Rapport BIPM-91/2

BUREAU INTERNATIONAL DES POIDS ET MESURES

On the Hyperfine Structure of Iodine:

2.

To Calculate Hyperfine Constants
on the Basis of Experimental Data

by

Susanne Picard-Fredin and Annick Razet[+]

[+] Institut National de Metrologie, Conservatoire National des Arts et Metiers, 292 rue St Martin, 75141 Paris cedex 03, France[*]

* Laboratory associated to C.N.R.S. UA827

PAVILLON DE BRETEUIL
92312 Sèvres Cedex
France

Rapport BIPM-91/2

On the Hyperfine Structure of Iodine:

2.

To Calculate Hyperfine Constants on the Basis of Experimental Data

by

Susanne Picard-Fredin and Annick Razet

Abstract

A method of calculating the hyperfine coupling constants of iodine is described. It has been transformed into a computer programme written in Turbo-Pascal. The accuracy of the calculations using the programme is compared with that of others published elsewhere. To exemplify the method the hyperfine constants of the R(12) 26-0 and the R(106) 28-0 transitions in the B-X system of iodine are calculated.

Table of Contents

1 Introduction

Investigations of hyperfine structure using sub-Doppler spectroscopy have been a popular subject over the past twenty years. Molecular iodine has probably been the most extensively and frequently studied diatomic molecule in this field. The ease with which it may be handled once contained in a glass cell and its rich, often strong, fluorescence spectrum in the visible, combined with its rather narrow hyperfine lines (some tens of kHz typically), make it particularely attractive for metrological applications. These relate to optical frequency standards. Indeed, simultaneously with the new definition of the Metre [1] based on a fixed value for the velocity of light in vacuum, the frequencies of five molecular transitions were recommended for the realization of the Metre [2]. Four of these correspond to hyperfine transitions in iodine.

The theory of hyperfine interaction has progressed in parallel with improved high precision experimental techniques. A good review of the most important steps in this developement up to 1981 is given by Bordé et al. [3]. Recently, effort has concentrated on understanding the influence of molecular vibration and rotation on the nuclear electric quadrupole and spin-rotation interaction [4-9]. An understanding of hyperfine structure is an important factor in the improvement of present optical frequency standards. Methods are needed for the calculation of spectra and characteristic constants in order to understand and control experimental results.

Here we present a computer programme for the calculation of hyperfine coupling constants for iodine, $^{127}I_2$, starting from experimental data. Our programme is written in Turbo-Pascal and is based on the first-order perturbation method first described by Bordé et al. [3]. We chose this iterative computational method as it has been shown to converge rapidly; three or four iterations are generally sufficient [10]. In Sec.2.1 we present our Hamiltonian and give a detailed description of the iterative first-order perturbation method in Sec.2.2. In Sec.3.1 we compare the results obtained using our programme with earlier calculations done by several other groups. Finally, we present in Sec.3.2 a calculation of hyperfine constants using recent experimental data obtained by Chartier et al. [11]. The programme is listed among the Appendices which also includes complementary information and guidance.

2 Theory

Before describing the calculation method, we first present the Hamiltoninan that was used. For further details on how to calculate a hyperfine spectrum we refer to [12].

2.1 The Hyperfine Hamiltonian

The hyperfine Hamiltonian, $\hat{H}_{hfs}$, can be written as

$$\hat{H}_{hfs} = eqQ\hat{H}_{EQ} + C\hat{H}_{SR} + d\hat{H}_{TSS} + \delta\hat{H}_{SSS} \tag{1}$$

where $\hat{H}_{EQ}$ [13], $\hat{H}_{SR}$ [13], $\hat{H}_{TSS}$ [14], $\hat{H}_{SSS}$ [15] represent the electric quadrupole, spin-rotation, tensorial spin-spin and scalar spin-spin interactions respectively[1]. We have chosen to use the conventional symbols eqQ, C, d and δ to represent the constants for electric quadrupole, spin-rotation, tensorial spin-spin and scalar spin-spin interactions respectively. The Hamiltonian can be extended in terms of higher degree [3] but in most cases these are not significant. In our calculation we use a basis set which contains the levels (v,J) and also (v,J±2), where v and J represent the vibrational and rotational quantum numbers respectively. This is why we must introduce the rotational Hamiltonian $\hat{H}_R$ into the calculation. We refer to the interaction between these levels as $\Delta J=0,\pm2$ from now on. The basis set can be extended by taking $\Delta J=\pm4$ into account and even using $\Delta J=\pm8$, but we consider these contributions also to be negligible.

The different interactions can be written as

1 Observe that the notation of the Hamiltonian is slightly different here than from notation used in [12].

$$< FIJ|\hat{H}_{EQ}|F'I'J' >= (-1)^{F+J+2I}\frac{1}{2}[(2I+1)(2I'+1)]^{1/2} \times$$

$$\begin{bmatrix}\begin{pmatrix} I_1 & I_1 & 2 \\ I_1 & -I_1 & 0\end{pmatrix}\begin{pmatrix} J' & J & 2 \\ J & -J & 0\end{pmatrix}\end{bmatrix}^{-1}\begin{Bmatrix} I_1 & I_1 & I \\ 2 & I' & I_1\end{Bmatrix}\begin{Bmatrix} F & J & I \\ 2 & I & J'\end{Bmatrix} \tag{2}$$

$$< FIJ|\hat{H}_{SR}|F'I'J' >= \delta_{JJ'}\delta_{II'}\frac{1}{2}[F(F+1)-I(I+1)-J(J+1)] \tag{3}$$

$$< FIJ|\hat{H}_{TSS}|F'I'J' >= \delta_{JJ'}(-1)^{F+I'+1}(2J+1)[I_1(I_1+1)(2I_1+1)]\,[30(2I+1)(2I'+1)]^{1/2} \times$$

$$\begin{pmatrix} J & 2 & J \\ 0 & 0 & 0\end{pmatrix}\begin{pmatrix} F & J & I \\ 2 & I' & J'\end{pmatrix}\begin{Bmatrix} I_1 & I_1 & 1 \\ I_1 & I_1 & 1 \\ I & I' & 2\end{Bmatrix} \tag{4}$$

$$< FIJ|\hat{H}_{SSS}|F'I'J' >= \delta_{JJ'}\delta_{II'}\frac{1}{2}[I(I+1)-2I_1(I_1+1)] \tag{5}$$

$$< FJI|\hat{H}_R|F'J'I' >= B_v J(J+1)-D_v J^2(J+1)^2+H_v J^3(J+1)^3+M_v J^4(J+1)^4 \tag{6}$$

using |FJI> as the basis set[2]. Here, F and I represent the total angular momentum quantum number and the nuclear spin quantum number respectively. B_v, D_v, H_v and M_v are the rotational constants [17]. The big parantheses represent 3-j symbols: small and large curly brackets correspond to 6-j and 9-j symbols.

When J" ,where (") indicates the lower state, is even we have a so called 'para state' in $^{127}I_2$ and obtain, using algebraic addition rules, a total of 15 hyperfine components. Including $\Delta J=0,\pm 2$ we obtain a Hamiltonian matrix with the dimension 45x45. It is block diag-

2 Explicit formulae for Eq.(2) can be found in [16].

onal with respect to F and contains 13 sub-matrices. When J" is odd we have an 'ortho state' in $^{127}I_2$ and the block diagonal Hamiltonian becomes 63x63 in dimension for $\Delta J=0,\pm 2$, including 15 sub-matrices.

The Hamiltonian matrix for a para state is represented in Fig.1 where the numbers indicate the dimensions of the submatrices, each labelled by a different F. A hyperfine transition is associated with the energy difference between the upper (') and lower (") state (for example the B and the X states in iodine). The Hamiltonian submatrices must be diagonalised to calculate the eigenvalues and eigenstates of each upper and lower energy level involved in the hyperfine transition. In such a way one can calculate a hyperfine spectrum, using known constants. Below, we show how to obtain such hyperfine constants from experimental data.

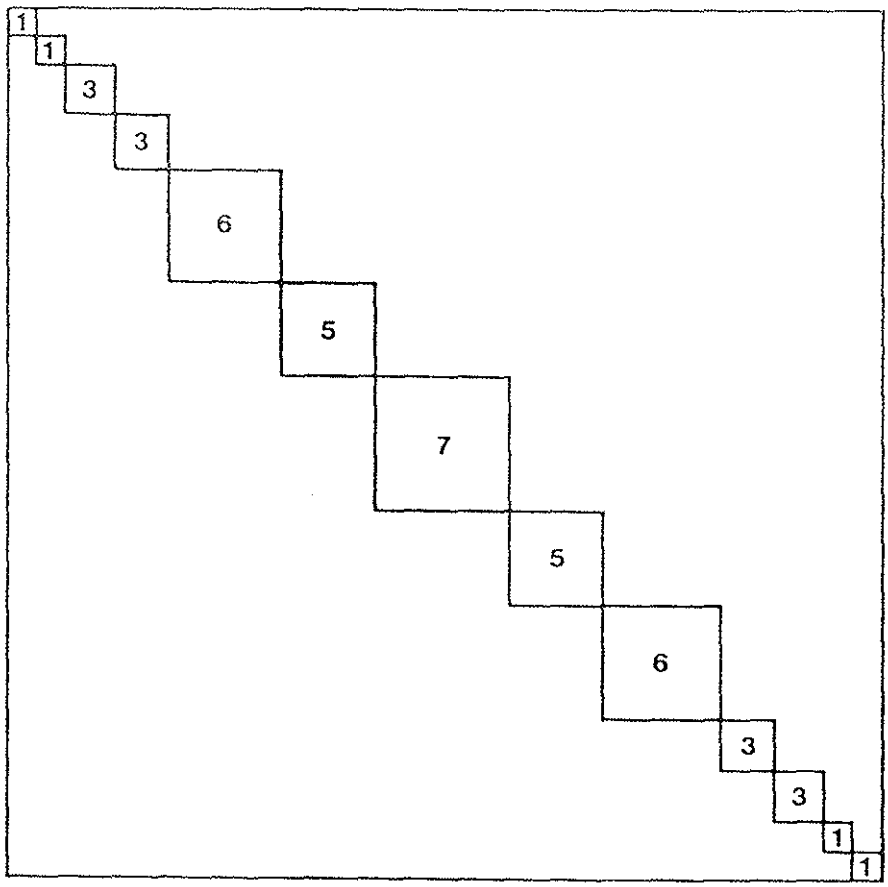


Fig1. Dimensional scheme of the Hamiltonian matrix and its sub-matrices for a para state including $\Delta J=0,\pm 2$.

2.2 Numerical Method

2.2.1 First Order Perturbation Iterative Computational Method

The first-order perturbation-iterative computational method was introduced by Bordé et al. [3]. We here describe in detail the application of this method to hyperfine transitions.

As the method is iterative, one needs an initial set of hyperfine constants. These approximate constants can for example be obtained from the empirical formulae derived by Gläser [4], Morinaga et al. [5] or Spirko and Blabla [7]. Using these constants a first approximation to the spectrum can be calculated. The difference between the observed and calculated spectrum gives a measure of the magnitude of the correction required in the initial constants. Hence, new constants are obtained which give a calculated spectrum closer to the observed one. The procedure is repeated until the correspondance between the observed and calculated spectrum is satisfactory.

The corrections to the constants are obtained in the following way:

The Hamiltonian equation can be written as

$$\hat{H}_{hfs} \mid A_i >= E_i \mid A_i > \tag{7}$$

where $\hat{H}_{hfs}$ is the Hamiltonian hyperfine operator, $|A_i>$ is an eigenstate and E_i is its eigenvalue.

A small variation of the hyperfine constants, dX_j, creates a new Hamiltonian $\hat{H}'_{hsf}$ (see Eq(1)):

$$\hat{H}'_{hsf} = \hat{H}_{hsf} + \sum_j d\hat{H}_j \tag{8}$$

and gives new eigenvalues:

$$E_i \rightarrow E_i + dE_i = E_i + \sum_j dE_{ij}. \tag{9}$$

where $d\hat{H}_j$ and dE_i are associated with the variation. From here on the index 'j' indicates the hyperfine constants, and the index 'i' indicates the hyperfine components. One can write

$$d\hat{H}_j = \hat{H}_j dX_j \tag{10}$$

Considering $d\hat{H}_j$ as a small perturbation and by applying the first-order perturbation theory we obtain

$$\frac{dE_{ij}}{dX_j} = \frac{< A_i \mid \hat{H}_j \mid A_i >}{< A_i \mid A_i >}. \tag{11}$$

For completeness, the first-order perturbation theory is reviewed in Appendix I.

Letting ε represent an observed state, one can write for a transition 'i'

$$\varepsilon'_i - \varepsilon''_i = (E'_i + dE'_i) - (E''_i + dE''_i). \tag{12}$$

This relation is illustrated by Fig. 2.

According to Eq.(9) we can hence write

$$\varepsilon'_i - \varepsilon''_i = E'_i - E''_i + \sum_j \frac{dE'_{ij}}{dX'_j} dX'_j - \sum_k \frac{dE'_{ik}}{dX''_k} dX''_k, \tag{13}$$

where E'_i and E''_i represent the energies obtained from the initial approximative constants.

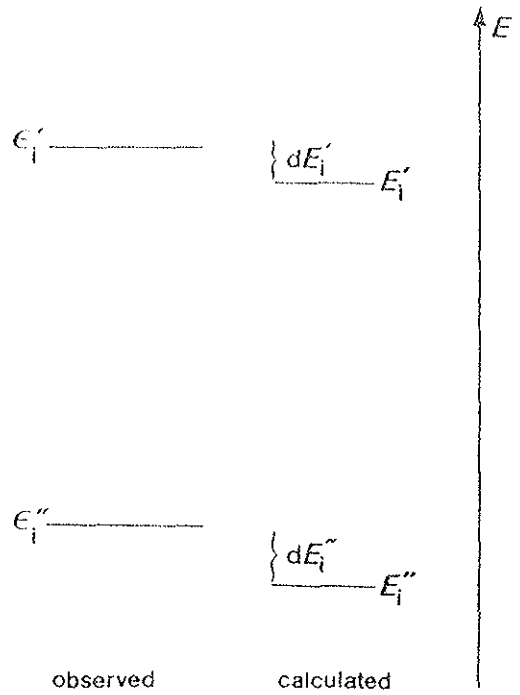


Fig2. Energy level scheme used for the approach of the calculation of hyperfine constants.

Using matrix representation, Eq.(13) can be written as

$$\begin{pmatrix} (\varepsilon'_1-\varepsilon''_1)-(E'_1-E''_1) \\ \vdots \\ (\varepsilon'_N-\varepsilon''_N)-(E'_N-E''_N) \end{pmatrix} = \begin{pmatrix} \frac{dE'_1}{dX'_1}\cdots\frac{dE'_1}{dX'_j} & \frac{-dE''_1}{dX''_1}\cdots\frac{-dE''_1}{dX''_k} \\ \vdots \quad\quad \vdots & \vdots \quad\quad \vdots \\ \frac{dE'_N}{dX'_1}\cdots\frac{dE'_N}{dX'_j} & \frac{-dE''_N}{dX''_1}\cdots\frac{-dE''_N}{dX''_k} \end{pmatrix} \begin{pmatrix} dX'_1 \\ \vdots \\ dX'_j \\ dX''_1 \\ \vdots \\ dX''_k \end{pmatrix} \qquad (14)$$

where the first matrix, represented below as Y, represents the difference between the observed and calculated spectra and N is the number of observed transitions. If we write Eq.(14) in the more compact form:

$$Y=A\theta, \tag{15}$$

the matrix θ contains the corrections to the hyperfine constants and can be explicitly written as

$$\theta = (A^T A)^{-1} A^T Y. \tag{16}$$

The calculations represented by Eqs (7-16) are repeated until the values dX'_j and dX''_k converge.

2.2.2 Statistical Weights and Standard Deviation

The precision of the measurement of each frequency interval is not necessarily the same for all experimental values. From a series of measurements, we can determine the standard deviation σ_i for each frequency separation. These experimental data are independent, and the variance-covariance matrix V_y can be defined as

$$V_y = \begin{pmatrix} \sigma_1^2 & \dots & 0 & \dots & 0 \\ . & . & . & . & . \\ 0 & \dots & \sigma_i^2 & \dots & 0 \\ . & . & . & . & . \\ 0 & \dots & 0 & \dots & \sigma_N^2 \end{pmatrix}. \tag{17}$$

The standard deviations can be used to determine the statistical weights g_i:

$$g_i = \frac{N\frac{1}{\sigma_i^2}}{\sum_i \frac{1}{\sigma_i^2}} \tag{18}$$

These, in turn, can be introduced in the calculation for the determination of the matrix θ which is then written as

$$\theta = (\mathbf{A}^T\mathbf{G}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{G}\mathbf{Y}, \tag{19}$$

where G is the matrix of statistical weights:

$$\mathbf{G} = \frac{N}{\sum_i \frac{1}{\sigma_i^2}}\mathbf{V}_y^{-1}. \tag{20}$$

The standard deviation, SD, of a fit is given by the expression:

$$SD = \sqrt{\sum_i \left(\frac{(\nu_{i,obs} - \nu_{i,calc})^2}{N-K} \right)} \tag{21}$$

where K is the number of fitted parameters. The variance-covariance matrix of the matrix θ is determined from

$$\mathbf{V}(\theta) = (\mathbf{A}^T\mathbf{G}\mathbf{A})^{-1}SD^2. \tag{22}$$

3 Results

The calculations described in Sec.2 were carried out using a computer programme written in Turbo-Pascal 4.0. Its content is listed in Appendix II. Below are listed the contents of Appendices I-III.

Appendix I	First-order perturbation theory.
Appendix II-a	Flow chart of the most important parts of the programme.
Appendix II-b	Main programme written in Turbo-Pascal 4.0
Appendix II-c	Subroutines for calculating the Hamiltonian matrix and deducing the energy differences.
Appendix II-d	Subroutines for calculating the rotational contribution.
Appendix II-e	Subroutines to diagonalise the symmetrical matrix and to calculate the energy eigenvalues.
Appendix II-f	Subroutines containing matrix calculations and evaluation of standard deviation.
Appendix III-a	Format of input file.
Appendix III-b	Example of output file.

In the rest of this section we present some results derived using this programme.

3.1 Comparison of Results

We have compared the calculations made by our programme with four different calculated structures for which three programmes with different origin have been used. These calculations have appeared in works by Foth and Spieweck [18], Bordé et al. [3] and Razet et al. [19].

These comparisons are listed in App.IV whose contents are listed below:

Appendix IV-a	Comparison with Foth and Spieweck [18] R(98) 58-1.

Appendix IV-b	Comparison with Bordé et al. [3] R(98) 58-1.
Appendix IV-c	Comparison with Bordé et al. [3] P(13) 43-0.
Appendix IV-d	Comparison with Razet [19] R(47) 9-2.

The frequency difference between the BIPM computation and the others is less than 1 kHz, except for that of Foth and Spieweck. These differences depend on several factors; for example, Bordé [3] uses an eigen matrix expanded to $\Delta J=\pm 4$. As the programmes are not identical, round-off errors are introduced. However, we have not used the same origin as Foth and Spieweck and our calculation gives recalculated lines which differ at most by 7kHz from that of Foth and Spieweck. Comparisons between the calculations by Foth and Spieweck and by Bordé et al. can be found in [3].

We have adapted our programme to calculate the hyperfine constants for $^{127}I_2$ (I=5/2). A few of the matrix dimensions in the programme have to be modified to allow calculation of the hyperfine structure of the $^{129}I_2$ molecule (I=7/2).

3.2 Calculated Hyperfine Constants of the R(12) 26-0 and R(106) 28-0 Transition in the B-X System of Iodine.

Recently, several groups have shown an interest in the green HeNe laser at 543 nm which was introduced on the commercial market in 1985. The 543 nm laser line has been shown to coincide with two rovibrational transitions in the B-X system of $^{127}I_2$, namely R(12) 26-0 and R(106) 28-0 [20]. At least four groups have observed the hyperfine spectrum emanating from these transitions [11,21-23]. Chartier et al. [11] have reported measured frequency differences between 15 hyperfine components. These were obtained by stabilizing two independent laser systems, using the third-derivative technique, each of them onto a different hyperfine component and measuring the beat frequency. We found it tempting to calculate the hyperfine constants for these components using our programme. Here we present calculated constants, using the experimental data from Chartier et al. [11].

The hyperfine coupling constants for the (v"=0,J"=13) and (v"=0,J"=15) levels of the ground state (X) calculated by Bordé at al. [3] are about the same. In addition, no evident J-dependence of the hyperfine coupling constants was observed for the 13-1 band in the B-X system for J=0-10 [8]. Precise hyperfine constants of the ground state have been determined for the (v"=0,J"=13) level by Yokozeki and Munter [24]. For this reason we have chosen to represent the (v"=0,J"=12) level of the ground state by the constants of Yokozeki and Munter [24], and they are maintained fixed in our calculations. The choice of the ground state constants mainly affect the calculated value of eqQ' and C', but has little influence on the calculated values of ΔeqQ and ΔC where ΔeqQ=eqQ'-eqQ" and ΔC=C'-C". We have, therefore, also used these constants to define the (v"=0,J"=106) level, as no experimental data close to this level is available. The ground state hyperfine constants are listed in Table 1. The rovibrational data used in the calculation are taken from Gerstenkorn and Luc [17].

Table 1. Hyperfine constants of the ground state for (v"=0,J"=13) given by Yokozeki and Munter [24][a].

eqQ"/MHz	-2452.5837(16)
C"/kHz	3.162(8)
d"/kHz	1.58(5)
δ"/kHz	3.66(3)

[a] Uncertainty in last digits is shown in parenthesis.

Table 2 shows the observed and calculated frequency differences for the **a** components (a_7-a_{15}) which correspond to the R(12) 26-0 transition. The a_{10} component is chosen as the origin. We have fitted the constants for this line in two ways. First we fitted the data without statistical weights. Then we used the inverse square of the given standard deviation of each component [6] as a weighting factor and fitted the data again. Table 3 shows the observed and calculated frequency differences for the **b** components (b_1-b_6) which corre-

spond to the R(106) 28-0 transition. Here we also fitted the constants first without and then with weights. We chose the b_1 component as origin as it had the smallest experimental standard deviation.

Table 2. Observed frequencies [11] and our calculated frequencies for the R(12) 26-0 transition in MHz without and with weights using the constants listed in Table 4. The weights used are the experimental uncertainties[a].

Component	I	F-J	Obs. in MHz[b]	Calc. in MHz without weights	Calc. in MHz with weights
a_7	4	-2	-156.26(1)	-156.242	-156.258
a_8	4	2	-136.99(2)	-136.984	-136.990
a_9	4	3	-83.25(5)	-83.251	-83.281
a_{10}	4	0	0	0	0
a_{11}	4	-1	110.507(10)	110.516	110.512
a_{12}	2	-2	119.779(7)	119.759	119.777
a_{13}	4	1	172.905(6)	172.911	172.903
a_{14}	2	2	186.104(7)	186.100	186.106
a_{15}	0	0	290.18(5)	290.197	290.224

[a] Ground state hyperfine constants fixed to the values listed in Table 1.

[b] Uncertainty in last digits is shown in parenthesis.

Table 3. Observed frequencies [11] and our calculated frequencies for the R(106) 28-0 transition in MHz without and with weights using the constants listed in Table 5. The weights used are the experimental uncertainties[a].

Component	I	F-J	Obs.-b_1 in MHz[bc]	Calc. in MHz[b] without weights	Calc. in MHz[b] with weights
b_1	2	0	0.000(9)	0	0
b_2	4	-4	253.297(20)	253.446	253.302
b_3	2	-1	282.10(11)	282.036	281.993
b_4	2	1	291.50(18)	291.516	291.542
b_5	4	4	320.28(20)	320.332	320.461
b_6	4	-3	401.11(20)	401.008	400.893

[a] Ground state hyperfine constants fixed to the values listed in Table 1.

[b] b_1 taken as fixed origin in the fitting.

[c] Uncertainty in last digits is shown in parenthesis.

Tables 4 and 5 give the calculated hyperfine constants for the R(12) 26-0 and the R(106) 28-0 transitions respectively, using the experimental values of Tables 2 and 3. The standard deviation of the fitted constants is represented by σ_c, and σ_{tot} represents the standard deviation of the fits. The standard deviations for the **a** components were 15 kHz and 6 kHz for the calculation first not using and then using weights respectively. The experimental standard deviations given for the components are in good agreement with the calculated ones. The standard deviation for the **b** components decreases from 115 kHz to 44 kHz first not using and then using weights respectively. This corresponds to a decrease of the standard deviation by about the same factor as for the **a** components. However, as the **b** components were few, and b_1 and b_2 had a considerably higher weight than the other components, the decrease in standard deviation is accompanied by an increase in uncertainty of the calculated constants. We have therefore chosen to fit only eqQ' and C' for the **b** components and conclude that higher precision measurements are required for a better determinantion of the hyperfine constants.

Table 4. Fitted hyperfine constants for the R(12) 26-0 transition[a].

Constant	Without weights	σ_c	With weights	σ_c
eqQ'/MHz	-534.936	0.078	-534.883	0.046
C'/kHz	62.41	0.22	62.37	0.12
d'/kHz	-27.4	2.1	-27.7	1.0
δ'/kHz	b	-	-7.3	1.6
ΔeqQ/MHz	1917.648	0.078	1917.701	0.046
ΔC/kHz	59.25	0.22	59.21	0.12
σ_{tot} /kHz	15		6	

[a] Ground state hyperfine constants fixed to the values listed in Table 1.
[b] δ' fixed to 0.

Table 5. Fitted hyperfine constants for the R(106) 28-0 transition[ab].

Constant	Without weights	σ_c	With weights	σ_c
eqQ'/MHz	-539.56	0.32	-539.62	0.47
C'/kHz	73.55	0.17	73.87	0.17
ΔeqQ/MHz	1913.02	0.32	1912.96	0.47
ΔC/kHz	70.39	0.17	70.71	0.17
σ_{tot} /kHz	115		44	

[a] Ground state hyperfine constants fixed to the values listed in Table 1.
[b] d and δ' fixed to 0.

Gläser [4] has derived empirical formulae for ΔeqQ and ΔC. Our fitted values for ΔeqQ and ΔC differ from his calculated values by 2 MHz and 7 kHz respectively for the **a** components, and by 1 MHz and 4 kHz respectively for the **b** components. However, our ΔeqQ and ΔC correspond to within 0.01 MHz and 0.2 kHz respectively for the **a** components with constants recently calculated by Gläser [25] and based on experimental data.

Morinaga et al. [5] have also derived empirical formulae to calculate eqQ' and ΔC. As pointed out by Morinaga et al., eqQ' differs from their formula for v'>15: our fitted values for eqQ'(v'=26)=-534.9 MHz and eqQ'(v'=28)=-539.6 MHz are effectively 3 MHz and 5 MHz smaller, respectively, than the values calculated from their formula, as well as for ΔC which differ by 9 kHz and 7 kHz respectively.

Conclusion

We have determined four significant hyperfine constants, namely eqQ', C', d' and δ', for the (v'=26,J'=13) level of the B state giving an unweighted standard deviation between observed [6] and calculated frequencies of 15 kHz. We have also calculated eqQ' and C' for the (v'=28,J'=107) level of the B state which give a standard deviation eight times higher than that found for the (v'=26,J'=13) level. We attribute this level of uncertainty to the relative frequencies of the components, caused by the less favourable experimental conditions obtained in this frequency region.

The constants eqQ'(v'=26), C'(v'=26), eqQ'(v'=28) and C'(v'=28) have been calculated using beat frequency data. These data will contribute to the understanding of the v-dependence of eqQ and C. New measurements are however needed to better determine the higher order interaction terms, especially for the R(106) 28-0 transition. This could be done by stabilizing a 543 nm HeNe laser onto one hyperfine component and measuring the beat frequency between the HeNe laser and a stabilized dye-laser.

4 Summary

The first-order perturbation-iterative computational method has been described in detail. It has been transformed into a computer programme written in Turbo-Pascal to calculate hyperfine constants. The accuracy of the calculations by our programme has been compared with calculations published elsewhere and differs from them by not more than 1 kHz. First hyperfine constants of the R(12) 26-0 and the R(106) 28-0 transitions in the B-X system of iodine, co-incident with the 543 nm HeNe laser line, have been calculated.

Acknowledgement We wish to thank P. Juncar for helpful criticism, J.-M. Chartier and L. Robertsson at BIPM for experimental data, and M. Gläser at PTB for useful discussion.

5 References

[1] Comptes Rendus 17e CGPM, (1983) 45-49.

[2] Documents Concerning the New Definition of the Metre, Metrologia 19 (1984) 163-177.

[3] Ch.J. Bordé, G. Camy, B. Decomps, J.-P. Descoubes and J. Vigué, J. Physique 42 (1981) 1393.

[4] M. Gläser, Opt. Comm. 54 (1985) 335.

[5] A. Morinaga, K. Sugiyama, N. Ito and J. Helmcke, J. Opt. Soc. Am. B 6 (1989) 1656.

[6] A. Morinaga, Jpn. J. Appl. Phys. 23 (1984) 774.

[7] V. Spirko and J. Blabla, J. Mol. Spec. 129 (1988) 59.

[8] M. Wakasugi, T. Horiguchi, M. Koizumi and Y. Yoshizawa, J. Opt. Soc. Am. B 5 (1988) 2298.

[9] R. Bacis, M. Broyer, S. Churassy, J. Vergès and J. Vigué, J. Chem. Phys. 73 (1980) 2641.

[10] A. Razet, Thèse: Transitions hyperfines de l'iode moléculaire au service de la métrologie des fréquences optiques (Paris XI - 1988).

[11] J.-M. Chartier, S. Fredin-Picard and L. Robertsson, Optics Comm. 74 (1989) 87.

[12] S. Fredin-Picard, Rapport BIPM-90/5.

[13] M. Kroll, Phys. Rev. Lett. 23 (1969) 631.

[14] P.R. Bunker and G.R. Hanes, Chem. Phys. Lett. 28 (1974) 377.

[15] L.A. Hackel, K.H. Casleton, S.G. Kukolich and S. Ezekiel, Phys. Rev. Lett. 35 (1975) 568.

[16] G.R. Hanes, J. Lapierre, P.R. Bunker and K.C. Shotton, J. Mol. Spec. 39 (1969) 506.

[17] S. Gerstenkorn and P. Luc, Atlas du spectre d'absorption de la molécule d'iode 14800-20000 cm^{-1}. Complement: Identification des transitions du système (B-X), (CNRS 1985).

[18] H.J. Foth and F. Spieweck, Chem. Phys. Lett. 65 (1979) 347.

[19] A. Razet, Y. Millerioux and P. Juncar, submitted to Metrologia.

[20] J.-M. Chartier, J.L. Hall and M. Gläser, Proc. CPEM-86, ISBN 86 CH2267-3 (1986) p.323.

[21] M.D. Rayman and M. Winters, JILA, Boulder, Colorado, USA, private communication.

[22] U. Brand and J. Helmcke, Frequency Standards and Metrology, ed. A. De Marchi (Springer-Verlag, 1988) p.455.

[23] H. Simonsen and O. Poulsen, Appl. Phys. B 50 (1990) 7.

[24] A. Yokozeki and J.S. Munter, J. Chem. Phys. 72 (1980) 3796.

[25] M. Gläser, PTB Bericht Opt-25 (1987) 173.

Appendix I

Stationary Perturbation Theory

The quantum study of time-independent physical systems is based on the characteristic equation of the Hamiltonian operator, the *Schrödinger equation*. This equation is of the form

$$\hat{H}_0 | \Phi_i^0 >= E_i^0 | \Phi_i^0 > \qquad (AII.1)$$

where $\hat{H}_0$ is the Hamiltonian operator, Φ^0_i is an eigenfunction of the operator and E^0_i is the eigenvalue of Φ^0_i. The normalization of these eigenfunctions can be written as

$$< \Phi_j^0 | \Phi_i^0 >= \delta_{ji} \qquad (AII.2)$$

When the difference between $\hat{H}$, the Hamiltonian of a particular system, and $\hat{H}_0$ (for which the eigenfunctions and eigenvalues are known) is sufficiently small, we can apply *the perturbation method*. This allows us to write $\hat{H}$ as

$$\hat{H} = \hat{H}_0 + \hat{W} \qquad (AII.3)$$

where $\hat{W}$ is the perturbation operator. As $\hat{W}$ is small, we can generally express $\hat{W}$ as

$$\hat{W} = \lambda \hat{W}' \qquad (AII.4)$$

where λ represents a small number ($\lambda << 1$). The Schrödinger equation of the system can therefore be written as

$$\hat{H} | \Phi_i >= E_i | \Phi_i > \qquad (AII.5)$$

where

$$E_i = \sum_{p=0}^{\infty} \lambda^p E_i^p \qquad (AII.6)$$

and

$$|\Phi_i>=\sum_{p=0}^{\infty}\lambda^p|\Phi_i^p> \qquad (AII.7)$$

By using Eqs (AII.5-7), and after identification of terms with equal power of λ^P, we obtain the following expressions for the zeroth order and first order perturbation theory:

Zeroth Order Perturbation

$$\hat{H}_0|\Phi_i^0>=E_i^0|\Phi_i^0>$$

$$E_i^0=<\Phi_i^0|\hat{H}_0|\Phi_i^0> \qquad (AII.8)$$

First Order Perturbation

$$(\hat{H}_0-E_i^0)|\Phi_i^1>+(\hat{W}'-E_i^1)|\Phi_i^0>=0 \qquad (AII.9)$$

To obtain the E^1_i, we expand the wavefunction Φ^1_i in the basis set Φ^0_j, that is

$$|\Phi_i^1>=\sum_j a_j|\Phi_j^0> \qquad (AII.10)$$

Substituting Eq.(AII.10) into Eq.(AII.9) we obtain

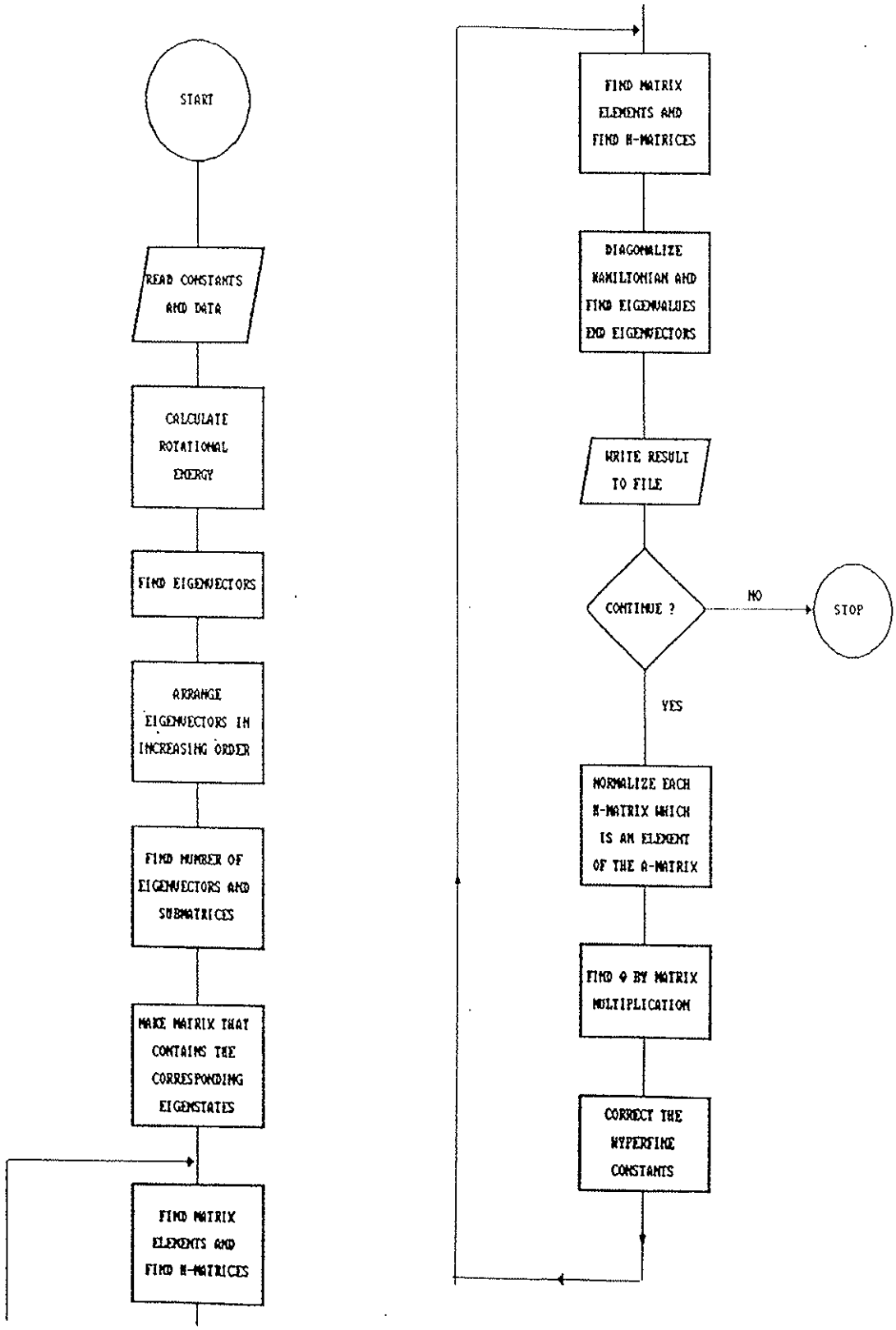
$$(\hat{H}_0-E_i^0)\sum_j a_j|\Phi_j^0>+(\hat{W}'-E_i^1)|\Phi_i^0>=0 \qquad (AII.11)$$

If we normalize Eq.(AII.11) by "multiplication" by $<\Phi^0_i|$ we finally obtain

$$E_i^1=\frac{<\Phi_i^0|\hat{W}'|\Phi_i^0>}{<\Phi_i^0|\Phi_i^0>} \qquad (AII.12)$$

that is, the first order energy contribution.

Appendix II


START
READ CONSTANTS AND DATA
CALCULATE ROTATIONAL ENERGY
FIND EIGENVECTORS
ARRANGE EIGENVECTORS IN INCREASING ORDER
FIND NUMBER OF EIGENVECTORS AND SUBMATRICES
MAKE MATRIX THAT CONTAINS THE CORRESPONDING EIGENSTATES
FIND MATRIX ELEMENTS AND FIND H-MATRICES
FIND MATRIX ELEMENTS AND FIND H-MATRICES
DIAGONALIZE HAMILTONIAN AND FIND EIGENVALUES AND EIGENVECTORS
WRITE RESULT TO FILE
CONTINUE ?
NO
STOP
YES
NORMALIZE EACH H-MATRIX WHICH IS AN ELEMENT OF THE A-MATRIX
FIND Φ BY MATRIX MULTIPLICATION
CORRECT THE HYPERFINE CONSTANTS

```
Program CalcConst;

(*     This programme fits hyperfine constants to      *)
(* a set of observed hyperfine component frequencies *)

Uses Crt,Dos,Turbo3,Printer;

LABEL    20,30,40,50,60;

CONST    convert = 29979.2458;

         IOerr : boolean = false;
         TNArraySize = 8;

TYPE     oo        = RECORD
                       lam, vikt, wi     : double;
                       nlin              : integer;
                     END;

         pp        = RECORD
                       F, JP, IP, JB, IB : integer;
                     END;

         qq        = RECORD
                       QEQQ, QCKON, QAKON, QDKON, QDelta : double;
                     END;

         rr        = RECORD
                       ko     : double;
                       fix    : char;
                     END;

         mattyp      = array [1..15,1..8,1..8] of qq;
         atyp        = array [1..63] of pp;
         xtyp        = array [1..8] of rr;
         otyp        = array [1..21] of oo;
         btyp        = array [1..63] of double;
         ftyp        = array [1..21] of double;
         ntyp        = array [1..21] of integer;
         ctyp        = array [1..15] of integer;
         dtyp        = array [1..15,1..8,1..8] of pp;
         antyp       = array [1..5,1..8,1..8] of double;
         etyp        = array [1..21,1..5] of double;
         tetyp       = array [1..5,1..21] of double;
         mtyp        = array [1..21,1..21] of double;
         sqvec       = array [1..5] of double;
         ytyp        = array [1..21,1..1] of double;
         ztyp        = array [1..1,1..21] of double;
         corrtyp     = array [1..5,1..1] of double;
         filtyp      = string[12];
         TNvector    = array[1..TNArraySize] of double;
         TNmatrix    = array[1..TNArraySize] of TNvector;
         TNIntVector = array[1..TNArraySize] of integer;

VAR      M                                    : mattyp;
         till, fran,
         EQQ, CKON ,DKON, AKON,
         tal, RMS1, sdev                      : double;
         Jbis, vprim, vbis, NofLines,
         NofConst, noll, NofCom,
         k1, n1, m1, x1, y1, z1, w1, w2,
         itter, jinit, iinit,
         antal, antmat, aja, Dimen, rak       : integer;
         ATAI, Corr, Vec, mille, hqp          : TNmatrix;
         TETA, Bii, sdevcon                   : corrtyp;
         h                                    : antyp;
         slask, egen, etikett, etsort         : atyp;
         eget, egetu, egetl                   : btyp;
         nostat                               : ctyp;
         punkt                                : dtyp;
         hkonst, hsort                        : etyp;
         linje, lsort, wiw                    : ftyp;
         nin                                  : ntyp;
         obslin                               : otyp;
```

```
        hk                                  : xtyp;
        obscalc, vpt                        : ytyp;
        vp                                  : ztyp;
        bok, state, ja                      : char;
        F1                                  : text;
        Fiol                                : filtyp;

{$I a:SUBER.pas }
{$I a:MATX.pas }
{$I a:ROTER.pas }
{$I a:JACOB.PAS }
{$I a:POP.pas }

(* SUBER  : Subroutines for basic calculations,      *)
(*          arrangements and the hamiltonian matrix  *)
(* ROTER  : Calculate rotational energy differences  *)
(* JACOB  : Diagonalize, calculate eigen vectors     *)
(*          and eigen values                         *)
(* POP    : Perform the diagonalization              *)
(* MATX   : Calculation with matrices                *)

BEGIN

(* --------- determine initial conditions ---------- *)

    SetNumber(vprim,vbis,bok,Jbis,aja,hk,NofLines,NofCom,
              NofConst,obslin,noll);
    IF aja=1 THEN GOTO 30;

(* ---------- Initialize for itteration ------------ *)

    Write('Give outfile for itteration results: ');
    Readln(fiol);
    Assign(F1,fiol);
    Rewrite(F1);
    writeln(F1,bok:1,'(',Jbis:3,') ',vprim:2,' - ',vbis:2);
    writeln(F1);

    itter:=0;

    FOR n1:=1 TO NofCom DO
    WITH obslin[n1] DO
    BEGIN
      nin[n1]:=nlin;
      wiw[n1]:=wi;
    END;

(* ---------- Set vectors = 0 ---------------------- *)

    FOR n1:=1 TO 5 DO
    BEGIN
      TETA[n1,1]:=0E+00;
      Bii[n1,1]:=0E+00;
    END;

    FOR n1:=1 TO TNArraySize DO
    FOR m1:=1 TO TNArraySize DO
    BEGIN
      ATAI[n1,m1]:=0E+00;
      Corr[n1,m1]:=0E+00;
    END;

(*  Calculate energy differences between rotational  *)
(* ------------ levels with delta-J=+-2 -----------  *)
(* First the lower state is executed, then the upper *)

20: state :='L';
40: jinit:=Jbis-2;
```

```
    IF state='U' THEN
    BEGIN
      IF bok='R' THEN jinit:=jinit+1
      ELSE jinit:=jinit-1;
      till:=Diffup(vprim,jinit+4)-Diffup(vprim,jinit+2);
      fran:=Diffup(vprim,jinit+2)-Diffup(vprim,jinit);
    END
    ELSE
    BEGIN
      till:=Difflow(vbis,Jbis+2)-Difflow(vbis,Jbis);
      fran:=Difflow(vbis,Jbis)-Difflow(vbis,Jbis-2);
    END;

(* ---------------- Find eigen vectors -------------- *)
(* ----------------- F=(J-I)...(J+I) ----------------- *)

    iinit:=0;
    IF Odd(Jbis) THEN iinit:=1;

    FindEigVec(jinit,iinit,slask,antal);

(* ----- Range eigen vectors F,J,I consecutif ------ *)

    RangeEig(slask,egen,etikett,antal,jinit,iinit,state);

(* --------- Counte number of eigen vectors -------- *)
(* ---------- Find number of sub matrices ---------- *)

    CalcEig(egen,nostat,antal,antmat);

(* ---- Make matrix containing the eigen states ---- *)

    PunktMat(punkt,egen,nostat,antmat);

(* -------------- Calculate matrix ----------------- *)

    MatElement(antmat,nostat,punkt,M,till,fran,state,hk);

(* ---------------- Set constants ------------------ *)

    IF state='U' THEN
    BEGIN
      EQQ:=hk[1].ko;
      CKON:=hk[2].ko;
      DKON:=hk[3].ko;
      AKON:=hk[4].ko;
    END
    ELSE
    BEGIN
      EQQ:=hk[5].ko;
      CKON:=hk[6].ko;
      DKON:=hk[7].ko;
      AKON:=hk[8].ko;
    END;

(* -------- Diagonalize and find eigenvalues -------- *)

    x1:=0;
    y1:=0;
    rak:=0;

    FOR n1:=1 TO antmat DO
    BEGIN
      Dimen:=nostat[n1];
      FOR m1:=1 TO Dimen DO
      BEGIN
        FOR k1:=1 TO Dimen DO
        WITH M[n1,m1,k1] DO
        BEGIN
          tal:=EQQ*QEQQ+CKON*QCKON+AKON*QAKON+DKON*QDKON+QDelta;
          mille[m1,k1]:=tal;
          IF state='U' THEN
          BEGIN
            h[1,m1,k1]:=QEQQ;
```

```
            h[2,m1,k1]:=QCKON;
            h[3,m1,k1]:=QDKON;
            h[4,m1,k1]:=QAKON;
          END
          ELSE
          h[5,m1,k1]:=-QEQQ;
        END;
      END;

      diag(eget,Dimen,rak,Vec,mille);

      z1:=0;
      FOR m1:=1 TO Dimen DO
      BEGIN
        z1:=z1+1;
        x1:=x1+1;

(* ----- Norma normalizes the correction terms------ *)

        WITH egen[x1] DO
        IF JP=(jinit+2) THEN
        BEGIN
          y1:=y1+1;
          FOR k1:=1 TO Dimen DO
          vp[1,k1]:=Vec[z1,k1];
          FOR k1:=1 TO Dimen DO
          vpt[k1,1]:=vp[1,k1];

          IF state='U' THEN
          FOR k1:=1 TO 4 DO
          BEGIN
            FOR w1:=1 TO Dimen DO
            FOR w2:=1 TO Dimen DO
            hqp[w1,w2]:=h[k1,w1,w2];
            IF hk[k1].fix='V' THEN
            hkonst[y1,k1]:=Norma(vp,vpt,hqp,Dimen)
            ELSE hkonst[y1,k1]:=0E+00;
          END
          ELSE
          BEGIN
            FOR w1:=1 TO Dimen DO
            FOR w2:=1 TO Dimen DO
            hqp[w1,w2]:=h[5,w1,w2];
            IF hk[5].fix='V' THEN
            hkonst[y1,5]:=Norma(vp,vpt,hqp,Dimen)
            ELSE hkonst[y1,5]:=0E+00;

(*    hkonst contain initial correction matrix A     *)

          END;
        END;
      END;
    END;

    IF state<>'L' THEN egetu:=eget
    ELSE egetl:=eget;

    writeln('Do not worry, I am working ! ');
    writeln;

    IF state<>'L' THEN GOTO 50;
    state:='U';
    GOTO 40;

(* ---------------- Calculate lines ---------------- *)

50: differ(egen,egetu,egetl,linje,nostat,jinit,antmat);

(* --- Arrange lines in raising frequency order ---- *)

    hsort:=hkonst;
    lsort:=linje;
    etsort:=etikett;
```

```
    SORT(lsort,etsort,hsort,NofLines);

    IF noll=0 THEN GOTO 60;

(* ------ Set one component to zero if needed ------ *)

    tal:=lsort[noll];
    FOR n1:=1 TO NofLines DO
    lsort[n1]:=lsort[n1]-tal;

    FOR m1:=1 TO 5 DO
    IF hk[m1].fix='V' THEN
    BEGIN
      tal:=hsort[noll,m1];
      FOR n1:=1 TO NofLines DO
      hsort[n1,m1]:=hsort[n1,m1]-tal;
    END;

(* -------------- Calculate obs-calc -------------- *)

60: Calcomc(obslin,lsort,NofCom,obscalc);

(* -------- Calculate standard deviations ---------- *)

    Felcalc(obscalc,Bii,NofCom,NofConst,RMS1,sdevcon,sdev,wiw);

(* ------ Write on screen and onto text file ------- *)

    writeln('Itteration = ',itter:5);
    writeln(F1,'Itteration = ',itter:5);

    writeln('RMS         : ',RMS1:12:6);
    writeln(F1,'RMS         : ',RMS1:12:6);

    writeln('sdev        : ',sdev:12:6);
    writeln(F1,'sdev        : ',sdev:12:6);
    writeln;
    writeln(F1);

    write('   no  correction   new const       ');
    writeln('sdev   status');
    write(F1,'   no  correction   new const       ');
    writeln(F1,'sdev   status');

    FOR n1:=1 TO 5 DO
    WITH hk[n1] DO
    BEGIN
      writeln(n1:5,TETA[n1,1]:12:6,' ',ko:12:6,
              sdevcon[n1,1]:12:6,' ',fix:3);
      writeln(F1,n1:5,TETA[n1,1]:12:6,' ',ko:12:6,
              sdevcon[n1,1]:12:6,' ',fix:3);
    END;

    FOR n1:=6 TO 8 DO
    WITH hk[n1] DO
    BEGIN
      writeln(n1:5,'              ',ko:12:6);
      writeln(F1,n1:5,'              ',ko:12:6);
    END;
    writeln;
    writeln(F1);
    write('Push RETURN to continue listing !');
    readln;
    writeln;

    write(' no comp   F    I    J');
    writeln('   observed     calculated     obs-calc   weight');
    write(F1,' no comp   F    I    J');
    write(F1,'   observed     calculated     obs-calc   ');
    writeln(F1,'weight');
    FOR n1:=1 TO NofCom DO
    WITH etsort[n1] DO
    WITH obslin[n1] DO
    BEGIN
```

```
      m1:=nlin;
      writeln(n1:4,m1:4,F:4,IP:4,JP:4,lam:12:6,' ',
              lsort[m1]:12:6,' ',obscalc[n1,1]:12:6,vikt:9:4);
      writeln(F1,n1:4,m1:4,F:4,IP:4,JP:4,lam:12:6,' ',
              lsort[m1]:12:6,' ',obscalc[n1,1]:12:6,vikt:9:4);
    END;

    writeln(F1);
    writeln(F1);

    writeln;
    writeln('Do you want to continue ? Y/N : ');
    readln(ja);
    ja:=UpCase(ja);
    IF ja='N' THEN GOTO 30;

(* Calculate the correction TETA to initial constants *)

    Matris(obscalc,hsort,TETA,Bii,NofCom,nin,wiw,ATAI);

    itter:=itter+1;

(* -------- Add correction to old constants -------- *)

    FOR n1:=1 TO 5 DO
    WITH hk[n1] DO
    IF fix='V' THEN
    ko:=ko+TETA[n1,1];

    GOTO 20;

(* ------ Calculate correlation coefficients ------- *)

30: Correlate(ATAI,Corr);
    writeln;
    writeln('Correlation coefficients: ');
    writeln;
    writeln(F1);
    writeln(F1,'Correlation coefficients: ');
    writeln(F1);
    FOR n1:=1 TO 5 DO
    BEGIN
      FOR m1:=1 TO 5 DO
      BEGIN
        write(Corr[n1,m1]:11:7);
        write(F1,Corr[n1,m1]:11:7);
      END;
      writeln;
      writeln(F1);
    END;

    Close(F1);
END.
```

```
(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
Function di0dj0(a,b,c:integer):real;
(* 3j: Delta-I=0 Delta-J=0 *)
CONST    I1 = 2.5;
VAR      F, I, J, d, frad, g, h    : real;
BEGIN
    F:=a;
    J:=b;
    I:=c;
    frad:=F*(F+1)-I*(I+1)-J*(J+1);
    d:=-3/(16*(2*J+3)*(2*J-1)*I1*(2*I1-1));
    g:=(F+I-J+1)*(F+J-I)*(F+I+J+2)*(I+J-F+1);
    g:=g*((2*I1+1)*(2*I1+1)-(I+1)*(I+1))/
        ((2*I+1)*(2*I+3));
    h:=(F+I-J)*(F+J-I+1)*(F+I+J+1)*(I+J-F);
    h:=h*((2*I1+1)*(2*I1+1)-I*I)/(4*I*I-1);
    di0dj0:=d*(frad*frad+2*frad+g+h-
                16*I1*(I1+1)*J*(J+1)/3);
END; (* Function di0dj0 *)
(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
Function dim2dj0(a,b,c:integer):real;
(* 3j: Delta-I=-2 Delta-J=0 *)
CONST    I1 = 2.5;
VAR      F, I, J, p, q, x, y, z    : real;
BEGIN
    F:=a;
    J:=b;
    I:=c;
    x:=3*SQRT((2*I1+1)*(2*I1+1)-I*I)
        *SQRT((2*I1+1)*(2*I1+1)-(I-1)*(I-1));
    y:=-16*(2*J+3)*(2*J-1)*I1*(2*I1-1)*SQRT((4*I*I-1)
        *(2*I-3)*(2*I-1));
    z:=SQRT((F+I-J-1)*(F+I-J)*(F+J-I+1));
    p:=SQRT((F+J-I+2)*(F+I+J)*(F+I+J+1));
    q:=SQRT((I+J-F-1)*(I+J-F));
    dim2dj0:=(x/y)*z*p*q;
END; (* Function dim2dj0 *)
(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
Function di0dj2(a,b,c:integer):real;
(* 3j: Delta-I=0 Delta-J=2 *)
CONST    I1 = 2.5;
VAR      x, y, z, p, q, F, I, J    : real;
BEGIN
    F:=a;
    J:=b;
    I:=c;
    x:=3*(3*I*(I+1)-3-4*I1*(I1+1));
    y:=16*(2*J+3)*(2*I+3)*(2*I1-1)*(2*I-1)*I1
        *SQRT((2*J+1)*(2*J+5));
    z:=SQRT((F+I-J-1)*(F+I-J)*(F+J-I+1));
    p:=SQRT((F+J-I+2)*(F+I+J+2)*(F+I+J+3));
    q:=SQRT((I+J-F+1)*(I+J-F+2));
    di0dj2:=(x/y)*z*p*q;
END; (* Function di0dj2 *)
(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
```

```
Function di2dj2(a,b,c:integer):real;
(* 3j: Delta-I=2 Delta-J=2 *)
CONST    I1 = 2.5;
VAR      F, I, J, x, y, p, q      : real;
BEGIN
    F:=a;
    J:=b;
    I:=c;
    x:=3*SQRT((2*I1+I+2)*(2*I1+I+3)
        *(2*I1-I-1)*(2*I1-I));
    y:=32*(2*J+3)*(2*I+3)*(2*I1-1)*I1
        *SQRT((2*J+1)*(2*J+5)*(2*I+1)*(2*I+5));
    p:=SQRT((F+I+J+5)*(F+J+I+4)*(F+I+J+2)*(F+I+J+3));
    q:=SQRT((I+J-F+1)*(I+J-F+2)*(I+J-F+3)*(I+J-F+4));
    di2dj2:=(x/y)*p*q;
END; (* Function di2dj2 *)
(* XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
Function dim2dj2(a,b,c:integer):real;
(* 3j: Delta-I=-2 Delta-J=2 *)
CONST    I1 = 2.5;
VAR      x, y, z, p, q, F, I, J          : real;
BEGIN
    F:=a;
    J:=b;
    I:=c;
    x:=3*SQRT((2*I1-I+2)*(2*I1-I+1)
        *(2*I1+I+1)*(2*I1+I));
    y:=32*(2*J+3)*(2*I-1)*(2*I1-1)*I1
        *SQRT((2*J+1)*(2*J+5)*(2*I+1)*(2*I-3));
    p:=SQRT((F+I-J)*(F+I-J-1)*(F+I-J-2)*(F+I-J-3));
    q:=SQRT((F+J-I+1)*(F+J-I+2)*(F+J-I+3)*(F+J-I+4));
    dim2dj2:=(x/y)*p*q;
END; (* Function dim2dj2 *)
(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
Function NioJ0(a:integer):real;
(* 9j: Delta-I=0 *)
CONST    I1 = 2.5;
VAR      I, S    : real;
BEGIN
    I:=a;
    S:=(4*I1*(I1+1)+I*(I+1))/
        (I1*(I1+1)*(2*I1+1)*SQRT(120));
    NioJ0:=S*SQRT((I*(I+1))/((2*I-1)*(2*I+1)*(2*I+3)));
END; (* Function NioJ0 *)
(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
Function NioJ2(a:integer):real;
(* 9j: Delta-I=2 *)
CONST    I1 = 2.5;
VAR      I, S    : real;
```

```
BEGIN
    I:=a;
    S:=SQRT(((2*I1+1)*(2*I1+1)-(I-1)*(I-1))*(I-1)*I/5);
    s:=S*SQRT(((2*I1+1)*(2*I1+1)-I*I)/
         ((2*I-3)*(2*I-1)*(2*I+1)));
    NioJ2:=-S/(2*I1*(2*I1+1)*(2*I1+2));
END; (* Function NioJ2 *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Function Cfunk(f1,j1,i1:integer):real;

(* HFS-SR *)

VAR      C, X, F, I, J        : real;

BEGIN
    F:=f1;
    J:=j1;
    I:=i1;
    X:=F*(F+1)-I*(I+1)-J*(J+1);
    Cfunk:=X/2;
END; (* Function Cfunk *)

(* XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Function Afunk(a:integer):real;

(* HFS-SSS *)

CONST    I1 = 2.5;

VAR      X, I         : real;

BEGIN
    I:=a;
    X:=I*(I+1)-2*I1*(I1+1);
    Afunk:=X/2;
END; (* Function Afunk *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Function Dfunk(f1,j1,a,b:integer):real;

(* HFS-TSS *)

CONST    I1 = 2.5;

VAR      C, X, F, IP, IB, J    : real;
         fas, s                : integer;

BEGIN
    fas:=1;
    F:=f1;
    J:=j1;
    IP:=a;
    IB:=b;
    s:=TRUNC(0.02+J+IP+F);
    IF Odd(s) THEN fas:=-1;
    X:=SQRT(30*(2*IB+1)*(2*IP+1))*(2*J+1)
                                 *I1*(I1+1)*(2*I1+1);
    X:=X*SQRT((J*(J+1))/((2*J+3)*(2*J+1)*(2*J-1)));
    Dfunk:=X*fas;
END; (* Function Dfunk *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Function sexj0(A1,B1,C1:integer):real;

(* 6j:  a b c  *)
(*      2 c b  *)

LABEL   10;
```

```
VAR     fas, s                           : integer;
        a, b, c, x1, y1, z1, X, se       : real;

BEGIN
    se:=0.0;
    a:=A1;
    b:=B1;
    c:=C1;
    s:=TRUNC(a+b+c+0.02);
    X:=b*(b+1)+c*(c+1)-a*(a+1);
    fas:=1;
    x1:=3*X*(X-1)-4*b*(b+1)*c*(c+1);
    IF x1=0.0 THEN GOTO 10;
    y1:=(2*b+3)*(2*b+2)*(2*b+1)*2*b*(2*b-1);
    z1:=(2*c+3)*(2*c+2)*(2*c+1)*2*c*(2*c-1);
    IF Odd(s) THEN fas:=-1;
    se:=x1/SQRT(y1*z1);
10: sexj0:=2*fas*se;
END; (* Function sexj0 *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Function sexj2(A1,B1,C1:integer):real;

(* 6j:  a  b  c   *)
(*      2 c-2 b   *)

VAR     fas, s1                          : integer;
        a, b, c, x1, y1, z1, X, s        : real;

BEGIN
    a:=A1;
    b:=B1;
    c:=C1;
    s1:=TRUNC(a+b+c+0.1);
    s:=a+b+c;
    fas:=1;
    x1:=6*s*(s+1)*(s-2*a-1)*(s-2*a);
    y1:=(s-2*b-1)*(s-2*b)/((2*b-1)
        *2*b*(2*b+1)*(2*b+2)*(2*b+3));
    z1:=(s-2*c+1)*(s-2*c+2)/((2*c-3)
        *(2*c-2)*(2*c-1)*2*c*(2*c+1));
    IF Odd(s1) THEN fas:=-1;
    sexj2:=fas*SQRT(x1*y1*z1);
END; (* Function sexj2 *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure SetNumber(VAR vprim,vbis        : integer;
                    VAR bok               : char;
                    VAR Jbis,aja          : integer;
                    VAR hk                : xtyp;
                    VAR NofLines,NofCom,
                        NofConst          : integer;
                    VAR obslin            : otyp;
                    VAR noll              : integer);

(* Infile : 1) Notation of transition                   *)
(*          2) Number of hyperfine constants            *)
(*          3) V/F Constants for upper state            *)
(*             eqQ C D=d A=delta                        *)
(*          4) V/F Constants for lower state            *)
(*             eqQ C D=d A=delta                        *)
(*             V - vary  F - fix                        *)
(*          5) Total number of possible lines           *)
(*          6) Number of lines observed                 *)
(*          7) Component number, frequency and          *)
(*             weight of lines in MHz                   *)
(*          8) Component number set to 0 frequency      *)

TYPE   filtyp = string[12];
```

```
VAR     fiol            : filtyp;
        Fl              : text;
        antal, nl, kl   : integer;
        tot             : double;

BEGIN
    aja:=0;
    Write('Give name of file containing the ');
    Write('hyperfine constants : ');
    Readln(fiol);
    Assign(Fl,fiol);
    Reset(Fl);

    read(Fl,bok);
    bok:=UpCase(bok);
    IF (bok<>'R') AND (bok<>'P') AND (bok<>'Q')
    THEN aja:=1;
    IF bok='Q' THEN
    BEGIN
      writeln('Forbidden transition !!!');
      aja:=1;
    END;

    read(Fl,Jbis);
    readln(Fl,vprim,vbis);

    NofConst:=0;

    readln(Fl,antal);
    FOR nl:=1 TO antal DO
    BEGIN
      WITH hk[nl] DO
      readln(Fl,fix,ko);
      IF hk[nl].fix='V' THEN
      NofConst:=NofConst+1;
      IF hk[nl].fix='F' THEN
    END;

 (* ---  NofLines - number of possible components --- *)
 (* ---  NofCom   - number of observed components --- *)

    readln(Fl,NofLines);
    readln(Fl,NofCom);

 (* Read number, frequ. in MHz and standard deviation *)

    FOR nl:=1 TO NofCom DO
    WITH obslin[nl] DO
    readln(Fl,nlin,lam,vikt);

    tot:=0;                        (* Calculate weights *)
    FOR nl:=1 TO NofCom DO
    WITH obslin[nl] DO
    tot:=tot+1/vikt/vikt;

    tot:=NofCom/tot;
    FOR nl:=1 TO NofCom DO
    WITH obslin[nl] DO
    wi:=tot/vikt/vikt;

 (*   noll - component number set to zero frequency   *)

    readln(Fl,noll);
    Close(Fl);
END; (* Procedure SetNumber *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure FindEigVec(jinit, iinit  :integer;
                  VAR slask        :atyp;
                  VAR antal        :integer);

 (* Find appropriate eigen-vectors *)
```

```
VAR     sum, n1, JC, m1, IC, ftal    : integer;

BEGIN
    sum:=0;
    FOR n1:=0 TO 2 DO
    BEGIN
      JC:=jinit+2*n1;
      FOR m1:=0 TO 2 DO
      BEGIN
        IC:=iinit+2*m1;
        FOR ftal:=(JC-IC) TO (JC+IC) DO
        BEGIN
          sum:=sum+1;
          WITH slask[sum] DO
          BEGIN
            F:=ftal;
            JP:=JC;
            IP:=IC;
          END;
        END;
      END;
    END;

(* ---------- antal = dimension of matrix ---------- *)

    antal:=sum;
END; (* Procedure FindEigVec *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure RangeEig(VAR slask,egen,etikett   : atyp;
                   antal,jinit,iinit        : integer;
                   state                    : char);

(* Arrange eigen-vectors by increasing F, I and J *)

VAR    sum, n1, m1, k1     : integer;

BEGIN
    sum:=0;
    k1:=0;
    FOR m1:=(jinit-(iinit+4)) TO (jinit+4+(iinit+4)) DO
    FOR n1:=1 TO antal DO
    IF slask[n1].F=m1 THEN
    BEGIN
      sum:=sum+1;
      egen[sum]:=slask[n1];

(* ------ Only eigenstates for J' in etikett ------- *)

      IF state='U' THEN
      IF egen[sum].JP=jinit+2 THEN
      BEGIN
        k1:=k1+1;
        etikett[k1]:=egen[sum];
      END;
    END;
END;

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure CalcEig(egen           :atyp;
              VAR nostat         :ctyp;
              VAR antal, antmat  :integer);

(* Find number of sub-matrices and dimensions *)

LABEL 1;

VAR   m1, sum, k1, n1   : integer;
```

```
BEGIN
    m1:=egen[1].F;
    sum:=0;
    k1:=1;

    FOR n1:=2 TO antal DO
    WITH egen[n1] DO
    BEGIN
      sum:=sum+1;

      IF F>m1 THEN
      BEGIN
        nostat[k1]:=sum;
        antmat:=k1;
        k1:=k1+1;
        sum:=0;
        m1:=F;
      END;

      IF n1=antal THEN
      BEGIN
        nostat[k1]:=sum+1;
        antmat:=k1;
        GOTO 1;
      END;
    END;

(* -------- antmat = number of sub-matrices -------- *)
(* ------- nostat = dimension of each matrix ------- *)

1:  antmat:=k1;
END; (* Procedure CalcEig *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure PunktMat(VAR punkt   :dtyp;
                       egen    :atyp;
                       nostat  :ctyp;
                       antmat  :integer);

(* Create matrix of eigen-vectors *)

VAR     n1, m1, k1, sum     : integer;

BEGIN
    sum:=0;
    FOR n1:=1 TO antmat DO
    BEGIN
      FOR m1:=1 TO nostat[n1] DO
      FOR k1:=1 TO nostat[n1] DO
      BEGIN
        punkt[n1, m1, k1].F:=egen[sum+1].F;
        punkt[n1, m1, k1].JP:=egen[sum+k1].JP;
        punkt[n1, m1, k1].IP:=egen[sum+k1].IP;
        punkt[n1, m1, k1].JB:=egen[sum+m1].JP;
        punkt[n1, m1, k1].IB:=egen[sum+m1].IP;
      END;
      sum:=sum+nostat[n1];
    END;
END; (* Procedure PunktMat *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure MatElement(antmat       : integer;
                     nostat       : ctyp;
                     punkt        : dtyp;
                 VAR M            : mattyp;
                     till,fran    : double;
                     state        : char;
                     hk           : xtyp);

(* Calculate raw matrix elements with the constants  *)
(* --- Calculate the hyperfine correction matrix --- *)
```

```
VAR     n1, m1, k1, ms, ns              : integer;
        C, A, D, p1, p2, p3, Delta      : double;

BEGIN

(*  Calculate half the matrix and symmetrize later.  *)

    FOR n1:=1 TO antmat DO
    BEGIN
      ns:=1;
      ms:=1;
      FOR m1:=ms TO nostat[n1] DO
      BEGIN
        FOR k1:=ns TO nostat[n1] DO
        WITH punkt[n1,m1,k1] DO
        BEGIN
          C:=0.0;  A:=0.0;  D:=0.0;
          p1:=0.0; p2:=0.0; p3:=0.0;
          Delta:=0.0;

          IF JP-JB=0 THEN
          BEGIN
            p2:=0.0;
            p3:=0.0;
            IF IP-IB=0 THEN
            BEGIN
              C:=Cfunk(F,JP,IP);
              A:=Afunk(IP);
              p1:=di0dj0(F,JB,IB);
              p2:=sexj0(F,JB,IB);
              p3:=NioJ0(IP);

(* ----- Delta = rotational energy difference ------ *)

              IF JP=(jinit+2) THEN Delta:=0
              ELSE IF JP=(jinit+4) THEN Delta:=till
              ELSE IF JP=jinit THEN Delta:=-fran;
              Delta:=Delta*convert;
            END

            ELSE IF IP-IB=-2 THEN
            BEGIN
              p1:=dim2dj0(F,JB,IB);
              p2:=sexj2(F,JB,IB);
              p3:=NioJ2(IB);
            END

            ELSE IF IP-IB=2 THEN
            BEGIN
              p1:=dim2dj0(F,JP,IP);
              p2:=sexj2(F,JP,IP);
              p3:=NioJ2(IP);
            END
            ELSE IF ABS(IP-IB)>2.05 THEN
            p1:=0;

            D:=Dfunk(F,JP,IP,IB)*p2*p3;
          END

          ELSE IF JP-JB=2 THEN
          BEGIN
            IF IP-IB=0 THEN
            p1:=di0dj2(F,JB,IB)
            ELSE IF IP-IB=2 THEN
            p1:=di2dj2(F,JB,IB)
            ELSE IF IP-IB=-2 THEN
            p1:=dim2dj2(F,JB,IB)
            ELSE IF ABS(IP-IB)>2.05 THEN
            p1:=0;
          END

          ELSE p1:=0;
```

```
(*    p1 is set to 0 - matrix is symmetrized later    *)
(* ----------- using the upper triangle -----------   *)

            M[n1,m1,k1].QEQQ:=p1;
            M[n1,m1,k1].QCKON:=C;
            M[n1,m1,k1].QDelta:=Delta;
            M[n1,m1,k1].QAKON:=A;
            M[n1,m1,k1].QDKON:=D;

(* ------------------- symmetrize ------------------ *)

            M[n1,k1,m1]:=M[n1,m1,k1];
          END;
          ms:=ms+1;
          ns:=ns+1;
        END;
      END;
END; (* Procedure MatElement *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

procedure differ(egen              : atyp;
                 egetu,egetl       : btyp;
             VAR linje             : ftyp;
                 nostat            : ctyp;
                 jinit,antmat      : integer);

(* Calculate differences between energy levels *)

VAR  n1, m1, x1, y1   : integer;
     diff             : double;

BEGIN
  y1:=0;
  x1:=0;

(* ----------- Write only central lines ----------- *)

  FOR n1:=1 TO antmat DO
  BEGIN
    FOR m1:=1 TO nostat[n1] DO
    BEGIN
      x1:=x1+1;
      WITH egen[x1] DO
      IF JP=(jinit+2) THEN
      BEGIN
        y1:=y1+1;
        linje[y1]:=egetu[x1]-egetl[x1];
        diff:=linje[y1];
      END;
    END;
  END;
END; (* Procedure differ *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure SORT(VAR differ : ftyp;
               VAR etikett: atyp;
               VAR hsort  : etyp;
                   x3     : integer);

(* Bubbel arrangement *)

VAR      i, j      : integer;
         m         : double;
         a         : atyp;
         b         : etyp;

BEGIN
    REPEAT
      m:=differ[1];
      FOR i:=1 TO x3-1 DO
      IF differ[i+1]<differ[i] THEN
      BEGIN
```

```
        m:=differ[i+1];
        differ[i+1]:=differ[i];
        differ[i]:=m;

        a[i]:=etikett[i+1];
        etikett[i+1]:=etikett[i];
        etikett[i]:=a[i];

        FOR j:=1 TO 5 DO
        BEGIN
          b[i,j]:=hsort[i+1,j];
          hsort[i+1,j]:=hsort[i,j];
          hsort[i,j]:=b[i,j];
        END;
      END;
    UNTIL m=differ[1];
END; (* SORT *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure Calcomc(VAR obslin        : otyp;
                  VAR lsort         : ftyp;
                      NofCom        : integer;
                  VAR obscalc       : ytyp);

(* Calculate observed-calculated frequency *)

VAR     n1, m1        : integer;
        sum           : double;

BEGIN
    sum:=0;
    FOR n1:=1 TO NofCom DO
    WITH obslin[n1] DO
    BEGIN
      m1:=nlin;
      obscalc[n1,1]:=lam-lsort[m1];
      sum:=sum+obscalc[n1,1];
    END;
END; (* Procedure Calcomc *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
```

```
(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
Function Diffup(v,a:integer):double;
VAR       J, Z                    : double;
          BvP, DvP, HvP,
          LvP, MvP                : array [6..75] of double;
BEGIN
     BvP[62]:=0.01115491;
     DvP[62]:=0.713553235E-7;
     HvP[62]:=-0.961453506E-12;
     LvP[62]:=-0.244446499E-16;
     MvP[62]:=-0.183447759E-20;

     BvP[58]:=0.01289293;
     DvP[58]:=0.557758756E-7;
     HvP[58]:=-0.542436409E-12;
     LvP[58]:=-0.977036282E-17;
     MvP[58]:=-0.507261119E-21;

     BvP[43]:=0.0187449932;
     DvP[43]:=0.261854525E-7;
     HvP[43]:=-0.102372922E-12;
     LvP[43]:=-0.671072385E-18;
     MvP[43]:=-0.140018365E-22;

     BvP[28]:=0.02337449;
     DvP[28]:=0.137436522E-7;
     HvP[28]:=-0.252229941E-13;
     LvP[28]:=-0.906430138E-19;
     MvP[28]:=-0.94976525E-24;

     BvP[26]:=0.0238890046;
     DvP[26]:=0.127196813E-7;
     HvP[26]:=-0.210436261E-13;
     LvP[26]:=-0.707695536E-19;
     MvP[26]:=-0.677686275E-24;

     BvP[17]:=0.0259407207;
     DvP[17]:=0.933472447E-8;
     HvP[17]:=-0.954371438E-14;
     LvP[17]:=-0.232226556E-19;
     MvP[17]:=-0.157482286E-24;

     BvP[16]:=0.0261448373;
     DvP[16]:=0.905736950E-8;
     HvP[16]:=-0.875954939E-14;
     LvP[16]:=-0.205409358E-19;
     MvP[16]:=-0.134313350E-24;

     BvP[15]:=0.0263447094;
     DvP[15]:=0.879576237E-8;
     HvP[15]:=-0.804290883E-14;
     LvP[15]:=-0.181804330E-19;
     MvP[15]:=-0.114494794E-24;

     BvP[14]:=0.0265404834;
     DvP[14]:=0.854896312E-8;
     HvP[14]:=-0.73878842E-14;
     LvP[14]:=-0.161026496E-19;
     MvP[14]:=-0.975127447E-25;

     BvP[12]:=0.0269203212;
     DvP[12]:=0.809626788E-8;
     HvP[12]:=-0.624191446E-14;
     LvP[12]:=-0.126602265E-19;
     MvP[12]:=-0.704383883E-25;
```

```
    BvP[11]:=0.0271046725;
    DvP[11]:=0.788872567E-8;
    HvP[11]:=-0.574173652E-14;
    LvP[11]:=-0.112368044E-19;
    MvP[11]:=-0.597084729E-25;

    BvP[10]:=0.0272854969;
    DvP[10]:=0.769269693E-8;
    HvP[10]:=-0.528454767E-14;
    LvP[10]:=-0.997813192E-20;
    MvP[10]:=-0.505146923E-25;

    BvP[9]:=0.0274629252;
    DvP[9]:=0.750747024E-8;
    HvP[9]:=-0.486654557E-14;
    LvP[9]:=-0.886268189E-20;
    MvP[9]:=-0.426572538E-25;

    BvP[8]:=0.02763708;
    DvP[8]:=0.733238172E-8;
    HvP[8]:=-0.448416290E-14;
    LvP[8]:=-0.787179347E-20;
    MvP[8]:=-0.35967667E-25;

    BvP[6]:=0.0279760289;
    DvP[6]:=0.701022026E-8;
    HvP[6]:=-0.381305617E-14;
    LvP[6]:=-0.62025631E-20;
    MvP[6]:=-0.255362733E-25;

    J:=a;
    Z:=J*(J+1);
    diffup:=BvP[v]*Z-DvP[v]*Z*Z+HvP[v]*Z*Z*Z
           +LvP[v]*Z*Z*Z*Z+MvP[v]*Z*Z*Z*Z*Z;
END; (* Function Diffup *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Function Difflow(v,a:integer):double;

VAR     J,Z                     : double;
        BvB,DvB,HvB             : array [1..6] of double;

BEGIN

    BvB[6]:=0.03673226380;
    DvB[6]:=0.46791357974E-8;
    HvB[6]:=-0.62560711081E-15;

    BvB[4]:=0.03696585854;
    DvB[4]:=0.4623730065E-8;
    HvB[4]:=-0.57795676696E-15;

    BvB[3]:=0.0370816135847;
    DvB[3]:=0.45975343386E-8;
    HvB[3]:=-0.55564294687E-15;

    BvB[2]:=0.03719670068;
    DvB[2]:=0.457220849E-8;
    HvB[2]:=-0.534032229E-15;

    BvB[1]:=0.03731114099;
    DvB[1]:=0.454754763E-8;
    HvB[1]:=-0.512751868E-15;

    J:=a;
    Z:=J*(J+1);
    difflow:=BvB[v+1]*Z-DvB[v+1]*Z*Z+HvB[v+1]*Z*Z*Z;
END; (* Function Difflow *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
```

```
procedure diag(VAR eget        : btyp;
                   Dimen       : integer;
               VAR rak         : integer;
               VAR Eigenvec    : TNmatrix;
                   Mat         : TNmatrix);

VAR       MaxIter, Iter  : integer;
          Tolerance      : double;
          Eigenval       : TNvector;
          Error          : byte;
          m1, n1         : integer;
          sum            : double;

BEGIN
    Tolerance:=1E-12;
    MaxIter:=200;

    Jacobi(Dimen,Mat,MaxIter,Tolerance,
           Eigenval,Eigenvec,Iter,Error);

    IF Error<>0 THEN writeln('Error in Jacobi !');

    FOR m1:=1 TO Dimen DO
    BEGIN
      rak:=rak+1;
      eget[rak]:=Eigenval[m1];
    END;
END; (* Procedure diag *)
```

```
Procedure Jacobi(Dimen      : integer;
                 Mat        : TNmatrix;
                 MaxIter    : integer;
                 Tolerance  : double;
             VAR Eigenval   : TNvector;
             VAR Eigenvec   : TNmatrix;
             VAR Iter       : integer;
             VAR Error      : byte);

CONST  TNNearZero = 1E-15;

VAR     Row, Column, Diag       : integer;
        SinTheta, CosTheta      : double;
        SumSquareDiag           : double;
        Done                    : boolean;

(* ------------------------------------------------ *)

Procedure TestData(Dimen      : integer;
               VAR Mat        : TNmatrix;
                   MaxIter    : integer;
                   Tolerance  : double;
               VAR Error      : byte);

VAR    Row, Column    : integer;

BEGIN
  Error:=0;
  IF Dimen<1 THEN Error:=1;
  IF Tolerance<=TNNearZero THEN Error:=2;
  IF MaxIter<1 THEN Error:=3;
  IF Error=0 THEN
  FOR Row:=1 TO Dimen-1 DO
  FOR Column:=Row+1 TO Dimen DO
  IF ABS(Mat[Row,Column]-Mat[Column,Row])>TNNearZero
  THEN Error:=4;
END; (* Procedure TestData *)

(* ------------------------------------------------ *)

Procedure Initialize(Dimen      : integer;
                 VAR Iter       : integer;
                 VAR Eigenvec   : TNmatrix);

VAR       Diag : integer;

BEGIN
  Iter:=0;
  FillChar(Eigenvec,SizeOf(Eigenvec),0);
  FOR Diag:=1 TO Dimen DO
  Eigenvec[Diag,Diag]:=1;
END; (* Procedure Initialize *)

(* ------------------------------------------------ *)

Procedure CalculateRotation(RowRow    : double;
                            RowCol    : double;
                            ColCol    : double;
                        VAR SinTheta  : double;
                        VAR CosTheta  : double);

VAR   TangentTwoTheta, TangentTheta, Dummy : double;

BEGIN
  IF ABS(RowRow-ColCol)>TNNearZero THEN
  BEGIN
    TangentTwoTheta:=(RowRow-ColCol)/(2*RowCol);
    Dummy:=SQRT(SQR(TangentTwoTheta)+1);
    IF TangentTwoTheta<0 THEN
    TangentTheta:=-TangentTwoTheta-Dummy
    ELSE TangentTheta:=-TangentTwoTheta+Dummy;
    CosTheta:=1/SQRT(1+SQR(TangentTheta));
```

```
    SinTheta:=CosTheta*TangentTheta;
  END
  ELSE
  BEGIN
    CosTheta:=SQRT(1/2);
    IF RowCol<0 THEN SinTheta:=-SQRT(1/2)
    ELSE SinTheta:=SQRT(1/2);
  END;
END; (* Procedure CalculateRotation *)

(* ------------------------------------------------ *)

Procedure RotateMatrix(Dimen    : integer;
                       SinTheta : double;
                       CosTheta : double;
                       Row      : integer;
                       Col      : integer;
                   VAR Mat      : TNmatrix);

VAR     CosSqr, SinSqr, SinCos             : double;
        MatRowRow, MatColCol, MatRowCol,
        MatRowIndex, MatColIndex           : double;
        Index                              : integer;

BEGIN
  CosSqr:=SQR(CosTheta);
  SinSqr:=SQR(SinTheta);
  SinCos:=SinTheta*CosTheta;
  MatRowRow:=Mat[Row, Row]*CosSqr+2*Mat[Row,Col]
             *SinCos+Mat[Col,Col]*SinSqr;
  MatColCol:=Mat[Row,Row]*SinSqr-2*Mat[Row,Col]
             *SinCos+Mat[Col,Col]*CosSqr;
  MatRowCol:=(Mat[Col,Col]-Mat[Row,Row])*SinCos+
              Mat[Row,Col]*(CosSqr-SinSqr);

  FOR Index:=1 TO Dimen DO
  IF NOT(Index in [Row,Col]) THEN
  BEGIN
    MatRowIndex:=Mat[Row,Index]*CosTheta+
                 Mat[Col,Index]*SinTheta;
    MatColIndex:=-Mat[Row,Index]*SinTheta+
                  Mat[Col,Index]*CosTheta;
    Mat[Row,Index]:=MatRowIndex;
    Mat[Index,Row]:=MatRowIndex;
    Mat[Col,Index]:=MatColIndex;
    Mat[Index,Col]:=MatColIndex;
  END;
  Mat[Row,Row]:=MatRowRow;
  Mat[Col,Col]:=MatColCol;
  Mat[Row,Col]:=MatRowCol;
  Mat[Col,Row]:=MatRowCol;
END; (* Procedure RotateMatrix *)

(* ------------------------------------------------ *)

Procedure RotateEigenvec(Dimen    : integer;
                         SinTheta : double;
                         CosTheta : double;
                         Row      : integer;
                         Col      : integer;
                     VAR Eigenvec : TNmatrix);

VAR  EigenvecRowIndex,
     EigenvecColIndex    : double;
     Index               : integer;

BEGIN
  (* Transform eigenvector matrix *)
  FOR Index:=1 TO  Dimen DO
  BEGIN
    EigenvecRowIndex:=CosTheta*Eigenvec[Row,Index]+
                      SinTheta*Eigenvec[Col,Index];
    EigenvecColIndex:=-SinTheta*Eigenvec[Row,Index]  +
```

```
                          CosTheta*Eigenvec[Col,Index];
      Eigenvec[Row,Index]:=EigenvecRowIndex;
      Eigenvec[Col,Index]:=EigenvecColIndex;
    END;
  END; (* Procedure RotateEigenvec *)

  (* ------------------------------------------- *)

  BEGIN (* Procedure Jacobi *)
    TestData(Dimen,Mat,MaxIter,Tolerance,Error);
    IF Error=0 THEN
    BEGIN
      Initialize(Dimen,Iter,Eigenvec);
      REPEAT
        Iter:=Succ(Iter);
        SumSquareDiag:=0;
        FOR Diag:=1 TO Dimen DO
        SumSquareDiag:=SumSquareDiag+SQR(Mat[Diag,Diag]);
        Done:=TRUE;
        FOR Row:=1 TO Dimen-1  DO
        FOR Column:=Row+1 TO Dimen DO
        IF ABS(Mat[Row,Column])>Tolerance*SumSquareDiag
        THEN
        BEGIN
          Done:=FALSE;
          CalculateRotation(Mat[Row,Row],Mat[Row,Column],
                            Mat[Column,Column],SinTheta,
                                               CosTheta);
          RotateMatrix(Dimen,SinTheta,CosTheta,Row,
                                              Column,Mat);
          RotateEigenvec(Dimen,SinTheta,CosTheta,Row,
                                             Column,Eigenvec);
        END;
      UNTIL Done or (Iter>MaxIter);
      FOR Diag:=1 TO Dimen DO
      Eigenval[Diag]:=Mat[Diag,Diag];

      IF Iter > MaxIter THEN
      Error:=5
    END;
  END; (* Procedure Jacobi *)
```

```
Procedure Inverse(Dimen : integer;
                      Data  : TNmatrix;
                  VAR Inv   : TNmatrix;
                  VAR Error : byte);

(* Inverse a square matrix *)

CONST     TNNearZero = 1E-015;

(* ------------------------------------------------ *)

Procedure Initial(Dimen : integer;
                  VAR Data  : TNmatrix;
                  VAR Inv   : TNmatrix;
                  VAR Error : byte);

VAR    Row : integer;

BEGIN
  Error:=0;
  IF Dimen<1 THEN
  Error:=1
  ELSE
  BEGIN
    FillChar(Inv, SizeOf(Inv),0);
    FOR Row:=1 TO Dimen DO
    Inv[Row,Row]:=1;
    IF Dimen=1 THEN
    IF ABS(Data[1,1])<TNNearZero THEN
    Error:=2    (* Singular matrix *)
    ELSE
    Inv[1,1]:=1/Data[1,1];
  END;
END; (* Procedure Initial *)

(* ------------------------------------------------ *)

Procedure EROdiv(Divisor : double;
                     Dimen   : integer;
                 VAR Row     : TNvector);

VAR    Term : integer;

BEGIN
  FOR Term:=1 TO Dimen DO
  Row[Term]:=Row[Term]/Divisor;
END; (* Procedure EROdiv *)

(* ------------------------------------------------ *)

Procedure EROswitch(VAR Row1 : TNvector;
                    VAR Row2 : TNvector);


VAR    DummyRow : TNvector;

BEGIN
  DummyRow:=Row1;
  Row1:=Row2;
  Row2:=DummyRow;
END; (* Procedure EROswitch *)

(* ------------------------------------------------ *)

Procedure EROmultAdd(Multiplier    : double;
                         Dimen         : integer;
                     VAR ReferenceRow : TNvector;
                     VAR ChangingRow  : TNvector);

VAR
  Term : integer;
```

```
BEGIN
  FOR Term:=1 TO Dimen DO
  ChangingRow[Term]:=ChangingRow[Term]+Multiplier
                     *ReferenceRow[Term];
END; (* Procedure EROmultAdd *)

(* ------------------------------------------------ *)

Procedure Inver(Dimen : integer;
            VAR Data  : TNmatrix;
            VAR Inv   : TNmatrix;
            VAR Error : byte);

VAR       Divisor, Multiplier       : double;
          Row, ReferenceRow         : integer;

Procedure Pivot(Dimen        : integer;
                ReferenceRow : integer;
            VAR Data         : TNmatrix;
            VAR Inv          : TNmatrix;
            VAR Error        : byte);

VAR       NewRow : integer;

BEGIN
  Error:=2;  (* No inverse exists *)
  NewRow:=ReferenceRow;
  WHILE (Error>0) AND (NewRow<Dimen) DO
  BEGIN
    NewRow:=Succ(NewRow);
    IF ABS(Data[NewRow,ReferenceRow])>TNNearZero THEN
    BEGIN
      EROswitch(Data[NewRow],Data[ReferenceRow]);
      EROswitch(Inv[NewRow],Inv[ReferenceRow]);
      Error:=0;
    END;
  END; (* WHILE *)
END; (* Procedure Pivot *)

(* ------------------------------------------------ *)

BEGIN (* Procedure Inver *)
  ReferenceRow:=0;
  WHILE (Error=0) AND (ReferenceRow<Dimen) DO
  BEGIN
    ReferenceRow:=Succ(ReferenceRow);
    IF ABS(Data[ReferenceRow,ReferenceRow])<TNNearZero
    THEN
    Pivot(Dimen,ReferenceRow,Data,Inv,Error);
    IF Error=0 THEN
    BEGIN
      Divisor:=Data[ReferenceRow,ReferenceRow];
      EROdiv(Divisor,Dimen,Data[ReferenceRow]);
      EROdiv(Divisor,Dimen,Inv[ReferenceRow]);
      FOR Row:=1 TO Dimen DO
      IF (Row <> ReferenceRow) AND
      (ABS(Data[Row,ReferenceRow])>TNNearZero) THEN
      BEGIN
        Multiplier:=-Data[Row,ReferenceRow]/
                    Data[ReferenceRow,ReferenceRow];
        EROmultAdd(Multiplier,Dimen,Data[ReferenceRow],
                                    Data[Row]);
        EROmultAdd(Multiplier,Dimen,Inv[ReferenceRow],
                                    Inv[Row]);
      END;
    END;
  END;
END; (* Procedure Inver *)

(* ------------------------------------------------ *)
```

```
BEGIN (* Procedure Inverse *)
  Initial(Dimen,Data,Inv,Error);
  IF Dimen>1 THEN
  Inver(Dimen,Data,Inv,Error);
END; (* Procedure Inverse *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure Transpose(VAR m1                : etyp;
                        numrow,numcol     : integer;
                    VAR m2                : tetyp);

(* Transpose a matrix *)

VAR i, j : integer;
    temp : double;

BEGIN
  FOR i:=1 TO numrow  DO
  FOR j:=1 TO numcol DO
  m2[j,i]:=m1[i,j];
END; (* Procedure Transpose *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure Product(VAR m1,m2  : mtyp;
                      l,m,n  : integer;
                  VAR m3     : mtyp);

(* Calculates the product of two matrices *)

VAR    i, j, k   : integer;

BEGIN
  FOR i:=1 TO l DO
  FOR j:=1 TO n DO
  BEGIN
    m3[i,j]:=0E+00;
    FOR k:= 1 TO m DO
    m3[i,j]:=m3[i,j]+m1[i,k]*m2[k,j];
  END;
END; (* Procedure Product *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure Matris(obscalc     : ytyp;
                 A           : etyp;
             VAR TETA,Bii    : corrtyp;
                 x3          : integer;
                 nin         : ntyp;
                 wiw         : ftyp;
             VAR ATAI        : TNmatrix);

(* Calculates the correction term TETA *)

TYPE      filtyp   = string[16];

VAR       B, BI                        : TNmatrix;
          M1, M2, M3                   : mtyp;
          a1, b1, fem, tje, ett,
          p1, q1                       : integer;
          Deter                        : double;
          AT,ATAIAT                    : tetyp;
          Error                        : byte;

BEGIN

(* ------------- Set matrices to zero -------------- *)
```

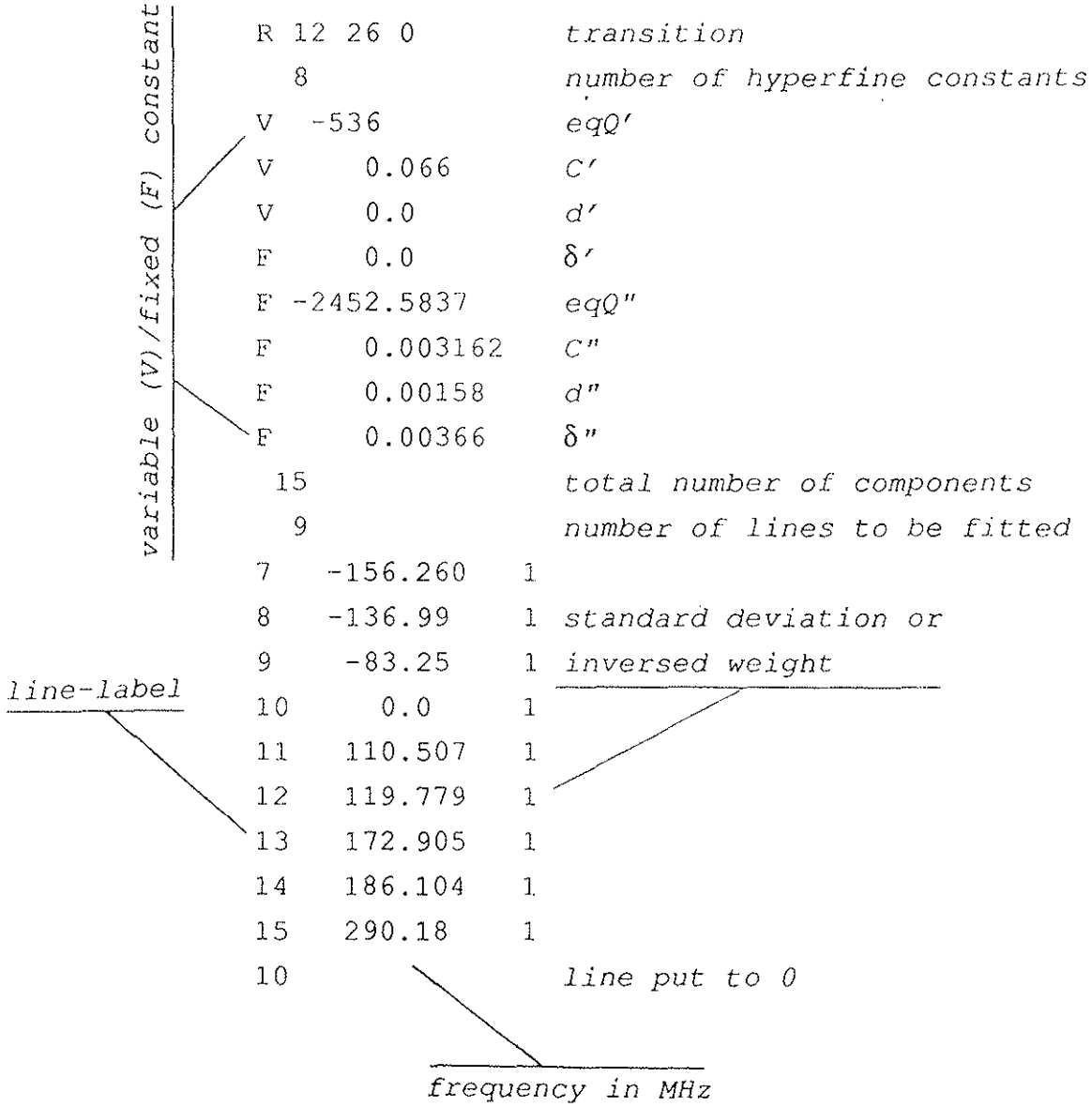
```
    FOR p1:=1 TO 21 DO
    FOR q1:=1 TO 5 DO
    BEGIN
      AT[q1,p1]:=0E+00;
      ATAIAT[q1,p1]:=0E+00;
    END;

    FOR p1:=1 TO 8 DO
    FOR q1:=1 TO 8 DO
    BEGIN
      B[p1,q1]:=0E+00;
      BI[p1,q1]:=0E+00;
    END;

    FOR p1:=1 TO 21 DO
    FOR q1:=1 TO 21 DO
    BEGIN
      M1[q1,p1]:=0E+00;
      M2[q1,p1]:=0E+00;
      M3[q1,p1]:=0E+00;
    END;

(* ------------------- Set variables ----------------- *)

    fem:=5;
    tje:=21;
    ett:=1;

(*   Put the lines together if  NofCom< 21 (or 15)    *)
(* -------- NofCom - Number of observed lines -------- *)
(* ------------- Matrix A = hkonst in Main ----------- *)

    FOR p1:=1 TO x3 DO
    BEGIN
      a1:=nin[p1];
      FOR q1:=1 TO 5 DO
      A[p1,q1]:=A[a1,q1];
    END;

(* ---------------------- Make AT ------------------- *)

    Transpose(A,x3,fem,AT);

(* --------------Include weights -------------------- *)

    FOR p1:=1 TO x3 DO
    FOR q1:=1 TO 5 DO
    A[p1,q1]:=A[p1,q1]*wiw[p1];

(* ---------------- Make ATA = AT x A --------------- *)

    FOR p1:=1 TO fem DO
    FOR q1:=1 TO x3 DO
    BEGIN
      M1[p1,q1]:=AT[p1,q1];
      M2[q1,p1]:=A[q1,p1];
    END;

    Product(M1,M2,fem,x3,fem,M3);

    FOR p1:=1 TO fem DO
    FOR q1:=1 TO fem DO
    B[p1,q1]:=M3[p1,q1];       (* ATA = M3 *)

(* ------------ Take away fixed elements ------------ *)

    a1:=0;
    FOR p1:=1 TO 5 DO
    IF hk[p1].fix='V' THEN
    BEGIN
      b1:=0;
      a1:=a1+1;
      FOR q1:=1 TO 5 DO
      IF hk[q1].fix='V' THEN
```

```
      BEGIN
        b1:=b1+1;
        BI[a1,b1]:=B[p1,q1];
      END;
    END;
    B:=BI;

(* ---------- Make ATAI= inverse of ATA ------------ *)

    Inverse(fem,B,BI,Error);

(* --------- Make space for fixed elements --------- *)

    FOR p1:=1 TO 8 DO
    FOR q1:=1 TO 8 DO
    B[p1,q1]:=0E+00;

    a1:=0;
    FOR p1:=1 TO 5 DO
    IF hk[p1].fix='V' THEN
    BEGIN
      b1:=0;
      a1:=a1+1;
      FOR q1:=1 TO 5 DO
      IF hk[q1].fix='V' THEN
      BEGIN
        b1:=b1+1;
        B[p1,q1]:=BI[a1,b1];
      END;
    END;
    BI:=B;
    ATAI:=B;     (* ATAI = Inverse of ATA *)

(* ----------- Make variance vectort Bii ----------- *)

    FOR p1:=1 TO fem DO
    Bii[p1,1]:=BI[p1,p1];

(* ------------ Make ATAIAT = ATAI x AT ------------ *)

    FOR p1:=1 TO 21 DO
    FOR q1:=1 TO 21 DO
    BEGIN
      M1[q1,p1]:=0E+00;
      M2[q1,p1]:=0E+00;
      M3[q1,p1]:=0E+00;
    END;

    FOR p1:=1 TO fem DO
    FOR q1:=1 TO x3 DO
    M2[p1,q1]:=AT[p1,q1];

    FOR p1:=1 TO fem DO
    FOR q1:=1 TO fem DO
    M1[p1,q1]:=BI[p1,q1];

    Product(M1,M2,fem,fem,x3,M3);   (* M3=ATAIAT *)

(* ------------ Make correction vector ------------- *)

    FOR p1:=1 TO 21 DO
    FOR q1:=1 TO 21 DO
    BEGIN
      M2[q1,p1]:=0E+00;
      M1[q1,p1]:=0E+00;
    END;

    FOR q1:=1 TO x3 DO
    M2[q1,1]:=obscalc[q1,1]*wiw[q1];

    Product(M3,M2,fem,x3,ett,M1);

    FOR q1:=1 TO fem DO
    TETA[q1,1]:=M1[q1,1];
```

```
END; (* Procedure Matris *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Function Norma(vp         : ztyp;
               vpt        : ytyp;
               hqp        : TNmatrix ;
               Dimen      : integer):double;

(* ---- Calculate the normalized matrix element ---- *)
(* ------------- to the A matrix ------------------- *)

VAR          M1, M2, M3                : mtyp;
             fem, tje, ett, p1, q1     : integer;

BEGIN
    FOR p1:=1 TO 21 DO
    FOR q1:=1 TO 21 DO
    BEGIN
      M1[q1,p1]:=0E+00;
      M2[q1,p1]:=0E+00;
      M3[q1,p1]:=0E+00;
    END;

    fem:=5;
    tje:=21;
    ett:=1;

    FOR q1:=1 TO Dimen DO
    M1[1,q1]:=vp[1,q1];

    FOR p1:=1 TO Dimen DO
    FOR q1:=1 TO Dimen DO
    M2[p1,q1]:=hqp[p1,q1];

    Product(M1,M2,ett,Dimen,Dimen,M3);

    FOR p1:=1 TO 21 DO
    FOR q1:=1 TO 21 DO
    BEGIN
      M2[q1,p1]:=0E+00;
      M1[q1,p1]:=0E+00;
    END;

    FOR q1:=1 TO Dimen DO
    M2[q1,1]:=vpt[q1,1];

    Product(M3,M2,ett,Dimen,ett,M1);

    Norma:=M1[1,1];
END; (* Function Norma *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure Felcalc(obscalc                : ytyp;
                  Bii                    : corrtyp;
                  NofCom,NofConst        : integer;
              VAR RMS                    : double;
              VAR sdevcon                : corrtyp;
              VAR sdev                   : double;
                  wiw                    : ftyp);

(* Calculates the standard deviation *)

VAR      n1            : integer;
         sum           : double;
BEGIN
    sum:=0E+00;
    FOR n1:=1 TO NofCom DO
    sum:=sum+obscalc[n1,1]*obscalc[n1,1]*wiw[n1];

    sdev:=SQRT(sum/(NofCom-NofConst));
    RMS:=SQRT(sum/NofCom);
```

```
    FOR n1:=1 TO 5 DO
    sdevcon[n1,1]:=SQRT(Bii[n1,1])*sdev;
END; (* Procedure Felcalc *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)

Procedure Correlate(VAR ATAI,Corr: TNmatrix);

(* Calculates the correlation coefficients *)

VAR     p1, q1     : integer;

BEGIN
    FOR p1:=1 TO 5 DO
    FOR q1:=1 TO 5 DO
    IF (ATAI[p1,p1]=0.0) OR (ATAI[q1,q1]=0.0) THEN
    Corr[p1,q1]:=1 ELSE
    Corr[p1,q1]:=ATAI[p1,q1]/
                 SQRT(ATAI[p1,p1]*ATAI[q1,q1]);
END; (* Procedure Correlate *)

(*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
```

Appendix III


variable (V)/fixed (F) constant
R 12 26 0 transition
8 number of hyperfine constants
V -536 eqQ'
V 0.066 C'
V 0.0 d'
F 0.0 δ'
F -2452.5837 eqQ"
F 0.003162 C"
F 0.00158 d"
F 0.00366 δ"
15 total number of components
9 number of lines to be fitted
7 -156.260 1
8 -136.99 1 standard deviation or
9 -83.25 1 inversed weight
line-label
10 0.0 1
11 110.507 1
12 119.779 1
13 172.905 1
14 186.104 1
15 290.18 1
10 line put to 0
frequency in MHz

R(12) 26 - 0

Itteration = 0
RMS : 0.118144
sdev : 0.158506

no	correction	new const	sdev	status
1	0.000000	-536.000000	0.000000	V
2	0.000000	0.066000	0.000000	V
3	0.000000	0.000000	0.000000	V
4	0.000000	0.000000	0.000000	F
5	0.000000	-2452.583700	0.000000	F
6		0.003162		
7		0.001580		
8		0.003660		

no	comp	F	I	J	observed	calculated	obs-calc	weight
1	7	13	2	13	-156.260000	-156.202148	-0.057852	1.0000
2	8	9	4	13	-136.990000	-136.793062	-0.196938	1.0000
3	9	14	2	13	-83.250000	-82.989767	-0.260233	1.0000
4	10	12	2	13	0.000000	0.000000	0.000000	1.0000
5	11	17	4	13	110.507000	110.462058	0.044942	1.0000
6	12	10	4	13	119.779000	119.763295	0.015705	1.0000
7	13	11	4	13	172.905000	172.886515	0.018485	1.0000
8	14	15	4	13	186.104000	186.217493	-0.113493	1.0000
9	15	16	4	13	290.180000	290.163276	0.016724	1.0000

Itteration = 1
RMS : 0.011506
sdev : 0.015437

no	correction	new const	sdev	status
1	1.063396	-534.936604	0.077823	V
2	-0.003586	0.062414	0.000218	V
3	-0.027372	-0.027372	0.002136	V
4	0.000000	0.000000	0.000000	F
5	0.000000	-2452.583700	0.000000	F
6		0.003162		
7		0.001580		
8		0.003660		

no	comp	F	I	J	observed	calculated	obs-calc	weight
1	7	13	2	13	-156.260000	-156.241584	-0.018416	1.0000
2	8	9	4	13	-136.990000	-136.983487	-0.006513	1.0000
3	9	14	2	13	-83.250000	-83.251123	0.001123	1.0000
4	10	12	2	13	0.000000	0.000000	0.000000	1.0000
5	11	17	4	13	110.507000	110.515510	-0.008510	1.0000
6	12	10	4	13	119.779000	119.758686	0.020314	1.0000
7	13	11	4	13	172.905000	172.910419	-0.005419	1.0000
8	14	15	4	13	186.104000	186.100200	0.003800	1.0000
9	15	16	4	13	290.180000	290.196725	-0.016725	1.0000

Itteration = 2
RMS : 0.011506
sdev : 0.015436

no	correction	new const	sdev	status
1	0.000311	-534.936294	0.077843	V
2	0.000000	0.062414	0.000218	V
3	0.000011	-0.027361	0.002134	V
4	0.000000	0.000000	0.000000	F
5	0.000000	-2452.583700	0.000000	F
6		0.003162		
7		0.001580		
8		0.003660		

no	comp	F	I	J	observed	calculated	obs-calc	weight
1	7	13	2	13	-156.260000	-156.241597	-0.018403	1.0000
2	8	9	4	13	-136.990000	-136.983495	-0.006505	1.0000
3	9	14	2	13	-83.250000	-83.251097	0.001097	1.0000
4	10	12	2	13	0.000000	0.000000	0.000000	1.0000
5	11	17	4	13	110.507000	110.515548	-0.008548	1.0000
6	12	10	4	13	119.779000	119.758762	0.020238	1.0000
7	13	11	4	13	172.905000	172.910461	-0.005461	1.0000
8	14	15	4	13	186.104000	186.100286	0.003714	1.0000
9	15	16	4	13	290.180000	290.196822	-0.016822	1.0000

Itteration = 3
RMS : 0.011506
sdev : 0.015436

no	correction	new const	sdev	status
1	-0.000000	-534.936294	0.077843	V
2	0.000000	0.062414	0.000218	V
3	0.000000	-0.027361	0.002134	V
4	0.000000	0.000000	0.000000	F
5	0.000000	-2452.583700	0.000000	F
6		0.003162		
7		0.001580		
8		0.003660		

no	comp	F	I	J	observed	calculated	obs-calc	weight
1	7	13	2	13	-156.260000	-156.241597	-0.018403	1.0000
2	8	9	4	13	-136.990000	-136.983495	-0.006505	1.0000
3	9	14	2	13	-83.250000	-83.251097	0.001097	1.0000
4	10	12	2	13	0.000000	0.000000	0.000000	1.0000
5	11	17	4	13	110.507000	110.515548	-0.008548	1.0000
6	12	10	4	13	119.779000	119.758762	0.020238	1.0000
7	13	11	4	13	172.905000	172.910461	-0.005461	1.0000
8	14	15	4	13	186.104000	186.100286	0.003714	1.0000
9	15	16	4	13	290.180000	290.196822	-0.016822	1.0000

Correlation coefficients:

1.0000000	0.0415754	-0.5968287	1.0000000	1.0000000
0.0415754	1.0000000	-0.0684054	1.0000000	1.0000000
-0.5968287	-0.0684054	1.0000000	1.0000000	1.0000000
1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
1.0000000	1.0000000	1.0000000	1.0000000	1.0000000

Appendix IV

Itteration = 5
RMS : 0.015960
sdev : 0.020346

no	correction	new const	sdev	status
1	0.000008	-426.158870	62.400314	V
2	0.000000	0.829225	0.000038	V
3	0.000000	-0.500525	0.003201	V
4	-0.000000	0.356043	0.003146	V
5	0.000008	-2306.355594	62.377951	V
6		0.000000		
7		0.001500		
8		0.000000		

no	comp	F	I	J	observed	calculated	obs-calc
1	1	95	4	99	-413.493000	-413.513685	0.020685
2	2	99	2	99	-359.551000	-359.552699	0.001699
3	3	96	4	99	-194.530000	-194.530275	0.000275
4	4	98	2	99	-159.159000	-159.142823	-0.016177
5	5	97	4	99	-105.771000	-105.761373	-0.009627
6	6	100	2	99	0.000000	0.000000	0.000000
7	7	97	2	99	172.203000	172.233733	-0.030733
8	8	99	4	99	200.482000	200.480607	0.001393
9	9	101	4	99	225.973000	225.972260	0.000740
10	12	103	4	99	314.122000	314.150416	-0.028416
11	13	98	4	99	426.698000	426.674695	0.023305
12	14	102	4	99	481.578000	481.573747	0.004253
13	15	100	4	99	510.252000	510.237485	0.014515

sdev : 0.0152

new const	sdev	status
-493.839	8.36194	V
0.82920	0.00004	V
-0.5004	0.0035	V
0.3568	0.0011	V
-2374.037	8.378841	V
0.000000		
0.001500		
0.000000		

calculated	obs-calc
-413.513	0.020
-359.558	0.007
-194.530	0.000
-159.139	-0.020
-105.759	-0.012
-0.010	0.010
172.228	-0.025
200.478	0.004
225.965	0.008
314.152	-0.030
426.677	0.021
481.573	0.005
510.243	0.009

Foth and Spieweck

Itteration = 3

RMS	:	0.019301	
sdev	:	0.023197	

no	correction	new const	sdev	status
1	0.000000	-572.463709	0.073699	V
2	-0.000000	0.832323	0.000036	V
3	-0.000000	-0.500442	0.003620	V
4	0.000000	0.360511	0.003537	V
5	0.000000	-2452.583700	0.000000	F
6		0.003162		
7		0.001580		
8		0.003660		

no	comp	F	I	J	observed	calculated	obs-calc
1	1	95	4	99	-413.487000	-413.507807	0.020807
2	2	99	2	99	-359.554000	-359.531227	-0.022773
3	3	96	4	99	-194.520000	-194.528426	0.008426
4	4	98	2	99	-159.158000	-159.120451	-0.037549
5	5	97	4	99	-105.771000	-105.751424	-0.019576
6	6	100	2	99	0.000000	0.000000	0.000000
7	7	97	2	99	172.197000	172.215611	-0.018611
8	8	99	4	99	200.476000	200.479989	-0.003989
9	9	101	4	99	225.982000	225.973742	0.008258
10	12	103	4	99	314.133000	314.169233	-0.036233
11	13	98	4	99	426.693000	426.679992	0.013008
12	14	102	4	99	481.574000	481.564547	0.009453
13	15	100	4	99	510.245000	510.250030	-0.005030

sdev	:	0.0246	

new const	sdev	status
-572.464	0.078	V
0.832322	0.000039	V
-0.50046	0.00384	V
0.36051	0.00375	V
-2452.583700		F
0.003162		
0.001580		
0.003660		

calculated	obs-calc
-413.508	0.021
-359.531	-0.023
-194.528	0.008
-159.120	-0.038
-105.751	-0.020
0.000000	
172.216	-0.019
200.480	-0.004
225.974	0.008
314.169	-0.036
426.680	0.013
481.564	0.010
510.250	-0.005

Itteration = 4
RMS : 0.000871
sdev . : 0.001037

no	correction	new const	sdev	status
1	0.000000	-557.714600	0.158686	V
2	-0.000000	0.186882	0.000009	V
3	-0.000000	-0.101595	0.000109	V
4	0.000000	-0.003120	0.000090	V
5	0.000000	-2451.679754	0.158604	V
6		0.000000		
7		0.000000		
8		0.000000		

no	comp	F	I	J	observed	calculated	obs-calc
1	1	7	5	12	0.000000	0.000000	0.000000
2	2	12	1	12	71.865200	71.864216	0.000984
3	3	17	5	12	131.770100	131.770949	-0.000849
4	4	8	5	12	207.818800	207.817671	0.001129
5	5	13	3	12	334.999100	334.999158	-0.000058
6	6	11	3	12	372.543900	372.545321	-0.001421
7	7	9	5	12	386.775400	386.774877	0.000523
8	8	10	3	12	470.469300	470.470573	-0.001273
9	9	16	5	12	481.488400	481.487007	0.001393
10	11	14	3	12	525.732000	525.732519	-0.000519
11	12	15	3	12	567.369800	567.368825	0.000975
12	13	9	3	12	631.481600	631.481796	-0.000196
13	15	12	3	12	719.166700	719.166339	0.000361
14	16	10	5	12	748.526200	748.525968	0.000232
15	17	11	5	12	792.702800	792.701619	0.001181
16	20	15	5	12	889.401500	889.402357	-0.000857
17	21	13	1	12	949.106600	949.107317	-0.000717

sdev : 0.00107

new const	sdev	status
-557.717	0.164	V
0.186880	0.000009	V
-0.101604	0.000113	V
-0.003116	0.000093	V
-2451.682	0.164	V

calculated	obs-calc
71.8642	0.0010
131.7708	-0.0007
207.8177	0.0011
334.9992	-0.0001
372.5453	-0.0014
386.7749	0.0005
470.4705	-0.0012
481.4870	0.0014
525.7326	-0.0006
567.3687	0.0011
631.4818	-0.0002
719.1662	0.0005
748.5260	0.0002
792.7018	0.0010
889.4023	-0.0008
949.1074	-0.0008

Itteration = 3
RMS : 0.001902
sdev : 0.002175

no	correction	new const	sdev	status	new const	sdev	status
1	-0.000000	-504.583055	0.003515	V	-504.583	0.003	V
2	0.000000	0.028152	0.000004	V	0.028152	0.000004	V
3	0.000000	-0.019410	0.000304	V	19410	0.0003	V
4	-0.000000	-0.015594	0.000214	V	-0.0156	0.0002	V
5	0.000000	-2452.600000	0.000000	F	-2452.600000		F
6		0.003000			0.003000		
7		0.000000			0.000000		
8		0.000000			0.000000		

no	comp	F	I	J	observed	calculated	obs-calc	calculated	obs-calc
1	1	43	5	48	-357.159500	-357.158290	-0.001210	-357.1583	-0.0012
2	2	48	1	48	-333.968600	-333.968641	0.000041	-333.9686	0.0000
3	5	53	5	48	-47.274500	-47.276786	0.002286	-47.2768	0.0023
4	6	44	5	48	-36.775800	-36.773685	-0.002115	-36.7737	-0.0021
5	7	49	3	48	0.000000	0.000000	0.000000	0.0000	
6	8	47	3	48	81.455700	81.453304	0.002396	81.4532	0.0025
7	9	52	5	48	99.106700	99.104671	0.002029	99.1047	0.0020
8	10	45	5	48	107.463000	107.466265	-0.003265	107.4663	-0.0033
9	11	46	3	48	119.046700	119.047429	-0.000729	119.0474	-0.0007
10	13	50	3	48	249.602200	249.601427	0.000773	249.6015	0.0007
11	14	51	3	48	284.304100	284.304887	-0.000787	284.3049	-0.0008
12	16	45	3	48	384.657100	384.659503	-0.002403	384.6595	-0.0024
13	17	48	3	48	403.767300	403.766353	0.000947	403.7663	0.0010
14	18	51	5	48	429.996500	429.993645	0.002855	429.9936	0.0029
15	19	46	5	48	527.158700	527.157317	0.001383	527.1573	0.0014
16	20	47	5	48	539.224400	539.227765	-0.003365	539.2278	-0.0034
17	21	49	5	48	555.097300	555.097146	0.000154	555.0971	0.0002