NPTS1)
        ELSE IF (Y.LT.0.3D0) THEN
        !Interpolation reg. 3  x<3.75,
        !13th order quadrature .53333x-y<2.4,
        !7th order quadrature, .2x-y<1.3
        !5th order quadrature .011765x-y<.235294,
        !3rd order quadrature elsewhere
          CALL QUADINTERP(3,13,7,5,3,3.75D0,4.5D0+1.875D0*Y&
          ,6.5D0+5.D0*Y,20.D0+85.D0*Y,SPEEDRE,SPEEDIM,NPTS1)
        ELSE
        !Interpolation reg. 4  x<3.75,
        !13th order quadrature .53333x-y<2.4,
        !7th order quadrature, .2x-y<1.3
        !5th order quadrature .011765x-y<.235294,
        !3rd order quadrature elsewhere
          CALL QUADINTERP(4,13,7,5,3,3.75D0,4.5D0+1.875D0*Y&
          ,6.5D0+5.D0*Y,20.D0+85.D0*Y,SPEEDRE,SPEEDIM,NPTS1)
```

```
      END IF
    ELSE IF (Y.LT.4.D0) THEN
      IF (Y.LT.1.25D0) THEN
      !Interpolation reg. 5  x<3.75,
      !13th order quadrature .53333x-y<2.4,
      !7th order quadrature, .2x-y<1.3
      !5th order quadrature .011765x-y<.235294,
      !3rd order quadrature elsewhere
        CALL QUADINTERP(5,13,7,5,3,3.75D0,4.5D0+1.875D0*Y&
        ,6.5D0+5.D0*Y,20.D0+85.D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (Y.LT.2.1D0) THEN
      !Interpolation reg. 5 3.4x+y<13,
      !15th order quadrature .53333x-y<2.4,
      !7th order quadrature, .2x-y<1.3
      !5th order quadrature .011765x-y<.235294,
      !3rd order quadrature elsewhere
        CALL QUADINTERP(5,15,7,5,3,3.82353D0-.294118D0*Y&
        ,4.5D0+1.875D0*Y,6.5D0+5.D0*Y,20.D0+85.D0*Y,&
        SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (Y.LT.2.5D0) THEN
      !Interpolation reg. 6 3.4x+y<13,
      !15th order quadrature .53333x-y<2.4,
      !7th order quadrature, .2x-y<1.3
      !5th order quadrature .011765x-y<.235294,
      !3rd order quadrature elsewhere
        CALL QUADINTERP(6,15,7,5,3,3.82353D0-.294118D0*Y&
        ,4.5D0+1.875D0*Y,6.5D0+5.D0*Y,20.D0+85.D0*Y,&
        SPEEDRE,SPEEDIM,NPTS1)
      ELSE
      !Interpolation reg. 6 3.4x+y<13
      !17th order quadrature .55x-y<1.35
      !9th order quadrature .225x-y<2
      !5th order quadrature .012x-y<.38,
      !3rd order quadrature elsewhere
        CALL QUADINTERP(6,17,9,5,3,3.82353D0-.294118D0*Y&
        ,2.45455D0+1.81818D0*Y,8.88889D0+4.4444D0*Y,&
        31.6667D0+83.3333D0*Y,SPEEDRE,SPEEDIM,NPTS1)
      END IF
    ELSE
    !Interpolation reg. 7 3.4x+y<13,
    !17th order quadrature .6x-y<2.2
    !9th order quadrature .242857x-y<2.07143
    !5th order quadrature .011429x-y<.571429,
    !3rd order quadrature elsewhere
      CALL QUADINTERP(7,17,9,5,3,3.82353D0-.294118D0*Y,&
      3.66667D0+1.66667D0*Y,8.52942D0+4.11765D0*Y,&
      49.9982D0+87.4967D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    END IF
  ELSE IF (Y.LT.30.D0) THEN
    !17th order quadrature 1.41667x-y<5,
    !13th order quadrature .708333x-y<7.83333
    !9th order quadrature .21111x-y<2.66667
    !5th order quadrature .011875x-y<2.875,
    !3rd order quadrature elsewhere
      CALL QUADRATURE5(17,13,9,5,3,3.52941D0+.705882D0*Y,&
      11.0588D0+1.41176D0*Y,12.6316D0+4.73684D0*Y,&
      242.105D0+84.2105D0*Y,SPEEDRE,SPEEDIM,NPTS1)
  ELSE IF (Y.LT.200.D0) THEN
    !17th order quadrature 1.36364x-y<14.5455,
    !13th order quadrature .613636x-y<16.3636
```

```
      !9th order quadrature .322222x-y<19.44444,
      !7th order quadrature .207692x-y<12.3077
      !5th order quadrature  elsewhere
        CALL QUADRATURE5(17,13,9,7,5,10.6667D0+.733333D0*Y,&
        26.66666D0+1.62963D0*Y,60.3447D0+3.10345D0*Y,&
        59.2594D0+4.81484D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    ELSE IF (Y.LT.2000.D0) THEN
      !17th order quadrature 1.9375x-y<231.25,
      !15th order quadrature 1.375x-y<162.5,
      !13th order quadrature .615385x-y<169.231
      !9th order quadrature .32x-y<180,
      !7th order quadrature elsewhere
        CALL QUADRATURE5(17,15,13,9,7,119.355D0+.516129D0*Y,&
        118.182D0+.727273D0*Y,275.D0+1.625D0*Y,562.5D0+&
        3.125D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    ELSE
      !17th order quadrature 2x-y<3000,
      !15th order quadrature 1.375x-y<1875,
      !13th order quadrature .615385x-y<1692.31,
      !9th order quadrature .31111x-y<1666.67,
      !7th order quadrature elsewhere
        CALL QUADRATURE5(17,15,13,9,7,1500.D0+.5D0*Y,&
        1363.64D0+.727273D0*Y,2750.D0+1.625D0*Y,&
        5357.15D0+3.21429D0*Y,SPEEDRE,SPEEDIM,NPTS1)
    END IF
  END IF
ELSE
!17th order quadrature only
  CALL QUADRATURE(17,SPEEDRE,SPEEDIM,NPTS1)
END IF
END SUBROUTINE SPEEDKL
```

```
      MODULE SPEEDDEPVAR
!***********************************************************************
!
!       SPEEDDEPVAR - Module for LABFIT which contains quadrature points
!       in an array as well as several variables involved in the
!       calculation of the function.
!
!       USED BY SPEEDREKL,QUADRATURE,QUADRATURE2,QUADRATURE3,
!               QUADRATURE4,QUADRATURE5,QUADRATURE6,QUADINTERP
!
!       8 April 2007 - Kendra L. Letchworth
!
!***********************************************************************

      INTEGER*4 NPTS   ! Number of different equally spaced x values at
                              ! which the function is to be evaluated
      REAL*8 XDELTA ! Spacing between x values for function evaluations
      REAL*8 XSTART                 ! First x value for the computations
      REAL*8 Y      ! Voigt y parameter equal to the square root of the
                      ! natural log of 2 times the ratio of the Lorentz
                                          !width to the Doppler width
      REAL*8 S                          ! Speed dependent parameter ~=.1
      REAL*8 H   ! Hard collision parameter used by Toronto, multiplied
                      ! by sqrt(ln(2)) and divided by the doppler width
      REAL*8 QPT(8,8),QW(8,8),QW0(8)

      DATA QPT/0.14999999999999997780D+01,1.D5                       ,&
               1.D5                       ,1.D5                      ,&
               1.D5                       ,1.D5                      ,&
               1.D5                       ,1.D5                      ,&
               !Holds squared points for 3 point quadrature
               0.91886116991581034963D+00,0.40811388300841890953D+01,&
               1.D5                       ,1.D5                      ,&
               1.D5                       ,1.D5                      ,&
               1.D5                       ,1.D5                      ,&
               !Holds squared points for 5 point quadrature
               0.66632590770237087874D+00,0.28007750541502565156D+01,&
               0.70328990381473719395D+01,1.D5                       ,&
               1.D5                       ,1.D5                      ,&
               1.D5                       ,1.D5                      ,&
               !Holds squared points for 7 point quadrature
               0.52352607673826911938D+00,0.21566487632690942711D+01,&
               0.51373875461767122275D+01,0.10182437613815926269D+02,&
               1.D5                       ,1.D5                      ,&
               1.D5                       ,1.D5                      ,&
               !Holds squared points for 9 point quadrature
               0.43139880714785150406D+00,0.17597536984236963331D+01,&
               0.41044653628283152003D+01,0.77467037795425577329D+01,&
               0.13457678352057580895D+02,1.D5                       ,&
               1.D5                       ,1.D5                      ,&
               !Holds squared points for 11 point quadrature
               0.36694987730837069773D+00,0.14885611336422577278D+01,&
               0.34340079684240714109D+01,0.63490679256803801422D+01,&
               0.10540469858448343388D+02,0.16820970077828384603D+02,&
               1.D5                       ,1.D5                      ,&
               !Holds squared points for 13 point quadrature
               0.31930363392063010330D+00,0.12907586229591525573D+01,&
               0.29583744586966496115D+01,0.54090315972444322767D+01,&
               0.88040795780567755457D+01,0.13468535743251482728D+02,&
               0.20249916365870880952D+02,1.D5                       ,&
```

```
        !Holds squared points for 15 point quadrature
        0.28263364811659907883D+00,0.11398738015816138880D+01,&
        0.26015248434060294080D+01,0.47241145375277913132D+01,&
        0.76052562992316135038D+01,0.11417182076545831393D+02,&
        0.16499410797655816197D+02,0.23730003995934712435D+02/
        !Holds squared points for 17 point quadrature

DATA QW/0.18806319451591876901D+00,1.D5                      ,&
       1.D5                       ,1.D5                      ,&
       1.D5                       ,1.D5                      ,&
       1.D5                       ,1.D5                      ,&
       !Holds weights divided by pi for 3 point quadrature
       !All weights also multiplied by 2
       0.25058584390466120961D+00,0.12702628417625060064D-01,&
       1.D5                       ,1.D5                      ,&
       1.D5                       ,1.D5                      ,&
       1.D5                       ,1.D5                      ,&
       !Holds weights divided by pi for 5 point quadrature
       0.27094999227465121905D+00,0.34705697924798682708D-01,&
       0.61865515504634063966D-03,1.D5                      ,&
       1.D5                       ,1.D5                      ,&
       1.D5                       ,1.D5                      ,&
       !Holds weights divided by pi for 7 point quadrature
       0.27543453700668624196D+00,0.56324633490137353264D-01,&
       0.31472089609631808543D-02,0.25214584849507344307D-04,&
       1.D5                       ,1.D5                      ,&
       1.D5                       ,1.D5                      ,&
       !Holds weights divided by pi for 9 point quadrature
       0.27333890780875741422D+00,0.74629583204401842011D-01,&
       0.75830298567198259688D-02,0.22079212970341425127D-03,&
       0.91645261015575683438D-06,1.D5                      ,&
       1.D5                       ,1.D5                      ,&
       !Holds weights divided by pi for 11 point quadrature
       0.26840927095801320190D+00,0.89332600473635989657D-01,&
       0.13281655260003705915D-01,0.76869290570797700612D-03,&
       0.13006371388959025602D-04,0.30721563119005440825D-07,&
       1.D5                       ,1.D5                      ,&
       !Holds weights divided by pi for 13 point quadrature
       0.26230560924446216786D+00,0.10089717749679338876D+00,&
       0.19595178157406731301D-01,0.17685735543966016065D-02,&
       0.63664804613181236658D-04,0.67425389889478559195D-06,&
       0.96923819993902427778D-09,1.D5                      ,&
       !Holds weights divided by pi for 15 point quadrature
       0.25581067552552255728D+00,0.10991131996238762136D+00,&
       0.26050502825691497499D-01,0.32259751765317161779D-02,&
       0.19012222119662986722D-03,0.45278238933327867930D-05,&
       0.31685068883412698676D-07,0.29160871162367655330D-10/
       !Holds weights divided by pi for 17 point quadrature

DATA QW0/0.37612638903183753802D+00,0.30090111122547003042D+00,&
        0.25791523819326001021D+00,0.22925798950512002450D+00,&
        0.20841635409556363845D+00,0.19238432685744336070D+00,&
        0.17955870506694715516D+00,0.16899642829830321955D+00/
        !Weight for quadrature point at 0 divided by pi

END MODULE
```

```
      MODULE SPINTERPDATA
!***********************************************************************
!
!      SPINTERPDATA - Module for LABFIT which contains
!      interpolation tables in arrays as well as constants involved
!      in the calculation of the speed dependent function.
!
!      USED BY
!
!      18 April 2007 - Kendra L. Letchworth
!
!***********************************************************************

      LOGICAL READIN(16)          !TRUE if data has not been read in
      REAL*8 TABLE1H0(:,:,:,:)
      REAL*8 TABLE2H0(:,:,:,:)
      REAL*8 TABLE3H0(:,:,:,:)
      REAL*8 TABLE4H0(:,:,:,:)
      REAL*8 TABLE5H0(:,:,:,:)
      REAL*8 TABLE6H0(:,:,:,:)
      REAL*8 TABLE7H0(:,:,:,:)
      REAL*8 TABLE8H0(:,:,:,:)
      REAL*8 TABLE9H0(:,:,:,:)
      REAL*8 TABLE10H0(:,:,:,:)
      REAL*8 TABLE11H0(:,:,:,:)
      REAL*8 TABLE12H0(:,:,:,:)
      REAL*8 TABLE13H0(:,:,:,:)
      REAL*8 TABLE14H0(:,:,:)
      REAL*8 TABLE15H0(:,:,:)
      REAL*8 TABLE16H0(:,:,:,:)
      ALLOCATABLE :: TABLE1H0,TABLE2H0,TABLE3H0,&
      TABLE4H0,TABLE5H0,TABLE6H0,TABLE7H0,&
      TABLE8H0,TABLE9H0,TABLE10H0,TABLE11H0,&
      TABLE12H0,TABLE13H0,TABLE14H0,TABLE15H0,TABLE16H0
      DATA READIN/.TRUE.,.TRUE.,.TRUE.,.TRUE.,.TRUE.,.TRUE.,.TRUE.&
      ,.TRUE.,.TRUE.,.TRUE.,.TRUE.,.TRUE.,.TRUE.,.TRUE.,.TRUE./
      END MODULE
```

```
SUBROUTINE QUADRATURE3(ORDER1IN,ORDER2IN,ORDER3IN&
,XDIV12,XDIV23,SPEEDRE,SPEEDIM,NPTS1)
      USE SPEEDDEPVAR
!**********************************************************************
!
!       QUADRATURE3 - Subroutine designed to compute the speed dependent
!                     profile for y values for which three x-value regions
!                     are required.
!
!       This subroutine is called by SPEEDKL.
!
!       15 APRIL 2007 - Kendra L. Letchworth
!
!**********************************************************************
       IMPLICIT NONE
       INTEGER*4 ORDER1IN             !first order of quadrature in use
                                                  !when abs(x)<xdiv12
       INTEGER*4 ORDER2IN             !second order of quadrature in use
                                          !when xdiv12<abs(x)<xdiv23
       INTEGER*4 ORDER3IN             !third order of quadrature in use
                                                  !when abs(x)>xdiv32
       INTEGER*4 ORDER1        !array index of first order of quadrature
       INTEGER*4 ORDER2       !array index of second order of quadrature
       INTEGER*4 ORDER3        !array index of third order of quadrature
       INTEGER*4 NPTS1  ! Number of different equally spaced x values at
                          ! which the function is to be evaluated
       REAL*8 SPEEDRE(NPTS1)    !Holds the real speed dependent function
       REAL*8 SPEEDIM(NPTS1) !Holds the imagin. speed dependent function
       REAL*8 XDIV12        !divides region 1 and region 2 of quadrature
       REAL*8 XDIV23        !divides region 2 and region 3 of quadrature
       REAL*8 XSIGNED   ! Signed value of voigt x parameter equal to the
                ! square root of the natural log of 2 times the positive
                     !ratio of the distance from line center divided by
                                                   !the Doppler width
       REAL*8 C,CS,YS,YSC,YDENOM,&
       A1,A2(3,8),A3(3,8),A4(3,8),A5(8),A6(8),X2    !Intermediate values
        !used to calculate the function, employed to increase efficiency
       INTEGER*4 IDELTA    !counter over the evaluations of the function
       INTEGER*4 K                          !Loop counter for quadrature
       INTEGER*4 NUMBER   !number of evaluations for a particular region
       INTEGER*4 NUMBER2        !also number of evaluations for a region

       XSIGNED=XSTART
       C=1.5D0
       CS=C*S
       YS=Y*S
       YSC=YS*C
       YDENOM=Y-YSC
       A1=Y*(YSC*(CS-2.D0)+Y)
       ORDER1=(ORDER1IN-1)/2
       ORDER2=(ORDER2IN-1)/2
       ORDER3=(ORDER3IN-1)/2

       NUMBER=0
       NUMBER2=0

       DO K=1,ORDER3
         A2(3,K)=QPT(K,ORDER3)*(YS*(YS*(QPT(K,ORDER3)-&
         2.D0*C)+2.D0*Y)+1.D0)+A1
         A3(3,K)=YDENOM-YS*QPT(K,ORDER3)
```

```
  A4(3,K)=YDENOM+YS*QPT(K,ORDER3)
END DO

IF (XSIGNED.LT.-XDIV23) THEN
  NUMBER=(-XDIV23-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER=MIN(NUMBER,NPTS)
  DO IDELTA=NUMBER2+1,NUMBER
    X2=XSIGNED*XSIGNED
      DO K=1,ORDER3
        A6(K)=A2(3,K)+X2
        A5(K)=4.D0*X2*QPT(K,ORDER3)/A6(K)
        SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
        +QW(K,ORDER3)*(A3(3,K)+A5(K)*YS)/(A6(K)-A5(K))
        SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
        +QW(K,ORDER3)*XSIGNED*(A3(3,K)*&
        (1.D0-2.D0*QPT(K,ORDER3)/A6(K))+&
        YS*(A5(K)-2.D0*QPT(K,ORDER3)))/(A4(3,K)*(A6(K)-A5(K)))
      END DO
      SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER3)*YDENOM/(A1+X2)
      SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER3)*XSIGNED/(A1+X2)
      XSIGNED=XSIGNED+XDELTA
  END DO
  IF (NUMBER.EQ.NPTS) RETURN
END IF

DO K=1,ORDER2
  A2(2,K)=QPT(K,ORDER2)*(YS*(YS*(QPT(K,ORDER2)-&
  2.D0*C)+2.D0*Y)+1.D0)+A1
  A3(2,K)=YDENOM-YS*QPT(K,ORDER2)
  A4(2,K)=YDENOM+YS*QPT(K,ORDER2)
END DO

IF (XSTART.LT.-XDIV12) THEN
  NUMBER2=(-XDIV12-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER2=MIN(NUMBER2,NPTS)
  IF (NUMBER2.GT.NUMBER) THEN
    DO IDELTA=NUMBER+1,NUMBER2
      X2=XSIGNED*XSIGNED
      DO K=1,ORDER2
        A6(K)=A2(2,K)+X2
        A5(K)=4.D0*X2*QPT(K,ORDER2)/A6(K)
        SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
        +QW(K,ORDER2)*(A3(2,K)+A5(K)*YS)/(A6(K)-A5(K))
        SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
        +QW(K,ORDER2)*XSIGNED*(A3(2,K)*&
        (1.D0-2.D0*QPT(K,ORDER2)/A6(K))+&
        YS*(A5(K)-2.D0*QPT(K,ORDER2)))/(A4(2,K)*(A6(K)-A5(K)))
      END DO
      SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER2)*YDENOM/(A1+X2)
      SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER2)*XSIGNED/(A1+X2)
      XSIGNED=XSIGNED+XDELTA
    END DO
    IF (NUMBER2.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER2=NUMBER
END IF
NUMBER=0

DO K=1,ORDER1
```

```
  A2(1,K)=QPT(K,ORDER1)*(YS*(YS*(QPT(K,ORDER1)-&
  2.D0*C)+2.D0*Y)+1.D0)+A1
  A3(1,K)=YDENOM-YS*QPT(K,ORDER1)
  A4(1,K)=YDENOM+YS*QPT(K,ORDER1)
END DO

IF (XSIGNED.LT.XDIV12) THEN
  NUMBER=(XDIV12-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER=MIN(NUMBER,NPTS)
  IF (NUMBER.GT.NUMBER2) THEN
    DO IDELTA=NUMBER2+1,NUMBER
      X2=XSIGNED*XSIGNED
      DO K=1,ORDER1
        A6(K)=A2(1,K)+X2
        A5(K)=4.D0*X2*QPT(K,ORDER1)/A6(K)
        SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
        +QW(K,ORDER1)*(A3(1,K)+A5(K)*YS)/(A6(K)-A5(K))
        SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
        +QW(K,ORDER1)*XSIGNED*(A3(1,K)*&
        (1.D0-2.D0*QPT(K,ORDER1)/A6(K))+&
        YS*(A5(K)-2.D0*QPT(K,ORDER1)))/(A4(1,K)*(A6(K)-A5(K)))
      END DO
      SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER1)*YDENOM/(A1+X2)
      SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER1)*XSIGNED/(A1+X2)
      XSIGNED=XSIGNED+XDELTA
    END DO
    IF (NUMBER.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER=NUMBER2
END IF
NUMBER2=0

IF (XSTART.LT.XDIV23) THEN
  NUMBER2=(XDIV23-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER2=MIN(NUMBER2,NPTS)
  IF (NUMBER2.GT.NUMBER) THEN
    DO IDELTA=NUMBER+1,NUMBER2
      X2=XSIGNED*XSIGNED
      DO K=1,ORDER2
        A6(K)=A2(2,K)+X2
        A5(K)=4.D0*X2*QPT(K,ORDER2)/A6(K)
        SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
        +QW(K,ORDER2)*(A3(2,K)+A5(K)*YS)/(A6(K)-A5(K))
        SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
        +QW(K,ORDER2)*XSIGNED*(A3(2,K)*&
        (1.D0-2.D0*QPT(K,ORDER2)/A6(K))+&
        YS*(A5(K)-2.D0*QPT(K,ORDER2)))/(A4(2,K)*(A6(K)-A5(K)))
      END DO
      SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER2)*YDENOM/(A1+X2)
      SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER2)*XSIGNED/(A1+X2)
      XSIGNED=XSIGNED+XDELTA
    END DO
    IF (NUMBER2.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER2=NUMBER
END IF
NUMBER=0
```

```
DO IDELTA=NUMBER2+1,NPTS
  X2=XSIGNED*XSIGNED
  DO K=1,ORDER3
    A6(K)=A2(3,K)+X2
    A5(K)=4.D0*X2*QPT(K,ORDER3)/A6(K)
    SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
    +QW(K,ORDER3)*(A3(3,K)+A5(K)*YS)/(A6(K)-A5(K))
    SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
    +QW(K,ORDER3)*XSIGNED*(A3(3,K)*&
    (1.D0-2.D0*QPT(K,ORDER3)/A6(K))+&
    YS*(A5(K)-2.D0*QPT(K,ORDER3)))/(A4(3,K)*(A6(K)-A5(K)))
  END DO
  SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER3)*YDENOM/(A1+X2)
  SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER3)*XSIGNED/(A1+X2)
  XSIGNED=XSIGNED+XDELTA
END DO

END SUBROUTINE QUADRATURE3
```

```
SUBROUTINE QUADINTERP(REGINTIN,ORDER1IN,ORDER2IN,ORDER3IN,ORDER4IN,&
XDIVIQ,XDIV12,XDIV23,XDIV34,SPEEDRE,SPEEDIM,NPTS1)
      USE SPEEDDEPVAR
!**********************************************************************
!
!       QUADINTERP - Subroutine designed to compute the speed dependent
!                    profile for y values for which interpolation could
!                    be necessary for some x-values.
!
!       This subroutine is called by SPEEDKL.
!
!       15 APRIL 2007 - Kendra L. Letchworth
!
!**********************************************************************
       IMPLICIT NONE
       INTEGER*4 ORDER1IN             !first order of quadrature in use
                                             !when xdiviq<abs(x)<xdiv12
       INTEGER*4 ORDER2IN             !second order of quadrature in use
                                             !when xdiv12<abs(x)<xdiv23
       INTEGER*4 ORDER3IN             !third order of quadrature in use
                                             !when xdiv23<abs(x)<xdiv34
       INTEGER*4 ORDER4IN             !fourth order of quadrature in use
                                               !when abs(x)>xdiv34
       INTEGER*4 ORDER1        !array index of first order of quadrature
       INTEGER*4 ORDER2       !array index of second order of quadrature
       INTEGER*4 ORDER3        !array index of third order of quadrature
       INTEGER*4 ORDER4       !array index of fourth order of quadrature
       INTEGER*4 NPTS1  ! Number of different equally spaced x values at
                              ! which the function is to be evaluated
       INTEGER*4 REGINTIN     !Integer value for region of interpolation
       REAL*8 SPEEDRE(NPTS1)    !Holds the real speed dependent function
       REAL*8 SPEEDIM(NPTS1) !Holds the imagin. speed dependent function
       REAL*8 XDIV12        !divides region 1 and region 2 of quadrature
       REAL*8 XDIV23        !divides region 2 and region 3 of quadrature
       REAL*8 XDIV34        !divides region 3 and region 4 of quadrature
       REAL*8 XDIVIQ       !divides interpolation and quadrature regions
       REAL*8 XSIGNED   ! Signed value of voigt x parameter equal to the
                ! square root of the natural log of 2 times the positive
                     !ratio of the distance from line center divided by
                                                 !the Doppler width
       REAL*8 C,CS,YS,YSC,YDENOM,&
       A1,A2(4,8),A3(4,8),A4(4,8),A5(8),A6(8),X2    !Intermediate values
        !used to calculate the function, employed to increase efficiency
       INTEGER*4 IDELTA    !counter over the evaluations of the function
       INTEGER*4 K                          !Loop counter for quadrature
       INTEGER*4 NUMBER   !number of evaluations for a particular region
       INTEGER*4 NUMBER2        !also number of evaluations for a region

       XSIGNED=XSTART
       C=1.5D0
       CS=C*S
       YS=Y*S
       YSC=YS*C
       YDENOM=Y-YSC
       A1=Y*(YSC*(CS-2.D0)+Y)
       ORDER1=(ORDER1IN-1)/2
       ORDER2=(ORDER2IN-1)/2
       ORDER3=(ORDER3IN-1)/2
       ORDER4=(ORDER4IN-1)/2
```

```
DO K=1,ORDER4
  A2(4,K)=QPT(K,ORDER4)*(YS*(YS*(QPT(K,ORDER4)-&
  2.D0*C)+2.D0*Y)+1.D0)+A1
  A3(4,K)=YDENOM-YS*QPT(K,ORDER4)
  A4(4,K)=YDENOM+YS*QPT(K,ORDER4)
END DO

NUMBER=0
NUMBER2=0
IF (XSTART.LT.-XDIV34) THEN
  NUMBER=(-XDIV34-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER=MIN(NUMBER,NPTS)
  DO IDELTA=1,NUMBER
    X2=XSIGNED*XSIGNED
    DO K=1,ORDER4
      A6(K)=A2(4,K)+X2
      A5(K)=4.D0*X2*QPT(K,ORDER4)/A6(K)
      SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
      +QW(K,ORDER4)*(A3(4,K)+A5(K)*YS)/(A6(K)-A5(K))
      SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
      +QW(K,ORDER4)*XSIGNED*(A3(4,K)*&
      (1.D0-2.D0*QPT(K,ORDER4)/A6(K))+&
      YS*(A5(K)-2.D0*QPT(K,ORDER4)))/(A4(4,K)*(A6(K)-A5(K)))
    END DO
    SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+&
    QW0(ORDER4)*YDENOM/(A1+X2)
    SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+&
    QW0(ORDER4)*XSIGNED/(A1+X2)
    XSIGNED=XSIGNED+XDELTA
  END DO
  IF (NUMBER.EQ.NPTS) RETURN
END IF

DO K=1,ORDER3
  A2(3,K)=QPT(K,ORDER3)*(YS*(YS*(QPT(K,ORDER3)-&
  2.D0*C)+2.D0*Y)+1.D0)+A1
  A3(3,K)=YDENOM-YS*QPT(K,ORDER3)
  A4(3,K)=YDENOM+YS*QPT(K,ORDER3)
END DO

IF (XSTART.LT.-XDIV23) THEN
  NUMBER2=(-XDIV23-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER2=MIN(NUMBER2,NPTS)
  IF (NUMBER2.GT.NUMBER) THEN
    DO IDELTA=NUMBER+1,NUMBER2
      X2=XSIGNED*XSIGNED
      DO K=1,ORDER3
        A6(K)=A2(3,K)+X2
        A5(K)=4.D0*X2*QPT(K,ORDER3)/A6(K)
        SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
        +QW(K,ORDER3)*(A3(3,K)+A5(K)*YS)/(A6(K)-A5(K))
        SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
        +QW(K,ORDER3)*XSIGNED*(A3(3,K)*&
        (1.D0-2.D0*QPT(K,ORDER3)/A6(K))+&
        YS*(A5(K)-2.D0*QPT(K,ORDER3)))/(A4(3,K)*(A6(K)-A5(K)))
      END DO
      SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER3)*YDENOM/(A1+X2)
      SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER3)*XSIGNED/(A1+X2)
      XSIGNED=XSIGNED+XDELTA
    END DO
```

```
    IF (NUMBER2.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER2=NUMBER
END IF
NUMBER=0

DO K=1,ORDER2
  A2(2,K)=QPT(K,ORDER2)*(YS*(YS*(QPT(K,ORDER2)-&
  2.D0*C)+2.D0*Y)+1.D0)+A1
  A3(2,K)=YDENOM-YS*QPT(K,ORDER2)
  A4(2,K)=YDENOM+YS*QPT(K,ORDER2)
END DO


IF (XSTART.LT.-XDIV12) THEN
  NUMBER=(-XDIV12-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER=MIN(NUMBER,NPTS)
  IF (NUMBER.GT.NUMBER2) THEN
    DO IDELTA=NUMBER2+1,NUMBER
      X2=XSIGNED*XSIGNED
      DO K=1,ORDER2
        A6(K)=A2(2,K)+X2
        A5(K)=4.D0*X2*QPT(K,ORDER2)/A6(K)
        SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
        +QW(K,ORDER2)*(A3(2,K)+A5(K)*YS)/(A6(K)-A5(K))
        SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
        +QW(K,ORDER2)*XSIGNED*(A3(2,K)*&
        (1.D0-2.D0*QPT(K,ORDER2)/A6(K))+&
        YS*(A5(K)-2.D0*QPT(K,ORDER2)))/(A4(2,K)*(A6(K)-A5(K)))
      END DO
      SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER2)*YDENOM/(A1+X2)
      SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER2)*XSIGNED/(A1+X2)
      XSIGNED=XSIGNED+XDELTA
    END DO
    IF (NUMBER.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER=NUMBER2
END IF
NUMBER2=0

DO K=1,ORDER1
  A2(1,K)=QPT(K,ORDER1)*(YS*(YS*(QPT(K,ORDER1)-&
  2.D0*C)+2.D0*Y)+1.D0)+A1
  A3(1,K)=YDENOM-YS*QPT(K,ORDER1)
  A4(1,K)=YDENOM+YS*QPT(K,ORDER1)
END DO

IF (XSTART.LT.-XDIVIQ) THEN
  NUMBER2=(-XDIVIQ-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER2=MIN(NUMBER2,NPTS)
  IF (NUMBER2.GT.NUMBER) THEN
    DO IDELTA=NUMBER+1,NUMBER2
      X2=XSIGNED*XSIGNED
      DO K=1,ORDER1
        A6(K)=A2(1,K)+X2
        A5(K)=4.D0*X2*QPT(K,ORDER1)/A6(K)
        SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
        +QW(K,ORDER1)*(A3(1,K)+A5(K)*YS)/(A6(K)-A5(K))
```

```
          SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
          +QW(K,ORDER1)*XSIGNED*(A3(1,K)*&
          (1.D0-2.D0*QPT(K,ORDER1)/A6(K))+&
          YS*(A5(K)-2.D0*QPT(K,ORDER1)))/(A4(1,K)*(A6(K)-A5(K)))
        END DO
        SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER1)*YDENOM/(A1+X2)
        SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER1)*XSIGNED/(A1+X2)
        XSIGNED=XSIGNED+XDELTA
      END DO
      IF (NUMBER2.EQ.NPTS) RETURN
    END IF
  ELSE
    NUMBER2=NUMBER
  END IF
  NUMBER=0

  IF (XSTART.LT.XDIVIQ) THEN
    NUMBER=(XDIVIQ-XSTART)/XDELTA+1.D0+1.D-13
    NUMBER=MIN(NUMBER,NPTS)
    IF (NUMBER.GT.NUMBER2) THEN
      IF (REGINTIN.EQ.1) THEN
      CALL SPINTERPREG1(NUMBER2+1,NUMBER,&
      XSIGNED,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (REGINTIN.EQ.2) THEN
      CALL SPINTERPREG2(NUMBER2+1,NUMBER,&
      XSIGNED,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (REGINTIN.EQ.3) THEN
      CALL SPINTERPREG3(NUMBER2+1,NUMBER,&
      XSIGNED,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (REGINTIN.EQ.4) THEN
      CALL SPINTERPREG4(NUMBER2+1,NUMBER,&
      XSIGNED,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (REGINTIN.EQ.5) THEN
      CALL SPINTERPREG5(NUMBER2+1,NUMBER,&
      XSIGNED,SPEEDRE,SPEEDIM,NPTS1)
      ELSE IF (REGINTIN.EQ.6) THEN
      CALL SPINTERPREG6(NUMBER2+1,NUMBER,&
      XSIGNED,SPEEDRE,SPEEDIM,NPTS1)
      ELSE
      CALL SPINTERPREG7(NUMBER2+1,NUMBER,&
      XSIGNED,SPEEDRE,SPEEDIM,NPTS1)
      END IF
      IF (NUMBER.EQ.NPTS) RETURN
    END IF
  ELSE
    NUMBER=NUMBER2
  END IF
  NUMBER2=0

  IF (XSTART.LT.XDIV12) THEN
    NUMBER2=(XDIV12-XSTART)/XDELTA+1.D0+1.D-13
    NUMBER2=MIN(NUMBER2,NPTS)
    IF (NUMBER2.GT.NUMBER) THEN
      DO IDELTA=NUMBER+1,NUMBER2
        X2=XSIGNED*XSIGNED
        DO K=1,ORDER1
          A6(K)=A2(1,K)+X2
          A5(K)=4.D0*X2*QPT(K,ORDER1)/A6(K)
          SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
          +QW(K,ORDER1)*(A3(1,K)+A5(K)*YS)/(A6(K)-A5(K))
```

```
        SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
        +QW(K,ORDER1)*XSIGNED*(A3(1,K)*&
        (1.D0-2.D0*QPT(K,ORDER1)/A6(K))+&
        YS*(A5(K)-2.D0*QPT(K,ORDER1)))/(A4(1,K)*(A6(K)-A5(K)))
      END DO
      SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER1)*YDENOM/(A1+X2)
      SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER1)*XSIGNED/(A1+X2)
      XSIGNED=XSIGNED+XDELTA
    END DO
    IF (NUMBER2.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER2=NUMBER
END IF
NUMBER=0

IF (XSTART.LT.XDIV23) THEN
  NUMBER=(XDIV23-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER=MIN(NUMBER,NPTS)
  IF (NUMBER.GT.NUMBER2) THEN
    DO IDELTA=NUMBER2+1,NUMBER
      X2=XSIGNED*XSIGNED
      DO K=1,ORDER2
        A6(K)=A2(2,K)+X2
        A5(K)=4.D0*X2*QPT(K,ORDER2)/A6(K)
        SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
        +QW(K,ORDER2)*(A3(2,K)+A5(K)*YS)/(A6(K)-A5(K))
        SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
        +QW(K,ORDER2)*XSIGNED*(A3(2,K)*&
        (1.D0-2.D0*QPT(K,ORDER2)/A6(K))+&
        YS*(A5(K)-2.D0*QPT(K,ORDER2)))/(A4(2,K)*(A6(K)-A5(K)))
      END DO
      SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER2)*YDENOM/(A1+X2)
      SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER2)*XSIGNED/(A1+X2)
      XSIGNED=XSIGNED+XDELTA
    END DO
    IF (NUMBER.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER=NUMBER2
END IF
NUMBER2=0

IF (XSTART.LT.XDIV34) THEN
  NUMBER2=(XDIV34-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER2=MIN(NUMBER2,NPTS)
  IF (NUMBER2.GT.NUMBER) THEN
    DO IDELTA=NUMBER+1,NUMBER2
      X2=XSIGNED*XSIGNED
      DO K=1,ORDER3
        A6(K)=A2(3,K)+X2
        A5(K)=4.D0*X2*QPT(K,ORDER3)/A6(K)
        SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
        +QW(K,ORDER3)*(A3(3,K)+A5(K)*YS)/(A6(K)-A5(K))
        SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
        +QW(K,ORDER3)*XSIGNED*(A3(3,K)*&
        (1.D0-2.D0*QPT(K,ORDER3)/A6(K))+&
        YS*(A5(K)-2.D0*QPT(K,ORDER3)))/(A4(3,K)*(A6(K)-A5(K)))
      END DO
      SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER3)*YDENOM/(A1+X2)
```

```
      SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER3)*XSIGNED/(A1+X2)
      XSIGNED=XSIGNED+XDELTA
    END DO
    IF (NUMBER2.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER2=NUMBER
END IF
NUMBER=0

DO IDELTA=NUMBER2+1,NPTS
  X2=XSIGNED*XSIGNED
    DO K=1,ORDER4
      A6(K)=A2(4,K)+X2
      A5(K)=4.D0*X2*QPT(K,ORDER4)/A6(K)
      SPEEDRE(IDELTA)=SPEEDRE(IDELTA)&
      +QW(K,ORDER4)*(A3(4,K)+A5(K)*YS)/(A6(K)-A5(K))
      SPEEDIM(IDELTA)=SPEEDIM(IDELTA)&
      +QW(K,ORDER4)*XSIGNED*(A3(4,K)*&
      (1.D0-2.D0*QPT(K,ORDER4)/A6(K))+&
      YS*(A5(K)-2.D0*QPT(K,ORDER4)))/(A4(4,K)*(A6(K)-A5(K)))
    END DO
    SPEEDRE(IDELTA)=SPEEDRE(IDELTA)+QW0(ORDER4)*YDENOM/(A1+X2)
    SPEEDIM(IDELTA)=SPEEDIM(IDELTA)+QW0(ORDER4)*XSIGNED/(A1+X2)
    XSIGNED=XSIGNED+XDELTA
END DO

END SUBROUTINE QUADINTERP
```

```
 SUBROUTINE READINTERPSP(NUM)
       USE VARFILIO
       USE LOGUNITS
       USE SPINTERPDATA
       USE VARLSTSQ
       USE VAROBSRV
!**********************************************************************
!
!        READINTERP - this subroutine was created to read in
!                     interpolation tables stored in binary files
!
!        CALLED BY DERIVINTERP,POLYINTERP
!
!        18 April 2007 - Kendra L. Letchworth
!
!**********************************************************************
       IMPLICIT NONE
       INTEGER*4 READCOUNTER1
       INTEGER*4 READCOUNTER2
       INTEGER*4 DERIVCOUNTER
       INTEGER*4 YCOUNTER
       INTEGER*4 XCOUNTER
       INTEGER*4 SCOUNTER
       INTEGER*4 KCOUNTER
       INTEGER*4 NUM
       INTEGER*4 IOS
       REAL*8 XYS1D(:)
       ALLOCATABLE :: XYS1D

       IF (NUM.EQ.1) THEN
         ALLOCATE(XYS1D(155040))
         CALL COPEN(LUINTTBL,'c:\tables\table1h0.bin',256)
         DO READCOUNTER1=0,155008,32
           CALL CREAD(LUINTTBL,0)
           DO READCOUNTER2=1,32
             XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
           END DO
         END DO
         CALL CLOSF(LUINTTBL)
         ALLOCATE(TABLE1H0(34,19,6,40))
         KCOUNTER=1
         DO DERIVCOUNTER=1,40
           DO SCOUNTER=1,6
             DO YCOUNTER=1,19
               DO XCOUNTER=1,34
               TABLE1H0(XCOUNTER,YCOUNTER,SCOUNTER,&
               DERIVCOUNTER)=XYS1D(KCOUNTER)
               KCOUNTER=KCOUNTER+1
               END DO
             END DO
           END DO
         END DO
         DEALLOCATE(XYS1D)
       ELSE IF (NUM.EQ.2) THEN
         ALLOCATE(XYS1D(103360))
         CALL COPEN(LUINTTBL,'c:\tables\TABLE2H0.bin',256)
         DO READCOUNTER1=0,103328,32
           CALL CREAD(LUINTTBL,0)
           DO READCOUNTER2=1,32
             XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
```

```
      END DO
    END DO
    CALL CLOSF(LUINTTBL)
    ALLOCATE(TABLE2H0(34,19,4,40))
    KCOUNTER=1
    DO DERIVCOUNTER=1,40
      DO SCOUNTER=1,4
        DO YCOUNTER=1,19
          DO XCOUNTER=1,34
          TABLE2H0(XCOUNTER,YCOUNTER,SCOUNTER,&
          DERIVCOUNTER)=XYS1D(KCOUNTER)
          KCOUNTER=KCOUNTER+1
          END DO
        END DO
      END DO
    END DO
    DEALLOCATE(XYS1D)
  ELSE IF (NUM.EQ.3) THEN
    ALLOCATE(XYS1D(187200))
    CALL COPEN(LUINTTBL,'c:\tables\TABLE3H0.bin',256)
    DO READCOUNTER1=0,187168,32
      CALL CREAD(LUINTTBL,0)
      DO READCOUNTER2=1,32
        XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
      END DO
    END DO
    CALL CLOSF(LUINTTBL)
    ALLOCATE(TABLE3H0(39,20,6,40))
    KCOUNTER=1
    DO DERIVCOUNTER=1,40
      DO SCOUNTER=1,6
        DO YCOUNTER=1,20
          DO XCOUNTER=1,39
          TABLE3H0(XCOUNTER,YCOUNTER,SCOUNTER,&
          DERIVCOUNTER)=XYS1D(KCOUNTER)
          KCOUNTER=KCOUNTER+1
          END DO
        END DO
      END DO
    END DO
    DEALLOCATE(XYS1D)
  ELSE IF (NUM.EQ.4) THEN
    ALLOCATE(XYS1D(124800))
    CALL COPEN(LUINTTBL,'c:\tables\TABLE4H0.bin',256)
    DO READCOUNTER1=0,124768,32
      CALL CREAD(LUINTTBL,0)
      DO READCOUNTER2=1,32
        XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
      END DO
    END DO
    CALL CLOSF(LUINTTBL)
    ALLOCATE(TABLE4H0(39,20,4,40))
    KCOUNTER=1
    DO DERIVCOUNTER=1,40
      DO SCOUNTER=1,4
        DO YCOUNTER=1,20
          DO XCOUNTER=1,39
          TABLE4H0(XCOUNTER,YCOUNTER,SCOUNTER,&
          DERIVCOUNTER)=XYS1D(KCOUNTER)
          KCOUNTER=KCOUNTER+1
```

```
        END DO
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.5) THEN
  ALLOCATE(XYS1D(219520))
  CALL COPEN(LUINTTBL,'c:\tables\TABLE5H0.bin',256)
  DO READCOUNTER1=0,219488,32
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE5H0(28,49,4,40))
  KCOUNTER=1
  DO DERIVCOUNTER=1,40
    DO SCOUNTER=1,4
      DO YCOUNTER=1,49
        DO XCOUNTER=1,28
        TABLE5H0(XCOUNTER,YCOUNTER,SCOUNTER,&
        DERIVCOUNTER)=XYS1D(KCOUNTER)
        KCOUNTER=KCOUNTER+1
        END DO
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.6) THEN
  ALLOCATE(XYS1D(55440))
  CALL COPEN(LUINTTBL,'c:\tables\TABLE6H0.bin',256)
  DO READCOUNTER1=0,55392,32
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
  CALL CREAD(LUINTTBL,0)
  READCOUNTER1=55424
  DO READCOUNTER2=1,16
    XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
  END DO
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE6H0(77,3,6,40))
  KCOUNTER=1
  DO DERIVCOUNTER=1,40
    DO SCOUNTER=1,6
      DO YCOUNTER=1,3
        DO XCOUNTER=1,77
        TABLE6H0(XCOUNTER,YCOUNTER,SCOUNTER,&
        DERIVCOUNTER)=XYS1D(KCOUNTER)
        KCOUNTER=KCOUNTER+1
        END DO
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.7) THEN
  ALLOCATE(XYS1D(36960))
```

```
  CALL COPEN(LUINTTBL,'c:\tables\TABLE7H0.bin',256)
  DO READCOUNTER1=0,36928,32
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE7H0(77,3,4,40))
  KCOUNTER=1
  DO DERIVCOUNTER=1,40
    DO SCOUNTER=1,4
      DO YCOUNTER=1,3
        DO XCOUNTER=1,77
        TABLE7H0(XCOUNTER,YCOUNTER,SCOUNTER,&
        DERIVCOUNTER)=XYS1D(KCOUNTER)
        KCOUNTER=KCOUNTER+1
        END DO
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.8) THEN
  ALLOCATE(XYS1D(133200))
  CALL COPEN(LUINTTBL,'c:\tables\TABLE8H0.bin',256)
  DO READCOUNTER1=0,133152,32
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
  CALL CREAD(LUINTTBL,0)
  READCOUNTER1=133184
  DO READCOUNTER2=1,16
    XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
  END DO
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE8H0(15,37,6,40))
  KCOUNTER=1
  DO DERIVCOUNTER=1,40
    DO SCOUNTER=1,6
      DO YCOUNTER=1,37
        DO XCOUNTER=1,15
        TABLE8H0(XCOUNTER,YCOUNTER,SCOUNTER,&
        DERIVCOUNTER)=XYS1D(KCOUNTER)
        KCOUNTER=KCOUNTER+1
        END DO
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.9) THEN
  ALLOCATE(XYS1D(88800))
  CALL COPEN(LUINTTBL,'c:\tables\TABLE9H0.bin',256)
  DO READCOUNTER1=0,88768,32
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
```

```
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE9H0(15,37,4,40))
  KCOUNTER=1
  DO DERIVCOUNTER=1,40
    DO SCOUNTER=1,4
      DO YCOUNTER=1,37
        DO XCOUNTER=1,15
        TABLE9H0(XCOUNTER,YCOUNTER,SCOUNTER,&
        DERIVCOUNTER)=XYS1D(KCOUNTER)
        KCOUNTER=KCOUNTER+1
        END DO
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.10) THEN
  ALLOCATE(XYS1D(602400))
  CALL COPEN(LUINTTBL,'c:\tables\TABLE10H0.bin',256)
  DO READCOUNTER1=0,602368,32
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE10H0(251,10,6,40))
  KCOUNTER=1
  DO DERIVCOUNTER=1,40
    DO SCOUNTER=1,6
      DO YCOUNTER=1,10
        DO XCOUNTER=1,251
        TABLE10H0(XCOUNTER,YCOUNTER,SCOUNTER,&
        DERIVCOUNTER)=XYS1D(KCOUNTER)
        KCOUNTER=KCOUNTER+1
        END DO
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.11) THEN
  ALLOCATE(XYS1D(401600))
  CALL COPEN(LUINTTBL,'c:\tables\TABLE11H0.bin',256)
  DO READCOUNTER1=0,401568,32
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE11H0(251,10,4,40))
  KCOUNTER=1
  DO DERIVCOUNTER=1,40
    DO SCOUNTER=1,4
      DO YCOUNTER=1,10
        DO XCOUNTER=1,251
        TABLE11H0(XCOUNTER,YCOUNTER,SCOUNTER,&
        DERIVCOUNTER)=XYS1D(KCOUNTER)
        KCOUNTER=KCOUNTER+1
        END DO
      END DO
```

```
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.12) THEN
  ALLOCATE(XYS1D(60720))
  CALL COPEN(LUINTTBL,'c:\tables\TABLE12H0.bin',256)
  DO READCOUNTER1=0,60672,32
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
  CALL CREAD(LUINTTBL,0)
  READCOUNTER1=60704
  DO READCOUNTER2=1,16
    XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
  END DO
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE12H0(11,23,6,40))
  KCOUNTER=1
  DO DERIVCOUNTER=1,40
    DO SCOUNTER=1,6
      DO YCOUNTER=1,23
        DO XCOUNTER=1,11
        TABLE12H0(XCOUNTER,YCOUNTER,SCOUNTER,&
        DERIVCOUNTER)=XYS1D(KCOUNTER)
        KCOUNTER=KCOUNTER+1
        END DO
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.13) THEN
  ALLOCATE(XYS1D(40480))
  CALL COPEN(LUINTTBL,'c:\tables\TABLE13H0.bin',256)
  DO READCOUNTER1=0,40448,32
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE13H0(11,23,4,40))
  KCOUNTER=1
  DO DERIVCOUNTER=1,40
    DO SCOUNTER=1,4
      DO YCOUNTER=1,23
        DO XCOUNTER=1,11
        TABLE13H0(XCOUNTER,YCOUNTER,SCOUNTER,&
        DERIVCOUNTER)=XYS1D(KCOUNTER)
        KCOUNTER=KCOUNTER+1
        END DO
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.14) THEN
  ALLOCATE(XYS1D(284529))
  CALL COPEN(LUINTTBL,'c:\tables\TABLE14H0.bin',256)
  DO READCOUNTER1=0,284480,32
```

```
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
  CALL CREAD(LUINTTBL,0)
  READCOUNTER1=284512
  DO READCOUNTER2=1,17
    XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
  END DO
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE14H0(21,797,17))
  KCOUNTER=1
  DO SCOUNTER=1,17
    DO YCOUNTER=1,797
      DO XCOUNTER=1,21
        TABLE14H0(XCOUNTER,YCOUNTER,SCOUNTER)=XYS1D(KCOUNTER)
        KCOUNTER=KCOUNTER+1
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.15) THEN
  ALLOCATE(XYS1D(332073))
  CALL COPEN(LUINTTBL,'c:\tables\TABLE15H0.bin',256)
  DO READCOUNTER1=0,332032,32
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
  CALL CREAD(LUINTTBL,0)
  READCOUNTER1=332064
  DO READCOUNTER2=1,9
    XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
  END DO
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE15H0(21,1757,9))
  KCOUNTER=1
  DO SCOUNTER=1,9
    DO YCOUNTER=1,1757
      DO XCOUNTER=1,21
        TABLE15H0(XCOUNTER,YCOUNTER,SCOUNTER)=XYS1D(KCOUNTER)
        KCOUNTER=KCOUNTER+1
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
ELSE IF (NUM.EQ.16) THEN
  ALLOCATE(XYS1D(250250))
  CALL COPEN(LUINTTBL,'c:\tables\TABLE16H0.bin',256)
  DO READCOUNTER1=0,250208,32
    CALL CREAD(LUINTTBL,0)
    DO READCOUNTER2=1,32
      XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
    END DO
  END DO
  CALL CREAD(LUINTTBL,0)
  READCOUNTER1=250240
  DO READCOUNTER2=1,10
```

```
    XYS1D(READCOUNTER1+READCOUNTER2)=R8IOARRAY(READCOUNTER2)
  END DO
  CALL CLOSF(LUINTTBL)
  ALLOCATE(TABLE16H0(1001,5,25,2))
  KCOUNTER=1
  DO DERIVCOUNTER=1,2
    DO SCOUNTER=1,25
      DO YCOUNTER=1,5
        DO XCOUNTER=1,1001
          TABLE16H0(XCOUNTER,YCOUNTER,SCOUNTER,&
          DERIVCOUNTER)=XYS1D(KCOUNTER)
          KCOUNTER=KCOUNTER+1
        END DO
      END DO
    END DO
  END DO
  DEALLOCATE(XYS1D)
END IF
READIN(NUM)=.FALSE.
WRITE (LUNTO,'("READ IN DATA FOR COMPLEX SPEED&
 DEPENDENT INTERPOLATION ROUTINE ",I2)')NUM

END SUBROUTINE READINTERPSP
```

```
SUBROUTINE SPINTERPREG1(IDELTABEG,IDELTAEND,XSIGNED,SPEEDRE,SPEEDIM,NPTS1)
      USE CONSTANT
      USE SPEEDDEPVAR
      USE SPINTERPDATA
      USE VARLSTSQ
!**********************************************************************
!
!       SPINTERPREG1 - Subroutine designed to handle the calculation
!                      of the speed dependent profile when 0<Y<.02,
!                      H=0 and interpolation is necessary.  The subroutine
!                      reads in the interpolation tables and stores
!                      them in a module within the program, then uses
!                      interpolation to perform calculation.
!
!       This subroutine is called by QUADINTERP.
!
!       15 APRIL 2007 - Kendra L. Letchworth
!
!**********************************************************************
      IMPLICIT NONE
      INTEGER*4 IDELTA                    !loop counter over all x points
      INTEGER*4 IDELTABEG                !first needed index for SPEEDDEP
      INTEGER*4 IDELTAEND                 !last needed index for SPEEDDEP
      INTEGER*4 ROUTNUM                   !array index of routine number
        PARAMETER (ROUTNUM=16)
      REAL*8 XSIGNED             !signed x-value corresponding to IDELTA
      REAL*8 X                                !absolute value of xsigned
      INTEGER*4 NPTS1  ! Number of different equally spaced x values at
                               ! which the function is to be evaluated
      REAL*8 SPEEDRE(NPTS1)    !array holding function values for all x
      REAL*8 SPEEDIM(NPTS1)   !array holding function values for all x
      INTEGER*4 INDX2,INDY2,INDS2,INDX1,INDX3,INDS1,INDS3,INDY1,INDY3
                         !indices in the table for each parameter X,Y,S
      REAL*8 DIFFX,DIFFY,DIFFS,DIFFXD2,DIFFYD2,DIFFSD2
          !difference between actual parameter value and value in table
      REAL*8 P1,P2,P3                                      !Holds points
      REAL*8 X1,X2,X3                                !Holds x polynomials
      REAL*8 Y1,Y2,Y3                                !Holds y polynomials
      REAL*8 DX,DY,DS
      IF (READIN(ROUTNUM)) CALL READINTERPSP(ROUTNUM)

      DATA DX,DY,DS/.005D0,.005D0,.01D0/

      INDY2=Y/DY+1.5D0+1.D-13
      IF (INDY2.EQ.1) THEN
        INDY2=2
      ELSE IF (INDY2.EQ.5) THEN
        INDY2=4
      END IF
      INDY1=INDY2-1
      INDY3=INDY2+1

      INDS2=S/DS+1.5D0+1.D-13
      IF (INDS2.EQ.1) THEN
        INDS2=2
      ELSE IF (INDS2.EQ.25) THEN
        INDS2=24
      END IF
      INDS1=INDS2-1
      INDS3=INDS2+1
```

```
DIFFY=Y/DY-FLOAT(INDY2-1)
DIFFYD2=.5D0*DIFFY
DIFFS=S/DS-FLOAT(INDS2-1)
DIFFSD2=.5D0*DIFFS

DO IDELTA=IDELTABEG,IDELTAEND
  X=ABS(XSIGNED)
  INDX2=X/DX+1.5D0+1.D-13
  IF (INDX2.EQ.1) THEN
    INDX2=2
  ELSE IF (INDX2.EQ.801) THEN
    INDX2=800
  END IF
  INDX1=INDX2-1
  INDX3=INDX2+1
  DIFFX=X/DX-FLOAT(INDX2-1)
  DIFFXD2=.5D0*DIFFX
  P1=TABLE16H0(INDX1,INDY1,INDS1,1)
  P2=TABLE16H0(INDX2,INDY1,INDS1,1)
  P3=TABLE16H0(INDX3,INDY1,INDS1,1)
  X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY2,INDS1,1)
  P2=TABLE16H0(INDX2,INDY2,INDS1,1)
  P3=TABLE16H0(INDX3,INDY2,INDS1,1)
  X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY3,INDS1,1)
  P2=TABLE16H0(INDX2,INDY3,INDS1,1)
  P3=TABLE16H0(INDX3,INDY3,INDS1,1)
  X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  Y1=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
  P1=TABLE16H0(INDX1,INDY1,INDS2,1)
  P2=TABLE16H0(INDX2,INDY1,INDS2,1)
  P3=TABLE16H0(INDX3,INDY1,INDS2,1)
  X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY2,INDS2,1)
  P2=TABLE16H0(INDX2,INDY2,INDS2,1)
  P3=TABLE16H0(INDX3,INDY2,INDS2,1)
  X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY3,INDS2,1)
  P2=TABLE16H0(INDX2,INDY3,INDS2,1)
  P3=TABLE16H0(INDX3,INDY3,INDS2,1)
  X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  Y2=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
  P1=TABLE16H0(INDX1,INDY1,INDS3,1)
  P2=TABLE16H0(INDX2,INDY1,INDS3,1)
  P3=TABLE16H0(INDX3,INDY1,INDS3,1)
  X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY2,INDS3,1)
  P2=TABLE16H0(INDX2,INDY2,INDS3,1)
  P3=TABLE16H0(INDX3,INDY2,INDS3,1)
  X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY3,INDS3,1)
  P2=TABLE16H0(INDX2,INDY3,INDS3,1)
  P3=TABLE16H0(INDX3,INDY3,INDS3,1)
  X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  Y3=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
  SPEEDRE(IDELTA)=Y2+DIFFSD2*((Y3-Y1)+DIFFS*(Y1+Y3-2.D0*Y2))

  P1=TABLE16H0(INDX1,INDY1,INDS1,2)
```

```
  P2=TABLE16H0(INDX2,INDY1,INDS1,2)
  P3=TABLE16H0(INDX3,INDY1,INDS1,2)
  X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY2,INDS1,2)
  P2=TABLE16H0(INDX2,INDY2,INDS1,2)
  P3=TABLE16H0(INDX3,INDY2,INDS1,2)
  X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY3,INDS1,2)
  P2=TABLE16H0(INDX2,INDY3,INDS1,2)
  P3=TABLE16H0(INDX3,INDY3,INDS1,2)
  X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  Y1=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
  P1=TABLE16H0(INDX1,INDY1,INDS2,2)
  P2=TABLE16H0(INDX2,INDY1,INDS2,2)
  P3=TABLE16H0(INDX3,INDY1,INDS2,2)
  X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY2,INDS2,2)
  P2=TABLE16H0(INDX2,INDY2,INDS2,2)
  P3=TABLE16H0(INDX3,INDY2,INDS2,2)
  X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY3,INDS2,2)
  P2=TABLE16H0(INDX2,INDY3,INDS2,2)
  P3=TABLE16H0(INDX3,INDY3,INDS2,2)
  X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  Y2=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
  P1=TABLE16H0(INDX1,INDY1,INDS3,2)
  P2=TABLE16H0(INDX2,INDY1,INDS3,2)
  P3=TABLE16H0(INDX3,INDY1,INDS3,2)
  X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY2,INDS3,2)
  P2=TABLE16H0(INDX2,INDY2,INDS3,2)
  P3=TABLE16H0(INDX3,INDY2,INDS3,2)
  X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  P1=TABLE16H0(INDX1,INDY3,INDS3,2)
  P2=TABLE16H0(INDX2,INDY3,INDS3,2)
  P3=TABLE16H0(INDX3,INDY3,INDS3,2)
  X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
  Y3=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
  SPEEDIM(IDELTA)=SIGN(Y2+DIFFSD2*((Y3-Y1)+DIFFS*&
  (Y1+Y3-2.D0*Y2)),XSIGNED)
  XSIGNED=XSIGNED+XDELTA
END DO

END SUBROUTINE
```

```
SUBROUTINE SPINTERPREG3(IDELTABEG,IDELTAEND,XSIGNED,&
SPEEDRE,SPEEDIM,NPTS1)
       USE CONSTANT
       USE SPEEDDEPVAR
       USE SPINTERPDATA
       USE VARLSTSQ
!**********************************************************************
!
!       SPINTERPREG3 - Subroutine designed to handle the calculation
!                      of the speed dependent profile when .2<Y<.3,
!                      H=0 and interpolation is necessary.  The subroutine
!                      reads in the interpolation tables and stores
!                      them in a module within the program, then uses
!                      interpolation to perform calculation.
!
!       This subroutine is called by QUADINTERP.
!
!       15 APRIL 2007 - Kendra L. Letchworth
!
!**********************************************************************
       IMPLICIT NONE
       INTEGER*4 IDELTA                  !loop counter over all x points
       INTEGER*4 IDELTABEG              !first needed index for SPEEDDEP
       INTEGER*4 IDELTAEND               !last needed index for SPEEDDEP
       INTEGER*4 ROUTNUM(6)               !array index of routine number
       REAL*8 XSIGNED            !signed x-value corresponding to IDELTA
       REAL*8 X                                !absolute value of xsigned
       INTEGER*4 NPTS1  ! Number of different equally spaced x values at
                                ! which the function is to be evaluated
       REAL*8 SPEEDRE(NPTS1)    !array holding function values for all x
       REAL*8 SPEEDIM(NPTS1)   !array holding function values for all x
       INTEGER*4 INDX2,INDY2,INDS2,INDX1,INDX3,INDS1,INDS3,INDY1,INDY3
                   !indices in the table for Lagrange interpolation for
                                                    !each parameter X,Y,S
       INTEGER*4 INDX,INDY(2),INDS
                   !indices in the table for Lagrange interpolation for
                                                    !each parameter X,Y,S
       REAL*8 DELTAX,DELTAY(2),DELTAS         !difference between actual
                  !parameter value and value in table for Taylor series
       REAL*8 DIFFX,DIFFY,DIFFS,DIFFXD2,DIFFYD2,DIFFSD2
           !difference between actual parameter value and value in table
       REAL*8 OFFY,OFFS,OFFYINT,OFFSINT,DXINT,DYINT,DSINT
       REAL*8 P1,P2,P3                                      !Holds points
       REAL*8 X1,X2,X3                                !Holds x polynomials
       REAL*8 Y1,Y2,Y3                                !Holds y polynomials
       REAL*8 DX(2),DY(2),DS                               !holds spacings
       REAL*8 BOUND    !Division between regions with different spacings
       INTEGER*4 NUMBER,NUMBER2

       DATA DX,DY,OFFY,DS,OFFS,OFFYINT,OFFSINT,DXINT,DYINT,DSINT,BOUND&
       /.02D0,.05D0,.02D0,.05D0,.2D0,.03D0,.15D0,&
       .02D0,.16D0,.005D0,.005D0,.01D0,.2D0/
       DATA ROUTNUM/6,12,14,7,13,15/

       IF (S.LT.0.16D0) THEN
         IF (READIN(ROUTNUM(1))) CALL READINTERPSP(ROUTNUM(1))
         IF (READIN(ROUTNUM(2))) CALL READINTERPSP(ROUTNUM(2))

         INDY(1)=(Y-OFFY)/DY(1)+1.5D0+1.D-13
         INDY(2)=(Y-OFFY)/DY(2)+1.5D0+1.D-13
```

```
INDS=S/DS+1.5D0+1.D-13
DELTAY(1)=Y-OFFY-FLOAT(INDY(1)-1)*DY(1)
DELTAY(2)=Y-OFFY-FLOAT(INDY(2)-1)*DY(2)
DELTAS=S-FLOAT(INDS-1)*DS

NUMBER=0
NUMBER2=0

IF (XSTART.LT.-BOUND) THEN
  NUMBER=(-BOUND-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER=MIN(NUMBER,NPTS)
  DO IDELTA=IDELTABEG,NUMBER
    INDX=(-XSIGNED-BOUND)/DX(2)+1.5D0
    DELTAX=-XSIGNED-BOUND-FLOAT(INDX-1)*DX(2)

    SPEEDRE(IDELTA)=TABLE6H0(INDX,INDY(2),INDS,1)+&
    DELTAX*(TABLE6H0(INDX,INDY(2),INDS,2)+&
    DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,8)+&
    TABLE6H0(INDX,INDY(2),INDS,20)*DELTAS)+&
    TABLE6H0(INDX,INDY(2),INDS,9)*DELTAS+&
    .5D0*DELTAX*(TABLE6H0(INDX,INDY(2),INDS,5)+&
    TABLE6H0(INDX,INDY(2),INDS,14)*DELTAY(2)+&
    TABLE6H0(INDX,INDY(2),INDS,15)*DELTAS+&
    .33333333333333D0*TABLE6H0(INDX,INDY(2),INDS,11)*DELTAX))&
    +DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,3)+&
    TABLE6H0(INDX,INDY(2),INDS,10)*DELTAS+&
    .5D0*DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,6)+&
    TABLE6H0(INDX,INDY(2),INDS,17)*DELTAS+&
    TABLE6H0(INDX,INDY(2),INDS,16)*DELTAX+&
    .33333333333333D0*TABLE6H0(INDX,INDY(2),INDS,12)*DELTAY(2)))&
    +DELTAS*(TABLE6H0(INDX,INDY(2),INDS,4)+&
    .5D0*DELTAS*(TABLE6H0(INDX,INDY(2),INDS,7)&
    +TABLE6H0(INDX,INDY(2),INDS,18)*DELTAX+&
    TABLE6H0(INDX,INDY(2),INDS,19)*DELTAY(2)&
    +.33333333333333333D0*TABLE6H0&
    (INDX,INDY(2),INDS,13)*DELTAS))

    SPEEDIM(IDELTA)=-(TABLE6H0(INDX,INDY(2),INDS,21)+&
    DELTAX*(TABLE6H0(INDX,INDY(2),INDS,22)+&
    DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,28)+&
    TABLE6H0(INDX,INDY(2),INDS,40)*DELTAS)+&
    TABLE6H0(INDX,INDY(2),INDS,29)*DELTAS+&
    .5D0*DELTAX*(TABLE6H0(INDX,INDY(2),INDS,25)+&
    TABLE6H0(INDX,INDY(2),INDS,34)*DELTAY(2)+&
    TABLE6H0(INDX,INDY(2),INDS,35)*DELTAS+&
    .33333333333333D0*TABLE6H0(INDX,INDY(2),INDS,31)*DELTAX))&
    +DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,23)+&
    TABLE6H0(INDX,INDY(2),INDS,30)*DELTAS+&
    .5D0*DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,26)+&
    TABLE6H0(INDX,INDY(2),INDS,37)*DELTAS+&
    TABLE6H0(INDX,INDY(2),INDS,36)*DELTAX+&
    .33333333333333D0*TABLE6H0&
    (INDX,INDY(2),INDS,32)*DELTAY(2)))&
    +DELTAS*(TABLE6H0(INDX,INDY(2),INDS,24)+&
    .5D0*DELTAS*(TABLE6H0(INDX,INDY(2),INDS,27)&
    +TABLE6H0(INDX,INDY(2),INDS,38)*DELTAX+&
    TABLE6H0(INDX,INDY(2),INDS,39)*DELTAY(2)&
    +.33333333333333333D0*TABLE6H0&
    (INDX,INDY(2),INDS,33)*DELTAS)))
    XSIGNED=XSIGNED+XDELTA
```

```
  END DO
  IF (NUMBER.EQ.NPTS) RETURN
END IF

IF (XSTART.LT.-DX(1)) THEN
  NUMBER2=(-DX(1)-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER2=MIN(NUMBER2,NPTS)
  IF (NUMBER2.GT.NUMBER) THEN
    DO IDELTA=NUMBER+1,NUMBER2
      INDX=-XSIGNED/DX(1)+1.5D0
      DELTAX=-XSIGNED-FLOAT(INDX-1)*DX(1)

      SPEEDRE(IDELTA)=TABLE12H0(INDX,INDY(1),INDS,1)+&
      DELTAX*(TABLE12H0(INDX,INDY(1),INDS,2)+&
      DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,8)+&
      TABLE12H0(INDX,INDY(1),INDS,20)*DELTAS)+&
      TABLE12H0(INDX,INDY(1),INDS,9)*DELTAS+&
      .5D0*DELTAX*(TABLE12H0(INDX,INDY(1),INDS,5)+&
      TABLE12H0(INDX,INDY(1),INDS,14)*DELTAY(1)+&
      TABLE12H0(INDX,INDY(1),INDS,15)*DELTAS+&
      .33333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,11)*DELTAX))&
      +DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,3)+&
      TABLE12H0(INDX,INDY(1),INDS,10)*DELTAS+&
      .5D0*DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,6)+&
      TABLE12H0(INDX,INDY(1),INDS,17)*DELTAS+&
      TABLE12H0(INDX,INDY(1),INDS,16)*DELTAX+&
      .33333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,12)*DELTAY(1)))&
      +DELTAS*(TABLE12H0(INDX,INDY(1),INDS,4)+&
      .5D0*DELTAS*(TABLE12H0(INDX,INDY(1),INDS,7)&
      +TABLE12H0(INDX,INDY(1),INDS,18)*DELTAX+&
      TABLE12H0(INDX,INDY(1),INDS,19)*DELTAY(1)&
      +.33333333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,13)*DELTAS))

      SPEEDIM(IDELTA)=-(TABLE12H0(INDX,INDY(1),INDS,21)+&
      DELTAX*(TABLE12H0(INDX,INDY(1),INDS,22)+&
      DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,28)+&
      TABLE12H0(INDX,INDY(1),INDS,40)*DELTAS)+&
      TABLE12H0(INDX,INDY(1),INDS,29)*DELTAS+&
      .5D0*DELTAX*(TABLE12H0(INDX,INDY(1),INDS,25)+&
      TABLE12H0(INDX,INDY(1),INDS,34)*DELTAY(1)+&
      TABLE12H0(INDX,INDY(1),INDS,35)*DELTAS+&
      .33333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,31)*DELTAX))&
      +DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,23)+&
      TABLE12H0(INDX,INDY(1),INDS,30)*DELTAS+&
      .5D0*DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,26)+&
      TABLE12H0(INDX,INDY(1),INDS,37)*DELTAS+&
      TABLE12H0(INDX,INDY(1),INDS,36)*DELTAX+&
      .33333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,32)*DELTAY(1)))&
      +DELTAS*(TABLE12H0(INDX,INDY(1),INDS,24)+&
      .5D0*DELTAS*(TABLE12H0(INDX,INDY(1),INDS,27)&
      +TABLE12H0(INDX,INDY(1),INDS,38)*DELTAX+&
      TABLE12H0(INDX,INDY(1),INDS,39)*DELTAY(1)&
      +.33333333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,33)*DELTAS)))
      XSIGNED=XSIGNED+XDELTA
```

```
    END DO
    IF (NUMBER2.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER2=NUMBER
END IF
NUMBER=0

IF (XSTART.LT.DX(1)) THEN
  NUMBER=(DX(1)-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER=MIN(NUMBER,NPTS)
  IF (NUMBER.GT.NUMBER2) THEN
    IF (READIN(ROUTNUM(3))) CALL READINTERPSP(ROUTNUM(3))
    INDY2=(Y-OFFYINT)/DYINT+1.5D0+1.D-13
    INDY1=INDY2-1
    INDY3=INDY2+1
    INDS2=S/DSINT+1.5D0+1.D-13
    IF (INDS2.EQ.1) THEN
      INDS2=2
    ELSE IF (INDS2.EQ.17) THEN
      INDS2=16
    END IF
    INDS1=INDS2-1
    INDS3=INDS2+1
    DIFFY=(Y-OFFYINT)/DYINT-FLOAT(INDY2-1)
    DIFFS=S/DSINT-FLOAT(INDS2-1)
    DIFFYD2=DIFFY*.5D0
    DIFFSD2=DIFFS*.5D0
    DO IDELTA=NUMBER2+1,NUMBER
      X=ABS(XSIGNED)
      INDX=X/DX(1)+1.5D0+1.D-13
      INDX2=X/DXINT+1.5D0+1.D-13
      IF (INDX2.EQ.1) INDX2=2
      INDX1=INDX2-1
      INDX3=INDX2+1
      DIFFX=X/DXINT-FLOAT(INDX2-1)
      DIFFXD2=.5D0*DIFFX
      DIFFXD2=.5D0*DIFFX

      DELTAX=XSIGNED-FLOAT(INDX-1)*DX(1)
      SPEEDRE(IDELTA)=TABLE12H0(INDX,INDY(1),INDS,1)+&
      DELTAX*(TABLE12H0(INDX,INDY(1),INDS,2)+&
      DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,8)+&
      TABLE12H0(INDX,INDY(1),INDS,20)*DELTAS)+&
      TABLE12H0(INDX,INDY(1),INDS,9)*DELTAS+&
      .5D0*DELTAX*(TABLE12H0(INDX,INDY(1),INDS,5)+&
      TABLE12H0(INDX,INDY(1),INDS,14)*DELTAY(1)+&
      TABLE12H0(INDX,INDY(1),INDS,15)*DELTAS+&
      .33333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,11)*DELTAX))&
      +DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,3)+&
      TABLE12H0(INDX,INDY(1),INDS,10)*DELTAS+&
      .5D0*DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,6)+&
      TABLE12H0(INDX,INDY(1),INDS,17)*DELTAS+&
      TABLE12H0(INDX,INDY(1),INDS,16)*DELTAX+&
      .33333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,12)*DELTAY(1)))&
      +DELTAS*(TABLE12H0(INDX,INDY(1),INDS,4)+&
      .5D0*DELTAS*(TABLE12H0(INDX,INDY(1),INDS,7)&
      +TABLE12H0(INDX,INDY(1),INDS,18)*DELTAX+&
```

```
      TABLE12H0(INDX,INDY(1),INDS,19)*DELTAY(1)&
      +.33333333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,13)*DELTAS))

      P1=TABLE14H0(INDX1,INDY1,INDS1)
      P2=TABLE14H0(INDX2,INDY1,INDS1)
      P3=TABLE14H0(INDX3,INDY1,INDS1)
      X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
      P1=TABLE14H0(INDX1,INDY2,INDS1)
      P2=TABLE14H0(INDX2,INDY2,INDS1)
      P3=TABLE14H0(INDX3,INDY2,INDS1)
      X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
      P1=TABLE14H0(INDX1,INDY3,INDS1)
      P2=TABLE14H0(INDX2,INDY3,INDS1)
      P3=TABLE14H0(INDX3,INDY3,INDS1)
      X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
      Y1=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
      P1=TABLE14H0(INDX1,INDY1,INDS2)
      P2=TABLE14H0(INDX2,INDY1,INDS2)
      P3=TABLE14H0(INDX3,INDY1,INDS2)
      X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
      P1=TABLE14H0(INDX1,INDY2,INDS2)
      P2=TABLE14H0(INDX2,INDY2,INDS2)
      P3=TABLE14H0(INDX3,INDY2,INDS2)
      X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
      P1=TABLE14H0(INDX1,INDY3,INDS2)
      P2=TABLE14H0(INDX2,INDY3,INDS2)
      P3=TABLE14H0(INDX3,INDY3,INDS2)
      X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
      Y2=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
      P1=TABLE14H0(INDX1,INDY1,INDS3)
      P2=TABLE14H0(INDX2,INDY1,INDS3)
      P3=TABLE14H0(INDX3,INDY1,INDS3)
      X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
      P1=TABLE14H0(INDX1,INDY2,INDS3)
      P2=TABLE14H0(INDX2,INDY2,INDS3)
      P3=TABLE14H0(INDX3,INDY2,INDS3)
      X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
      P1=TABLE14H0(INDX1,INDY3,INDS3)
      P2=TABLE14H0(INDX2,INDY3,INDS3)
      P3=TABLE14H0(INDX3,INDY3,INDS3)
      X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
      Y3=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
      SPEEDIM(IDELTA)=SIGN(Y2+DIFFSD2*((Y3-Y1)+&
      DIFFS*(Y1+Y3-2.D0*Y2)),XSIGNED)
      XSIGNED=XSIGNED+XDELTA
    END DO
    IF (NUMBER.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER=NUMBER2
END IF
NUMBER2=0

IF (XSTART.LT.BOUND) THEN
  NUMBER2=(BOUND-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER2=MIN(NUMBER2,NPTS)
  IF (NUMBER2.GT.NUMBER) THEN
    DO IDELTA=NUMBER+1,NUMBER2
      INDX=XSIGNED/DX(1)+1.5D0
```

```
      DELTAX=XSIGNED-FLOAT(INDX-1)*DX(1)

      SPEEDRE(IDELTA)=TABLE12H0(INDX,INDY(1),INDS,1)+&
      DELTAX*(TABLE12H0(INDX,INDY(1),INDS,2)+&
      DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,8)+&
      TABLE12H0(INDX,INDY(1),INDS,20)*DELTAS)+&
      TABLE12H0(INDX,INDY(1),INDS,9)*DELTAS+&
      .5D0*DELTAX*(TABLE12H0(INDX,INDY(1),INDS,5)+&
      TABLE12H0(INDX,INDY(1),INDS,14)*DELTAY(1)+&
      TABLE12H0(INDX,INDY(1),INDS,15)*DELTAS+&
      .33333333333333D0*TABLE12H0(INDX,INDY(1),INDS,11)*DELTAX))&
      +DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,3)+&
      TABLE12H0(INDX,INDY(1),INDS,10)*DELTAS+&
      .5D0*DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,6)+&
      TABLE12H0(INDX,INDY(1),INDS,17)*DELTAS+&
      TABLE12H0(INDX,INDY(1),INDS,16)*DELTAX+&
      .33333333333333D0*TABLE12H0(INDX,INDY(1),INDS,12)*DELTAY(1)))&
      +DELTAS*(TABLE12H0(INDX,INDY(1),INDS,4)+&
      .5D0*DELTAS*(TABLE12H0(INDX,INDY(1),INDS,7)&
      +TABLE12H0(INDX,INDY(1),INDS,18)*DELTAX+&
      TABLE12H0(INDX,INDY(1),INDS,19)*DELTAY(1)&
      +.33333333333333333D0*TABLE12H0(INDX,INDY(1),INDS,13)*DELTAS))

      SPEEDIM(IDELTA)=TABLE12H0(INDX,INDY(1),INDS,21)+&
      DELTAX*(TABLE12H0(INDX,INDY(1),INDS,22)+&
      DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,28)+&
      TABLE12H0(INDX,INDY(1),INDS,40)*DELTAS)+&
      TABLE12H0(INDX,INDY(1),INDS,29)*DELTAS+&
      .5D0*DELTAX*(TABLE12H0(INDX,INDY(1),INDS,25)+&
      TABLE12H0(INDX,INDY(1),INDS,34)*DELTAY(1)+&
      TABLE12H0(INDX,INDY(1),INDS,35)*DELTAS+&
      .33333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,31)*DELTAX))&
      +DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,23)+&
      TABLE12H0(INDX,INDY(1),INDS,30)*DELTAS+&
      .5D0*DELTAY(1)*(TABLE12H0(INDX,INDY(1),INDS,26)+&
      TABLE12H0(INDX,INDY(1),INDS,37)*DELTAS+&
      TABLE12H0(INDX,INDY(1),INDS,36)*DELTAX+&
      .33333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,32)*DELTAY(1)))&
      +DELTAS*(TABLE12H0(INDX,INDY(1),INDS,24)+&
      .5D0*DELTAS*(TABLE12H0(INDX,INDY(1),INDS,27)&
      +TABLE12H0(INDX,INDY(1),INDS,38)*DELTAX+&
      TABLE12H0(INDX,INDY(1),INDS,39)*DELTAY(1)&
      +.33333333333333333D0*TABLE12H0&
      (INDX,INDY(1),INDS,33)*DELTAS))
      XSIGNED=XSIGNED+XDELTA
    END DO
    IF (NUMBER2.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER2=NUMBER
END IF
NUMBER=0

DO IDELTA=NUMBER2+1,IDELTAEND
  INDX=(XSIGNED-BOUND)/DX(2)+1.5D0
  DELTAX=XSIGNED-BOUND-FLOAT(INDX-1)*DX(2)

  SPEEDRE(IDELTA)=TABLE6H0(INDX,INDY(2),INDS,1)+&
```

```
      DELTAX*(TABLE6H0(INDX,INDY(2),INDS,2)+&
      DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,8)+&
      TABLE6H0(INDX,INDY(2),INDS,20)*DELTAS)+&
      TABLE6H0(INDX,INDY(2),INDS,9)*DELTAS+&
      .5D0*DELTAX*(TABLE6H0(INDX,INDY(2),INDS,5)+&
      TABLE6H0(INDX,INDY(2),INDS,14)*DELTAY(2)+&
      TABLE6H0(INDX,INDY(2),INDS,15)*DELTAS+&
      .33333333333333D0*TABLE6H0&
      (INDX,INDY(2),INDS,11)*DELTAX))&
      +DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,3)+&
      TABLE6H0(INDX,INDY(2),INDS,10)*DELTAS+&
      .5D0*DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,6)+&
      TABLE6H0(INDX,INDY(2),INDS,17)*DELTAS+&
      TABLE6H0(INDX,INDY(2),INDS,16)*DELTAX+&
      .33333333333333D0*TABLE6H0&
      (INDX,INDY(2),INDS,12)*DELTAY(2)))&
      +DELTAS*(TABLE6H0(INDX,INDY(2),INDS,4)+&
      .5D0*DELTAS*(TABLE6H0(INDX,INDY(2),INDS,7)&
      +TABLE6H0(INDX,INDY(2),INDS,18)*DELTAX+&
      TABLE6H0(INDX,INDY(2),INDS,19)*DELTAY(2)&
      +.33333333333333333D0*TABLE6H0&
      (INDX,INDY(2),INDS,13)*DELTAS))

      SPEEDIM(IDELTA)=TABLE6H0(INDX,INDY(2),INDS,21)+&
      DELTAX*(TABLE6H0(INDX,INDY(2),INDS,22)+&
      DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,28)+&
      TABLE6H0(INDX,INDY(2),INDS,40)*DELTAS)+&
      TABLE6H0(INDX,INDY(2),INDS,29)*DELTAS+&
      .5D0*DELTAX*(TABLE6H0(INDX,INDY(2),INDS,25)+&
      TABLE6H0(INDX,INDY(2),INDS,34)*DELTAY(2)+&
      TABLE6H0(INDX,INDY(2),INDS,35)*DELTAS+&
      .33333333333333D0*TABLE6H0(INDX,INDY(2),INDS,31)*DELTAX))&
      +DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,23)+&
      TABLE6H0(INDX,INDY(2),INDS,30)*DELTAS+&
      .5D0*DELTAY(2)*(TABLE6H0(INDX,INDY(2),INDS,26)+&
      TABLE6H0(INDX,INDY(2),INDS,37)*DELTAS+&
      TABLE6H0(INDX,INDY(2),INDS,36)*DELTAX+&
      .33333333333333D0*TABLE6H0(INDX,INDY(2),INDS,32)*DELTAY(2)))&
      +DELTAS*(TABLE6H0(INDX,INDY(2),INDS,24)+&
      .5D0*DELTAS*(TABLE6H0(INDX,INDY(2),INDS,27)&
      +TABLE6H0(INDX,INDY(2),INDS,38)*DELTAX+&
      TABLE6H0(INDX,INDY(2),INDS,39)*DELTAY(2)&
      +.33333333333333333D0*TABLE6H0&
      (INDX,INDY(2),INDS,33)*DELTAS))
      XSIGNED=XSIGNED+XDELTA
    END DO

  ELSE
    IF (READIN(ROUTNUM(4))) CALL READINTERPSP(ROUTNUM(4))
    IF (READIN(ROUTNUM(5))) CALL READINTERPSP(ROUTNUM(5))

    INDY(1)=(Y-OFFY)/DY(1)+1.5D0+1.D-13
    INDY(2)=(Y-OFFY)/DY(2)+1.5D0+1.D-13
    INDS=(S-OFFS)/DS+1.5D0+1.D-13
    DELTAY(1)=Y-OFFY-FLOAT(INDY(1)-1)*DY(1)
    DELTAY(2)=Y-OFFY-FLOAT(INDY(2)-1)*DY(2)
    DELTAS=S-OFFS-FLOAT(INDS-1)*DS

    NUMBER=0
    NUMBER2=0
```

```
IF (XSTART.LT.-BOUND) THEN
  NUMBER=(-BOUND-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER=MIN(NUMBER,NPTS)
  DO IDELTA=IDELTABEG,NUMBER
    INDX=(-XSIGNED-BOUND)/DX(2)+1.5D0
    DELTAX=-XSIGNED-BOUND-FLOAT(INDX-1)*DX(2)

    SPEEDRE(IDELTA)=TABLE7H0(INDX,INDY(2),INDS,1)+&
    DELTAX*(TABLE7H0(INDX,INDY(2),INDS,2)+&
    DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,8)+&
    TABLE7H0(INDX,INDY(2),INDS,20)*DELTAS)+&
    TABLE7H0(INDX,INDY(2),INDS,9)*DELTAS+&
    .5D0*DELTAX*(TABLE7H0(INDX,INDY(2),INDS,5)+&
    TABLE7H0(INDX,INDY(2),INDS,14)*DELTAY(2)+&
    TABLE7H0(INDX,INDY(2),INDS,15)*DELTAS+&
    .33333333333333D0*TABLE7H0&
    (INDX,INDY(2),INDS,11)*DELTAX))&
    +DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,3)+&
    TABLE7H0(INDX,INDY(2),INDS,10)*DELTAS+&
    .5D0*DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,6)+&
    TABLE7H0(INDX,INDY(2),INDS,17)*DELTAS+&
    TABLE7H0(INDX,INDY(2),INDS,16)*DELTAX+&
    .33333333333333D0*TABLE7H0&
    (INDX,INDY(2),INDS,12)*DELTAY(2)))&
    +DELTAS*(TABLE7H0(INDX,INDY(2),INDS,4)+&
    .5D0*DELTAS*(TABLE7H0(INDX,INDY(2),INDS,7)&
    +TABLE7H0(INDX,INDY(2),INDS,18)*DELTAX+&
    TABLE7H0(INDX,INDY(2),INDS,19)*DELTAY(2)&
    +.33333333333333333D0*TABLE7H0&
    (INDX,INDY(2),INDS,13)*DELTAS))

    SPEEDIM(IDELTA)=-(TABLE7H0(INDX,INDY(2),INDS,21)+&
    DELTAX*(TABLE7H0(INDX,INDY(2),INDS,22)+&
    DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,28)+&
    TABLE7H0(INDX,INDY(2),INDS,40)*DELTAS)+&
    TABLE7H0(INDX,INDY(2),INDS,29)*DELTAS+&
    .5D0*DELTAX*(TABLE7H0(INDX,INDY(2),INDS,25)+&
    TABLE7H0(INDX,INDY(2),INDS,34)*DELTAY(2)+&
    TABLE7H0(INDX,INDY(2),INDS,35)*DELTAS+&
    .33333333333333D0*TABLE7H0&
    (INDX,INDY(2),INDS,31)*DELTAX))&
    +DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,23)+&
    TABLE7H0(INDX,INDY(2),INDS,30)*DELTAS+&
    .5D0*DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,26)+&
    TABLE7H0(INDX,INDY(2),INDS,37)*DELTAS+&
    TABLE7H0(INDX,INDY(2),INDS,36)*DELTAX+&
    .33333333333333D0*TABLE7H0&
    (INDX,INDY(2),INDS,32)*DELTAY(2)))&
    +DELTAS*(TABLE7H0(INDX,INDY(2),INDS,24)+&
    .5D0*DELTAS*(TABLE7H0(INDX,INDY(2),INDS,27)&
    +TABLE7H0(INDX,INDY(2),INDS,38)*DELTAX+&
    TABLE7H0(INDX,INDY(2),INDS,39)*DELTAY(2)&
    +.33333333333333333D0*TABLE7H0&
    (INDX,INDY(2),INDS,33)*DELTAS)))
    XSIGNED=XSIGNED+XDELTA
  END DO
  IF (NUMBER.EQ.NPTS) RETURN
END IF
```

```
IF (XSTART.LT.-DX(1)) THEN
  NUMBER2=(-DX(1)-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER2=MIN(NUMBER2,NPTS)
  IF (NUMBER2.GT.NUMBER) THEN
    DO IDELTA=NUMBER+1,NUMBER2
      INDX=-XSIGNED/DX(1)+1.5D0
      DELTAX=-XSIGNED-FLOAT(INDX-1)*DX(1)

      SPEEDRE(IDELTA)=TABLE13H0(INDX,INDY(1),INDS,1)+&
      DELTAX*(TABLE13H0(INDX,INDY(1),INDS,2)+&
      DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,8)+&
      TABLE13H0(INDX,INDY(1),INDS,20)*DELTAS)+&
      TABLE13H0(INDX,INDY(1),INDS,9)*DELTAS+&
      .5D0*DELTAX*(TABLE13H0(INDX,INDY(1),INDS,5)+&
      TABLE13H0(INDX,INDY(1),INDS,14)*DELTAY(1)+&
      TABLE13H0(INDX,INDY(1),INDS,15)*DELTAS+&
      .33333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,11)*DELTAX))&
      +DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,3)+&
      TABLE13H0(INDX,INDY(1),INDS,10)*DELTAS+&
      .5D0*DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,6)+&
      TABLE13H0(INDX,INDY(1),INDS,17)*DELTAS+&
      TABLE13H0(INDX,INDY(1),INDS,16)*DELTAX+&
      .33333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,12)*DELTAY(1)))&
      +DELTAS*(TABLE13H0(INDX,INDY(1),INDS,4)+&
      .5D0*DELTAS*(TABLE13H0(INDX,INDY(1),INDS,7)&
      +TABLE13H0(INDX,INDY(1),INDS,18)*DELTAX+&
      TABLE13H0(INDX,INDY(1),INDS,19)*DELTAY(1)&
      +.33333333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,13)*DELTAS))

      SPEEDIM(IDELTA)=-(TABLE13H0(INDX,INDY(1),INDS,21)+&
      DELTAX*(TABLE13H0(INDX,INDY(1),INDS,22)+&
      DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,28)+&
      TABLE13H0(INDX,INDY(1),INDS,40)*DELTAS)+&
      TABLE13H0(INDX,INDY(1),INDS,29)*DELTAS+&
      .5D0*DELTAX*(TABLE13H0(INDX,INDY(1),INDS,25)+&
      TABLE13H0(INDX,INDY(1),INDS,34)*DELTAY(1)+&
      TABLE13H0(INDX,INDY(1),INDS,35)*DELTAS+&
      .33333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,31)*DELTAX))&
      +DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,23)+&
      TABLE13H0(INDX,INDY(1),INDS,30)*DELTAS+&
      .5D0*DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,26)+&
      TABLE13H0(INDX,INDY(1),INDS,37)*DELTAS+&
      TABLE13H0(INDX,INDY(1),INDS,36)*DELTAX+&
      .33333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,32)*DELTAY(1)))&
      +DELTAS*(TABLE13H0(INDX,INDY(1),INDS,24)+&
      .5D0*DELTAS*(TABLE13H0(INDX,INDY(1),INDS,27)&
      +TABLE13H0(INDX,INDY(1),INDS,38)*DELTAX+&
      TABLE13H0(INDX,INDY(1),INDS,39)*DELTAY(1)&
      +.33333333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,33)*DELTAS)))
      XSIGNED=XSIGNED+XDELTA
    END DO
    IF (NUMBER2.EQ.NPTS) RETURN
  END IF
ELSE
```

```
  NUMBER2=NUMBER
END IF
NUMBER=0

IF (XSTART.LT.DX(1)) THEN
  NUMBER=(DX(1)-XSTART)/XDELTA+1.D0+1.D-13
  NUMBER=MIN(NUMBER,NPTS)
  IF (NUMBER.GT.NUMBER2) THEN
    IF (READIN(ROUTNUM(6))) CALL READINTERPSP(ROUTNUM(6))
    INDY2=(Y-OFFYINT)/DYINT+1.5D0+1.D-13
    INDY1=INDY2-1
    INDY3=INDY2+1
    INDS2=(S-OFFSINT)/DSINT+1.5D0+1.D-13
    IF (INDS2.EQ.1) THEN
      INDS2=2
    ELSE IF (INDS2.EQ.9) THEN
      INDS2=8
    END IF
    INDS1=INDS2-1
    INDS3=INDS2+1
    DIFFY=(Y-OFFYINT)/DYINT-FLOAT(INDY2-1)
    DIFFS=(S-OFFSINT)/DSINT-FLOAT(INDS2-1)
    DIFFYD2=DIFFY*.5D0
    DIFFSD2=DIFFS*.5D0
    DO IDELTA=NUMBER2+1,NUMBER
      X=ABS(XSIGNED)
      INDX=X/DX(1)+1.5D0+1.D-13
      INDX2=X/DXINT+1.5D0+1.D-13
      IF (INDX2.EQ.1) INDX2=2
      INDX1=INDX2-1
      INDX3=INDX2+1
      DIFFX=X/DXINT-FLOAT(INDX2-1)
      DIFFXD2=.5D0*DIFFX
      DIFFXD2=.5D0*DIFFX

      DELTAX=XSIGNED-FLOAT(INDX-1)*DX(1)
      SPEEDRE(IDELTA)=TABLE13H0(INDX,INDY(1),INDS,1)+&
      DELTAX*(TABLE13H0(INDX,INDY(1),INDS,2)+&
      DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,8)+&
      TABLE13H0(INDX,INDY(1),INDS,20)*DELTAS)+&
      TABLE13H0(INDX,INDY(1),INDS,9)*DELTAS+&
      .5D0*DELTAX*(TABLE13H0(INDX,INDY(1),INDS,5)+&
      TABLE13H0(INDX,INDY(1),INDS,14)*DELTAY(1)+&
      TABLE13H0(INDX,INDY(1),INDS,15)*DELTAS+&
      .33333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,11)*DELTAX))&
      +DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,3)+&
      TABLE13H0(INDX,INDY(1),INDS,10)*DELTAS+&
      .5D0*DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,6)+&
      TABLE13H0(INDX,INDY(1),INDS,17)*DELTAS+&
      TABLE13H0(INDX,INDY(1),INDS,16)*DELTAX+&
      .33333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,12)*DELTAY(1)))&
      +DELTAS*(TABLE13H0(INDX,INDY(1),INDS,4)+&
      .5D0*DELTAS*(TABLE13H0(INDX,INDY(1),INDS,7)&
      +TABLE13H0(INDX,INDY(1),INDS,18)*DELTAX+&
      TABLE13H0(INDX,INDY(1),INDS,19)*DELTAY(1)&
      +.33333333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,13)*DELTAS))
```

```
        P1=TABLE15H0(INDX1,INDY1,INDS1)
        P2=TABLE15H0(INDX2,INDY1,INDS1)
        P3=TABLE15H0(INDX3,INDY1,INDS1)
        X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
        P1=TABLE15H0(INDX1,INDY2,INDS1)
        P2=TABLE15H0(INDX2,INDY2,INDS1)
        P3=TABLE15H0(INDX3,INDY2,INDS1)
        X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
        P1=TABLE15H0(INDX1,INDY3,INDS1)
        P2=TABLE15H0(INDX2,INDY3,INDS1)
        P3=TABLE15H0(INDX3,INDY3,INDS1)
        X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
        Y1=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
        P1=TABLE15H0(INDX1,INDY1,INDS2)
        P2=TABLE15H0(INDX2,INDY1,INDS2)
        P3=TABLE15H0(INDX3,INDY1,INDS2)
        X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
        P1=TABLE15H0(INDX1,INDY2,INDS2)
        P2=TABLE15H0(INDX2,INDY2,INDS2)
        P3=TABLE15H0(INDX3,INDY2,INDS2)
        X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
        P1=TABLE15H0(INDX1,INDY3,INDS2)
        P2=TABLE15H0(INDX2,INDY3,INDS2)
        P3=TABLE15H0(INDX3,INDY3,INDS2)
        X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
        Y2=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
        P1=TABLE15H0(INDX1,INDY1,INDS3)
        P2=TABLE15H0(INDX2,INDY1,INDS3)
        P3=TABLE15H0(INDX3,INDY1,INDS3)
        X1=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
        P1=TABLE15H0(INDX1,INDY2,INDS3)
        P2=TABLE15H0(INDX2,INDY2,INDS3)
        P3=TABLE15H0(INDX3,INDY2,INDS3)
        X2=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
        P1=TABLE15H0(INDX1,INDY3,INDS3)
        P2=TABLE15H0(INDX2,INDY3,INDS3)
        P3=TABLE15H0(INDX3,INDY3,INDS3)
        X3=P2+DIFFXD2*((P3-P1)+DIFFX*(P1+P3-2.D0*P2))
        Y3=X2+DIFFYD2*((X3-X1)+DIFFY*(X1+X3-2.D0*X2))
        SPEEDIM(IDELTA)=SIGN(Y2+DIFFSD2*((Y3-Y1)+&
        DIFFS*(Y1+Y3-2.D0*Y2)),XSIGNED)
        XSIGNED=XSIGNED+XDELTA
      END DO
      IF (NUMBER.EQ.NPTS) RETURN
    END IF
  ELSE
    NUMBER=NUMBER2
  END IF
  NUMBER2=0

  IF (XSTART.LT.BOUND) THEN
    NUMBER2=(BOUND-XSTART)/XDELTA+1.D0+1.D-13
    NUMBER2=MIN(NUMBER2,NPTS)
    IF (NUMBER2.GT.NUMBER) THEN
      DO IDELTA=NUMBER+1,NUMBER2
        INDX=XSIGNED/DX(1)+1.5D0
        DELTAX=XSIGNED-FLOAT(INDX-1)*DX(1)

        SPEEDRE(IDELTA)=TABLE13H0(INDX,INDY(1),INDS,1)+&
        DELTAX*(TABLE13H0(INDX,INDY(1),INDS,2)+&
```

```
      DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,8)+&
      TABLE13H0(INDX,INDY(1),INDS,20)*DELTAS)+&
      TABLE13H0(INDX,INDY(1),INDS,9)*DELTAS+&
      .5D0*DELTAX*(TABLE13H0(INDX,INDY(1),INDS,5)+&
      TABLE13H0(INDX,INDY(1),INDS,14)*DELTAY(1)+&
      TABLE13H0(INDX,INDY(1),INDS,15)*DELTAS+&
      .33333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,11)*DELTAX))&
      +DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,3)+&
      TABLE13H0(INDX,INDY(1),INDS,10)*DELTAS+&
      .5D0*DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,6)+&
      TABLE13H0(INDX,INDY(1),INDS,17)*DELTAS+&
      TABLE13H0(INDX,INDY(1),INDS,16)*DELTAX+&
      .33333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,12)*DELTAY(1)))&
      +DELTAS*(TABLE13H0(INDX,INDY(1),INDS,4)+&
      .5D0*DELTAS*(TABLE13H0(INDX,INDY(1),INDS,7)&
      +TABLE13H0(INDX,INDY(1),INDS,18)*DELTAX+&
      TABLE13H0(INDX,INDY(1),INDS,19)*DELTAY(1)&
      +.33333333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,13)*DELTAS))

      SPEEDIM(IDELTA)=TABLE13H0(INDX,INDY(1),INDS,21)+&
      DELTAX*(TABLE13H0(INDX,INDY(1),INDS,22)+&
      DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,28)+&
      TABLE13H0(INDX,INDY(1),INDS,40)*DELTAS)+&
      TABLE13H0(INDX,INDY(1),INDS,29)*DELTAS+&
      .5D0*DELTAX*(TABLE13H0(INDX,INDY(1),INDS,25)+&
      TABLE13H0(INDX,INDY(1),INDS,34)*DELTAY(1)+&
      TABLE13H0(INDX,INDY(1),INDS,35)*DELTAS+&
      .33333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,31)*DELTAX))&
      +DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,23)+&
      TABLE13H0(INDX,INDY(1),INDS,30)*DELTAS+&
      .5D0*DELTAY(1)*(TABLE13H0(INDX,INDY(1),INDS,26)+&
      TABLE13H0(INDX,INDY(1),INDS,37)*DELTAS+&
      TABLE13H0(INDX,INDY(1),INDS,36)*DELTAX+&
      .33333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,32)*DELTAY(1)))&
      +DELTAS*(TABLE13H0(INDX,INDY(1),INDS,24)+&
      .5D0*DELTAS*(TABLE13H0(INDX,INDY(1),INDS,27)&
      +TABLE13H0(INDX,INDY(1),INDS,38)*DELTAX+&
      TABLE13H0(INDX,INDY(1),INDS,39)*DELTAY(1)&
      +.33333333333333333D0*TABLE13H0&
      (INDX,INDY(1),INDS,33)*DELTAS))
      XSIGNED=XSIGNED+XDELTA
    END DO
    IF (NUMBER2.EQ.NPTS) RETURN
  END IF
ELSE
  NUMBER2=NUMBER
END IF
NUMBER=0

DO IDELTA=NUMBER2+1,IDELTAEND
  INDX=(XSIGNED-BOUND)/DX(2)+1.5D0
  DELTAX=XSIGNED-BOUND-FLOAT(INDX-1)*DX(2)

  SPEEDRE(IDELTA)=TABLE7H0(INDX,INDY(2),INDS,1)+&
  DELTAX*(TABLE7H0(INDX,INDY(2),INDS,2)+&
```

```
      DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,8)+&
      TABLE7H0(INDX,INDY(2),INDS,20)*DELTAS)+&
      TABLE7H0(INDX,INDY(2),INDS,9)*DELTAS+&
      .5D0*DELTAX*(TABLE7H0(INDX,INDY(2),INDS,5)+&
      TABLE7H0(INDX,INDY(2),INDS,14)*DELTAY(2)+&
      TABLE7H0(INDX,INDY(2),INDS,15)*DELTAS+&
      .33333333333333D0*TABLE7H0&
      (INDX,INDY(2),INDS,11)*DELTAX))&
      +DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,3)+&
      TABLE7H0(INDX,INDY(2),INDS,10)*DELTAS+&
      .5D0*DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,6)+&
      TABLE7H0(INDX,INDY(2),INDS,17)*DELTAS+&
      TABLE7H0(INDX,INDY(2),INDS,16)*DELTAX+&
      .33333333333333D0*TABLE7H0&
      (INDX,INDY(2),INDS,12)*DELTAY(2)))&
      +DELTAS*(TABLE7H0(INDX,INDY(2),INDS,4)+&
      .5D0*DELTAS*(TABLE7H0(INDX,INDY(2),INDS,7)&
      +TABLE7H0(INDX,INDY(2),INDS,18)*DELTAX+&
      TABLE7H0(INDX,INDY(2),INDS,19)*DELTAY(2)&
      +.33333333333333333D0*TABLE7H0&
      (INDX,INDY(2),INDS,13)*DELTAS))

      SPEEDIM(IDELTA)=TABLE7H0(INDX,INDY(2),INDS,21)+&
      DELTAX*(TABLE7H0(INDX,INDY(2),INDS,22)+&
      DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,28)+&
      TABLE7H0(INDX,INDY(2),INDS,40)*DELTAS)+&
      TABLE7H0(INDX,INDY(2),INDS,29)*DELTAS+&
      .5D0*DELTAX*(TABLE7H0(INDX,INDY(2),INDS,25)+&
      TABLE7H0(INDX,INDY(2),INDS,34)*DELTAY(2)+&
      TABLE7H0(INDX,INDY(2),INDS,35)*DELTAS+&
      .33333333333333D0*TABLE7H0&
      (INDX,INDY(2),INDS,31)*DELTAX))&
      +DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,23)+&
      TABLE7H0(INDX,INDY(2),INDS,30)*DELTAS+&
      .5D0*DELTAY(2)*(TABLE7H0(INDX,INDY(2),INDS,26)+&
      TABLE7H0(INDX,INDY(2),INDS,37)*DELTAS+&
      TABLE7H0(INDX,INDY(2),INDS,36)*DELTAX+&
      .33333333333333D0*TABLE7H0&
      (INDX,INDY(2),INDS,32)*DELTAY(2)))&
      +DELTAS*(TABLE7H0(INDX,INDY(2),INDS,24)+&
      .5D0*DELTAS*(TABLE7H0(INDX,INDY(2),INDS,27)&
      +TABLE7H0(INDX,INDY(2),INDS,38)*DELTAX+&
      TABLE7H0(INDX,INDY(2),INDS,39)*DELTAY(2)&
      +.33333333333333333D0*TABLE7H0&
      (INDX,INDY(2),INDS,33)*DELTAS))
      XSIGNED=XSIGNED+XDELTA
    END DO
  END IF

  END SUBROUTINE
```