Table of contents

SpecView and Data Analysis

1. Introduction.

The analysis of the experimental data is one of the most important parts of the researcherís cycle. In the world of spectroscopy that consists of putting together experimental data, identifying spectral lines and fitting spectra. To automate this process scientists usually write a new program or try to adapt a previously written one. As a result many research groups have collected and developed a number of ìhome-grownî programs and utilities that help in the analysis of data. Many such programs serve as a tool to analyze the spectrum of a specific molecule and have a poor user interface. The ìcommunicationî with such a program involves creation and editing of large tables and text files describing the systemís parameters. Obviously when the user of the utility is a well-skilled programmer (especially the guy who wrote the code) that approach can be acceptable, though the repetitive usage of the program with a error prone data input (like assignment tables) leads to an effective loss of the researcherís valuable time.

The process of data analysis in spectroscopy can be done in a few steps:

1. Measure the exact frequency positions of lines in an experimental spectrum.

2. Simulate the spectrum of the molecule for a given set of Hamiltonianís constants and a set of modelís parameters.

3. Making the assignment for experimental lines (associate them in a unique way with spectral line profiles in a simulated spectrum).

4. Run a fitting procedure in an attempt to get the molecular constants best describing experimental spectrum.

Following this generic approach a standard package for data analysis consists of number of programs doing simulating or fitting with standardized input and output data files and a set of graphical user interface programs that help to visualize data files (CalFit). Though such an approach dramatically simplifies the process of data analysis, it does not make it easier to develop software for a new molecular system.

The purpose of our work was to create an intuitive user - friendly instrument that would consolidate in one all the tools needed for detailed spectroscopic analysis of experimental spectra. More, the application allows one to work with **any** molecular system. Additionally, the generalized fitting procedure is designed not to ì loseî assignment between consecutive iterations ñ a wellñknown problem for many fitting programs (to be discussed in details later).

2. Principles of SpecView.

SpecView is not just a graphical user interface (GUI) program for running trial simulation. Unlike many other predecessors it does not need any other program to measure positions of experimental lines, simulate the spectrum or run a fitting procedure on molecular constants. All those engines are already included in SpecView. The only limitation of the program (that could be avoided by a scientist who writes a model for molecular system) is a two ñ state approach. That means that all transitions are assumed to occur between two main states of the molecule, referred in a program as a ground state and an excited state (for example in a linear molecule a good choice for the two would be two electronic states between which the transitions are considered: Σ, Π, Δ ...) . Usually each of these two states has a structure of internal states defined by the Hamiltonian. For the case of Asymmetric rotor the Hamiltonian defines rotational structure for both states, though with a different Hamiltonian constants.

As was stipulated above SpecView is a program that is able to simulate and fit the spectrum of any molecular system. To accomplish that the program is logically and programmatically divided in two parts ñ a model independent part and a set of functions that completely describe the particular molecular model. The model-independent part includes simulation and fitting engines and is compiled in one executable file ñ SpecView.exe . The model dependent part that contains the body of functions is loaded from dynamic link libraries (dll) at run time. Any user, even with minor knowledge in programming, should be able to extend the model library by creating his own *.dll file and registering it via the SpecView.ini file in the Windows system directory of the computer.

The graphical user interface part of the program provides convenient tools to:

- ✓ edit values of the molecular constants;
- ✓ edit values of the parameters (temperature, line width);
- ✓ simulate spectral line profiles with a given Doppler or Lorentz line width;
- ✓ import experimental spectra and display them along with simulated spectra;
- ✓ assign the frequency of lines in the experimental spectrum to selected transitions in the simulated one;
- ✓ select a group of molecular constants for adjustment in the fitting procedure;
- ✓ run non-linear least square (NLSQ) fitting and display its results;

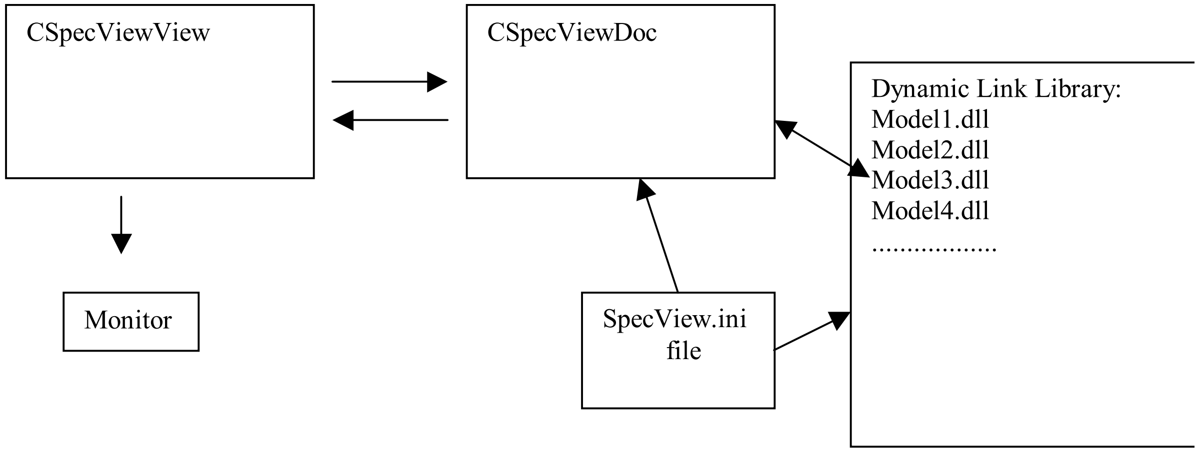
- ✓ change selected constants dynamically;
- ✓ select a group of simulated transitions based on energy criteria or constrains on quantum numbers ñ an excellent tool to analyze the influence of given constants on certain molecular transitions;
- ✓ Draw state energy diagrams for a group of states;
- ✓ export results of simulations and fitting;
- ✓ Print the simulated as well as the experimental spectrum.

Although the GUI part of the program is written specifically for Windows operating system, spectrum simulation and fitting procedures are written with plain C++ that allows to port the calculational part of the program to different operating systems (Linux or Appleís MacOS).

In many senses SpecView is similar to the earlier program ìSpecSimî developed by Sergey Panov. I have to note that while taking good genes from SpecSim, SpecView is a completely different program with its own internal structure.

2.1 Internal structure of the program.

SpecView is written in C++ object oriented language with the extensive use of Microsoft Foundation Classes (MFC) as well as Windows API. All supplied dynamic link modules are written in plain C to make it easier for one not very familiar with programming to edit them. As standard for MFC programming the project contains a number of classes describing the behavior of all Dialog Boxes in the GUI part of the program, but the main code is put inside two core classes of the application ñ CSpecViewDoc ñ inherited from CDocument and CSpecViewView inherited from CView class. All procedures and data not directly related to GUI are embedded in an object of CSpecViewDoc class, and an object of CSpecViewView class is responsible for displaying the processed data on a monitor of the computer. The object of class CSpecViewDoc contains the information about a current set of constants, parameters, current assignment and simulation. It is bound to one of the dynamically loadable modules at a time. Each of the modules provides the same set of routines (21 functions with the same declaration), although the implementation of each of them is dependent on a particular molecular model. A more detail description how those model work and how to write a new model are given in the last section.



Several C++ classes were defined for internal representation of spectroscopic information such as states, transitions and assignments. The following diagram shows some data members of the listed above structures.

CState:

A set of quantum numbers identifying the state;
Wavefunction of the state;
Energy of the state relative to the origin of the ìelectronicî state;
Stateís ID (used for internal representation of the state)

CTransition

A pointer to an object of CState (ground state);
A pointer to an object of CState (excited state);
Intensity of transition;
Frequency of transition

CAssignment:

A pointer to an object of class CTransition
Frequency assigned to a transition (frequency of a line from experimental spectrum)
A number of flags;
GarbageCollection ñ static member of the class that provides memory managing and prompt deletion of not used objects of a class.

Numerous other member functions that help to manipulate data members of the described above classes are not shown on a diagram.

For large calculations (which for example is typical for spectral simulations at room temperature) the information about molecular states can occupy megabytes of memory. To manage the memory in a smart way, once created, the object of one of those classes is never duplicated. Instead to manipulate data members of the object the program uses references to the object.

2.2 Calculation of a spectrum.

The core of the simulation of the spectrum is the member function CalculateSpectr() of the class CSpecViewDoc. We will describe this function in detail since it uses model dependent functions and a full understanding of its structure is required for those who write models.

As was mentioned before the program calculates transitions between two main electronic states. The internal level structure of the upper and lower states is determined by the Hamiltonian. Typically the ground and excited states would be electronic or vibronic states with their own rotational structure. Most rotational and ro - vibrational Hamiltonians have a block-diagonal form. The Hamiltonian is defined in a particular basis set, and every basis function is labeled by its own set of quantum numbers.

Quantum numbers can be divided in two categories ñ good quantum numbers and bad quantum numbers. Good quantum numbers are those that uniquely identify blocks in a block ñ diagonal Hamiltonian matrix (the corresponding operator commutes with the Hamiltonian).

Bad quantum numbers help to identify the eigenstates within a block (the operator relative to quantum number does not commute with the Hamiltonian).

The search for eigenstates starts with the diagonalization of the Hamiltonian matrix. Since it has block-diagonal shape it is much more efficient to diagonalize each of the blocks (characterized by a set of good quantum numbers) individually.

So on the first step the program creates two arrays ñ array of objects of the class CState that describe states in the ground electronic state and array of objects of the class CState that describe states in the excited electronic state.

Here is the algorithm:

1. Generate a set of good quantum numbers for the first time (function StartQN() of the DLL module);

2. For the given set of quantum numbers find the size of the Hamiltonian matrix (function HamSize() of the DLL module);
3. Allocate memory for the matrix;
4. Initialize the matrix for the current set of molecular constants and good quantum numbers (function Hamilt() of the module);
5. Diagonalize the matrix, calculating eigenvectors and eigenvalues.
6. Try the default assignment (bad quantum numbers, since good ones are already defined): for each eigenvector find the maximum coefficient in the wavefunction. The quantum numbers of the corresponding basis function can be accepted as bad quantum numbers of the state.
7. Run custom defined assignment function (Assign() function from DLL module), memorize a final set of quantum numbers.
8. Increment the set of good quantum numbers ñ switch to another block of the global Hamiltonian matrix (NextQN() function of DLL module).
9. Check if you have to stop the loop (ContQN() function of the module) and if yes ñ quit, go to step 2 otherwise.

While creating an array of states in the ground electronic state they are sorted in energy order for reasons explained later. After all the information about states is collected, the program starts to calculate the intensity of the each possible transition. For each state from the ground electronic state try every state in the excited electronic state. Calculate the intensity of the transition (Inten() function of the module) and if itís non-zero ñ create an object of class CTrans and memorize it. The first time the Inten() function is called it remembers the energy of the lowest state in the ground electronic state, which is used for the calculation of the Bolzman factor. That is why states in the ground electronic state are sorted in energy order before the intensities of the transitions are calculated.

The new feature of the program ñ dynamic change of a constant ñ allows changing one of the molecular constants just with a touch of mouse wheel. It lets the user to explore how the simulation behaves with the change of a particular constant. Unfortunately for the models with a large size of Hamiltonian matrices (for example asymmetric top with spin-rotation) the time-greedy process of matrix diagonalization necessary for spectra simulation may take up to a few seconds even for fast machines (the program was tested mostly on computers with 1.0GHz CPUs). That inevitably leads to noticeable delay between the moment the dynamic constant is incremented and the recalculated spectrum is drawn on a screen.

To avoid that an interesting approach was taken. The program offers approximate and fast simulation of the spectral profiles (CalculateFast() and CalculateAccurate() member functions of SpecViewDoc class). It is based on two assumptions:

1. Since the constant is incremented by a small amount, the wavefunctions of the eigenstates to the first approximation do not change and since the intensities of the transitions stay the same.
2. The frequency of the transition is calculate based on

$E_{tr} = T_e + E_{excited\ state} - E_{ground\ state}$

Hence if the constant changed dynamically is the ground state constant than the correction to the frequency position of every transition in the spectrum could be calculated as

$dE_{tr} = -dE_{ground\ state}$

Correspondingly, if the dynamic constant belongs to excited electronic stateís set of constants

$dE_{tr} = dE_{excited\ state}$

Now the correction to the energy of the state is calculated as

$dE_{state} = (\partial E_{state}/\partial a_i)da_i$

where a_i is the incremented constant.

The program generates a spectrum using this algorithm only while incrementing dynamic constant and the application is in ì Fast modeî (function CalculateFast() of SpecViewDoc class). If the application is in the default ì Normal modeî , the spectrum is always recalculated using the precise CalculateSpectr() procedure.

To get the energy correction with the accuracy of up to second order terms in the Taylorís expansion the program uses the following expression:

$$dE_{state} = 1/2(\partial E_{state}(\mathbf{a_0})/\partial a_i + \partial E_{state}(\mathbf{a_0} + d\mathbf{a_i})/\partial a_i)da_i$$

This algorithm is utilized in CalculateAccurate() function of CSpecViewDoc class. This mode is available at any time and is activated when PgUp or PgDown buttons are pressed.

Although giving a clear advantage of fast respond of the program to the change of dynamic constant, the described above modes should be used carefully because:

1. The wavefunctions of the states are not recalculated, since the intensities could not be trusted if the mode is run for a long time. To avoid that we advise to recalculate all states as well as intensities of transitions between them by clicking "*Enter*" button once in while (that invokes the main calculational function CalculateSpectr());

2. If any assignment exist, the assigned lines of the simulated spectrum are not updated during fast generation of spectrum and the next time the spectrum will be recalculated in a normal mode part of the assignment can be lost if total change of vector of molecular constants since the last time wavefunction were updated is significant. So as a precaution: SAVE CURRENT DOCUMENT (File->Save option of the application Menu or Save Toolbar button) to *.svw file if any assignment was done.

Another downside of the fast mode is that the derivative $\partial E_{state}(\mathbf{a_0})/\partial a_i$ is provided by current module. For all models included currently in SpecView package (9 *.dll files) that function is defined and the fast mode provides great opportunity to explore multidimensional surface of spectrum in a space of molecular constants with a speed of rolling a mouse wheel. However if a writer of a new model does not provide the body of the function DHamilt() the fast calculational mode will not be operation able.

2.3. The Assignment and fit.

In many other programs the process of assignment requires from a user to import into the program large tables containing in general a list of pairs of data: the set of quantum numbers completely describing the ground and excited state of the transition (essentially the unique label for the transition) and the measured frequency of the corresponding experimental line profile. The standard method to get a label (set of quantum numbers) for an experimental line profile is to get a trial simulation that sufficiently resembles the experimental spectrum. Then the quantum numbers for labeling experimental profiles are taken from the simulated transitions that match experimental profiles. In case the match can not be done unambiguously, some programs (e.g. CalFit) still allow for the assignment to be done, but a third number accompanies the pair of label ñ frequency - the weight of the assignment (a measure of ì trustî put in a transition) that is less than unity.

This approach works in many cases, though to do a fit on the selected assignment two conditions should be satisfied:

1. The assignment should be unique, that is each simulated line should be assigned to a unique set of quantum numbers;
2. The assignment of the simulated line should not change when molecular constants are varied in a reasonable region;

The requirement #1 is clear - the fitting procedure minimizes the uncertainty between the experimental data and modeling function which in our case is not analytical (could not be given as analytical expression but rather is a solution of circular equation) and as for any function there could be only one value (frequency of the measured profile) for a given a argument (label of a line). The first condition is automatically satisfied if the Hamiltonian matrix is close to diagonal with far from degenerate eigenvalues. In that case the default assigning procedure described in main calculational part of the program CalculateSpectr() gives a unique assignment. Unfortunately for a highly blended eigenstates it is possible that the maximum coefficient in the wavefunctions of two states correspond to the same function of the basis set. Clearly in that case the default assignment procedure fails, and the module provided Assign() function is needed.

Another good example that proves the need of the custom Assign() function is the simple case of asymmetric rotor. The natural choice of basis functions for that model is a set eigenfunctions of the Hamiltonian of symmetric top:

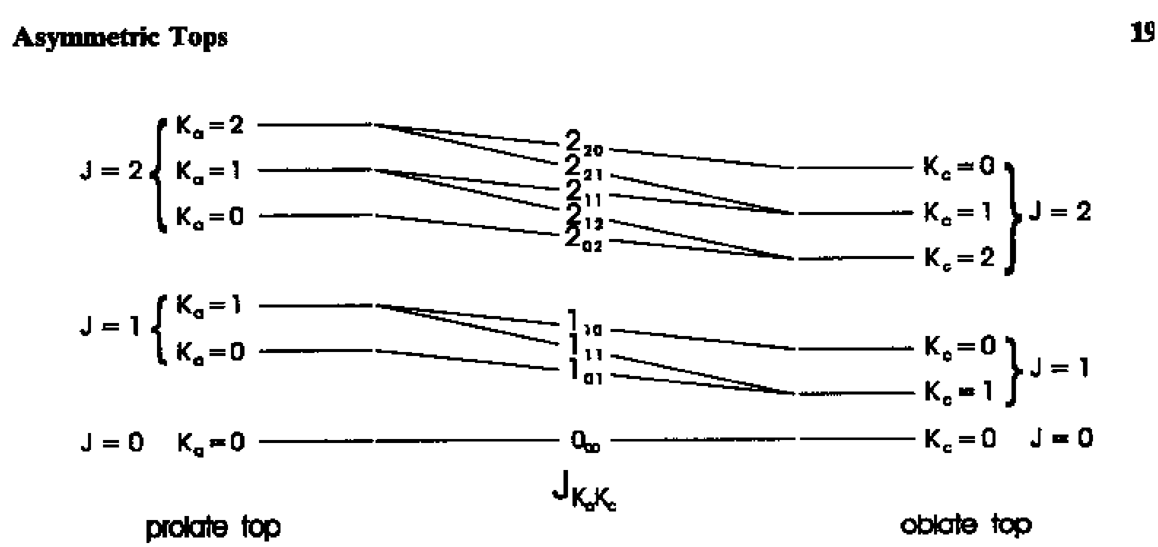
| J, K>

Since the eigenfunctions of the asymmetric rotorís states are represented by symmetric and antisymmetric combinations with equal weights of two basis functions (for K ≠ 0), the default procedure can not be used, the parity ì badî quantum number has to be introduced and the function Assign() is obviously a must.

The requirement for condition #2 comes from a fact that NLSQ method used for a fitting procedure is an iterative method, and while converging to a minimum recalculates merit function in a few points in a space of molecular constants. In order for the assignment not to be ì lostî or ì jumpî the label of the simulated line should not change under any change of the adjusted molecular constants.

That requirement makes difficult the writing a ì goodî Assign() function (good in a sense that it will allow to run fitting procedure successfully).

An interesting approach is taken in SpecSim program. Instead



of using a set of quantum numbers as a label for a transition, the program labels states by a position of the state in energy order. This method works perfectly for the modeling of the spectra of asymmetric rotor:

The diagram shows that one index completely replaces two ì badî quantum numbers K_a and K_c standard for the labeling of eigenstates of asymmetric rotor. We have to mention that that trick often does not work when large distortion constants are included in a Hamiltonian and never works when spin-rotation terms distort the energy diagram of asymmetric rotor.

In SpecView the tedious process of measuring the positions of experimental lines, analyzing the sets of quantum numbers for simulated spectrum and creating the tables for the assignments is replaced by a simple and intuitive action : the experimental profile ñ simulated profile pair is memorized in the program when you sequentially click on the matching lines with a mouse.

We also elaborated the generic approach to label molecular states to satisfy 2 assignment requirements for successful fit. Instead of using quantum numbers or any other index replacing them we consider a wavefunction of state as a unique label, that changes continuously with a change of molecular constants. On each iteration of fitting procedure the wavefunction of the each of the state of the assigned line is compared with a set of wavefunctions of eigenstates of Hamiltonian matrix with the set of good quantum numbers matching those for assigned line. The eigenstate with the ì closestî wavefunction is chosen as the state representing the new state of the assigned line and the info about the assigned line is updated in the memory of the computer (energies and wavefunctions of the excited and ground states of the CAssign object are refreshed). The ì closestî is defined as the minimum of

$\Sigma(w_{i\,k}$ñ $w_{a\,k})$ or $\Sigma(w_{i\,k}$ñ $w_{a\,k})^2$ depending on the current *caption mode.* The default one is *Wavefunction, Squared.* $w_{i\,k}$ stands for k-th coefficient in the i-th eigenvector of a recalculated Hamiltonian matrix and $w_{a\,k}$ stands for k-th coefficient of an assigned line wavefunction.

The program also allows using energy order position index as a label for the eigenstates of the assigned lines ñ *Energy Order* caption mode. And the last caption mode used for the labeling in the program is called *Default.* In that mode the position of the energy eigenvalue of the molecular state in a diagonalized Hamiltonian matrix serves as the label of the state. Though historically it was the first caption mode implemented in a program, it did not prove to be effective when large Hamiltonian matrixes are involved . In some cases for the sake of accuracy the algorithm used for matrix diagonalization forces the order of rows in Hamiltonian matrix to be changed. As the least successful out of available four caption modes that labeling method will probably be deprecated in future versions of SpecView.

2.4. Numerical algorithms used in a program.

Most of numerical algorithms are taken from the ì Numerical recipes in Cî book by S. Teukolsky, W. Press, W. Vetterling, and B. Flannery. The program code in the book was edited to comply with the contemporary version of C++, effective memory management and is interfaced with a GUI input-output of the program.

To diagonalize the Hamiltonian matrix first it is transformed into tridiagonal form by the Householder method (~ $4n^3/3$ operations) and then diagonalized with the QL algorithm with implicit shifts (~$3n^3$ operations since eigenvectors are required). The whole process is much more effective than straightforward Jacobi transformation algorithm with a total workload of $12n^3$ to $20n^3$ operations.

The NLSQ fitting procedure used in SpecView is a well-known Levenberg-Marquardt method. This method has become a standard solution in many statistical problems requiring nonlinear least square fit.

The main insight of the method is that the merit function (the uncertainty between a simulated spectrum and experimental profiles) in the close vicinity of the minimum can be approximated with the quadratic equation. If the approximation is perfect, the minimum can be calculated in one step (inverse-Hessian method). In reality the complex surface of the merit function in a multidimensional space of molecular constants can rarely be approximated with quadratic function. In that case the steepest descent method is used making a step along the gradient of the merit function toward the minimum. The length of the step should be reasonable enough not exceed the downhill direction. The Levenberg-Marquardt routine smoothly transforms between steepest descent method when far from the minimum and inverse-Hessian when minimum is approached. The size of the step in each iteration is decreased by a coefficient λ if the iteration is successful, i.e.

$X^2(\mathbf{a}) > X^2(\mathbf{a} + \partial\mathbf{a})$

and is increased otherwise.

The scale coefficient λ could be varied by changing *ì Step parameterî* from the programís ì Set constant incrementî dialog box that can be open by using Menuís *Fit* ➔ *Numerical* derivatives option.

The Hessian matrix used by the routine (the matrix of second derivatives of merit function $X^2(\mathbf{a})$) is calculated using only first derivatives of the frequency of the transitions $\partial E_{state}/\partial a_i$. Remaining in the analytical expression for Hessian matrix, terms with second derivatives could be omitted due to statistical reasons. They average out when summed over large number of assigned lines if the molecular model is good enough and the uncertainty distribution of experimental line profiles is close to natural, which is the case for most experiments. As in ì calculate fastî mode, the first derivatives should be provided by the DHamilt() function implemented in a user-defined module. Though quite straightforward when the dependence of the Hamiltonian on the molecular constants is linear, the process of writing the body for the DHamilt() function can be difficult for arbitrary Hamiltonians. In SpecView the fitting procedure still could be run even if the DHamilt() function is not provided by the module. For that case the program uses numerical derivatives calculated as

$\partial E_{state}/\partial a_i = (E_{state}(a + \delta a_i) - E_{state}(a))/ \delta a_i$

The increment parameter δa_i can be varied at any moment from the ì Set constants incrementî dialog box described above. To use numerical derivatives in fitting procedure one has to check *Fit* ➔ ì *Numerical derivatives*î option of the main applicationís Menu. Our numerous experiments with the program showed that even if DHamilt() function is implemented in a module in some cases using numerical derivatives in fitting routine gives more robust results than when analytical derivatives are used. Though useful in fitting procedure numerical derivatives are not optimal for CalculateFast() method of calculating spectrum when dynamic constant is incremented. Clearly if they were used, the method would converge to the main calculational function of SpecView CalculateSpectr().

The analytical derivatives $\partial E_{state}/\partial a_i$ are calculated based on the following useful in applications theorem:

$\partial E_{state\ n}/\partial a_i = (\partial H /\partial a_i)_{nn}$

This expression allows for the quick calculation of the derivatives of the energy of the states.

2.5. Importing and exporting capabilities.

2.5.1. Importing files.

At this time importing capabilities of SpecView are limited to the ability to load a group of experimental data files in the program. It is also possible to load a simulated spectrum or a stick plot previously exported to a data file. To load the file successfully the experimental data file should comply with the following requirements:

1. Data should be represented as a list of pairs frequency ñ intensity, separated with spaces or tabs.
2. Two numbers in each frequency ñ intensity pair should be delimited with any number of white spaces or tabs.

If there are any characters except for digits, ì ,î , ì .î and ì -ì preceding frequency number of any pair they will be discarded when reading the file.

2.5.2. *Exporting files.*

Exporting capabilities conveniently allow extracting various spectroscopic information from the memory of the computer into data or text file for the future analysis. The program supports:

1. Export of the convoluted simulated spectrum in data file (*.dat). The format of the created data file (a list of frequency ñ intensity pairs) allows for the file to be open with any of the spread-sheet applications (ì Excelî , ì Originî or ì Sigma Plotî).
2. Export of the stick plot in data file. Every molecular transition is represented by a triplet of data pairs:

frequency_N 0

frequency_N intensity_N

frequency_N 0

Such data format allows for a stick plot to be plotted with SpecView or spread-sheet programs using ì XY pairsî graphing mode.

3. Export the energy diagram of molecular states selected by an energy constrain or selected by quantum numbers. File format is the same as for a stick plot data file that allows drawing calculated energy diagrams in plotting programs.

4. Export general information about the model used for the simulations such as name of the model, comments, the information about current molecular constants, parameters, and assignment. The assignment part of the file lists quantum numbers labels for each of the assigned transitions, there calculated frequencies for the current set of molecular constants and parameters, the frequencies of the corresponding experimental spectral line profile and the differences between calculated and assigned frequencies ñ residuals. All this easy to read information lets a user to reproduce the simulation and assignment completely if entered manually in a program.

5. Export the result of the fit. If the fit was run at least once for the current model the program automatically appends the information about the result of the routine to the end of general information file described above. In an exported file you will see results of the very last fit, no matter if the adjusted constants were accepted by a user or if old once were remained (ìCancelî button was clicked in a ìFit resultî dialog box). Fit results include:

   - ✓ The $X^2(\mathbf{a})$ of the fit
   - ✓ The standard deviation of the fit sigma.
   - ✓ The covariance matrix of the fit
   - ✓ The correlation matrix of the fit

- ✓ The set of fitted values for adjusted molecular constants
- ✓ Estimated error bars of molecular constants

6. Export all information about molecular transitions or even molecular states selected either by quantum numbers or energy in a text file. In that file the user can find the complete wavefunction of the state under interest, including the information on the basis functions of the model, the energy of the transition or molecular state, and the set of quantum numbers assigned to the transition by the program. Of course the file also contains general model information.
7. And export all the simulated and experimental spectra from the screen of the computer to the printer. Combined with axis, axis labels and even with the title of the graph provided by SpecView this feature becomes a powerful tool replacing in many cases the need in other plotting applications.

2.6. Available models.

The models created in our group provide great versatility to emulate and refine the spectra of a large class of molecules. The standard SpecView Package installs and registers the following 8 models on you computer:

ïLinear.dll - Linear doublet, $^2S^+$ $^2S^-$ ^2P ^2D states;

ïLinearHF.dll - Linear doublet with hyperfine interaction

(I = 1/2), $^2S^+$ $^2S^-$ ^2PïAsyRot.dll - Asymmetric rotor, Watsonís Hamiltonian

ïA2E2.dll - Symmetric rotor doublet A and doublet E states

with hyperfine structure

ïA2E2JT.dll - Symmetric rotor doublet A and E states with

Jan Teller terms that mix the same J but different K

quantum numbers.

ïAsyRotsr.dll - Asymmetric rotor with spin - rotation, case A,

Cs symmetry of the molecule

ïMasyrot.dll - Asymmetric rotor on top of vibrational structure,

supports up to 6 vibrational levels.

ïAsyrotsrBnr1.dll - asymmetric rotor with spin-rotation, case B,

Cs symmetry

ïAsyrot_par.dll - asymmetric rotor with spin-rotation, case B,

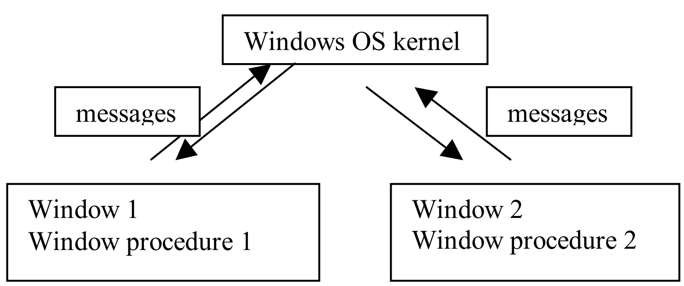
C1 symmetry (6 components of spin-rotation tensor)

3. Introduction to graphical User Interface (GUI).

This section will describe the GUI part of the program. That part of the program makes the trial simulations, experimental spectra analysis, assignment and fit ñ all spectroscopic manipulations ñ a very convenient and intuitive process that still hides beyond the complex and accurate mathematical calculations.

3.1. Windows.

First we put a few words about Windows OS. The basic unit of the Windows operating system is the window. It can be shown on a screen of the computer, hidden or put beyond the limits of the Desktop. Usually window has a process function associated with it that helps to communicate with the operating system. Window sends so-called messages to the operating system and responds to the messages from OS through windowís procedure.



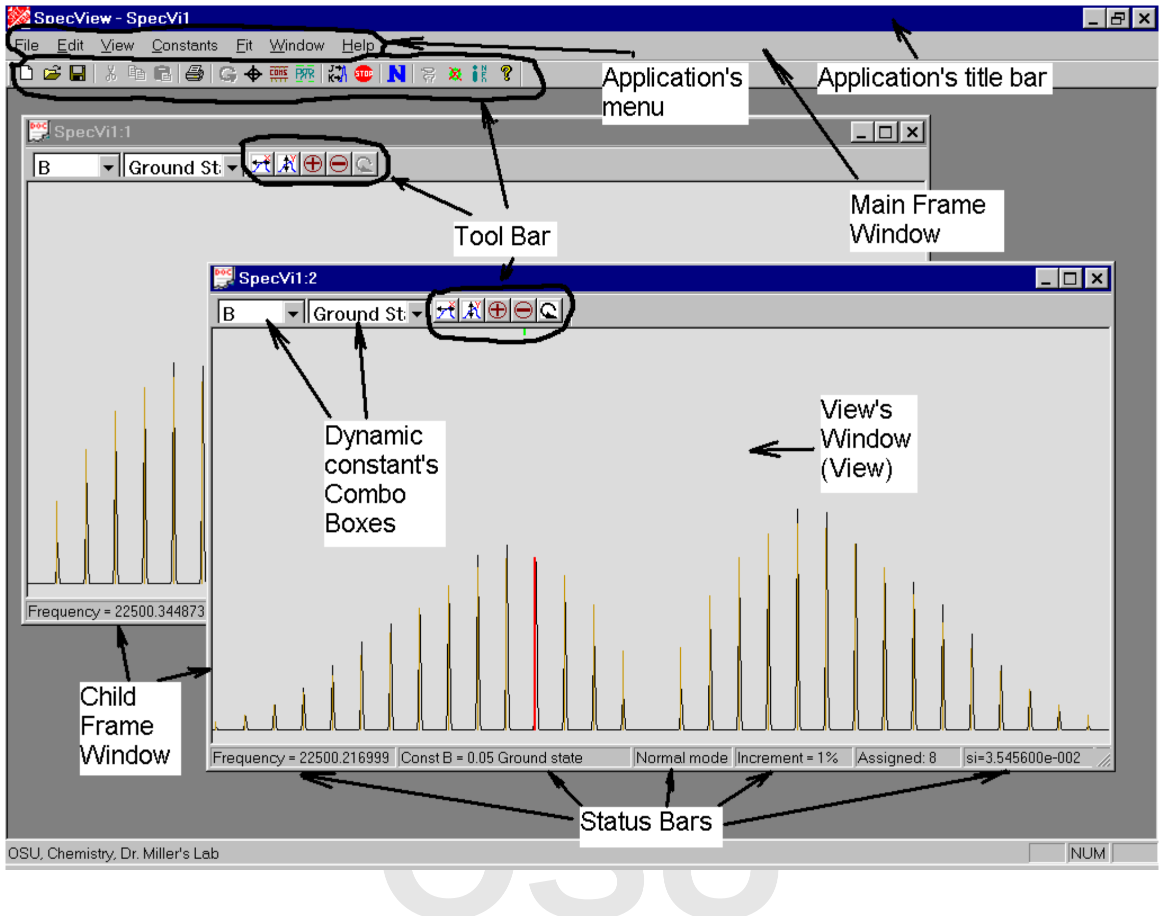
A usual Microsoft GUI application encloses hundreds of windows. For example, every toolbar button or control in WinWord application is a independent unit that communicates with the rest of the application with messages. We will list the most functional windows in SpecView application and describe their spectroscopical significance.

To start the discussion we will stipulate several definition that though difficult to understand in the beginning, are common for all contemporary Microsoftís applications, and in a long-run will help to easily comprehend the GUI features of SpecView.

Definitions:

Document and View:

Document ñ to put it in other words ñ this is the set of constants that completely describes the simulation ñ condition parameters (like temperature), molecular constants, calculated molecular states and transitions between them, assignment and result of the fit etc. This is your simulation. If you want to show that on a screen and manipulate this core information in any visually conceivable way(and that is the ultimate goal of a program) you start talking about View. View is the way you put the results on a screen of the Computer (graphs and tags in our case). View takes care of scaling, colors, fonts and everything that allows you to reflect the result of the simulation (Document) on a Display.



Pic.1 Main Frame Window, Child Frame Window, Toolbars, Status Bars and Views.

Main Frame Window ñ The main Window of the application. Contains ì Title Barî (blue horizontal strip at the top with the Icon and the name of the Application and current Document ñ ì SpecView ñ SpecVi1î, right mouse click on a title bar invokes System menu, common for most MS Windows windows), Application Menu (main entries: File, Edit, View, Constants, Fit, Windows, About) and Tool Bar with 18 buttons.

Child Frame Windows ñ floating Windows inside the Main Frame Window. You can have as many of them per open Application but all of them are enclosed in a Main Frame Window. Each of the Child Frame Windows represents either the different Views of the same Document (or spectrum) or different Documents.

Each Child Frame Window of the application has 2 Combo Boxes in a Dockable Tool Bar area with 5 buttons.

3.1.1. Status Bars.

At the bottom of each child frame you can find 7 Status Bar windows. Here is the description of each of the Status Bar Window (from left to right):

Status Bar Windows (small gray windows at the bottom of Child Frame Window):

Status Bar #1: Frequency at the current position of the cursor;

Status Bar #2: Dynamic constantís name and its current magnitude;

Status Bar #3: Current mode (Fast Mode or Normal Mode);

Status Bar #4: Current increment value of the dynamic constant;

Status Bar #5: Number of currently assigned lines (active only if the assignment exists);

Status Bar #6: The uncertainty σ (Sigma) of the current assignment (active if at least one line was assigned);

Status Bar #7: The name of the current model;

3.1.2 Toolbars.

Next two sections list all the buttons of the application as well as their functionality. So called ì tool tipsî are provided for all of them ñ a small yellowish window with a short text note that appears near the button if mouse arrow hovers over the button for a few moments.

Main Frame Toolbars

Main Frame Toolbar includes 17 buttons. The vast majority of the buttons is duplicated in Applicationís Menu or Context Menu. Although the functionality of some of them (for example ì Clean the assignmentî) is unique.

Button #1: New. Opens a new default Document ñ the spectrum of Asymmetric rotor with default parameters.

Button #2: Open. Opens a standard ì Open fileî dialog that allows to open a previously saved *.svw file.

Button #3: Save. Saves current simulation in the SpecViewís *.svw file.

Button #4, 5, 6: Standard for Microsoft cut, copy, paste buttons. Not used by the application (always grayed out).

Button #7: Print. Sends currently active View to the printer.

Button #8: Undo constants. Sets molecular constants to the values before the last change occurred.

Button #9: Change cursor. Changes the standard systems mouse cursor to the cross hair convenient for the measurement of the frequencies of experimental or simulated line profiles.

Button #10: Change constants. Invokes ìChange constantsî dialog, a convenient tool to modify and analyze molecular constants and model parameters. One can also choose a set of constants to be adjusted during fitting routine from that dialog box.

Button #11: Change parameters. Invokes ìChange parametersî dialog box. You can change any of the parameters of the model from that dialog box.

Button #12: Edit assignment. Invokes ìPick assigned linesî dialog box. In that dialog box you can choose those transitions that are used for the fitting procedure.

Button #13: Remove assignment. Delete assignment completely. This operation will be prompted with ì Yes ñ Noî dialog.

Button #14: Start fit. Starts the fitting routine. After the procedure is complete, the ì Fit resultsî dialog will appear on a screen. You will have the option to accept the fitted molecular constants as a new set of parameters or, if not satisfied with the result of the fit, discard changes with ì Cancelî button.

Button #15: Recalculate spectrum. Duplicated with ì Enterî keypad button. Recalculates spectrum precisely no matter what is the current mode of the simulation (ì Normalî or ì Fastî). Used in ì Fastî mode.

Button #16: Clean the assignment. Deletes all assigned lines from the assignment if their intensity becomes 0 (the effect of ì lostî assignment).

Button #17: Get Info about levels and transitions. Invokes ì Get Infoî dialog. This unique tool allows to simulate, analyze, plot and export information about molecular states and allowed transitions.

Button #18: About. Invokes ì Aboutî dialog that displays information about current version of the program and the authors.

Child Frame Toolbar

The five buttons of Child Frame Toolbar provide basic manipulation with the View, the user clicks on those buttons if he wants to stretch, zoom or scroll the spectrum on a screen or undo changes in View. Every View window can display the spectrum in its own way, that explains why this group of buttons was separated from the Main Frame Toolbar.

Button #1: Show all. Stretches the spectrum in a window to fit the window.

Button #2: Scale vertically to fit a screen. Gives the best fit of the part of the spectrum on a screen in y-direction.

Button #3: Zoom in. Self-explanatory. The scale is increased by 10%.

Button #4: Zoom out. Self-explanatory. The scale is decreased by 10%.

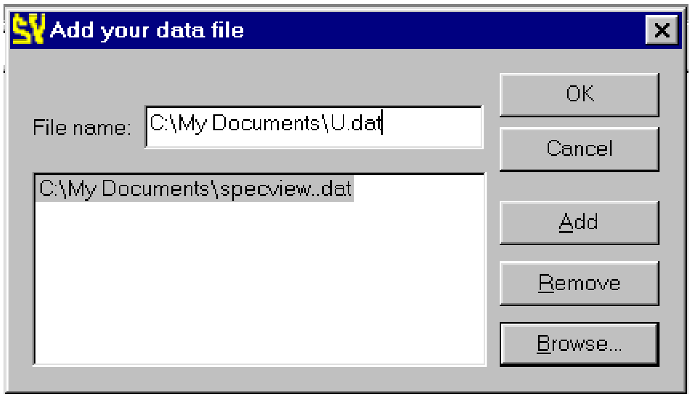
Button #5: Undo View. Discards the very last change in View (zoom or scrolling).

3.1.3. Main Applicationís Menu.

The main applicationís menu has 7 main entrances: File, Edit, View, Constants, Window, Help.

The File main entry subsides 11 subentries, most of which are common for all windowing applications and are therefore self-explanatory. So we will describe those that are specific for SpecView program.

Import Data files ñ invokes ì Add your data fileî dialog box.



Pic 2. ì Add your data fileî dialog box.

It is possible to display any number of experimental data files simultaneously. All currently included in a Document data files are listed in a List Box (lower on a picture). To remove a file from the list select the file with a mouse (that activates ì Removeî button of the dialog box) and click the ì Removeî button. To add a file either click ì BrowseÖ î button that starts standard ì Open fileî MS dialog box, or type the name of the file along with its complete path in the File name Edit Control (in the top of the dialog). While you type the program checks if the file with a name from the string in the Edit Control exist and when it finds a file the ì Addî button is activated. To add a file click the ì Addî button and the file appears in a List.

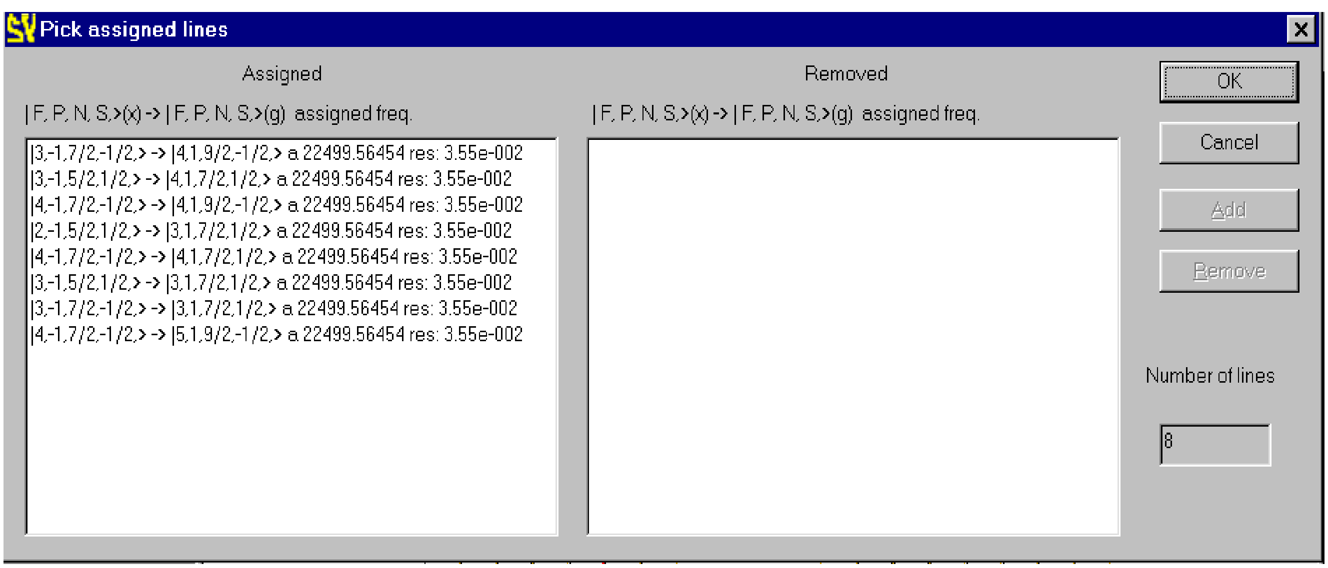
Export simulation has two subentries ñ stick plot and spectrum. Obviously those options allow to export the current stick plot or convoluted spectrum correspondingly in a data file.

Export assignment exports the text and data information about the model and its parameters and constants as well as the list of all assigned lines and the results of the last fit in a text file.

The last two menu options invoke MS standard ì Save asî dialog to choose the name of the target file.

The *Edit* Menuís option includes 5 entries:

Edit assignment ñ invokes ì Pick assigned lines dialogî



Pic 3. ì Pick assigned linesî dialog box.

At any time the program remembers two lists of assigned lines. One of them is called ì Activeî ñ that part of the assignment is used by the program while running fitting routine. The other one is ì Inactiveî . This is the list of assigned transitions that you do not trust but still be nice to store in a memory of the computer in case they become more ì favorableî after molecular constants are changed.

To move the transition between the lists you can just double click on it or select the transition with a mouse click and click ì Removeî button to move it from Active list to Inactive or click ì Addî button to move it from Inactive List to Active.

Every line in a List Box of Active or Inactive transitions shows the following information of the assigned line:

- Character ë%í or (and) ë@í, that signify that the wavefunction of the excited or (and) ground state correspondingly is a mixture of highly blended basis set functions, two maximum coefficients in a wavefunction are different by less than 10% and the default labeling procedure is not giving meaningful results. For such cases the model should provide custom Assign() function
- the set of quantum numbers that labels the ground state
- the set of quantum numbers that labels the excited state

- a small ëaí or ërí that allows to remember to which list the transition belonged before the dialog started
- the frequency of the experimental profile corresponding to the assigned line
- the residual, which is calculated as f_{theor} ñ f_{exper}, where f_{theor} is the calculated frequency of the transition and f_{exper} is the measured frequency of the profile.

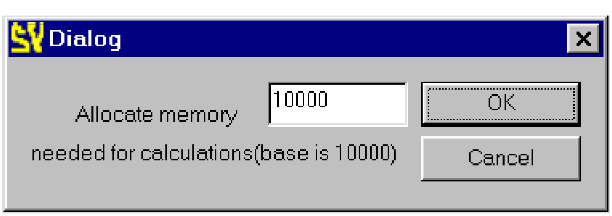
Clear assignment ñ this function deletes the lines with zero intensities from the assignment list.

The process of making an assignment suggests that you create a list of pairs experimental profile ñ simulated spectral line. Since the only transitions kept in a memory of the computer are those with non-zero intensities, when the assignment is created all of the simulated lines in assignment list have non-zero intensities.

The situation changes when any parameter or molecular constant of the model is modified. The intensity of the transition can become lower than intensity threshold of the model or, even worse, the assignment can be ì lostî. When the parameters of the ground and excited states of the transition are recalculated for a new set of molecular constants often the state can easily be mistaken for other state with the same set of ì goodî quantum numbers (the chances to track successfully ì correctî excited and ground state of the transition depend on a current labeling mode ñ default, by wavefunction, or by energy order, described in the theoretical section). Of course the selection rules can prohibit such ì mistakenî transition and the recalculated intensity of the transition is zero. The phenomena of the ì lostî assignment usually causes many troubles in the fitting procedure of the application and leads to the inaccurately adjusted parameter.

We advise to delete zero intensity transitions from the list of assigned lines after large change of constants occurred, and particularly after fitting procedure was run.

Memory ñ this Menuís option opens dialog that controls maximum number of transitions in a simulation.



Pic 4. ì Allocated memoryî dialog box.

The default value is 10000 transitions. The maximum possible is 1000.000. For moderate temperature the default settings satisfy needs of the application for the allocated memory. For room temperature the parameter of the dialog box should be set to the maximum value. If the number of non-zero transitions in the simulations exceeds that parameter, the program crashes, General Windows Protection Error is generated and the program shuts down.

Acute cursor - this function switches between MS Windows standard mouse pointer and cross hair pointer convenient for measuring of the spectral line profilesí frequency.

Custom colors ñ calls standard Windowsí Color Dialog that allows to customize the appearance of the View (dialogís and background colors, fonts).

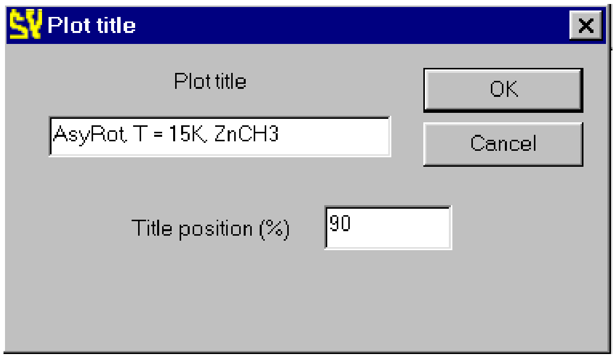
View main entry of the applicationís Menu has 14 subentries.

The first two ñ *Tool Bar* and *Status Bar* are self-explanatory, so we will describe the functionality of the application specific features.

Show Stick Plot ñ displays stick plot of the simulation.

Remove Stick Plot ñ hides the stick plot. When the simulation is done the first time, stick plot is on and *Show Stick Plot* menu option is grayed out.

Plot tag ñ opens ì Plot titleî Dialog



Pic 5. ì Plot titleî dialog box.

This information is displayed on a screen above the simulated spectrum. The tag of the graph is shown in a Plot title window, ì Title positionî edit control handles the position of the title on a screen. Plot title should be used as a supplementary information to every file prepared to export text information about spectral transitions, molecular states or assignment, since it is recorded in the very beginning of an exported file, and helps to distinguish and organize the files.

Axes labels On/Off - shows and hides frequency labels on the horizontal axes and maximum intensity label on the vertical axes.

Allow Zoom ñ the program is in that mode by default, zoom is allowed. Used to switch from *Show All* regime (next).

Show All ñ shows all the spectrum in one piece in the Viewís window no matter what are the current molecular constants or module parameters. In that mode zooming and scrolling capabilities are disabled.

Overlap ñ available if any experimental data file is uploaded to a program. This function allows to match the origin of the band of the simulation to the origin of the experimental spectrum on a large scale. Fine tuning should be done manually (Shift + right mouse button).

Auto Y-scaling ñ for exploring of the weak transitions it is nice to stretch barely visible transitions to some significant fraction of the window. In that mode the most intensive transition shown on a screen is stretched to fill the portion of the screen indicated in *Theor. Plot Offset*.

Theor. Plot Offset ñ invokes dialog with 2 Edit Controls. The ì Y-scalingî control allows to change the portion of the screen occupied by the simulation, the second ì Y-positionî control changes the position of the simulated spectrum on a screen.

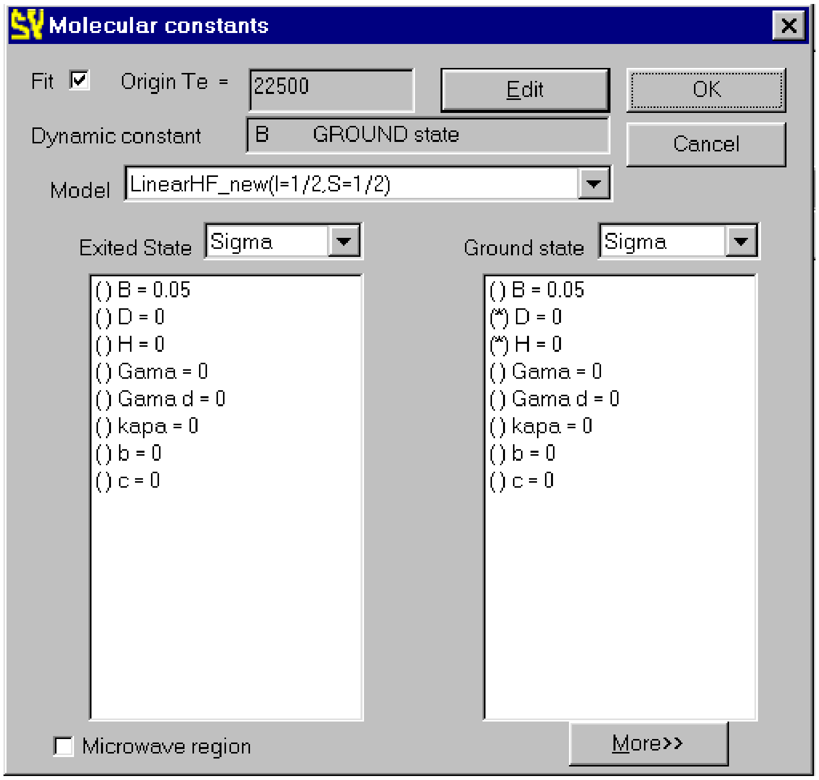
Exper. Plot Offset ñ same as previous, controls the y-position and vertical size of the experimental spectrum on a screen.

Color ñ calls ì Colorî dialog. One can choose the color of the convoluted simulated spectrum. Available colors are: black (default), green, red and blue.

Choose View Window ñ sets the frequency size of the View window. You can look at the particular part of the spectrum by setting maximum and minimum frequencies of the View window.

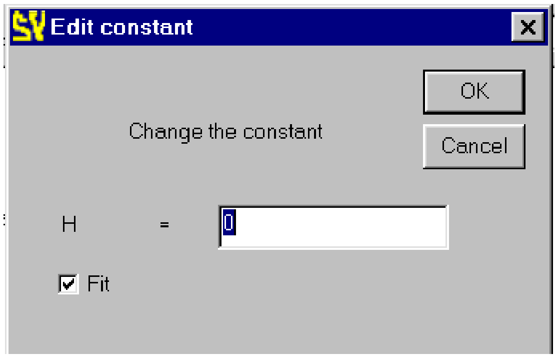
Constants ñ that main entry on the applicationís Menu has 5 subentries:
Mol. Constants ñ opens ì Molecular constantsî dialog. This is the main GUI tool to manipulate molecular constants, module parameters and to select constants to be adjusted while fitting procedure is running. You can choose the module used for the simulation from the ì Modelî combo box, that lists

all models registered on you computer through specview.ini file. The dialog box also have two combo boxes (ì Excited Stateî and ì Ground Stateî) that



Pic 6. ì Molecular constantsî dialog box.

lists all available for the simulation electronic states of the molecule. Two List Boxes show molecular constants used for the simulation. Selecting the constant with a mouse click and pressing ì Edit buttonî or double click on a constant opens ì Edit constantî dialog.

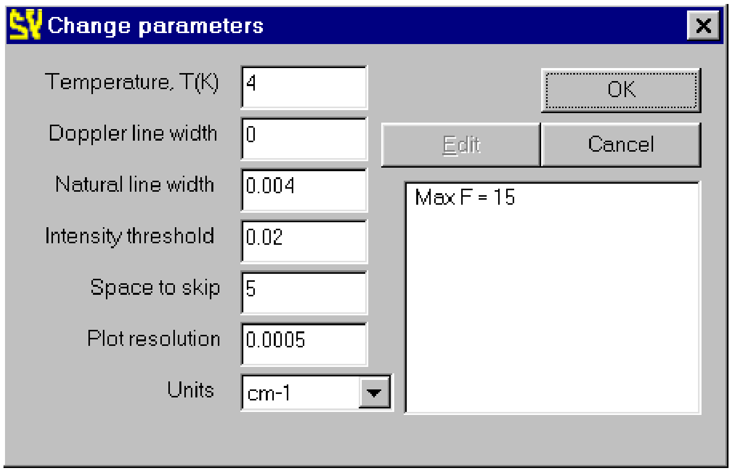


Pic 7. ì Edit constantî dialog box.

The dialog shows the name of the constant to be changed and has an edit control for a new magnitude of the constant. The ì Fitî check box is used to notify the program that the constant is going to be adjusted in NLSQ routine. Such ì to be adjustedî constants are indicated by the asterisk in parenthesis (*) in ì Molecular constantsî dialog box (constants H and D of the excited state on a picture).

ì Molecular constantsî dialog box can be extended to show the list of module parameters. For that press ì Moreî button, to hide press ì Lessî . The ì Microwave regionî check box in the left bottom corner of the dialog window disables Excited State list box and sets the application in the microwave mode. In that case all transitions occur between energy levels of one electronic state (Ground State). If any assignments exist before switching the modes, they are permanently deleted.

Parameters ñ invokes ìChange parametersî dialog. Contains 7 edit controls with standard parameters (temperature, line width, intensity threshold, etc.)



Pic 8. ìChange parametersî dialog box.

All modelís custom parameters are listed in a List Box. To change values of the custom parameters one should double click on a item in a List Box that calls ìEdit Parameterî dialog box. The procedure resembles changing the constant from the ìChange Molecular Constantî dialog box.

Increment ñ this pop-up menuís entry allows to change the increment value for the current dynamic constant. The available values are by percentage - 0.1%, 0.5%, 1%, 5% and to increment by a certain magnitude ñ by 0.01 and by 0.1. The last two a usually used to change dynamically initially zeroed constant.

Quick simulation ñ switches the program to Fast simulation mode, when changing dynamic molecular constant the spectrum is recalculated using approximate expressions and linesí intensities are not recalculated. The user wins in terms of time ñ performance of the program but loose in accuracy of the simulation. To get good results in that mode the ìRecalculateî button from the applicationís Main Frame Tool Bar should be clicked once per few increments of the dynamic constants. That invokes precise recalculation of all transitions, standard in Normal Simulation mode.

Normal Simulation ñ switches the application run in Fast mode back to Normal simulation.

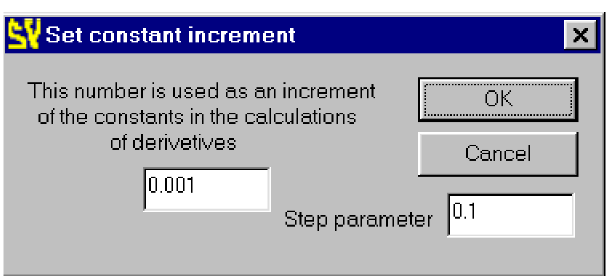
Here we want to stress that those two modes affect the simulation only if the simulation is calculated due to the change of dynamic constant. If the molecular constants or parameters of the model were changed from the ìChange Molecular constantsî or ìChange Parametersî dialog boxes, the programs recalculates all transitions precisely, independently of the current simulation mode.

The fifth main entry of the applicationís menu - *Fit* has 3 subentries:

Normal Derivatives, *Numerical Derivatives* and pop-up entry *Assignment Capture.* All these menu options affect the performance of the fitting engine of the program.

Normal Derivatives is the default setting, the fitting procedure calculates derivatives used to get the elements of the Hessian matrix using analytical derivatives provided by DHamilt() function provided by the model. If the DHamilt() function is not implemented in the model, to use fitting routine successfully numerical derivatives must be used.

Numerical derivatives ñ signals the fitting routine that the derivatives have to be calculated numerically. This menuís option invokes ì Set constant incrementî dialog, used to change step parameter λ, a parameter of the fitting procedure discussed in theoretical section, and the increment of the



Pic 9. ì Set constant incrementî dialog box.

argument while calculating the numerical derivatives.

This number should be set as a moderately small number in comparison to the average magnitude of the constants. Probably good results will be obtained if $C_{ave}/\Delta C \cong 10^4$, where C_{ave} is the magnitude of the average adjusted during fitting molecular constant and is the increment of the constant.

Assignment capture menu option offers four choices for the tracking of the list of the assigned lines. The most universal so far is the default *Wavefunction, square*. The comparably effective one is *Wavefunction, linear*. The *Energy order* proves to give reliable results while fitting spectra with the model of Asymmetric Rotor with negligible distortion coefficients. The default one leads to frequent ì lostî of the assignment and can be used only for models with small Hamiltonian matrices. To get the complete description of the caption modes refer to the Theoretical part of the manual.

And finally the last two main entries of the main applicationís Menu ñ the *Window* and *Help*. These two entries are common for all MFC designed windowing applications. *Window* allows to manage conveniently the position of the open Views on a screen of the computer, and *Help* opens ì Aboutî dialog with the basic information about SpecView application.

View:

By view we will also call the area in a Child Frame Window where the results of the simulations are shown.

3.2. Customizing the View.

3.2.1. Zoom.

After the spectrum is simulated for the first time the View is scaled to show the user the whole spectrum. To zoom in the spectrum the user have to hold the Shift button down and left-click the mouse button on a position on a screen that will appear in the center of View after the picture is redrawn. To zoom out click left mouse button while Ctrl button is down. The magnifying coefficient is 2. To compare several spectra it is sometimes useful to fine-zoom simulations. For that you have to use one of the Child Frame Windowís Tool Bar Buttons ñ Zoom In or Zoom Out (+ or -). In that case the magnifying coefficient is 10%).

You can also select the particular part of the spectrum to be shown in a screen. For that go to menuís View ➔ Select View Window option and provide maximum and minimum frequencies for the part of a spectrum.

3.2.2. Scrolling.

If you want to scroll the simulation you have to press the left button of the mouse and drag to the left or to the right. The spectrum will be shifted after you release the mouse button.

3.2.3. Show All Lines.

Sometimes it is nice to see the entire spectrum no matter what are the current constantsí values. For that go to View ➔ Show All. Off course that locks Zoom functionality up, so if you want to scroll or zoom spectrum again, go to View ➔ Allow Zoom.

3.2.4. Graphs Offset and Vertical Size.

You can change the vertical size of the simulated and experimental spectrum and their position in a View Window. To do that go to Main menuís View ➔ Theor. Plot Offset or View ➔ Exper. Plot Offset correspondently. The numbers there are in fractions of a Viewís entire size.

3.2.5. Vertical (Y) Autoscaling.

If you are looking at the weak wings of the simulated spectrum it is useful to scale up the height of the low ñ intensity lines. In a Y-autoscaling mode the line on a Screen with a maximum intensity will be scaled to the default magnitude in a View ➔ Theor. Plot Offset. To go to this mode check View ➔ Auto Y scaling, to get back to absolute intensities mode uncheck View ➔ Auto Y scaling.

3.2.6. Axes labels and Simulation Name.

When you save the document for the future references or send the simulation to the Printer sometimes itís extremely useful to put a nametag on a simulation that would appear it the top of the View or draw frequency labels along X-axes. To do that go to menuís View ➔ Axes Labels On/Off to add or remove axes labels and View ➔ Plot Tag to edit the title of the simulation as well as the position of the tag in a Window.

3.2.7. Graph Color.

The default color of the simulation is Black. But you can change it either from ìChange the colorî Dialog (menu ➔ View ➔ Color) or from

Context Menu (click right mouse button inside the View Windowís rectangle). Available colors are Black, Red, Green, and Blue.

3.2.8. Background color, Text color, Dialogís color, View.

You can customize the appearance of the Dialog Boxes, Text color and the Viewís Background color. All those settings implemented as persistent properties and written directly to the central repository of the Windows OS ñ registry. That means ones you make any changes they would appear each time you start the application.

You can change settings by going to the Menuís Edit ➔ Custom Colors ➔ î Your settingî . The color of acute cursor changes automatically to suite the purposes the best.

3.2.9. Acute Cursor.

You can always change standard Windowís cursor to custom Cross Hair cursor that will help you to find the current frequency in the most accurate way. For that check Menuís Edit ➔ Acute Cursor. To get back Systemís cursor uncheck the Menuís option.

Show Stick Plot

In that mode the program shows all simulated lines as they are. The color of simulated lines is orange.

By default the program always shows the convoluted spectrum ñ result of overlapping of broadened lines. To remove that go to parameters and zero Natural and Doppler line width. Itís useful trick for relatively narrow line widths when the calculation of correct convoluted spectra can take more than a few seconds.

The default option is Stick Plot On, so to remove stick plot go to View ➔ Remove Stick Plot and to get it back click on View ➔ Show Stick Plot.

Viewís ToolBar Buttons (Child Frame Window)

(the order is from left to right)

Button #1 Show all nonzero-intensity transitions in a window.

Button #2 Scale the spectrum in vertical direction. The size of the strongest line will occupy the portion of the screen specified in Menuís View->Choose View Window

Button #3 + Zoom In, 10%

Button #4 - Zoom Out, 10%

Button #5 Undo last view change (Zooming or scrolling).

3.3. Controlling simulations.

When loading the model for the first time the parameters and constants used for the simulation are set to default values in the model. You can change them at any moment from Menuís Constants ➔ Parameters and Constants ➔ Mol. Constants correspondently. You can also access Parametersí and Constantsí Dialog Boxes from Main Frameís Toolbar (buttons number 10 and 11) or Context Menu that appears in a Viewís Window after right mouse click (click either Molecular Constants or Parameters). Itís also possible to access simulationís parameters from Constantsí Dialog Box by pressing ìMoreî push button which extends the dialog. This feature is extremely useful when the calculations take a lot of time and the user desires to change parameters and constants at once.

Please note that you can change molecular model from Constantsí Dialog Box.

3.3.1. Dynamic Change of Constants.

One of the unusual features of SpecView is Dynamic Change Of Constants. Often the result of abinitio calculations of molecular constants gives a simulation far from experimental spectrum. In that case before you do initial assignment and fitting you have to play around with constants to see matching lines. SpecView greatly facilitates this procedure. Each View Window of the current Document has a set of two Combo Boxes. One of them allows choosing the constant you want to change dynamically, the second explains if the constant belongs to the set of constants of ground or excited state of the molecule.

After you make you choice you can easily change the constant pressing arrow Up or Down Key. The constantís name and current value is conveniently shown in Status Bar #2. There are a few predefined ways to change the constant:

1. By percentage (0.1%, 0.5%, 1%, 5%);
2. By small predefined magnitude (0.01, 0.1).

The default one is 0.1%. Two change the increment value go to Constants ➔ Increment menu option.

The current increment magnitude is shown in Status Bar Window #4 at the bottom of your View.

For complex spectra with a large number of lines the calculations time can approach tens of seconds or maybe even minutes. For those cases program allows you to use different calculation modes.

There are three modes of simulations. The default one is called ìNormal modeî. In that regime each time the constant is changed the spectrum is calculated completely. The second one is ìFast modeî. In that mode when you press Up or Down arrow Key the program does not solve for eigenvalues in order to obtain true transition frequencies. It uses the results of previous calculations and Hamiltonian function derivatives over corresponding constant to get the corrected eigenvalues for ground and excited state of a given transition and, following, frequency of the transition.

Note that the intensities are not recalculated for each individual line in that mode; also they visually change on a screen due to overlapping of lines. When you use ì Fast modeî the Tool Bar button ì Refreshî in a Main Frame Window is Activated. After you simulated spectrum a few times in a ì Fast Modeî it is advised to press this button in order to recalculate the spectrum for the current set of constants using ì Normal Modeî . That would improve the quality of the future simulation in a ì Fast Modeî (cause that allows wavefunctions and intensities to be updated).

You can switch between the two modes using menuís Constants ➔ Quick Simulation or Constants ➔Normal Simulation options. You can observe current mode in the lowest part of a View Window ñStatus Bar #3 shows either ì Quick modeî or ì Normal modeî message. There is third mode ñ it is always activated and works as more accurate version of ì Fast Modeî . It uses second derivatives for the calculations of transitionsí frequencies and does not recalculate intensities. For changing constants dynamically in that mode use Page Up and Page Down keyboard Keys.

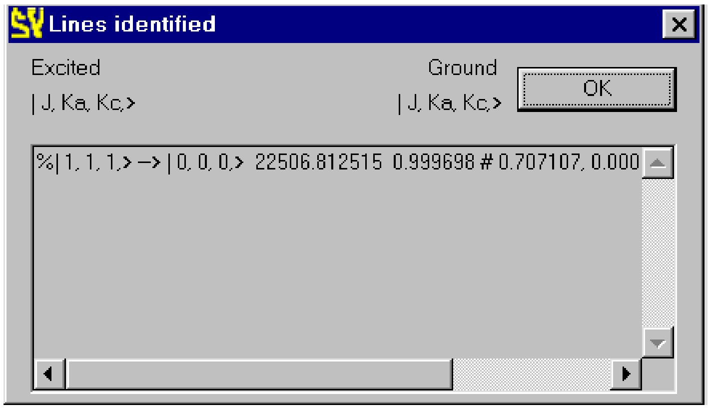
It is worth to mention that if your change one of the constants from ì Constants Dialog Boxî the spectrum is recalculated completely (as in ì Normalî modeî) no matter what the current mode is.

The functionality of Arrow Up and Down Keys is duplicated with a mouse wheel.

3.3.2. Making an assignment.

As we stated in the first part of the manual, to run a fitting routine a scientist has to do the assignment of the line profiles of the experimental spectrum. To do so a user has to measure the frequency of the line profile and label it in acceptable for the program way (for example with a set of quantum numbers). Measuring the frequency of the profile is greatly simplified with a program. The user has to change the cursor to the cross hair for the accurate determination of the cursor position and take the reading from the frequency Status Bar #1.

Labeling of the line profile is easy when the simulated spectrum closely matches experimental data. In that case the set of quantum numbers for experimental line profile could be defined as the set of quantum numbers corresponding to the nearest line of the simulated spectrum. To read the set of ì goodî and ì badî quantum numbers that define simulated spectral line the user has to click right mouse button in the vicinity of the transition under interest. A context menu around the current position of the mouse pointer offers one option ñ *Show assign* . Clicking on context menuís window opens ì Lines identifiedî dialog box with the following information about transitions:



Pic 11. ì Lines identifiedî dialog box.

1. Character ì %î or (and) ì @î that signify that the wavefunction of the excited or (and) ground state is highly blended (the maximum two coefficients of the wavefunction are different by less than 10%). If ì smartî model defined Assign() function is not provided by *.dll module, the default assigning mechanism of the program fails in that case and the following quantum numbers assignment of ì badî quantum numbers is not reliable (ì goodî quantum numbers are always well ñ defined and correctly provided by the program).
2. A set of quantum numbers that uniquely define the excited and ground state involved in the particular transition.

3. The calculated frequency of the transition.
4. The calculated intensity of the transition.
5. The array of the wavefunction coefficients in the decomposition of the eigenfunctions of the state into basis functions of the model (first coefficients of the excited state and then the coefficients of the ground state are listed).

Indeed the process of making the assignment is even easier in SpecView. To make an assignment of the experimental line, or to add the assignment of the experimental profile to already open assignment list the user just has to double click with the right mouse button on the simulated line that going to be assigned to the experimental profile. That triggers the assigning mode of the application and the usual brown color of the line that denote the line on a simulation turns black. The dialog ì Pick assigned linesî pops up (Pic. 3). The complete description of the dialog is done in section listing all Menuís entries of the application. The program keeps two lists of assigned lines ñ Active list and Inactive list. Active list is used for the fitting routine. Inactive list keeps assigned lines for the user to add them later to the Active list if the fit improves the quality of the simulation. For example in the spectrum of the Asymmetric Rotor the scientist has to move all transitions with high Js to Inactive list in the beginning and add them gradually add them to Active list when the constants are getting closer to the constants of the real molecule.

After the initial work with the list of assignments is done the user has to click ì OKî button of the dialog. The program now expects the user to define the experimental profile that corresponds to the simulated line. The cursor changes to cross hair that allows to define precisely the frequency of the maximum of the experimental profile. In that mode you can blow up the profile to get the best results (Shift + left mouse click on the lineís maximum). Clicking the right mouse button just once memorizes the frequency position of the line of experimental spectrum. That completes the assigning cycle of the line.

To correct the assignment of the particular spectral line the user can double click on the previously assigned line that as always select the line by changing the color of the simulated line to black. The big green line on a screen shows the frequency of the experimental line profile being assigned to the simulated one. Press ì OKî button and click on a new experimental spectrum profile. The program memorizes the new frequency of the experimental line.

To delete a particular transition from the Active list the user has to click Edit Assignment button of the Main applicationís Toolbar or go to Edit ➔ Edit assignment menuís option. The dialog that appears lists all active and inactive assigned transitions and allows to move them from the list of active transitions to the list of inactive transitions and vise versa.

4 Writing a new model.

4.1 Overview

As was stated above one of the biggest benefits of SpecView is the ability, I principle, to generate the spectrum of any molecular system. This, however, may require some programming from the userís side. In a SpecView package we supply a sample project for creating dynamic link library. A scientist even with minimal programming skills should be able to write itís own model in plain C, and then compile the file and build the *.dll file used by SpecView application.

More than 10 molecular models that describe wide range of molecular systems have been designed in our lab over the years. In many cases a scientist can design a new model just by slightly editing an already existent one. For example, a user can add new Hamiltonian terms, or do minor changes in the intensity function. For fast module development we provide source code for every model supplied in standard module library. The procedure of creating of a new model becomes simple:

1. Open the sample project supplied in a package with Microsoft Visual C++ 6.0 Environment (project workspace file asyrot.dsw).
2. Add to project the *model.cpp file that will serve as a core of your model.
3. Edit the *model.cpp file as desired.
4. Compile the project (Ctrl + F7).
5. Build the project (F7).

The resultant asyrot.dll file should be renamed and placed in a SpecViewModels directory containing standard *.dll models. After being copied to the final destination the model should be registered in a SpecView model database ñ specview.ini file located in Windows directory.

In order to write a successful module that would give all functionality to SpecView application the programmer has to implement 21 functions. Logically those functions can be segregated in 4 groups:

- ✓ 3 core functions that are used rigorously by SpecView for calculations, simulations and fit. These are 2 mandatory functions ñ the Hamilt() function that initializes matrix elements of the Hamiltonian matrix and the Inten() function that calculates transition strength for a given pair of molecular states, and one that is recommended but not required ñ DHamilt() - the function that calculates the derivative of the energy of the state with respect to a given molecular constant. That last function allows running simulations in a ì Fast modeî and running fitting procedure using analytical derivatives. If this function is not implemented you will still be able to run simulations in a normal mode and fitting procedure with numerical derivatives.
- ✓ The functions that are mostly used by GUI part of the SpecView program and provide the names of constants, parameters or quantum numbers.
- ✓ The Service functions, the functions that supply generic information about the model, helps the calculational engine of SpecView program to work correctly, allocating memory for the calculations and providing conditions for the loop when calculating the allowed transitions in the model.

- ✓ Miscellaneous group. Comprises one function ñ the function that makes intelligent assignment based on the current set of quantum numbers and wavefunction of the state.

The next section discusses the functionality of each of the 21. To make it clear we will follow the example of the model of Rigid Asymmetric Rotor.

The writing of the module starts with a decision about the appropriate set of basis functions for the Hamiltonian matrix. As was explained above the process of diagonalization is not done in one big Hamiltonian matrix but rather the matrix is diagonalized by running diagonalization routines on each block of the matrix. So a good model (and good in this context is a characteristic of the simulation time) should have a basis set that minimizes the size of blocks in block-diagonal Hamiltonian matrix. From another point of view the basis set should be natural for the model in the sense that the Hamiltonian matrix should be close to diagonal in order to give a meaningful assignment for the lines, i.e. basis functions are not mixed heavily by non-diagonal matrix elements. However if a smart Assign() function is implemented even highly mixed states can be labeled. Another argument in the search for a convenient basis set for a model is the simplicity of the intensity Inten() function for the given choice of basis set functions.

All models should be developed with a thought in mind of using them for simulation of spectra of microwave transitions. In that situation all transitions occur within one electronic state. Special attention should be paid to the calculation of the population factors of the lower and excited state (see comments in a sample Inten() function).

Our choice for the basis functions for our model of asymmetric rotor will be eigenfunctions of rigid symmetric rotor Hamiltonian:

|J, K>, where J is total angular momentum and K is its projection on the intermolecular z-axis.

Although not the most efficient basis set for the Hamiltonian diagonalization (the best would be symmetric and antisymmetric combinations like |J, K> ± |J, -K> that would decrease the size of all diagonalized matrixes by a factor of 2) this choice makes easy implementation of the intensity function. Since the eigenfunctions of states will be a linear combinations of |J, K> functions with equal weights of |J, K> and |J, -K> the default labeling algorithm would fail and the custom user-defined function Assign() is needed.

Here we would note that this basis set nearly satisfies our ìrule of thumbî for the correct choice of basis set functions:

1. The set nearly minimizes the size of blocks in a Hamiltonian matrix.
2. The set is only slightly blended by small off-diagonal matrix elements.
3. The intensity function expressed in terms of the basis set functions is simple.

4.2 Model functions

We will start from the first function as it is listed in an example file asyrot.cpp. Each function has a certain defined type for the return value and a number of parameters passed as arguments. Described function properties called function declarations are set in SpecView application and by that standardized for all models. Any change in function declaration would lead to a run-time error in SpecView when the user attempts to load a faulty *.dll module.

In what follows we list each function, then generally describe it, finishing with the code for that function in the file asyrot.cpp.

4.2.1 GUI functions

4.2.1.1 Name().

char* Name() - This function should return the name of the model. The only restriction on the name of the model is that to get the full functionality of SpecView the returned name should be unique among the group of modules registered in the SpecView.ini file. When registering a built *.dll file in a SpecView.ini file the user has to specify the name of the model and specify the path to the dynamic link library file location. In case there are two modules registered at the same name, the program will function normally if save ñ open activity is not used. When the current document is saved in a *.svw file, the file remembers the name of the model used for the simulation. The next time you open the file SpecView searches for a module registered in a SpecView.ini file with the name of the model recorded in a *.svw file. If the file is generated using the second module with the duplicated name, when opened SpecView tries to interpret the data with the very first module from SpecView.ini file registered with the same name. This inevitably leads to a program crash.

The char* Name() function in asyrot.cpp is as follows:

```
char* Name()
{
        return "Asymmetric Rotor ";  //return the unique model name
```

```
}
```

4.2.1.2 StaNum()

UINT StaNum() ñ this function returns the number of electronic states of the molecule. For example to describe the linear molecule in Σ, Π and Δ states this function should return 3, for the molecule with C_s symmetry the function would return 2, in accordance with two electronic states of different symmetry ñ Aí and Aî. In our case our simple model of rigid asymmetric rotor does not need any specific behavior on different electronic state, so the function returns integer 1. (Both ground and excited states of the molecule are described with the same Hamiltonian, though different molecular constants).


```
UINT StaNum()
{
        return 1;  //for this model we do not need different electronic states. The symmetry of the
        // electronic states  are taken care of by the appropriate choice of the transitional moments ñ a, b,
        //or c ñtype transitions.
}
```

4.2.1.3 StName()

char* StName(UINT nStateType) ñ this function returns a string that names electronic state, which number is passed as parameter nStateType.

Electronic states are numbered from 0 to StaNum()-1. This function is mainly used by the programís GDI engine. The way the model numbers electronic states is totally your choice. For linear molecule a logic choice would be 0 for Σ state, 1 for Π and 2 for Δ state. The functionís body consist with a series of ì ifî statements checking the value for the argument or one ì switchî statement. In our case of asymmetric rotor the function is trivial ñ since there is only one state possible we always return ì AsyRotî .

```
char* StName( UINT nStateType )
{
        return "AsyRot";  /* that will be the name of the electronic state. In case we have a few electronic
        states to choose from, the body of the function will have a switch(nStateType) statement that will
        analyze the type of the state variable and return the corresponding name string. (for example for
        Cs symmetry model that will look like:
        switch(nStateType)
        {
                case 0: return ì A1î ;
                case 1: return ì A2î ;
                default: return ì stateî ;
        }
        It is a well known fact that the process of debugging all of the *.dll modules can become a difficult
        task. So even in case you list all possible constant magnitudes of nStateType variable after ì caseî
        statement, we encourage you to use ì defaultî statement in the end, so that in case of the run-time
        error if the variable goes out of bounds the program will signal you. */
}
```

4.2.1.4 ConNum()

UINT ConNum(UINT nStateType) - this function takes the number assigned to the given electronic state and returns the number of Hamiltonian constants used to describe the state. Usually that should be a switch statement, though in our simple case the Hamiltonian of the model always has same number of constants A, B and C, so we return 3.

```
UINT ConNum( UINT nStateType )
{
  return 3;   /*  the Hamiltonian has 3 constants ñ A, B and C in excited and ground states(index={0,...,
N})*/
}
```


4.2.1.5 ConName()

char* ConName(UINT nStateType, UINT nConst) ñ function takes as arguments two unsigned integers ñ the number assigned to a given state and the number of constant in that electronic state, which varies from 0 to ConNum(nStateType) ñ1. It returns a name of the constant. The function used mainly by the SpecViewís GUI. Of course naming of the constants is your choice. We will return string ì Aî for the first constant, ì Bî for the second and ì Cî for the last.

```
char* ConName( UINT nStateType, UINT nConst )
{
	switch(nConst)
	{
		case 0 : return "A"; /* the array of molecular constants is zero based so the first constant
		A corresponds to the 0-th index nConst of the array*/
		case 1 : return "B";
		case 2 : return "C";
		default : return ì Constantî ;
	} /*because we have only one type of electronic state for our model, we do not need to analyze
nStateType here, which is always 0. Though the more carefully conducted  model would include the
switch(nStateType) statement with ì  return ì Errorî ; ì  string to simplify the debugging.*/
}
```


4.2.1.6 InitCo()

void InitCo(UINT nStateType, double* pdConst) ñ this function does not return any value. The arguments are the number assigned to a given electronic state and the array of variables of type of double. The size of the zero-based array is ConNum(nStateType). The naming convention defined by the previous function suggests that the element pdConst[0] correspond to A constant,

pdConst[1] corresponds to B constant of the Hamiltonian and pdConst[2] corresponds to C. The very first time the model is loaded in the memory of the computer, constant arrays are initialized by that function. It is advised to keep constants giving contribution to the non-diagonal elements equal to zero, so that the initial simulation is fast. When working on a particular molecule, it is convenient to initialize Hamiltonian constants with known constants of the molecule.

```
void InitCo( UINT nStateType, double* pdConst )
{
	int i;
	for( i = 0; i < 3; pdConst[i++] = 0.0 ); /*All 3 const.  set to 0 in C language variables are not
	automatically initialized to 0s, so to ensure that zero all the elements of the constants array*/
	pdConst[0] = 1.3;
	 pdConst[1] = 1.0;
	 pdConst[2] = 1.0;
	/* this default set of constants will generate the spectrum of the prolate symmetric top */
}
```


4.2.1.7 ParNum()

UINT ParNum() ñ this function returns number of parameters used in a model. Parameters are passed as an argument to the intensity function

Inten(). This number includes 7 default parameters and any number of user-defined parameters. The first 7 parameters are natural for any model and are embedded in a SpecView application. Those are:

- ✓ the temperature of the gas. Though playing role of the coefficient in Bolzmanís distribution in all standard models in the SpecView package, this parameters is used for calculations only in custom Inten() function supplied by the module. That allows using this parameter also as characteristic of non ñ equilibrium media such as plasma. To characterize complex population distribution the writer of the model can use in addition any number of custom parameters
- ✓ the Dopplerís line width used for the calculations of a line profile
- ✓ the natural line width
- ✓ intensity threshold. In order to run the program efficiently the spectrum generator discards all transitions for which the intensity is less than the given value. To see all the transition, set this parameter to zero.
- ✓ space to skip. This parameter is not used by the application. Kept for compatibility reasons.
- ✓ Theoretical plot resolution. In a current version of SpecView this parameter is used only to define the frequency step in a data file of the exported simulation. If this step is larger than the width of a line profile,

the warning message will be generated if the attempt to export the simulation is made. The program will also generate a warning message if the plot resolution is set to very small magnitude so that the potential size of the exported file becomes unacceptably large (more than 10MB)

- ✓ units. This parameter can span three discrete values ñ 0.0(cm^{-1}), 1.0(MHz) and 2.0(GHz). The parameter specifies the energy units for the current simulation.

Custom parameters are often used as a tool to select certain transitions or, on the contrary, extinguish a group of lines. The beauty of custom parameters is that all parameters as well as molecular constants could be easily changed while running the program. By that you can control the behavior of the model from SpecView with a few mouse clicks. A good example would be to create a custom parameter called Jeq. The intensity function is written such that all intensities are zero, if the total angular momentum J in the ground state of the considered transition is not equal to Jeq. In that case you will be able to change Jeq at run time and observe a simulation of a certain group of molecular lines. Taking this approach you can program even more complex manipulation of the spectrum.

Another significant role of custom parameters is the main calculational loop of the program. The program enumerates states in ground and excited electronic states of the molecule. Since the number of states is infinite and for high-energy states due to Bolzmanís distribution and small line strength the transitionís intensity is close to zero, a cut-off procedure should be used. As a parameter for that cut-off procedure model-writers often use custom parameter. The advantage of using parameter instead of numeric constant is that in most cases for effective simulation the cut-off parameter depends on other parameters, e.g. temperature, of the simulation. In our model of asymmetric top we use 5 custom parameters. So our ParNum() function should return 12.

```
UINT ParNum()
{
  return 12;   /* 7 Standard + 5 extra*/
}
```


4.2.1.8 ParName()

char* ParName(UINT nParam) ñ this used by SpecViewís GDI function and returns names for the custom parameters. The index of the parameter in the array of parameters is passed as an argument. The first custom parameter occupies 8th place in a zero-based array (parameter[7]). In our model we use

5 custom parameters - J_{max} , K_{delta} , and transitional dipole moments along molecular axes a, b and c.

The 4 last constants are used in Inten() function of the module. The first J_{max} is used by ContQN() function. K_{delta} imposes selection rules and accelerates computation in the Inten() function.

```
char* ParName( UINT nParam )
{
    switch( nParam )
    {
        case 7 : return "MaxJ";
        case 8 : return "MaxDltK";
        case 9 : return "Weight of a-type";
        case 10 : return "Weight of b-type";
        case 11 : return "Weight of c-type";
        default : return "Parameter";/* Non of the defined*/
    }
}
```

4.2.1.9 InitPa()

void InitPa(double* pdParam) ñ this function takes the zero-based array of parameters and does not return any value. It initializes the array of parameters to a reasonable set of values for a default simulation.

```
void InitPa( double* pdParam )
{
```

```
        pdParam[0] = 3.0;    /* Parameters. T(K)*/
        pdParam[1] = 0.004;  /* Parameters. DelW(Dopp)*/
        pdParam[2] = 0.004;  /* Parameters. DelW(Norm)*/
        pdParam[3] = 0.001;    /* Parameters. Min Intensity*/
        pdParam[4] = 10.0;   /* Parameters. SpaceToSkip*/
        pdParam[5] = 0.0005; /* Parameters. TheorPlotRes*/
        pdParam[6] = 0.0;    /* Units 0.0<=>cm-1; 1.0<=>MHz; 2.0<=>GHz */
        pdParam[7] = 11;     /* Parameters. Jmax*/
        pdParam[8] = 3;      /* Maximum Delta K*/
        pdParam[9] = 1.0;     /* weight of a-type transition intensities.*/
        pdParam[10] = 0.0;   /* weight of b-typetransition intensities.*/
        pdParam[11] = 0.0;    /* weight of c-type transition intensities.*/
}
```


4.2.1.10 QNnumG()

UINT QNnumG(UINT nStateType) ñ this function takes the number assigned to a given molecular electronic state as a parameter and returns the number of ì goodî quantum numbers ñ quantum numbers needed to describe the block in a block-diagonal matrix of electronic stateís Hamiltonian matrix. For our model the operator of the total angular momentum *J* commutes with a Hamiltonian so we will choose this number as a descriptor. Our function will return 1.

```
UINT QNnumG(UINT nStateType)
{
  return 1; /* total angular momentum J*/
```

```
}
```

4.2.1.11 QNnumB()

UINT QNnumB(UINT nStateType) ñ the return value of that function is the number of ì badî quantum numbers used in a module in particular electronic state nStateType. The concept of ì badî quantum number is explained in the introduction. These numbers allow one to distinguish molecular states within the block of the block-diagonal matrix. Obviously it is always possible to distinguish the states with a unique number. In our case we will try to number the states with two quantum numbers from the limiting cases of prolate and oblate tops (K_a and K_c), which give physical insight into a problem. For asymmetric top model the return value is 2.

```
UINT QNnumB(UINT nStateType)
{
        return 2;  /* Ka and Kc*/
}
```

4.2.1.12 QNName()

char* QNName(UINT nStateType, UINT nQNumb) This function is used by the GDI part of the program and returns the names of the quantum numbers that completely define basis functions of the model. The argument list of the function consists of the number of the electronic state and the number of a quantum number. An array of quantum numbers always starts with a set of ì goodî quantum numbers. The model of asymmetric top should return ì Jî , ì K_aî and ì K_cî correspondingly for nQNnumb 0, 1, 2.

```
char* QNName( UINT nStateType, UINT nQNumb )
{
	switch(nQNumb)
	{
		case 0 : return "J";
		case 1 : return "Ka";
		case 2 : return "Kc";
		default : return "QN"; /* Non of the defined*/
	}
}
```


4.2.1.13 QNBase()

UINT QNBase(UINT nStateType, UINT nQN) This function is used by the GDI part of the program. The SpecView program internally uses only integers as values for quantum numbers. In case the physical value of the quantum number is half odd integer, the number kept in the memory of the computer is the quantum number value doubled. In that case the denominator used to get the correct value for the quantum number is called a ì baseî for the quantum number. WARNING: while accessing quantum numbers in any function of the model, remember, that those numbers always odd integers and do not necessarilly correspond to the physical value of the quntum number (except for a trivial case when the base is equal to 1). The function takes two arguments ñ the number of the electronic state and the number of the modelís quantum number. For the case of the asymmetric rotor all the quantum numbers are integers, so our function returns 1.

```
UINT QNBase( UINT nStateType, UINT nQN )
{
	return 1;
}
```

The next 3 functions are used in the main calculational loop of the SpecView application for the generation of the information on the set of eigenstates in the ground and excited states of the molecule. They govern the iteration of the program through the space of good quantum numbers. As a standard loop it requires the function that initializes the set of ì goodî quantum numbers (function StartQN()), the function that increments the set of ì goodî quantum numbers (function ContQN()) and the function that checks the condition when to exit the loop (function NextQN()).

4.2.2 Service functions

4.2.2.1 StartQN()

void StartQN(UINT nStateType, int* pnQNd) ñ the function takes the number of the electronic state and the pointer to the array of ì goodî quantum numbers. It does not return any value. This function starts the iteration of the loop. In our example model we will set our only quantum number J to zero.

```
void StartQN( UINT nStateType, int* pnQNd )
{
        pnQNd[0]=0;  /* for the first iteration J is set to 0 then in a loop it is increased to Jmax;*/
}
```


4.2.2.2 NextQN()

void NextQN(UINT nStateType, int* pnQNd, double* pdParam) ñ in addition to the number of electronic state and the pointer to the array of ì goodî quantum numbers, this function takes the pointer to the array of modelís parameters. The function does not return any value. This function increments the set of good quantum numbers effectively switching from one block of the block-diagonal Hamiltonian matrix to another. In asymmetric rotor this function will increase the magnitude of J by 1. Do not forget that if

J would be a half-integer, to increase J by one you would need to increase the magnitude of the corresponding quantum number array element by 2!

```
void NextQN( UINT nStateType, int* pnQNd, double* pdParam )
{
	pnQNd[0] ++;
}
```

4.2.2.3 ContQN()

BOOL ContQN(UINT nStateType, int* pnQNd, double* pdParam) - this function returns boolian value (0 (false) or 1 (true)). The list of functioní s arguments includes the number of electronic state, the pointer to the array of integer quantum numbers and the pointer to the array of modelís parameters. This function is used to stop the iterations of the good quantum numbers and quite the loop. The custom parameter J_{max} is used for that. When the J quantum number of the model exceeds custom parameter J_{max} , the function returns 0. It returns 1 otherwise.

```
BOOL ContQN( UINT nStateType, int* pnQNd, double* pdParam )
{
	return(pnQNd[0] <  pdParam[7]); //continue the loop untill J is less than Jmax
}
```

4.2.2.4 HamSize()

UINT HamSize(UINT nStateType, int* pnQNd) The HamSize() function takes the number of the electronic state and the pointer to the array of ì goodî quantum numbers. It returns the size of the block of block-diagonal Hamiltonian matrix for the current set of ì goodî quantum numbers. This function is used by SpecView to allocate the appropriate memory in the memory of the computer for the following Hamiltonian matrix initialization and processing. For a given J we have 2J+1 different functions of type |J,K>, a complete basis set for eigenstates belonging to the block with ì goodî quantum number J. So the block of the Hamiltonian will be represented by the (2J+1)x(2J+1) matrix. The return value of that function is 2J+1.


```
UINT HamSize( UINT nStateType, int* pnQNd )
{
  return 2*pnQNd[0] + 1;  /* 2J+1*/
}
```

4.2.3 Miscellaneous functions

4.2.3.1 Assign()

void Assign(UINT nStateType, double* pdStWF, int* StQN, int StNum) ñ this function takes as a parameter the number of the electronic state, the pointer to the array of the coefficients in a linear combination of basis wavefunctions, the pointer to the array of quantum number and the number of the state in energy order among the states with a current set of ì goodî quantum numbers. This function is used in order to make an intellegent assignment of the line. If the default assignment algorithm is sufficient (as described in ì Assighnment and fitî section), an empty body could be supplied for that function. In our case, as was mentioned before, the eigenvectors of the asymmetric top Hamiltonian matrix will have equal coefficients and custom Assign() function is necessary. In our case Assign() function can rely totally on the diagram on the page 17. We make an assignment of ì badî quantum numbers K_a and K_c based on the energy order of the states:

$K_a = (n+1)\%2$

$K_c = (2J+1-n)\%2$,

where %2 means ì divide by 2 and truncate the result to the next lower integer.

In general the assignment of the ì badî quantum numbers requires thorough analysis of the state wavefunctioní s coefficients. The Assign() function helps the user of the program to determine the labels for the molecular states and transitions. It doesní t affect the simulating engine of the program, neither is it needed by the fitting routine. At the same time the right assignment of the quantum numbers often helps to understand the behavior of the spectrum as a function of molecular constants.

```
void Assign( UINT nStateType, double* pdStWF, int* StQN, int StNum )
{
        StQN[1] = (StNum+1)/2; //Ka
        StQN[2] = (2*StQN[1] - StNum+1)/2; /* Kc  here the assignment is made based on the energy
order position number of the state.*/
}
```

4.2.4 Core calculational functions

4.2.4.1 Hamilt()

void Hamilt(UINT nStateType, double* pdConst, int* pnQNd, double** ppdH, int** ppnQNm) This function fills the passed as the argument Hamiltonian matrix with matrix elements for the current set of molecular constants. The arguments are: the number of the electronic state, the pointer to the array of molecular constants of the electronic state, the pointer to the array of ì goodî quantum numbers, the pointer to the matrix allocated with help of previous function and, finally, the pointer to the matrix of ì badî quantum numbers used for the purposes of basis set identification. The size of the last matrix is Number_of_bad_QN X HamSize() . This last matrix labels all the basis set functions from left to right (or from top to bottom) used to construct eigenvectors. Before any initialization started we have to decide about the rule that relates matrix elements to basis set functions. We assume that 2xJ+1 rows of the matrix are labeled with |J, -J>, |J, -J+1>,........ |J, J> from the top down.

The asymmetric rotor Hamiltonian has a simple form of

$$H = (B+C)/2 * J(J+1) + (A- (B+C)/2) * K^2 + (B-C)/2*(K_-^2 + K_+^2)$$

The first two terms give contribution only to diagonal elements of the Hamiltonian matrix ppdH[i][i], and the last defines elements ppdH[i][i±2]. All other elements of the Hamiltonian matrix have to be initialized to zero.

THE MATRIX ELEMENTS ARE NOT ZEROED BY DEFAULT!


```
void Hamilt( UINT nStateType,  double* pdConst,  int* pnQNd,  double** ppdH,int** ppnQNm)
{
        double F( double J, double K ); /*this function is included in any *.dll module project developed
sofar and calculates the matrix elements of operator K± : <J, K ± 1| K± | J, K> = √(J-K)(J+K-1) */
        double J;
        double  A,B,C;
        A = pdConst[0];
        B = pdConst[1];
        C = pdConst[2];
        J = pnQNd[0]); /* J QUANTUM NUMBER*/
        int N = 2*pnQNd[0] + 1;  //Number of states for current J
        int i,j;

        for(i = 0; i < N; i++)  /* INITIALIZE THE HAMILTONIAN MATRIX*/
```

for(j = 0; j < N; j++) ppdH[i][j] = 0.0;

/* initialize all the elements of the matrix initially to zero. When memory is allocated on the heap by any C or C++ command except for the C command calloc() the elements placed in the portion of the allocated memory are not zeroed automatically*/

for(i = 0; i < N; i++)

{

K = i - N; /* K QUANTUM NUMBER*/ this arrangment sugests that we order the basis set | J, K> of the Hamiltonian matrix from |J, -J> to |J, J> from left to right (and from the top to the bottom of the matrix).*/

/* DIAGONAL ELEMENTS*/

ppdH[i][i] = (A-.5*(B+C))*K*K + .5*(B+C)*N*(N+1);

if(i<N-1)

{

ppdH[i+2][i] = ppdH[i][i+2] = (B-C)/2*F(N, K)*F(N, K+1); /*these non-zero elements are due to (B-C) $K^{\pm 2}$ part of the Hamiltonian */

}

ppnQNm[i][0] = K;

/* the array of bad quantum numbers is passed to the function as the argument Bad quantum numbers should be initialized in the function to make clear to the user what is the basis set for the Hamiltonian. This information is used by the GUI part of the program when displaying the wavefunction of the particular state.*/

ppnQNm[i][1] = 0;

/* In our example we have two bad quantum numbers K_a and K_c and only one bad quantum number K in the basis set. Therefor the second bad quantum number of the model is initialized to zero. */

}
}

4.2.4.2 DHamilt()

void DHamilt(UINT nStateType, double* pdConst, int* pnQNd, double** ppdDH, int k) ñ This function computes the derivative of the Hamiltonian matrix with respect to the molecular constant with arrayís index *k*. The function does not return any value and takes as parameters

- the number of electronic state;
- the pointer to the double array of molecular constants;
- the pointer to the integer array of the assigned quantum numbers;
- the pointer to the two dimensional double precision array matrix used by the function to store the derivatives matrix;
- an integer index of the constant with respect to which the Hamiltonian is differentiated.

The implementation of the DHamilt() function written in all models included in SpecView standard package is identical. The reason for it is that if the molecular Hamiltonian is linearly dependent on molecular constants, the Hamiltonian matrix can be represented as a sum

$H_{ij} = \Sigma C_k H^k_{ij}$

where $\{C_k\}$, $(k = 1, N)$ is an array of N molecular constants and $\{H^k_{ij}\}$, $(k=1,N)$ is an array of N constant matrices.

In that case,

$\partial H_{ij}/\partial C_k = H^k_{ij}$

That expression makes finding the Hamiltonian derivatives trivial. If the Hamilt() function is called when the elements of the constants array are equal to zero for all indexes except one, for which the array element is equal to 1, the resultant matrix would be H^k_{ij}, where k is the index of the non-zero constants array. That approach makes it possible to use the same DHamilt() function in all models where the Hamiltonian has linear dependence on molecular constants. The function was not moved from the list of functions in a dynamic link library module to the main executable part of the program because it is possible for the Hamiltonian matrix to be non-linearly dependent upon molecular constants in some cases. Still in many non-linear cases the initialization of the derivative matrix converges to the number of calls of Hamilt() function and doesnít present any significant problem for a programmer.

```
void DHamilt( UINT nStateType,   double* pdConst,    int* pnQNd,    double** ppdDH, int k )
{
```

```
/* THIS IS A STANDARD FUNCTION THAT INITIALIZES THE DERIVATIVE MATRIX IN CASE
OF THE HAMILTONIAN LINEAR  DEPENDENT ON MOLECULAR CONSTANTS */
	UINT i;
	double *pdIndicators;
	int **ppnQNm;
	UINT nDim;

	pdIndicators = (double *)calloc( ConNum(nStateType), sizeof( double ) ); /*Allocate the memory
	on heap for the array of variables of double precision. All elements are initialized to zero.*/
	nDim = HamSize(nStateType,pnQNd);  /* Hamiltonian is (2*J+1)x(2*J+1) */
	pdIndicators[k]=1.0;
	ppnQNm = (int**)calloc( nDim, sizeof( int* ) );

	for ( i = 0; i < nDim; i++ )
		ppnQNm[i] = (int*)calloc( QNnumB(nStateType), sizeof(int) );  /* Although the
	information about bad quantum numbers is not used by the DHamilt() function, the memory still
	need to be allocated on heap since the elements will be accessed by the Hamilt() function called
	next: */

	Hamilt( nStateType, pdIndicators, pnQNd, ppdDH, ppnQNm);
	for ( i = 0; i < nDim; i++ )
		free( ppnQNm[i] );  /* The memory was allocated on heap in a memory space of the
		application (loaded *.dll module shares the memory of the main application).. It is not
		cleaned automatically, so after part of the memory was used it should be released by the
		C command free() for the following use by the application or other program. If the
		allocated memory is not promptly released that leads to a memory leak and finally to the
		crash of the program*/
	free(ppnQNm);
```

```
        free( pdIndicators );
}
```

4.2.4.3 Inten()

void Inten(int* pnCount, BOOL* pbCont, double* pdParam,

double dLoStEnerg, UINT nLoStateType, double* pdLoStWF,

int* LoStQN, double dUpStEnerg, UINT nUpStateType,

double* pdUpStWF, int* UpStQN, float* fInten)

the function takes 12 arguments and returns no value. The list of arguments includes:

- the pointer to the integer variable pnCount. This argument is not used by the current version of SpecView and is kept for the compatibility with models written for SpecSim models and recompiled for SpecView with minor changes.
- The pointer pbCont to the variable of type BOOL.
- a pointer to the array of parameters of type *double.*
- a *double* variable that stores the energy of the state of the ground electronic state.
- a variable of type UINT with a type of a ground electronic state.

- a pointer to the array of double precision coefficients of the linear combination of the basis set functions that gives the wavefunction of the state in the ground electronic state.
- a pointer to the integer array of ì goodî and ì badî quantum numbers of the state of the ground electronic state assigned by the default procedure or Assign() function.
- a double variable that holds the energy of the state of the excited electronic state.
- a variable of type UINT with a type of the excited electronic state.
- a pointer to the array of double precision coefficients of the linear combination of the basis set functions that gives the wavefunction of the state of the excited electronic state.
- a pointer to the integer array of ì goodî and ì badî quantum numbers of the state of the excited electronic state assigned by the default procedure or Assign() function.
- a pointer to the *float* variable that is used to return to the main executable application SpecView the value of the intensity for the transition between the two selected molecular states.

The 12 arguments of the function allow the retrieval of all the necessary information about the states between which the transition occurs, and to access the appropriate module parameters. We will concentrate our discussion on the first two parameters ñ the pointer to integer variable pnCount and the pointer to the BOOL variable pbCont. The first variable as we stated above is not used by the calculational engine of SpecView and should not be referenced in the body of the Inten() function. The second argument passes the address of the BOOL variable to the Inten() function that manipulates the variable. The memory occupied by the BOOL variable is allocated by the main program SpecView. This variable serves two purposes. The first time the function Inten() is the called, the BOOL value of *pbCount is set to FALSE. That signals the function that the energy of the state from the ground electronic state is the lowest. Since the algorithm of SpecView sorts states so that the state of the lower electronic state has the minimum energy the first time the Inten() function is called, that sets the origin for the calculation of the Bolzmanís factor for all consecutive transitions. The other purpose of the variable is the following. In order to enumerate all non-zero transitions between the ground and excited electronic states, SpecView should call the Inten() function M x N times, where M is the number of states in the ground electronic state and N is the number of states in the excited electronic state. In principle the selection rules for the transition strength could limit the number of non-zero transition from the particular state of the ground electronic state of the molecule from N possible down to a few. This can be used by Inten() function that counts the number of non-zero transitions and sets the *pbCont variable to FALSE when the number of allowed transition has been reached. In that case SpecView simulation engine immediately switches to the next state in the ground electronic state and the procedure continues until the last state from the ground electronic state is considered.

In a certain sense this is the most flexible and tricky function of the module. Here you can exploit the list of parameters to the fullest and implement custom behavior of the model. For example in one of the models that we used for the analysis of the molecule with large spin-rotation coupling we introduced 4 custom parameters ñ GroundStateRestriction, ExcitedStateRestriction, GroundState_K, ExcitedState_K. Then in a Inten() function the first 2 parameters were used as flags (if it is 0 ñ that means no restriction imposed, and if it is non-zero ñ the restriction is applied), and the last two worked as filters, in essence calculating correct intensities of the transitions between states with quantum numbers specified by GroundState_K, ExcitedState_K and zeroing all others. The behavior of different K-stacks of the spectrum is affected only by a few constants of the Hamiltonian which drastically simplifies the assignment of experimental profiles. We also have to mention that for the debugging of complex modules it is often useful to build a qualitative picture of the spectrum, setting all non-zero transitions to some constant, for example a unity. This sanity check is done by only one C-language condition statement and clearly visualizes selection rules of the model.

Here we show the sample Inten() function appropriate for our Asymmetric top molecule. The intensities are calculated with accordance to the equation 6.123 (p.287) of ì Angular momentumî by Richard Zare.

```
void Inten( int* pnCount,
                    BOOL* pbCont,
                    double* pdParam,
                    double dLoStEnerg,
                    UINT nLoStateType,
                    double* pdLoStWF, int* LoStQN,
                    UINT nUpStateType,
                    double* pdUpStWF, int* UpStQN,
                    float* fInten )
{
        int f, nJL, nJU, nKaL, nKaU, nDelKmax;
        double dNorm;
        double JL, KL, JU;
        double static dWeightA, dWeightB, dWeightCsq, EMIN, RK; /*variables declared as static within
the body of the function are stored in the memory of the computer even after the function returns.*/
        BOOL static bFlagA, bFlagBC;
```

```
*fInten = 0.0f; /* by default return zero intensity */
nJL = LoStQN[1]; // total angular momentum J of the lower state
nJU = UpStQN[1];  // total angular momentum J of the upper state.
nKaL = LoStQN[2];  // Ka quantum number of the lower state
nKaU = UpStQN[2];  //Ka quantum number of the upper state
JL = (double)nJL;
JU = (double)nJU;
f = nJU - nJL;
nDelKmax = (int)pdParam[8];

if( abs(f)< 2 &&  abs(nKaL-nKaU) < nDelKmax )
{       // dJ <= 1, dK < dKmax  and Parities is diff.  -  selection rules, transition strength is 0 otherwise
    double  dTmp, SUM1, SUM2, SUM3, SUMAB, BF, LS, RINTE;
    int kL,kU,kl;


    // *pbCont = FALSE only for a first call
    if (!*pbCont) // This Ground Level was the first one, so you should
    {           // setup trap for various ladders. Keep in mind that
     EMIN = dLoStEnerg; // states are sorted and lowest
     *pbCont = TRUE;     // are coming first. Here we have just one

 // BOLTZMANN FACTOR hv/kt-->v/(k/h)t, k/h in Hz/K
     RK = 1.380662E-23 / 6.626176E-34;
     RK = (pdParam[6] == 2.0)?RK * 1E-9:     // GHz
                  ((pdParam[6] == 1.0)?RK * 1E-6:RK * 1E-9 / 29.9792458 );
             //MHz            cm-1
```

```
            dNorm = fabs(pdParam[9]) + fabs(pdParam[10]) + fabs(pdParam[11]);
        dNorm = (dNorm==0.0)?1.0:dNorm;
            dWeightA = sqrt(fabs(pdParam[9]/dNorm));    // weight of a-type transition intensities.
            dWeightB = sqrt(fabs(pdParam[10]/dNorm));   // weight of b-typetransition intensities.
            dWeightCsq = fabs(pdParam[11]/dNorm);       // weight of c-type transition intensities.
            bFlagA =  (dWeightA != 0.0);                                //ladder.
            bFlagBC = ( dWeightB!=0.0 || dWeightCsq!=0.0);
           }

//          (*pnCount)++; // One more transition from
 //                     //  a current ground state level

            //  SUMMATIONS FOR a-type, b-type and c-type TRANSITIONS
            SUM1 = 0.0, SUM2 = 0.0, SUM3 = 0.0;
            for ( kl=-nJL; kl < nJL+1; kl++ )
            {
                KL = (double)( kl );
                kL = kl + nJL;
                kU = kL+nJU-nJL;
                if( kU >= 0 && kU < 2*nJU+1 && bFlagA )
                    SUM1 += pdLoStWF[kL]*pdUpStWF[kU]*CG1(JL,1.0,JU,KL,0.0,KL);
                kU -= 1;
                if( kU >= 0 && kU<2*nJU+1 && bFlagBC )
                {
                    dTmp = pdLoStWF[kL]*pdUpStWF[kU]*CG1(JL,1.0,JU,KL,-1.0,KL-1.0);
                    SUM2 += dTmp;
                    SUM3 += dTmp;
```

```
            }
            kU += 2;
            if( kU >= 0 && kU<2*nJU+1 && bFlagBC )
            {
                dTmp = pdLoStWF[kL]*pdUpStWF[kU]*CG1(JL,1.0,JU,KL,1.0,KL+1.0);
                SUM2 -= dTmp;
                SUM3 += dTmp;
            }
        }
        SUMAB = dWeightA*SUM1 + 0.707*dWeightB*SUM2;
        LS = (2*JL+1)*(SUMAB * SUMAB + 0.5*dWeightCsq*SUM3*SUM3);

        BF = exp(-(dLoStEnerg - EMIN)/(RK*pdParam[0])) - exp(-(dUpStEnerg -
EMIN)/(RK*pdParam[0]));          // Boltzmann factor, takes into account the population of the upper
state, which is critical in case of microwave transitions.
        RINTE = BF * LS;
       // Intensity

        if( RINTE >= pdParam[3] && dLoStEnergy<dUpStEnrgy)  /*the returned intensity is not
compared against intensity threshold parameter in the SpecView executable, it memorizes the transition
only if non-zero intensity returns. But we do not want to show weak transition so we must to make
comparison with the threshold partameter here. The second requirement for the energy of the ground state
to be lower than the energy of the excited state is satisfied automatically for electronic transitions and is
natural for microwave transitions where transition intensities are calculated between states of the same
electronic state (if not that condition, all the transition pairs would be calculated twice)*/
            *fInten = (float)RINTE;
    }
    return;
```

```
}
```

The somewhat challenging but always rewarding process of writing of a new model is drastically simplified by the extended library mathutil.c included in a sample project of the SpecView package. This library of spectroscopic functions provides precise and effective routines that calculate

- Clebsch-Gordon coefficients
- Racah coefficients
- 3 ñ J symbols
- 6 ñ J symbols
- 9 ñ J symbols

We welcome everybody to extend the mathutil.c library. If you send us the renewed edition of the library we will gladly include it in a new version of SpecView package as tool to simplify greatly the model development. We will also appreciate if you send us the code for the newly created models. Revised, these modules will probably be included in a list of models distributed with a package, making a considerable contribution to the spectroscopic community.

5. Registering a new model.

Registering a new model is simple. A newly created asyrot.dll file should be renamed and placed in a SpecViewModels folder or any other folder with dynamic link library files used by the application. Then the user has to add a few lines to specview.ini text file. Opened with text viewer the standard specview.ini file should look like that:

```
[Settings]
PreviewPages=2

[Models]
lib1="Asym.Rotor(&int.rotation)"
lib2="Asym.Rotor(&spin-rot.)"
lib3="Sym.Rotor&JT(A-E,I=1/2,S=1/2,C3v)"
lib4="Linear(S=1/2)"
lib5="LinearHF_new(I=1/2,S=1/2)"
lib6="AsyRot with Mult States"
lib7="Asym.Rotor(&spin-rot.) case B Watson"
lib8="Asym.Rotor(&spin-rot. C1 symmetry) case B Watson"
lib9="Asym.Rotor(&spin-rot.) case B"
lib10="Asym.Rotor(&spin-rot. C1 symmetry) case B Watson sym"

[DllLibs]
lib1="C:\Program Files\OSU\Program\SpecViewModels\asyrot.dll"
lib2="C:\Program Files\OSU\Program\SpecViewModels\asyrotsr.dll"
lib3="C:\Program Files\OSU\Program\SpecViewModels\symrotJT.dll"
lib4="C:\Program Files\OSU\Program\SpecViewModels\linear.dll"
lib5="C:\Program Files\OSU\Program\SpecViewModels\linearHF_new.dll"
lib6="C:\Program Files\OSU\Program\SpecViewModels\masyrot.dll"
lib7="C:\Program Files\OSU\Program\SpecViewModels\asyrotsrBnr1.dll"
lib8="C:\Program Files\OSU\Program\SpecViewModels\asyrot_par.dll"
lib9="C:\Program Files\OSU\Program\SpecViewModels\asyrotsrB1.dll"
lib10="C:\Desktop\Dll6\Release\asyrot_sym.dll"
```

The file has two sections with a critical model information ñ [Models] and [DllLibs].

The first part ñ [Models] ñ enumerates all models available for SpecView executable. That list you can see in a list of models in ì Modelsî combo box of the ì Molecular constantsî dialog box of SpecView. The name of each model should coincide with the text string returned by the Name() function implemented in a model. If it is not so the program looses the ability to open the previously saved SpecView Document file and crashes if such an attempt is made.

The section [DllLibs] provides the path to the corresponding *.dll module location.

To register our newly created test.dll dynamic link library module with the Name() function returning ì New modelî , we obviously have to add a line

```
lib11="New model"
```

to the [Models] section of the specview.ini file and the line with the location of the test.dll file (default models folder created by the self-extracting installation program SpecVi.exe)

```
lib11="C:\Program Files\OSU\Program\SpecViewModels\test.dll"
```

The edited specview.ini file is shown lower:

```
[Settings]
PreviewPages=2

[Models]
lib1="Asym.Rotor(&int.rotation)"
lib2="Asym.Rotor(&spin-rot.)"
Ö Ö Ö Ö Ö Ö Ö Ö Ö Ö Ö Ö
lib10="Asym.Rotor(&spin-rot. C1 symmetry) case B Watson sym"
lib11=î New modelî

[DllLibs]
lib1="C:\Program Files\OSU\Program\SpecViewModels\asyrot.dll"
lib2="C:\Program Files\OSU\Program\SpecViewModels\asyrotsr.dll"
Ö Ö Ö Ö Ö Ö Ö Ö Ö Ö Ö Ö
lib10="C:\Desktop\Dll6\Release\asyrot_sym.dll"
lib11="C:\Program Files\OSU\Program\SpecViewModels\test.dll"
```

6. Self-extracting SpecVi.exe package.

The recent SpecView package is available on our website Ö Ö Ö .

The package installs:

1. The latest version of SpecView.exe executable.

2. The library of 10 models.

3. An example dynamic link library project of the model of Asymmetric rotor ready to be edited, compiled and built with Microsoft Visual C++ 6.0 environment. The project includes mathutil.c file with the variety of spectroscopic functions (Clebsh-Gordon coefficients, 3,6 and 9 ñ J symbols).

5. The source code for all 10 currently provided in a package models.

6. The instruction and the user manual in PDF, Word and HTML formats.

7. An Excel Macro personal.xls that allows to convert spectral line data file (for example from HITRAN data base) into a format readable by SpecView. To make use of that utility double click personal.xls icon and click ì Enable microî button on the pop-up dialog box. Then go to ì Toolsî menu option and choose (Play) Macro option. A Dialog Box will show up. Double click PERSONAL.XLS!Macro2. this will start a micro. On the first step the dialog box will help your to locate the data file with spectral data. After that a script will be run and the newly formatted file will be created. You will see a ì Save Asî Dialog Box that allows you to save the result of the transformation. The new file is now compatible with SpecView and can be loaded in an environment through the applicationís *File->Export Data File* menu option. In essence, the excel-based VB script transfers the data in format

Line frequency::::::::::Intensity of the line

Into

Line frequency::::::::::0

Line frequency::::::::::Intensity of the line

Line frequency::::::::::0

where ::::::::::: stands for any reasonable delimiter, like space or tab.

The described above operation allows for the spectral line data to be treated as experimental spectrum with negligible line widths.

Although this option could be implemented in the main application code ñ SpecView, the use of VB Excel control helps to manage the data in a more versatile manner.

Acknowledgments

I would like to acknowledge Terry A. Miller, Chris Carter, Sandhya Gopolakrishnan and other members of the group for invaluable help and comments during the development and testing of SpecView program, models and SpecVi package.