JPL Publication 87-2

Parametrization of Electron Impact Ionization Cross Sections for CO, CO_2, CH_4, NH_3, and SO_2

Santosh K. Srivastava
Hùng P. Nguyễn

April 1, 1987

Prepared for

Air Force Office of Scientific Research

and

National Aeronautics and Space Administration

by

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL Publication 87-2

Parametrization of Electron Impact Ionization Cross Sections for CO, CO_2, CH_4, NH_3, and SO_2

Santosh K. Srivastava
Hùng P. Nguyễn

April 1, 1987

Prepared for

Air Force Office of Scientific Research

and

National Aeronautics and Space Administration

by

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the Air Force Office of Scientific Research and the National Aeronautics and Space Administration.

ABSTRACT

The electron impact ionization and dissociative ionization cross section data of CO, CO_2, CH_4, NH_3, and SO_2, measured in our laboratory, have been parametrized utilizing an empirical formula based on the Born approximation. For this purpose a χ^2 minimization technique was employed which provided an excellent fit to the experimental data.

TABLE OF CONTENTS

Figures

Table

I. INTRODUCTION

Cross sections for the production of positive ions by electron impact on molecules find application in a wide variety of plasmas. The collision process may be represented by the following relations:

$$e^- + MN \xrightarrow{\sigma} MN^+ + 2e^- \tag{1}$$

$$\xrightarrow{\sigma} M^+ + N + 2e^- \tag{2a}$$

$$\xrightarrow{\sigma} M + N^+ + 2e^- \tag{2b}$$

$$\xrightarrow{\sigma} M^+ + N^+ + 3e^- \tag{3}$$

$$\xrightarrow{\sigma} \textit{Species of higher stages of ionization,} \tag{4}$$

where σ is the cross section, MN is a diatomic molecule with M and N as component atoms and the "+" sign indicates a positive ion. Although equations (1) through (4) have been written for a diatomic molecule, similar relations hold for polyatomic molecules.

There are various definitions of the cross section, σ. They are given below:

a) Partial cross section, σ_p : It represents the individual process given by equations (1) through (4).

b) Total cross section, $\sigma(T)$: The total cross section is defined according

to the method used for its determination:

i) If the particle counting method is employed for obtaining the total number of ions (singly as well as multiply ionized) produced as a result of electron impact then it is known as *total counting ionization cross section*, $\sigma_c(T)$, and is given by:

$$\sigma_c(T) = \sum_p \sigma_p + \sum_i \sigma_p^i, \tag{5}$$

where σ_p is the partial ionization process described by eqs.(1) through (3) and σ_p^i is the partial cross section for the i^{th} stage of ionization.

ii) If the cross section data are generated by measuring the total ion current then the *total ionization cross section*, $\sigma_I(T)$, is as follows:

$$\sigma_I(T) = \sum_p \sigma_p + \sum_i Z_i \sigma_p^i, \tag{6}$$

where Z_i is the stage of ionization.

Cross sections for ionization have been measured since the 1930's. The various methods employed, in the past, for this purpose have been described in detail in several review articles[1,2,3,4] previously published in the literature. Theoretical calculations for these cross sections are difficult due to many channels in the continuum contributing to the ionization process. Recently, however, the status of the theory of ionization of atoms and ions has been reviewed by Younger[5]. A similar survey for molecules has also been made

by Rudge[6].

The first approach to the calculation of ionization cross sections was based on the classical theory and was presented by Thomson[7] in 1912. His expression for the ionization is as follows:

$$\sigma_p = 4\xi \left(\frac{I_H}{I}\right)^2 X^{-1}(1 - X^{-1})\pi a_o^2 \tag{7}$$

where ξ is the number of electrons in the target with binding energy I, $X \approx EI^{-1}$ is the reduced ionization energy and I_H is the ionization energy of the hydrogen atom.

After Thomson, several classical, semi-classical, and empirical formulas have been proposed for the calculation of ionization cross sections. They are by: Elewert (1952)[8], Gryzinski (1959)[9], Post (1961)[10], Drawin (1961)[11], Burgess (1963)[12], Stabler (1964)[13], Seaton (1964)[14], Gryzinski I (1965, Simple ionization)[15], Gryzinski (1965, Double ionization)[16], Vriens (1966)[17], Lotz (1967)[18], McFarland (1967)[19], Jain and Khare (1976)[20], Green and Sawada (1972)[21] and Bell et al. (1982)[22].

Quantum mechanical calculations by Bethe[23], however, showed that the simple asymptotic behavior of σ_p (eq.7) is incorrect and it should vary as $\frac{\ln(E)}{E}$ at high electron impact energies. Based on Bethe's theory, Bell et al.[22] proposed an empirical formula for atoms and ions to fit the experimental data at all energies of the colliding electron. It is as follows:

$$\sigma_p(E) = \frac{1}{IE}\left[A \ln\left(\frac{E}{I}\right) + \sum_{i=1}^{N} a_i \left(1 - \frac{I}{E}\right)^i\right] \tag{8}$$

where A and a_i are fitting coefficients and all other quantities have been defined previously. The above formula takes care of the behavior of $\sigma_p(E)$ at high electron impact energies. The coefficient A can be calculated by fitting to the Bethe relation at high energies:

$$\sigma_p(E) = \frac{1}{IE}[A \ln(E) + B]. \tag{9}$$

It can also be obtained from the following relation:

$$A = \frac{1}{\pi\alpha}\int_I^\infty \frac{\sigma_{ph}}{E} dE \tag{10}$$

where σ_{ph} is the photon-ionization cross sections for CO, CO_2, CH_4, NH_3, and SO_2. For the sake of convenience of the modelers to utilize our data, we have parametrized them using equations (8) and (9). Experimental apparatus and procedures for obtaining these data are described in appendix I. Section II describes the fitting procedure, and in section III, results are presented and discussed.

II. PROCEDURE FOR PARAMETRIZATION

a) Methods

Cross sections for electron impact ionization of CO, CO_2, CH_4, NH_3, and SO_2 were previously measured in our laboratory. Equation (8) was then fitted to these cross sections by a χ^2 minimization technique. The details are described below. For in-depth discussion of the methodology, refer to Bevington[24].

For this purpose, we define a parameter χ^2 in the following way:

$$\chi^2 = \frac{1}{N-M} \sum_{i}^{N} \frac{1}{\sigma_i^2} [f_e(x_i) - f_t(x_i)]^2 , \tag{11}$$

where M is the number of parameters, N the number of data points, $\frac{1}{\sigma_i^2}$ the weight for each data point, $f_e(x_i)$ the experimental data at x_i, and $f_t(x_i)$ is the theoretical value at x_i, calculated from eq.(8). Since x_i's (experimental data) do not change in the fitting procedure, $f_t(x)$ is a function of a_i's alone, i.e., $f_t(x) \equiv f_t(a_1, a_2, \ldots, a_M)$. Consequently, χ^2 is also a function of the parameters a_i's alone,

$$\chi^2 \equiv \chi^2(a_1, a_2, \ldots, a_M).$$

Thus we can obtain a minimal value for χ^2, at least a locally minimal value, by manipulating a_i's on the a_i-space. There are many ways to minimize χ^2, two of which are used in this report. They are the gradient search and the linearization of the fitting function. These methods are described in detail below:

i) Gradient search

In the gradient search, the negative gradient of χ^2

$$-\nabla\chi_o^2 = \sum_i^M \delta a_i \vec{\mathbf{a}}_i, \tag{12}$$

is calculated at some point $(a_1^o, a_2^o, \ldots, a_M^o)$. The parameters a_i's are then incremented simultaneously by δa_i, that is in the direction of the negative gradient $-\nabla\chi_o^2$. $\vec{\mathbf{a}}_i$, in this case, is the unit orthogonal vector in the direction of a_i in the a_i-space, δa_i is the a_i-component of the gradient $-\nabla\chi_o^2$.

Expanding χ^2, using Taylor's series expansion, as a function of parameters a_i's, we have

$$\chi^2 = \chi_o^2 + \sum_{j=1}^M \left[\frac{\partial\chi_o^2}{\partial a_j}\delta a_j\right]. \tag{13}$$

The optimum values for parameter increments, δa_j's, are those for which χ^2 is a minimum, that is when

$$\frac{\partial \chi^2}{\partial a_k} = \frac{\partial \chi^2_o}{\partial a_k} + \sum_{j=1}^{M} \left[\frac{\partial^2 \chi^2_o}{\partial a_j \partial a_k} \delta a_j \right] = 0, \qquad k = 1 \ldots M \tag{14}$$

We have, from eq.(14), a set of M simultaneous linear equations in δa_j's. To solve for δa_j's, we first express these equations in matrix form. Letting

$$\beta = \begin{pmatrix} \frac{\partial \chi^2_o}{\partial a_1} \\ \vdots \\ \frac{\partial \chi^2_o}{\partial a_M} \end{pmatrix},$$

$$\delta \mathbf{a} = \begin{pmatrix} \delta a_1 \\ \vdots \\ \delta a_M \end{pmatrix}, \tag{15}$$

$$\alpha = \begin{pmatrix} \frac{\partial^2 \chi^2_o}{\partial a_1 \partial a_1} & \cdots & \frac{\partial^2 \chi^2_o}{\partial a_M \partial a_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 \chi^2_o}{\partial a_1 \partial a_M} & \cdots & \frac{\partial^2 \chi^2_o}{\partial a_M \partial a_M} \end{pmatrix};$$

and substituting eq.(15) into eq.(14), we have

$$\beta = \delta \mathbf{a} \cdot \alpha. \tag{16}$$

Multiplying both sides of eq.(16) by α^{-1} we have a matrix of the parameter increments $\delta \mathbf{a}$:

$$\delta \mathbf{a} = \beta \cdot \alpha^{-1}. \tag{17}$$

ii) Linearization of the fitting function.

Instead of expanding χ^2 as in the gradient search method, we expand $f_t(x)$ as a function of parameters a_i's,

$$f_t(x) = f_o(x) + \sum_{j=1}^{M} \left[\frac{\partial f_o(x)}{\partial a_j} \delta a_j \right]. \tag{18}$$

Together with eq.(11), χ^2 can then be expressed as,

$$\begin{aligned} \chi^2 &= \frac{1}{N-M} \sum_{i}^{N} \frac{1}{\sigma_i^2} [f_e(x_i) - f_t(x_i)]^2 \\ &= \frac{1}{N-M} \sum_{i}^{N} \frac{1}{\sigma_i^2} \left\{ f_e(x_i) - f_o(x_i) - \sum_{j=1}^{M} \left[\frac{\partial f_o(x)}{\partial a_j} \delta a_j \right] \right\}^2 . \end{aligned} \tag{19}$$

Minimizing χ^2 with respect to δa_k, we have

$$\begin{aligned} \frac{\partial \chi^2}{\partial (\delta a_k)} &= \frac{-2}{N-M} \sum_{i}^{N} \frac{1}{\sigma_i^2} \left\{ f_e(x_i) - f_o(x_i) - \sum_{j=1}^{M} \left[\frac{\partial f_o(x_i)}{\partial a_j} \delta a_j \right] \right\} \frac{\partial f_o(x_i)}{\partial a_k} \\ &= 0, \qquad\qquad k = 1 \ldots M \end{aligned} \tag{20}$$

As before, we have $\beta = \delta \mathbf{a} \cdot \alpha$, in matrix form, or in a more explicit form,

$$\begin{aligned} \beta_k &= \sum_{i}^{N} \frac{1}{\sigma_i^2} [f_e(x_i) - f_o(x_i)] \frac{\partial f_o(x_i)}{\partial a_k} \\ &= \frac{\partial}{\partial a_k} (\chi_o^2) \end{aligned} \tag{21}$$

and

$$\alpha_{jk} \approx \sum_{i}^{N} \left[\frac{1}{\sigma_i^2} \frac{\partial f_o(x_i)}{\partial a_j} \frac{\partial f_o(x_i)}{\partial a_k} \right] \qquad j,k = 1..M. \tag{22}$$

b) Advantages and disadvantages of the above two methods

Theoretically, the gradient search will take the search to the minimal point of χ^2. However, from a computational point of view, it presents some difficulties. Computing the gradient of χ^2 requires the computer to go through many calculations. Thus at a near-minimal point, where the increments δa_j's are considerably small, the gradient search has to be utilized many times and thus renders the method inefficient. The linearization of the fitting function, on the other hand, gives good results near the minimum point. However, by the nature of the Taylor's expansion, this method is unreliable when χ_o^2 is too far away from the minimum point.

c) Gradient-Expansion search method

It is obvious then that these two methods are complementary. As discussed in the last section, the gradient search is effective when χ_o^2 is at a point far away from the minimum point, and the linearization method is effective nearby. The best features of these two methods can be incorporated into one by increasing the diagonal terms of matrix α by a factor of λ, so that:

$$\beta = \delta \mathbf{a} \cdot \alpha, \tag{23}$$

where

$$\alpha_{jk} = \begin{cases} \alpha_{jk}(1+\lambda), & \text{if j} = \text{k}, \\ \alpha_{jk}, & j \neq k. \end{cases} \tag{24}$$

If λ is small, the above equation is similar to that obtained from the linearization method. If λ is large, the diagonal terms dominate the matrix, thus we have

$$\beta_i \approx \lambda \cdot \delta \mathbf{a}_i \cdot \alpha_{ii} \tag{25}$$

which yields similar solutions to those obtained from the gradient search method.

d) Algorithm

The above search method can be summarized as follows:

1) *Set* $\lambda = 0.0001$*; Compute* $\chi^2(\mathbf{a})$,

2) *Compute* $\delta\mathbf{a}$ *and* $\chi^2(\mathbf{a}+\delta\mathbf{a})$,

3) *If* $\chi^2(\mathbf{a}+\delta\mathbf{a}) > \chi^2(\mathbf{a})$, *increase* λ *by a factor of 10 and repeat step 2,*

4) *If* $\chi^2(\mathbf{a}+\delta\mathbf{a}) < \chi^2(\mathbf{a})$, *decrease* λ *by a factor of 10 and return to repeat step 2, substituting* $\mathbf{a}+\delta\mathbf{a}$ *for* $\mathbf{a}$,

where $\mathbf{a} = (a_1, a_2, \ldots, a_M)$, and $\delta\mathbf{a} = (\delta a_1, \delta a_2, \ldots, \delta a_M)$

The above steps can be terminated when the difference of $\chi^2(\mathbf{a}+\delta\mathbf{a})$ and $\chi^2(\mathbf{a})$ is smaller than some predetermined constant; 0.0001 was used in this report.

The programs, using the methods discussed above, were written in Pascal language and were executed on the HP9836C and Apple IIe computers and implemented by Caltech's CS10 graphics library. The program for HP9836C is listed in appendix II. Flow diagrams from these programs are given in Figures 1 and 2.

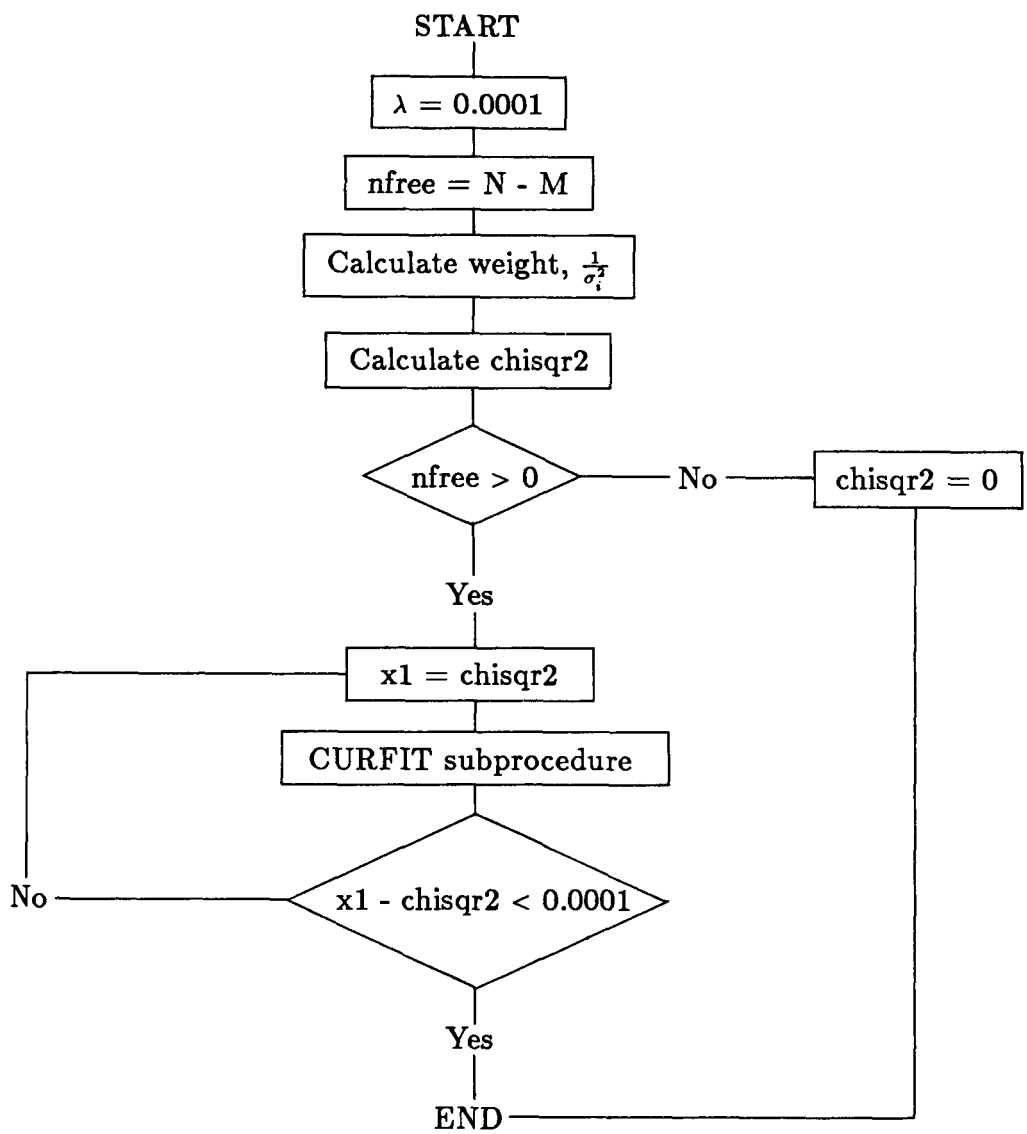

Figure 1. Flow diagram

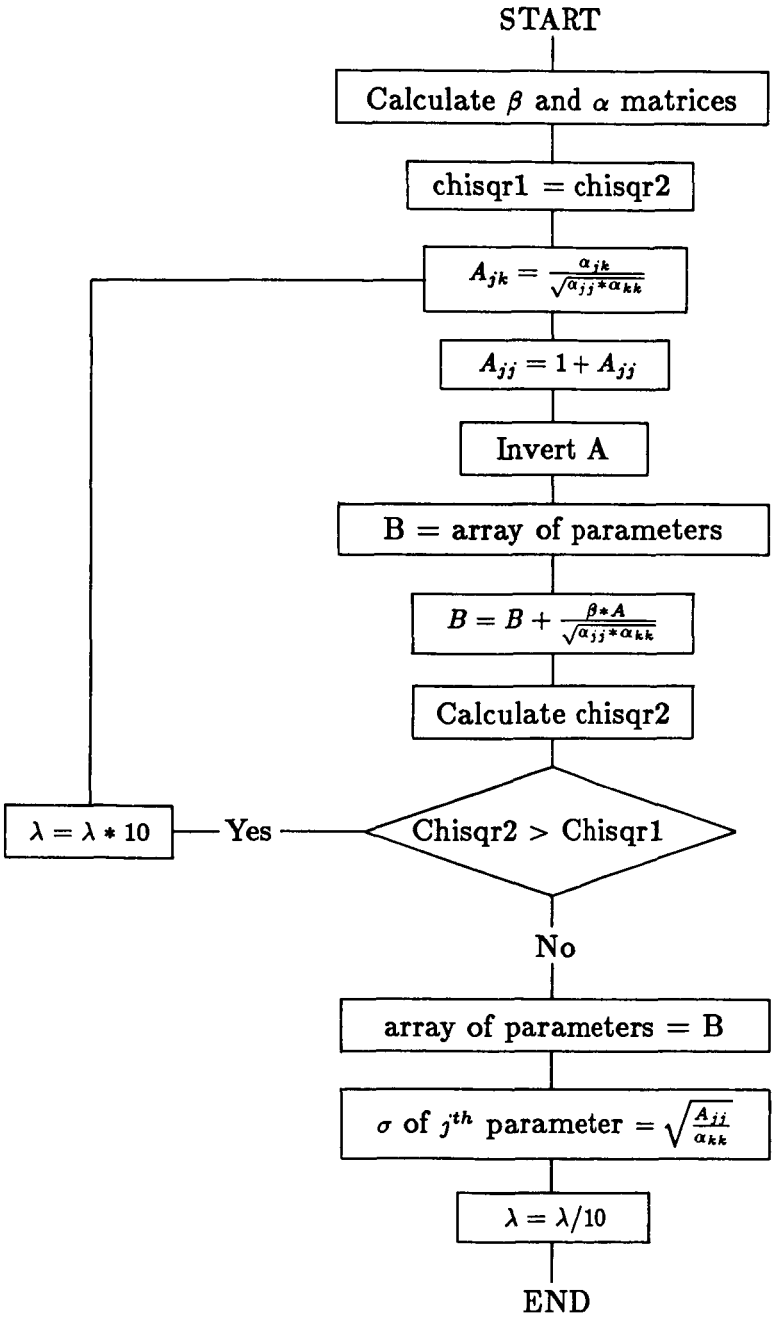

Figure 2. Subprocedure CURFIT

III. RESULTS AND DISCUSSION

Utilizing eq.(8) and the fitting procedure described in section II, the values of coefficients A and a_i's were determined. They are presented in Table I. Fitted data are also shown in graphical forms in figures 2 through 26. Except for SO_2, the experimental measurements were up to 510 eV electron impact energy. The data uncertainty was about 15%. The error bars are shown in all figures. In the cases of CO, CO_2, CH_4, and NH_3, eq.(8) has provided an excellent fit to the experimental data. However, the data for SO_2 are only up to 200 eV and the fits are not very satisfactory.

IV. ACKNOWLEDGEMENT

We would like to thank the SURF program at Caltech for providing a financial grant to one of us (HPN). We would also like to thank Mr. C. Thoms and Dr. E. Krishnakumar for their help and discussion during the course of this work. The research described in this report was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by SURF, AFOSR, and the National Aeronautics and Space Administration.

V. REFERENCES

1. L. J. Kieffer and G. H. Dunn, Rev. Mod. Phys. *38*, 1 (1986).

2. T.D. Märk, in *Electron Impact Ionization*, Springer Verlag ed. T. D. Märk and G. H. Dunn (1985); pp.137.

3. F. J. de Heer and M. Inokuti, Ibid; pp.232.

4. T. D. Märk, in *Electron-Molecule Interactions and Their Application*, Academic Press Inc. (1984), ed. L.G. Christophoron; pp.251.

5. S. M. Younger, in *Electron Impact Ionization*, Springer Verlag, ed. T. D. Märk and G. H. Dunn(1985); pp.1.

6. M. R. Rudge, Rev. Mod. Phys. *40*, 564 (1968).

7. J. J. Thomson, Phil. Mag. *23*, 449 (1912).

8. G. Elwert, Z. Naturforchg *7a*, 432(1952).

9. M. Gryzinski, Phys. Rev. *115*, 374 (1959).

10. R. F. Post, Plasma Physics *3*, 273 (1961).

11. H. W. Drawin, Z. Physik *164*, 513 (1961).

12. A. Burgess, Proc. 3rd ICPEAC, London, 237 (1963).

13. R. C. Stabler, Phys. Rev. *A133*, 1268 (1964).

14. M. J. Seaton, Planet. Space Science *12*, 55 (1964).

15. M. Gryzinski, Phys. Rev. *A138*, 305 (1965).

16. M. Gryzinski, Phys. Rev. Lett. *14*, 1059 (1965).

17. l. Vriens, Phys. Rev. *141*, 88 (1966).

18. W. Lotz, Astrophys. J.,Suppl. *14*, 207 (1967).

19. R. H. McFarland, Phys. Rev. *159*, 20 (1967).

20. D. K. Jain and S. P. Khare, J. Phys. *B9*, 1429 (1976).

21. A. E. S. Green and T. Sawada, J. Atm. Terr. Phys. *34*, 1719 (1972).

22. K. L. Bell, H. B. Gilbody, J. G. Hughs, A. E. Kingston, and F. J. Smith; J. Phys. Chem. Ref. Data *12*, 891 (1983).

23. H. Bethe, Ann. Physik *5*, 325 (1930); also Z. Physik *76*, 293 (1932).

24. P. R. Bevington, *Data Reduction and Error Analysis for the Physical Sciences*, New York: McGraw-Hill (1969); pp.204.

25. S. K. Srivastava, A. Chutjian, and S. Trajmar, J. Chem. Phys. *63*, 2659 (1975).

26. D. Rapp and P. Englander-Golden, J. Chem. Phys. *43*, 1464 (1965).

27. K. Stephan, H. Helm, and T. D. Märk, J. Chem. Phys. *73*, 3763 (1980).

28. T. C. Elhert, J. Phys. *E3*, 237 (1970).

29. O. J. Orient and S. K. Srivastava, J. Chem. Phys. *78*, 2949 (1983).

TABLE I

The parameters A and a_i's of (8) for the fitted cross sections. These parameters are in the units of $10^{-14} eV^2 \cdot cm^2$.

Mol.	Prod.	A	a_1	a_2	a_3	a_4	a_5	a_6
	CO_2^+	65.459	-65.00	-22.427	-26.250			
	CO^+	3.849	-3.203	-3.138	6.189	-6.089		
CO_2	C^+	1.433	-1.262	-2.514	6.436	-3.055		
	O^+	9.355	-9.252	-5.309	8.227	-11.275		
	Total	77.266	-75.035	-46.899	27.582	-41.401		
	CO^+	49.342	-46.397	-39.366	45.765	-51.574		
CO	C^+	6.781	-6.653	5.229	7.254	-11.443		
	O^+	2.384	-2.368	0.984	-10.776	29.389	-18.058	
	Total	52.961	-51.003	-35.790	24.074	-31.205		
	CH_4^+	25.660	-24.545	-9.057	-1.971			
	CH_3^+	30.133	-27.286	-43.680	169.936	-276.50	140.548	
CH_4	CH_2^+	3.428	-2.609	-7.047	14.061	-7.379		
	CH^+	0.819	0.0684	-12.478	88.954	-219.99	245.070	-100.40
	C^+	-0.0435	0.330	-1.828	24.783	-74.772	100.194	-47.281
	Total	60.704	-54.546	-82.229	287.768	-447.52	225.250	
	NH_3^+	24.677	-23.387	-8.199	-35.344	65.208	-48.343	
	NH_2^+	57.455	-58.280	-1.297	-31.188			
NH_3	NH^+	4.780	-4.231	-2.982	-7.752	31.153	-19.699	
	N^+	2.359	-1.796	-10.851	68.642	-186.8	233.891	-104.63
	Total	60.008	-59.746	-15.417	-88.234	113.977	-71.625	
	SO_2^+	66.607	-63.735	-75.659	331.36	-674.18	472.058	-91.948
	SO^+	-9.180	11.802	-46.235	403.755	-730.14	454.224	
SO_2	S^+	29.372	-24.266	-57.076	196.126	-329.76	173.684	
	O^+	2.010	-1.707	-2.537	9.661			
	Total	80.341	-77.366	5.886	-311.92	1414.13	-2309.7	1252.17

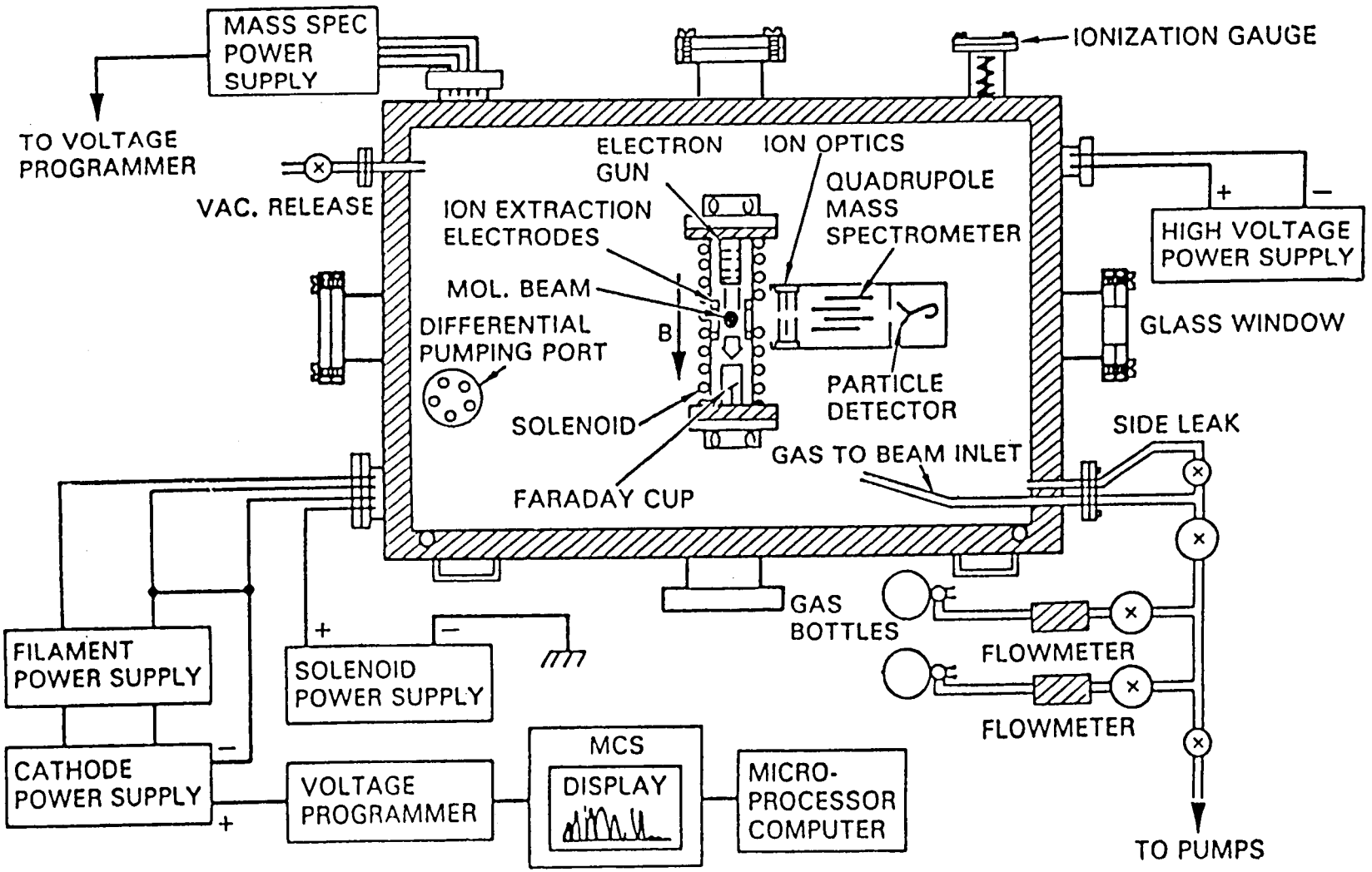

Figure 3. Dissociative ionization and attachment spectrometer (dimensions not to scale)

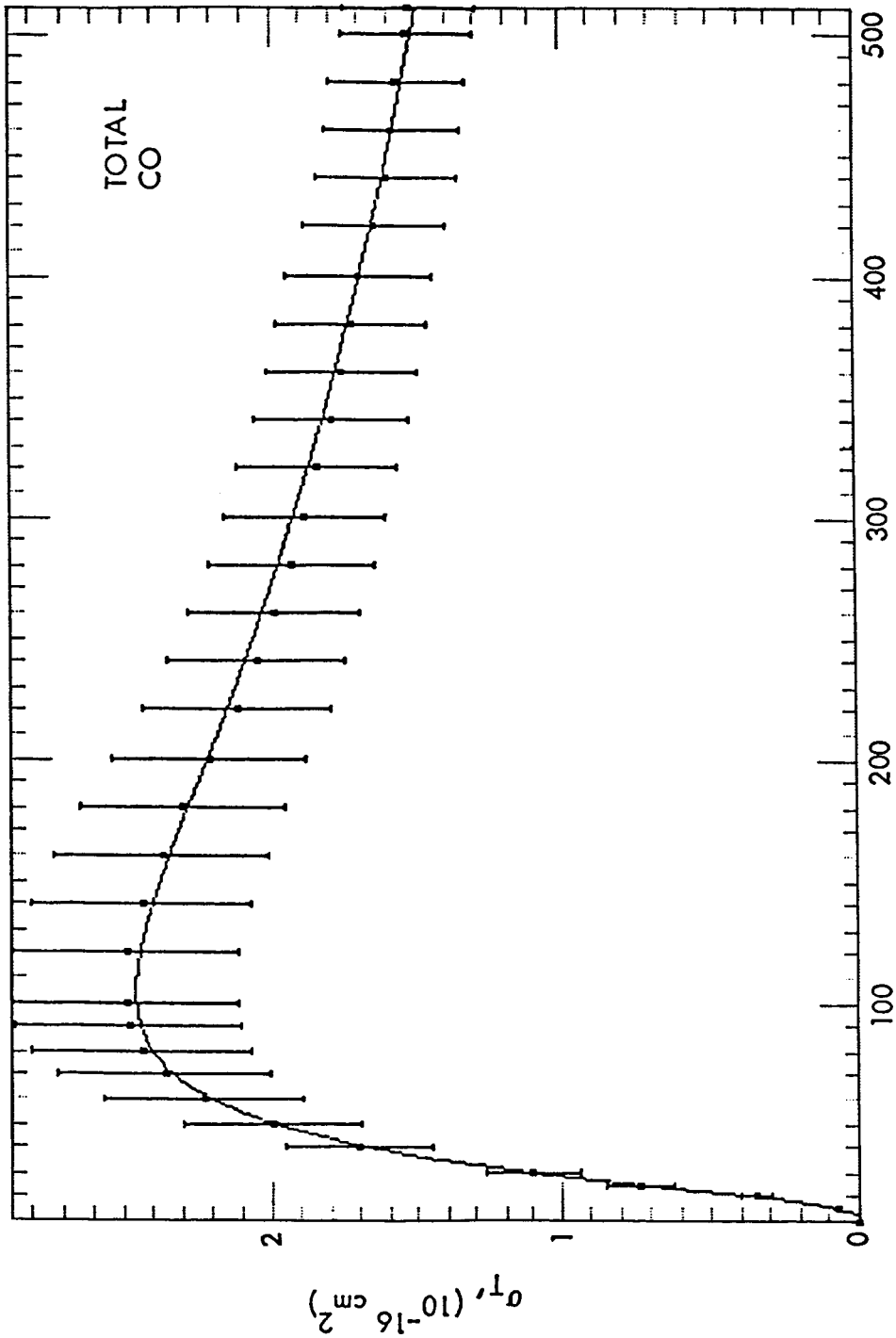

Figure 4. Total electron impact ionization cross section for CO

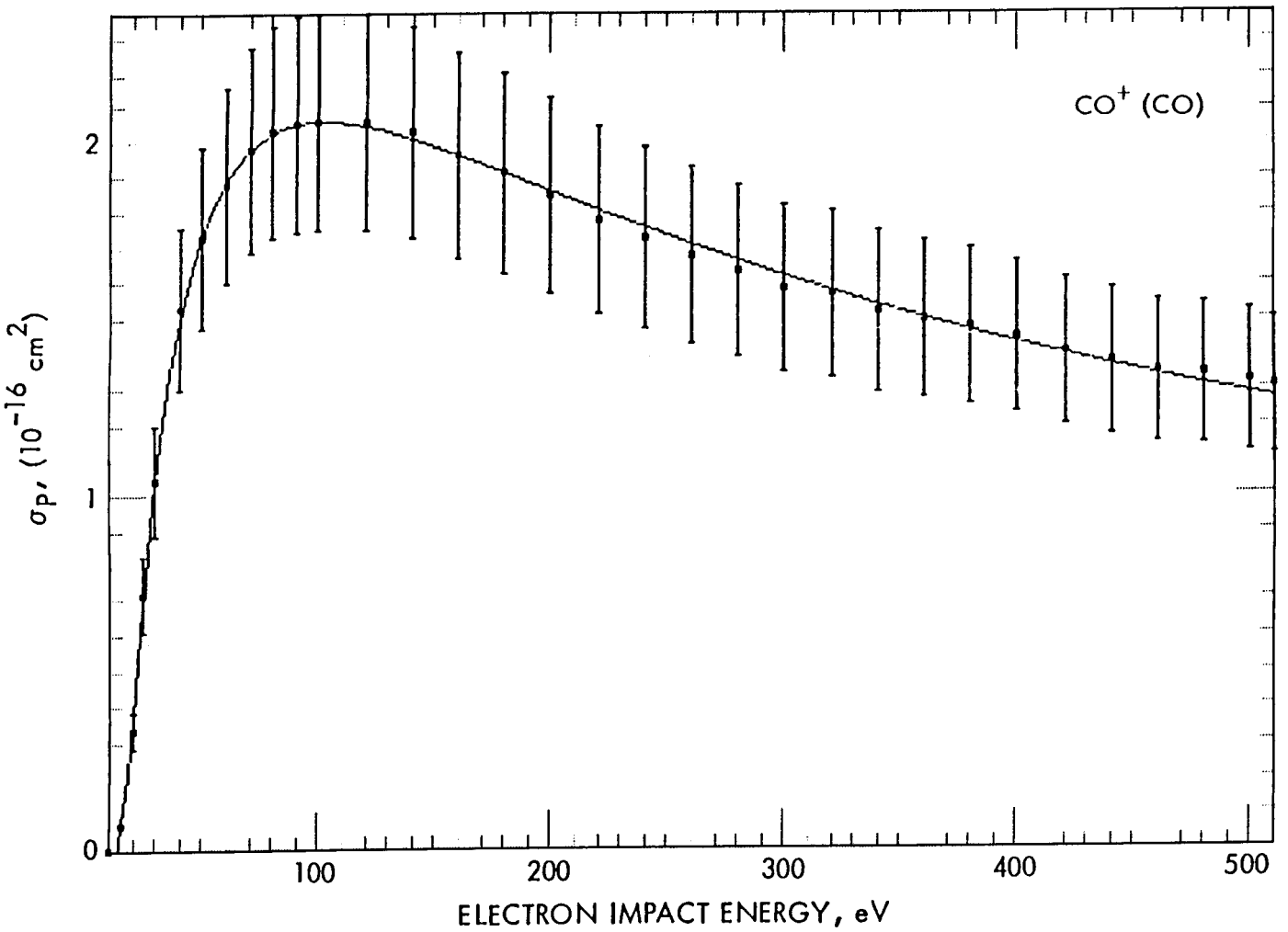

Figure 5. Electron impact ionization cross section for the production of CO^+ from CO

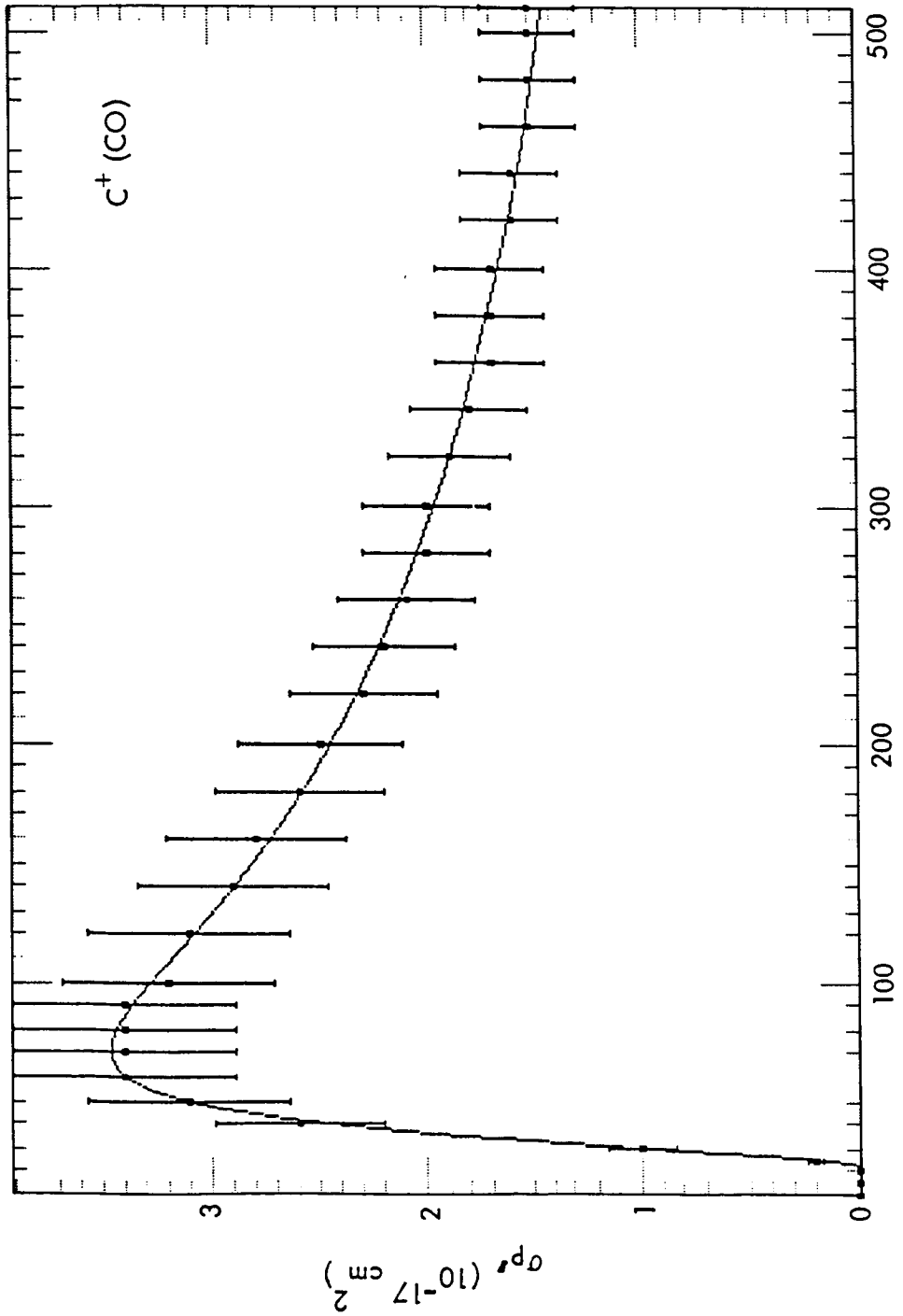

Figure 6. Electron impact ionization cross section for the production of C^+ from CO

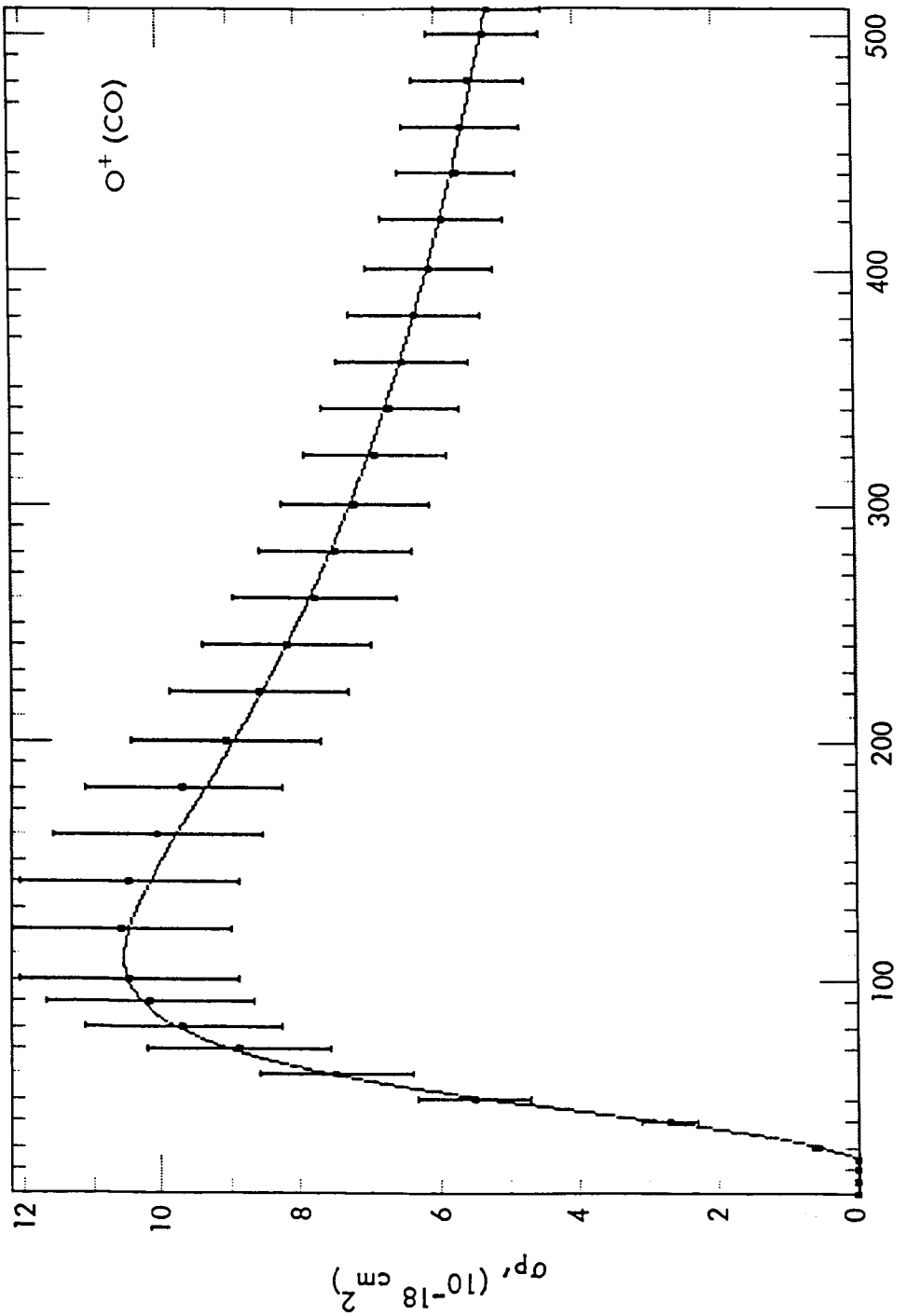

Figure 7. Electron impact ionization cross section for the production of O^+ from CO

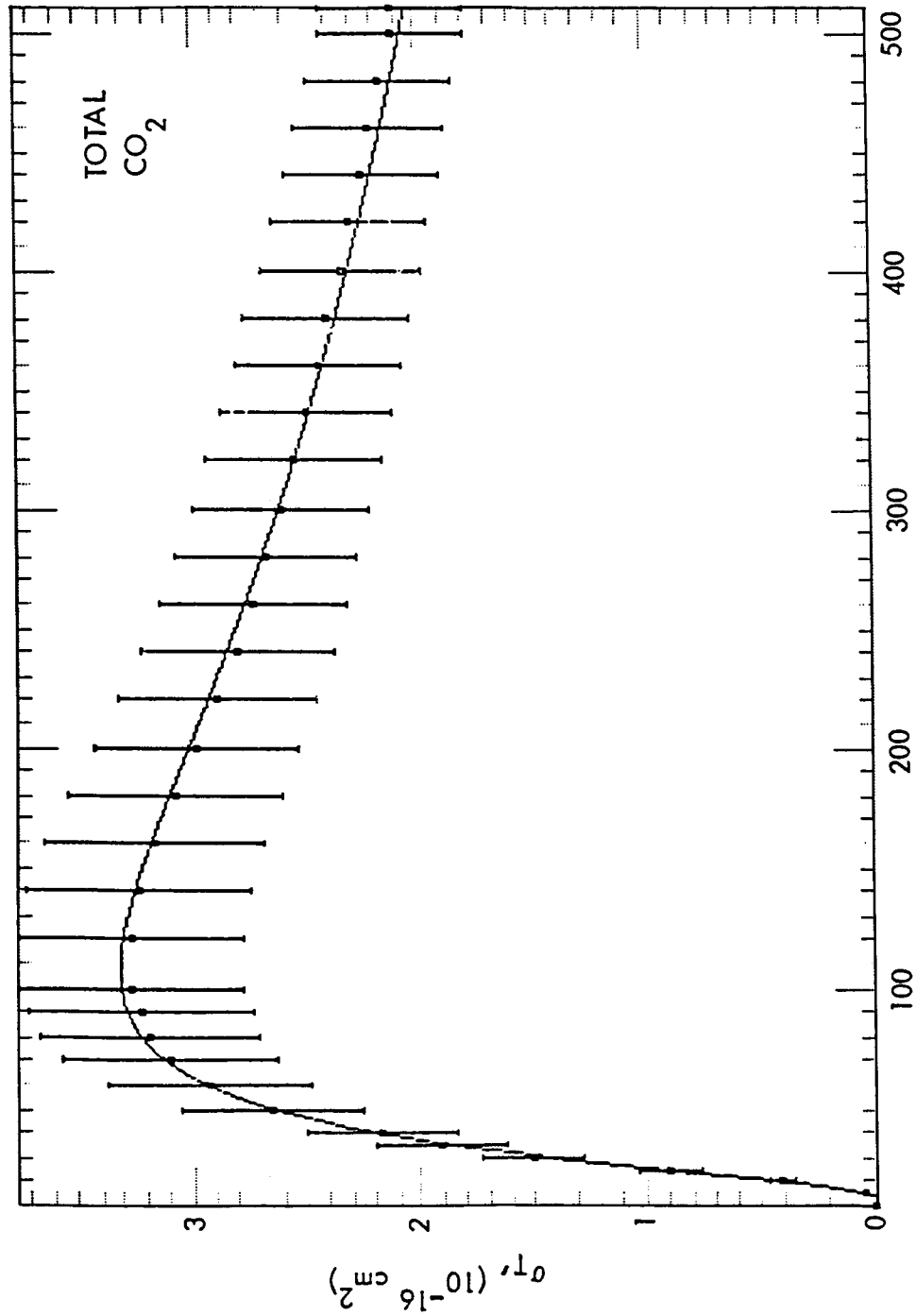

Figure 8. Total electron impact ionization cross section for CO_2

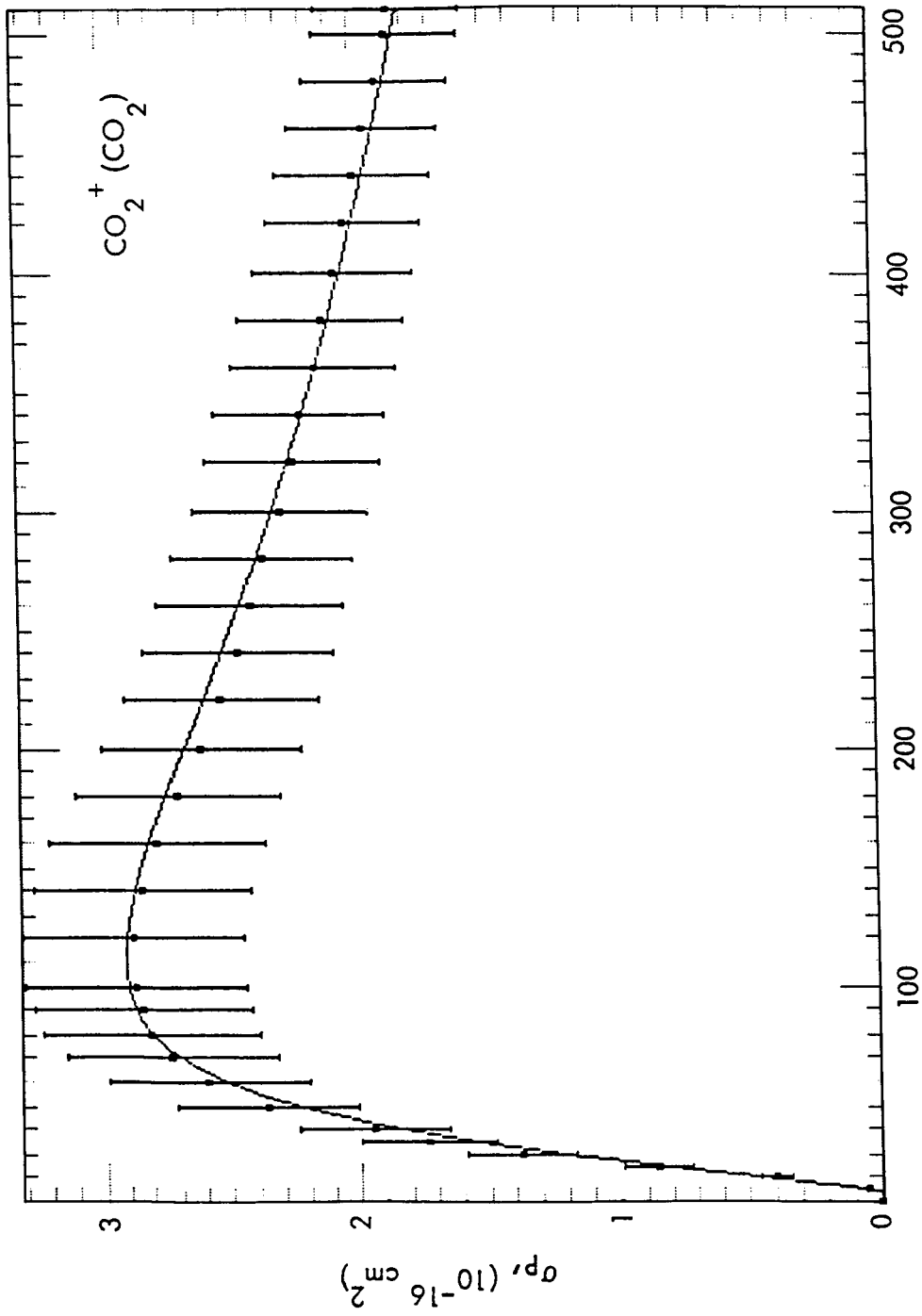

Figure 9. Electron impact ionization cross section for the production of CO_2^+ from CO_2

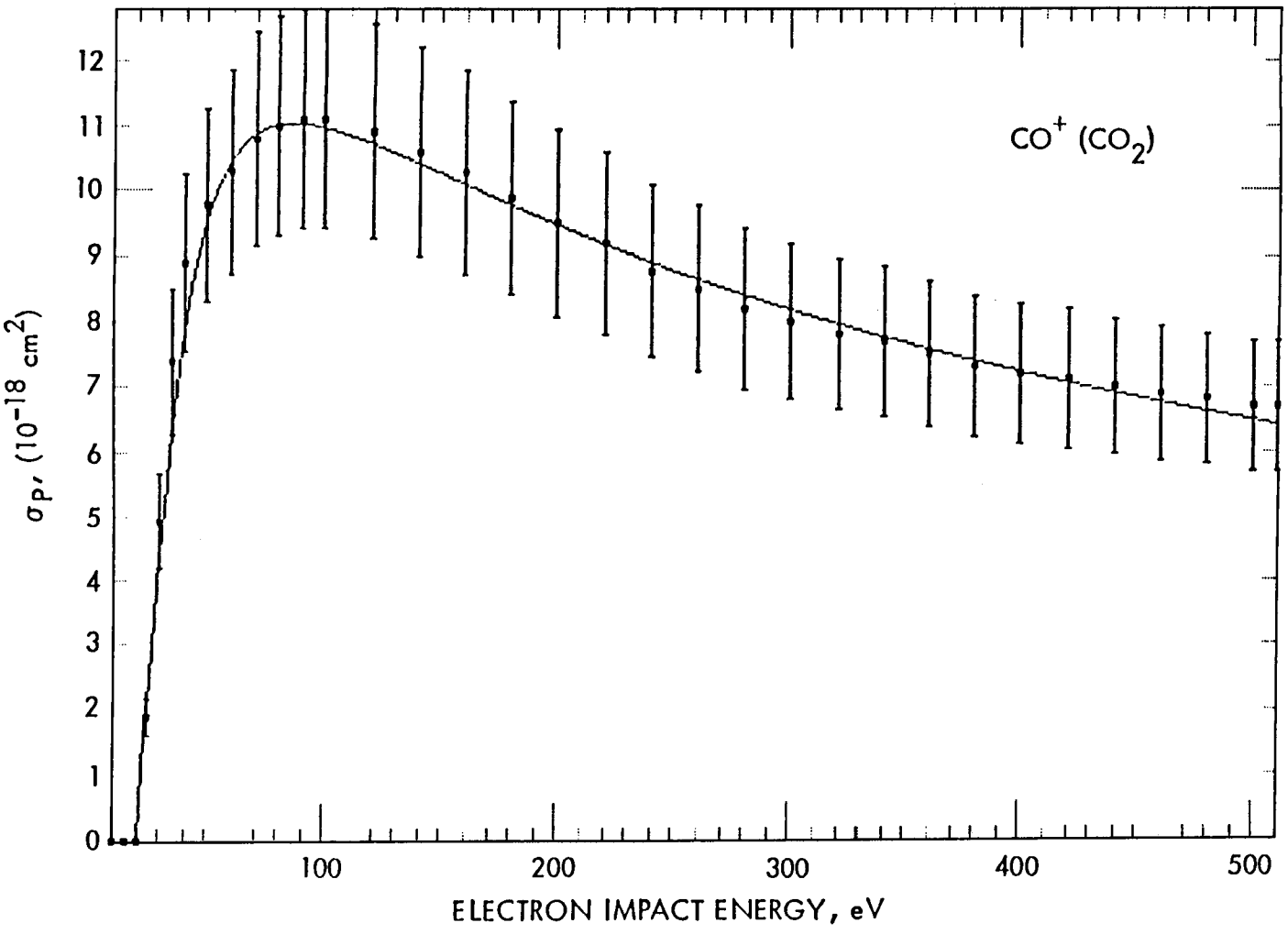

Figure 10. Electron impact ionization cross section for the production of CO^+ from CO_2

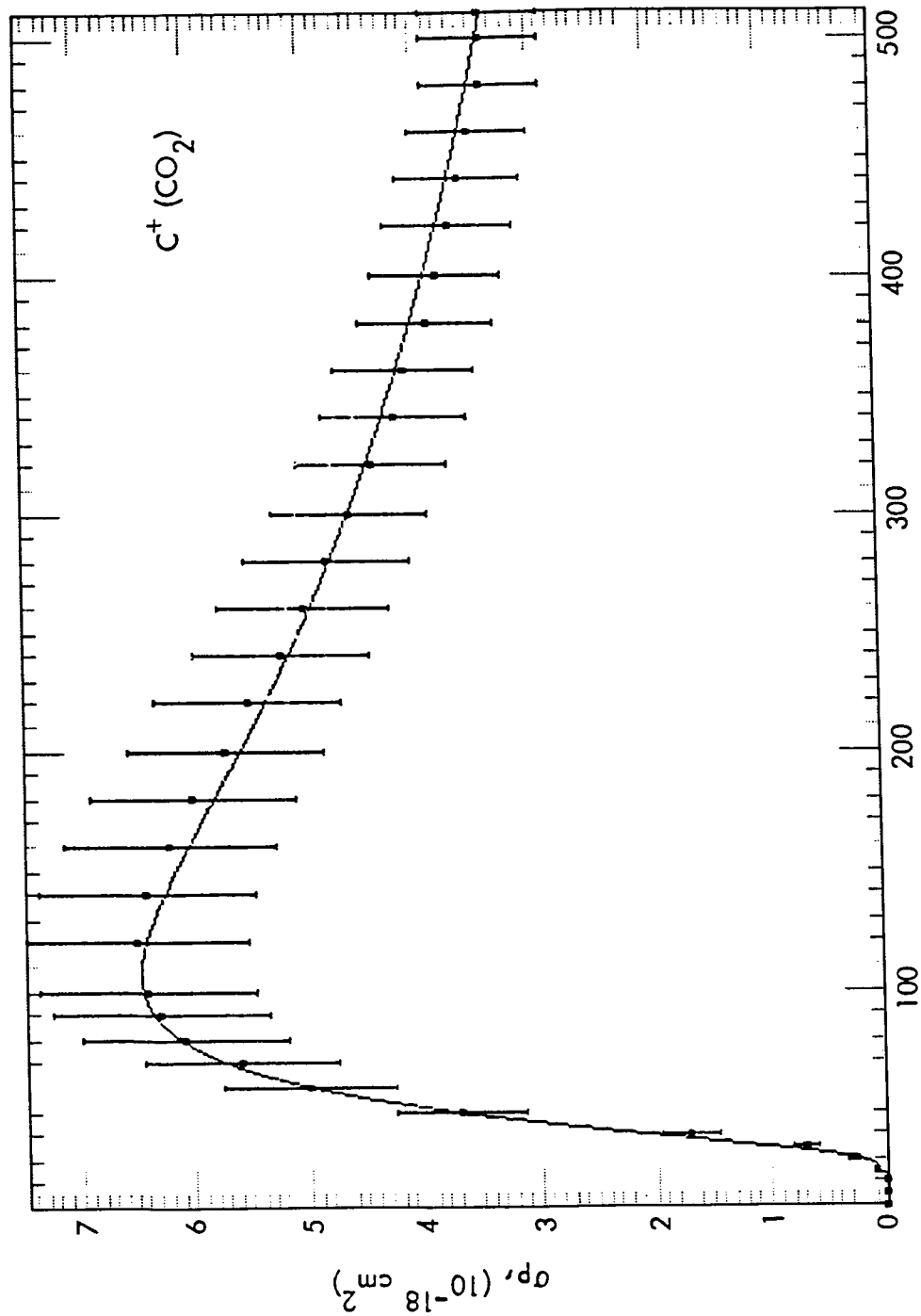

Figure 11. Electron impact ionization cross section for the production of C^+ from CO_2

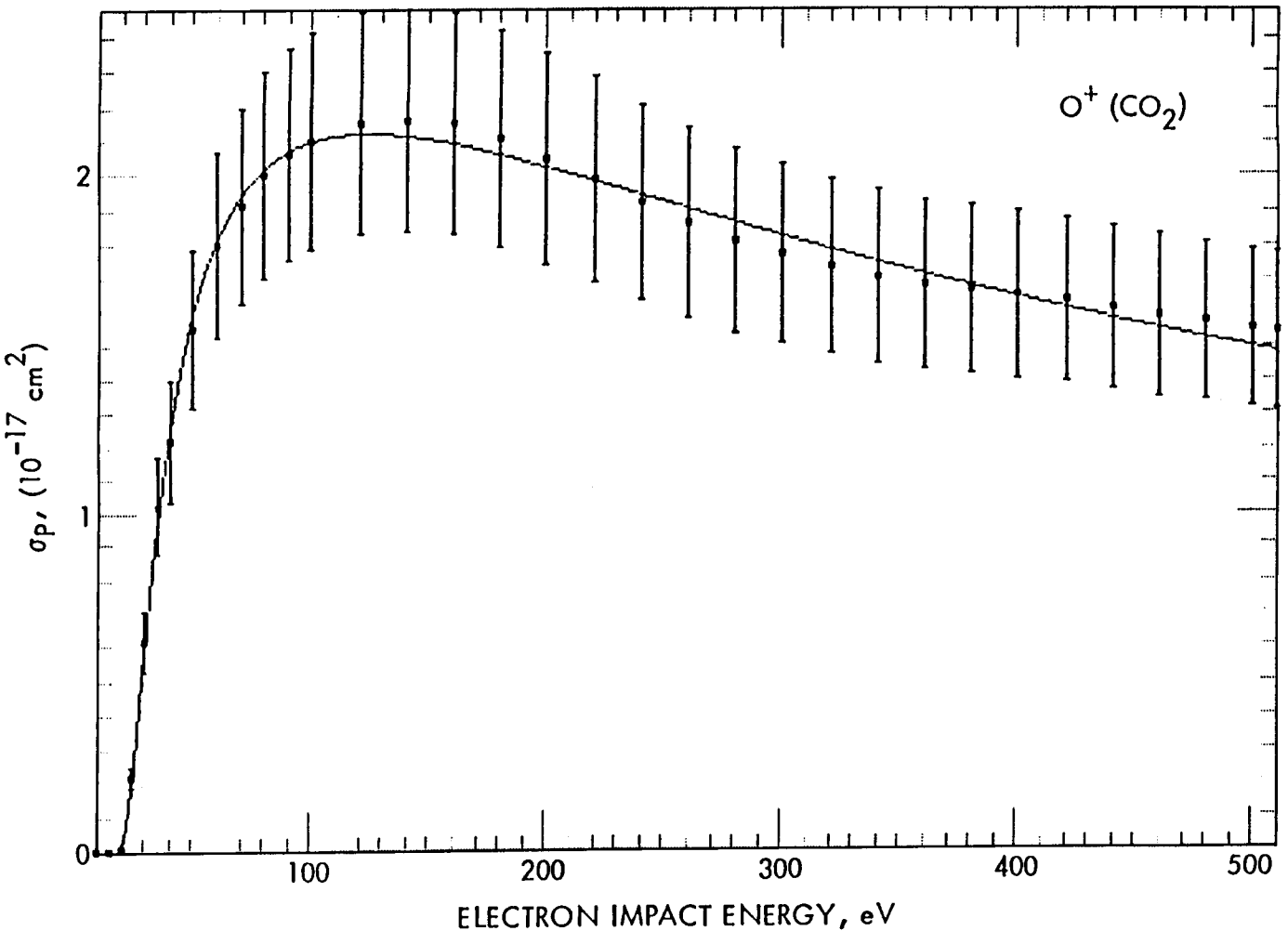

Figure 12. Electron impact ionization cross section for the production of O^+ from CO_2

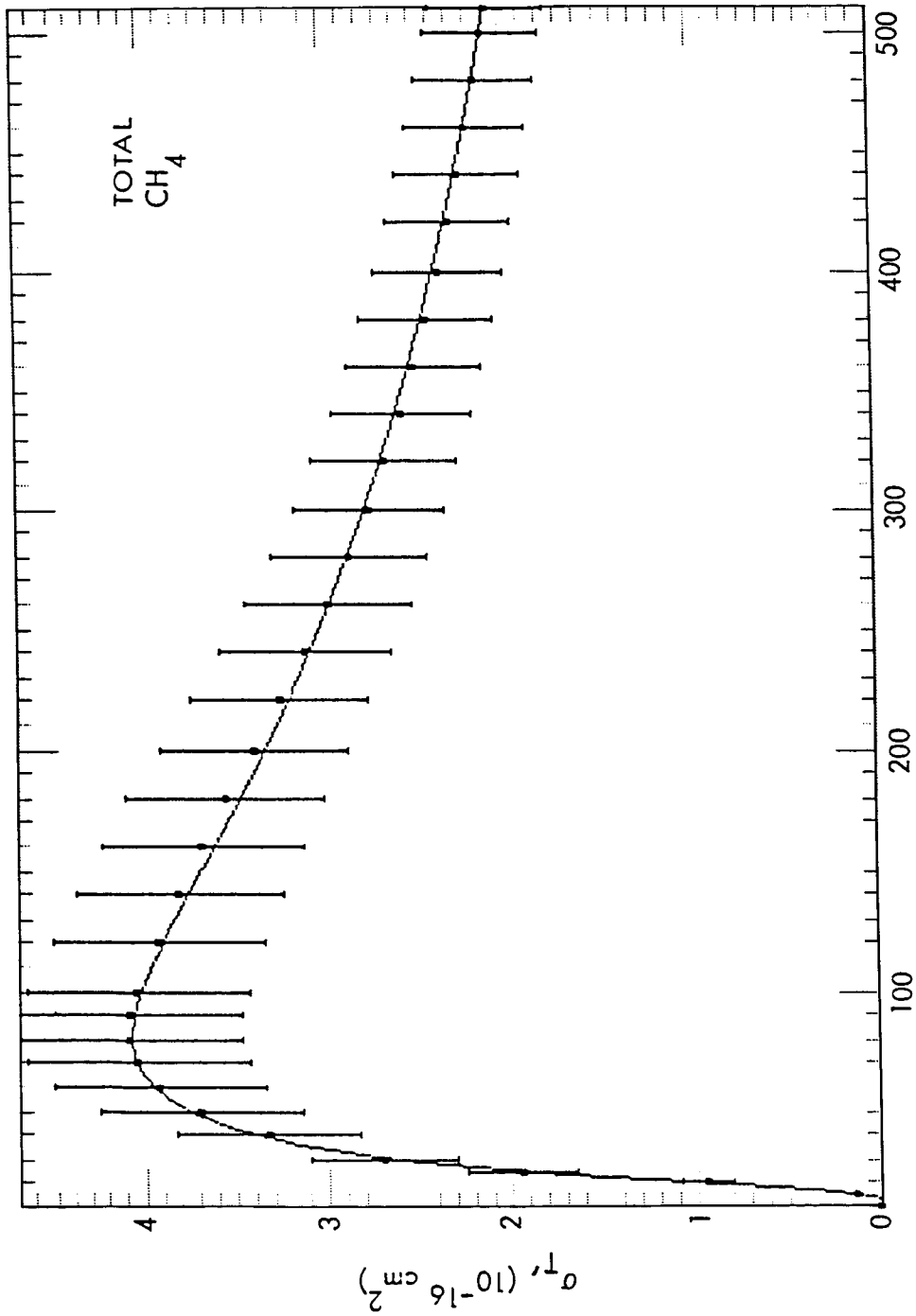

Figure 13. Total electron impact ionization cross section for CH_4

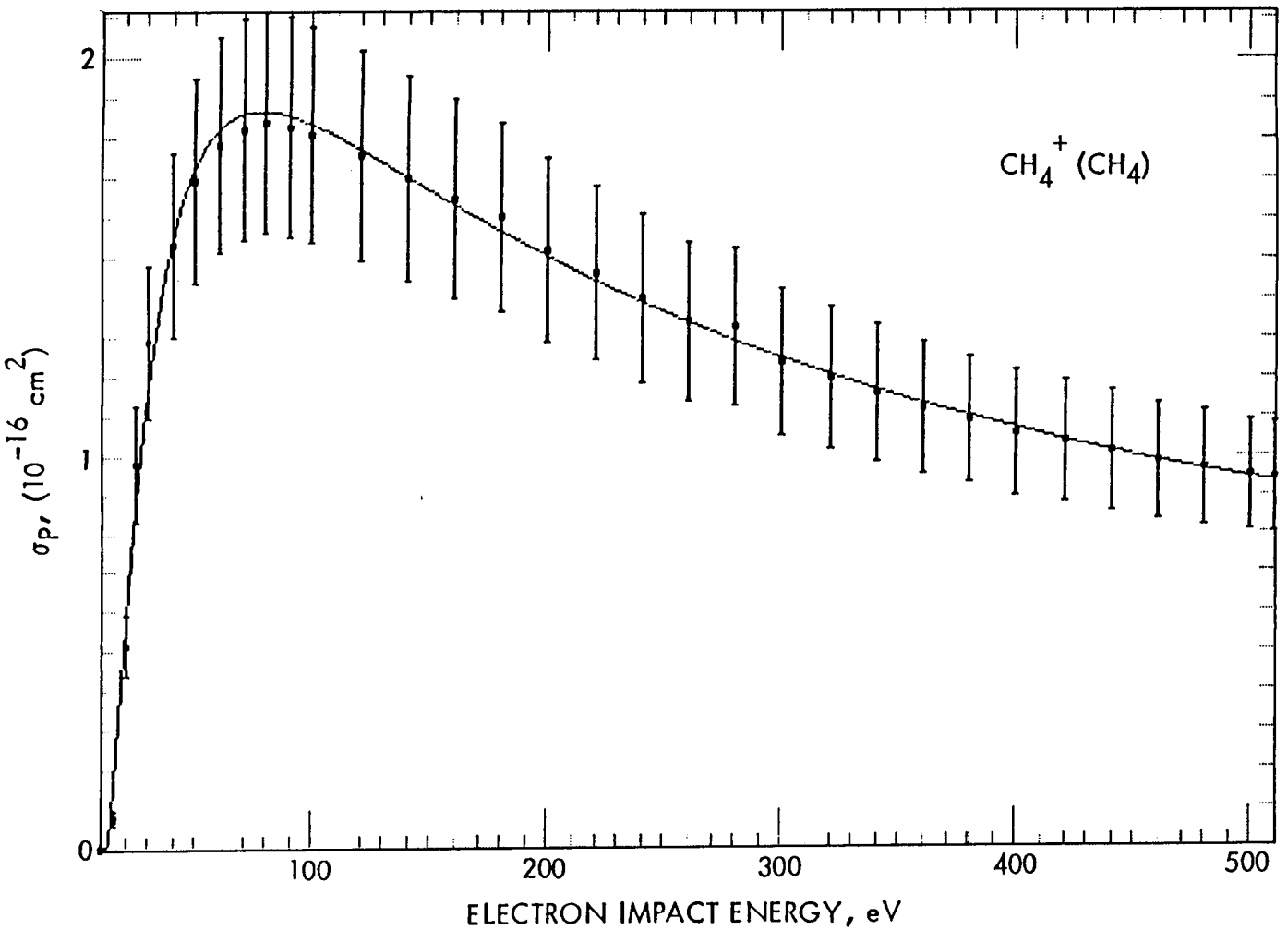

Figure 14. Electron impact ionization cross section for the production of CH_4^+ from CH_4

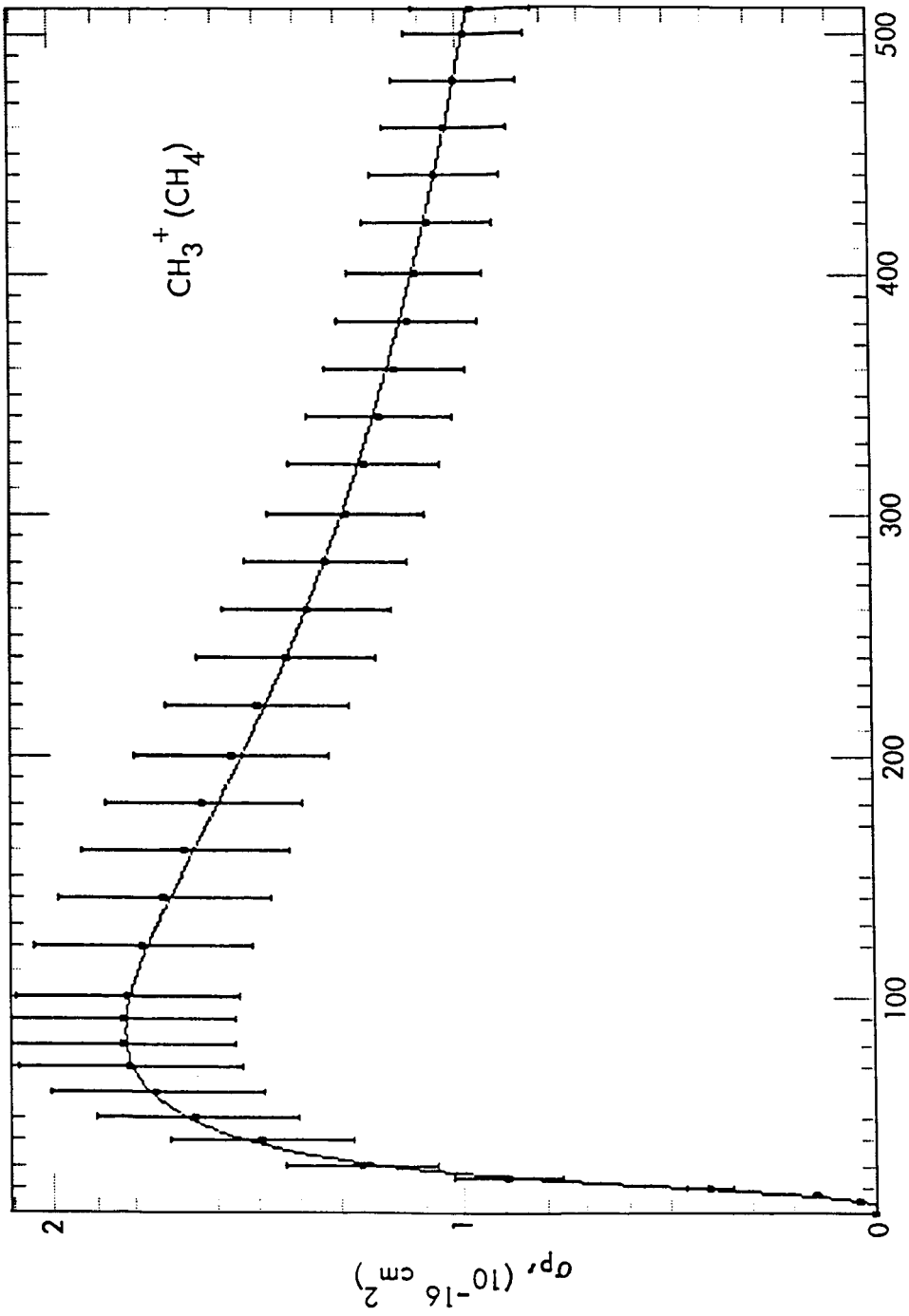

Figure 15. Electron impact ionization cross section for the production of CH_3^+ from CH_4

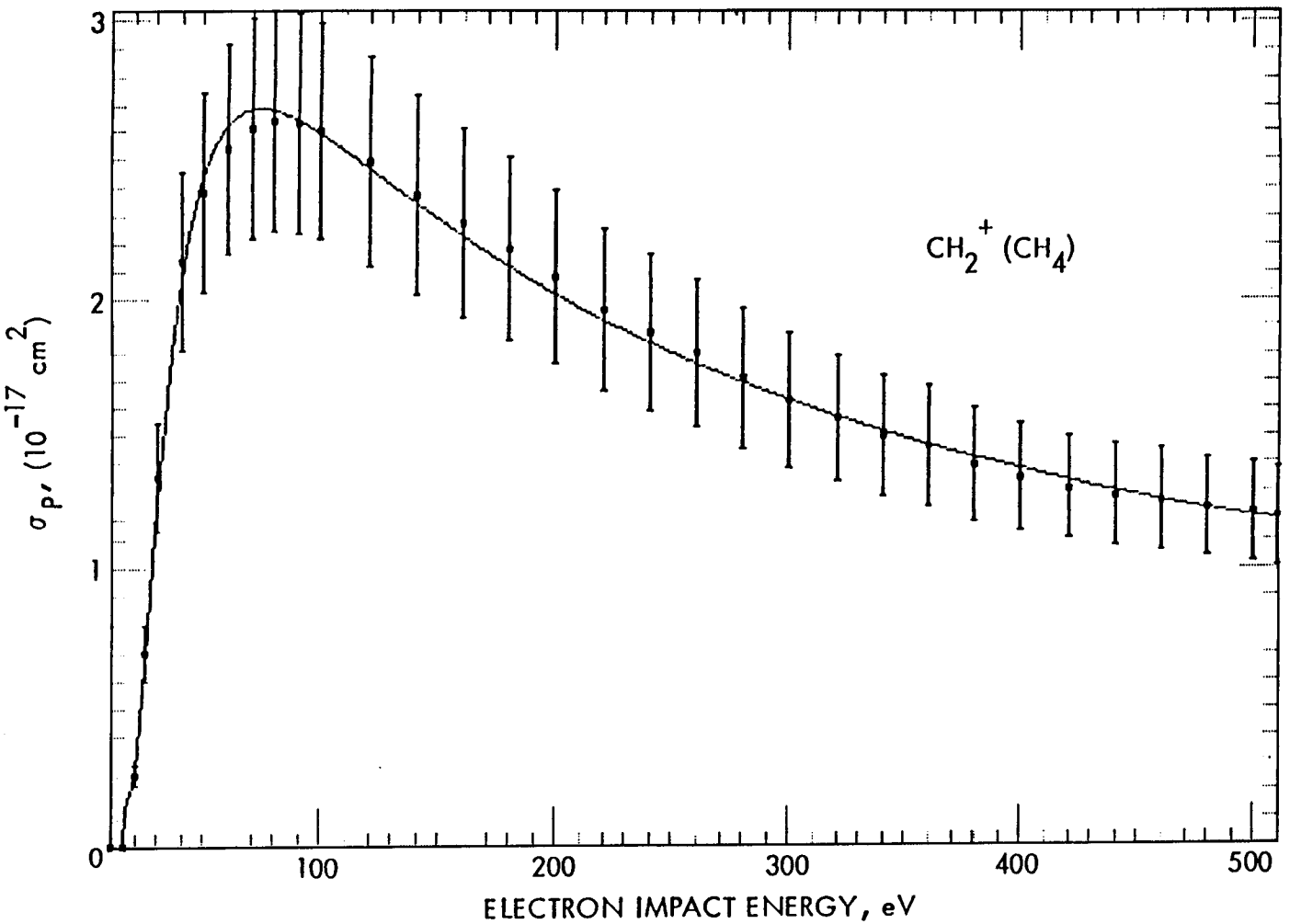

Figure 16. Electron impact ionization cross section for the production of CH_2^+ from CH_4

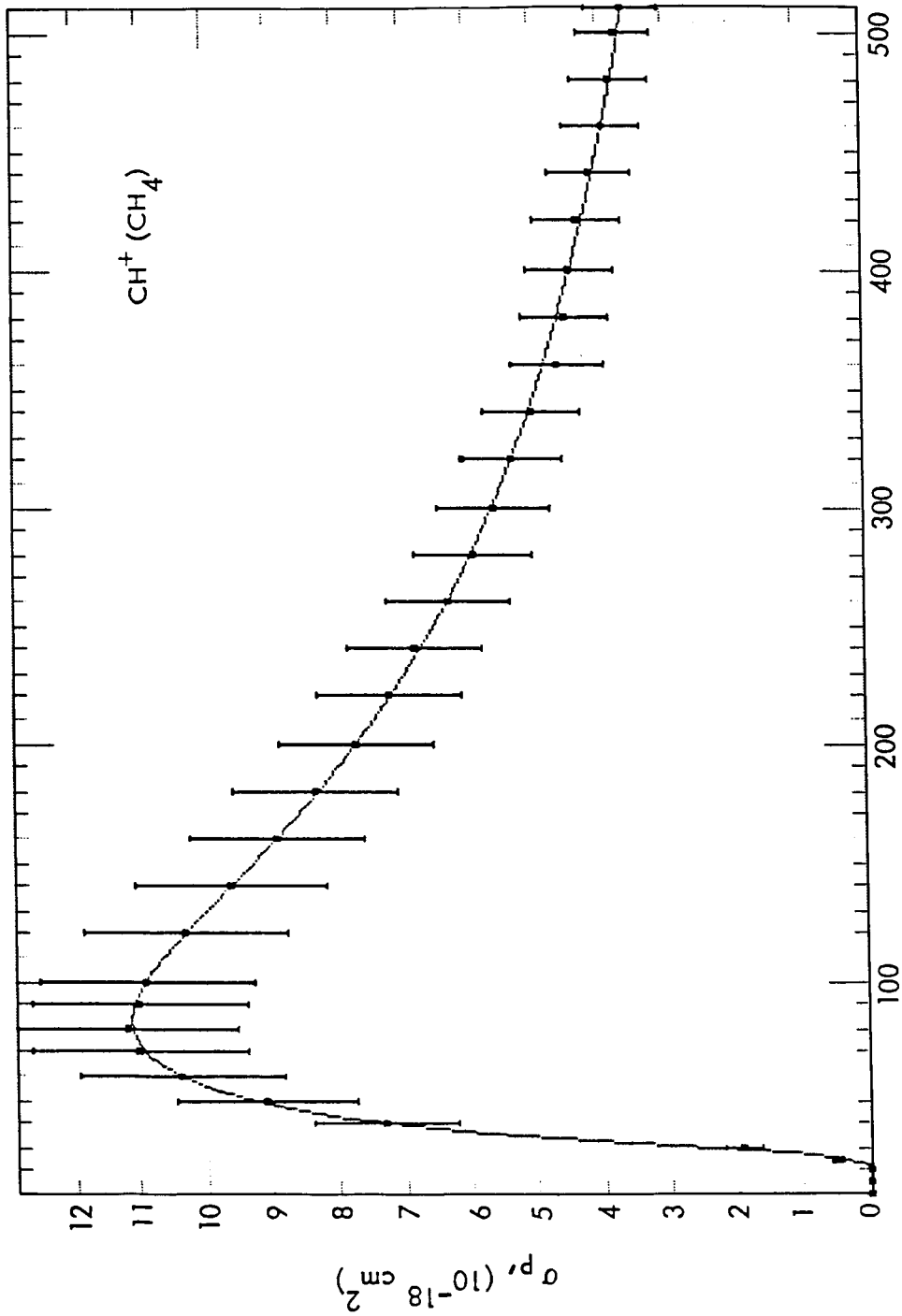

Figure 17. Electron impact ionization cross section for the production of CH^+ from CH_4

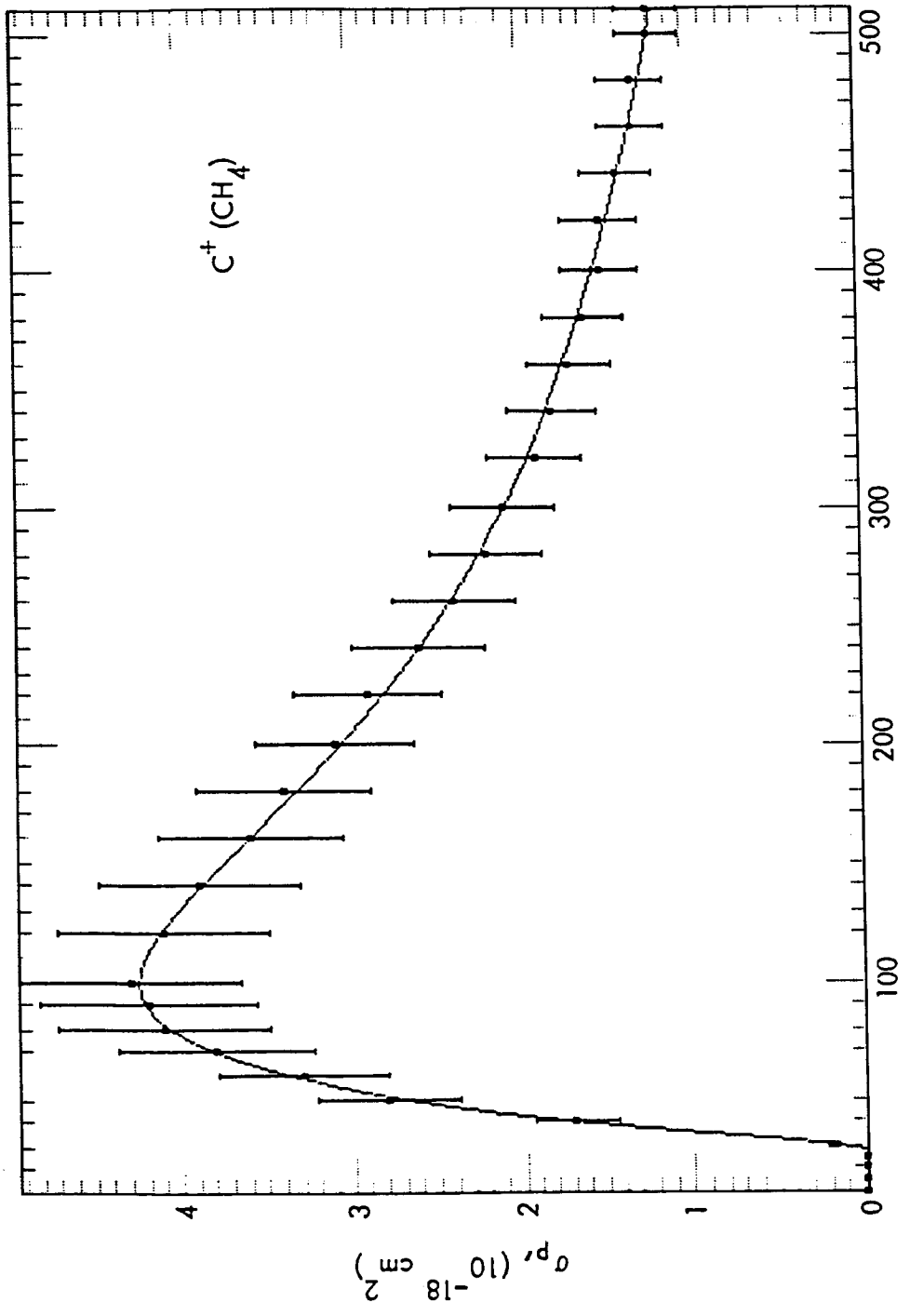

Figure 18. Electron impact ionization cross section for the production of C^+ from CH_4

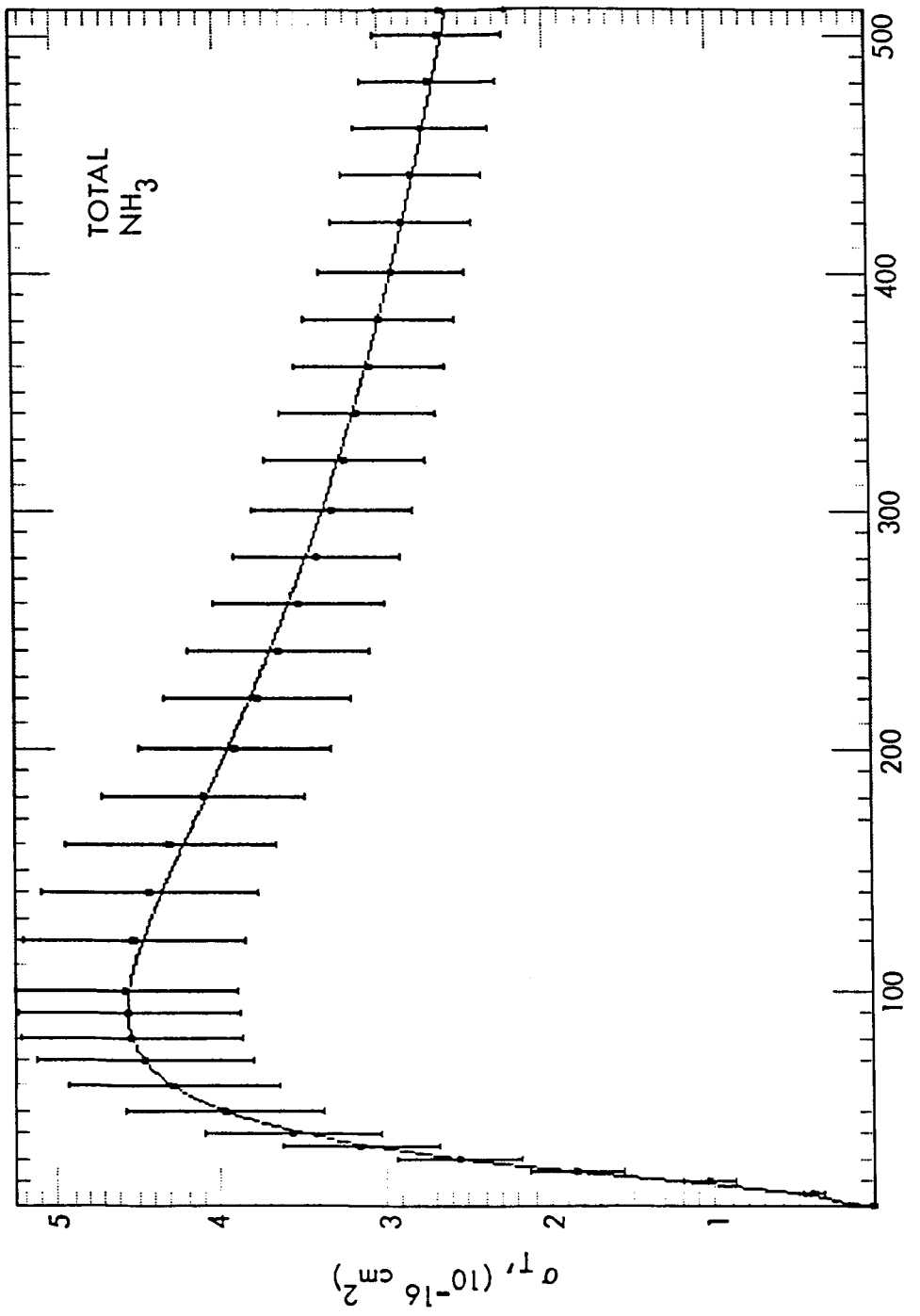

Figure 19. Total electron impact ionization cross section for NH_3

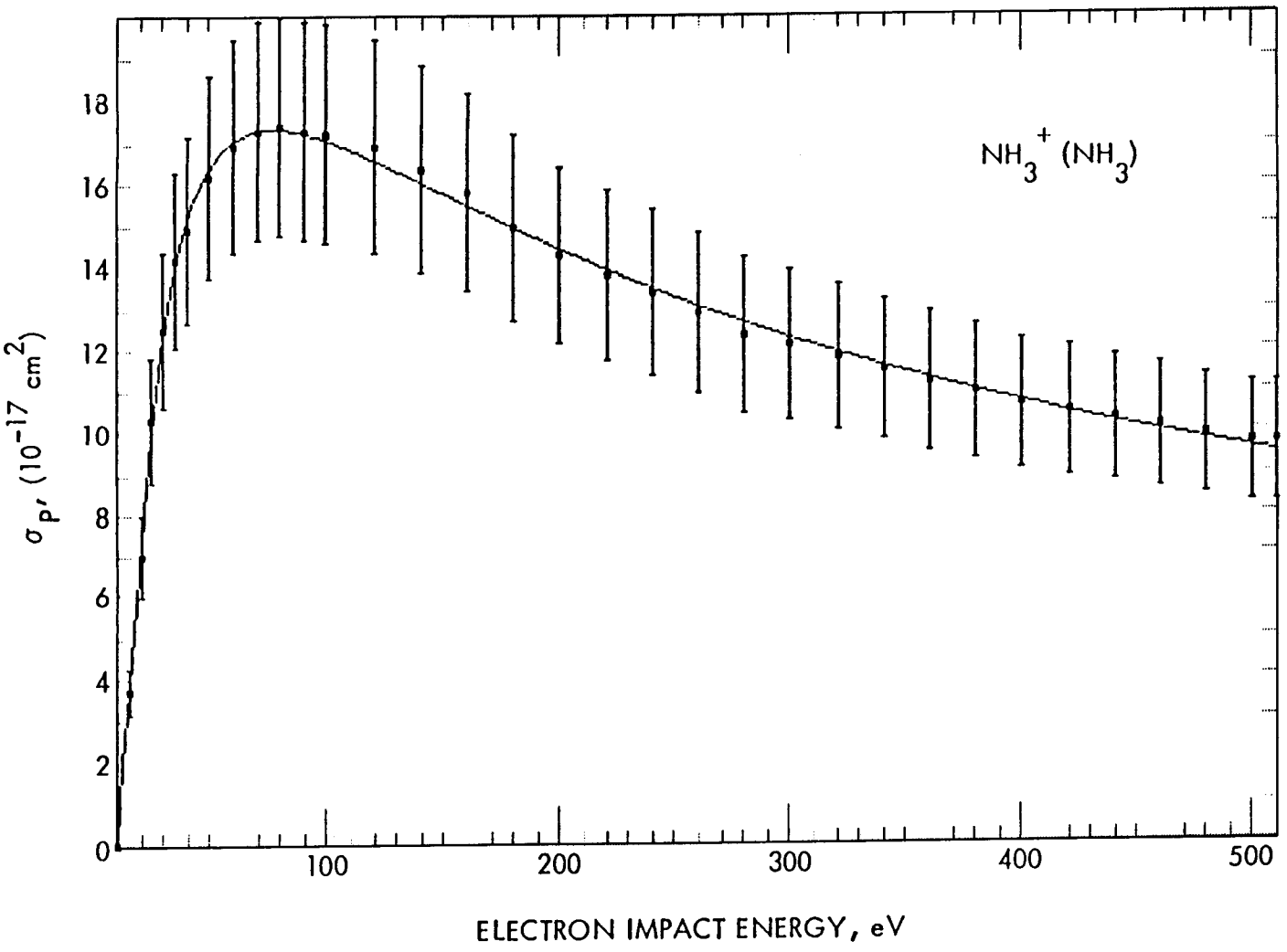

Figure 20. Electron impact ionization cross section for the production of NH_3^+ from NH_3

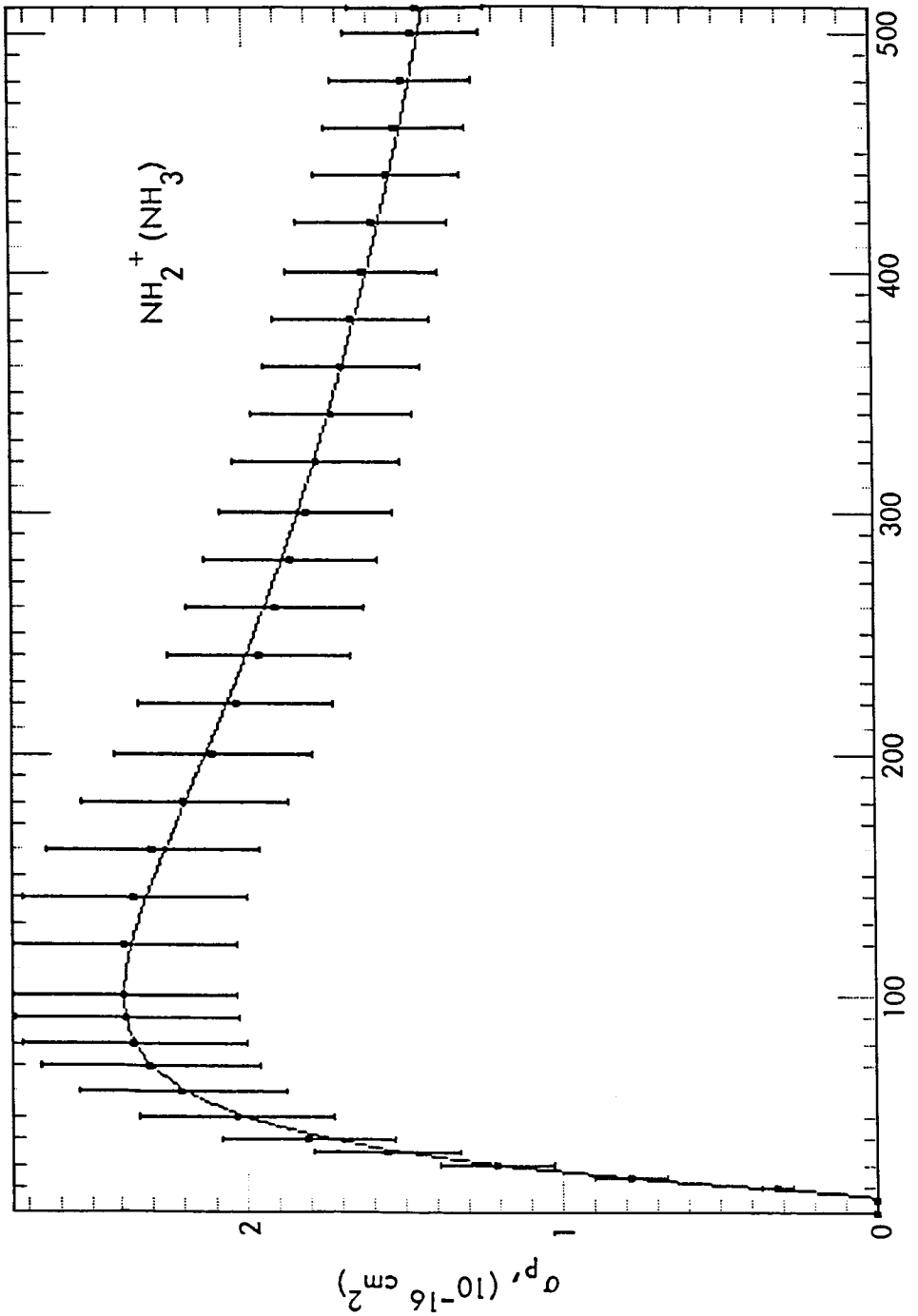

Figure 21. Electron impact ionization cross section for the production of NH_2^+ from NH_3

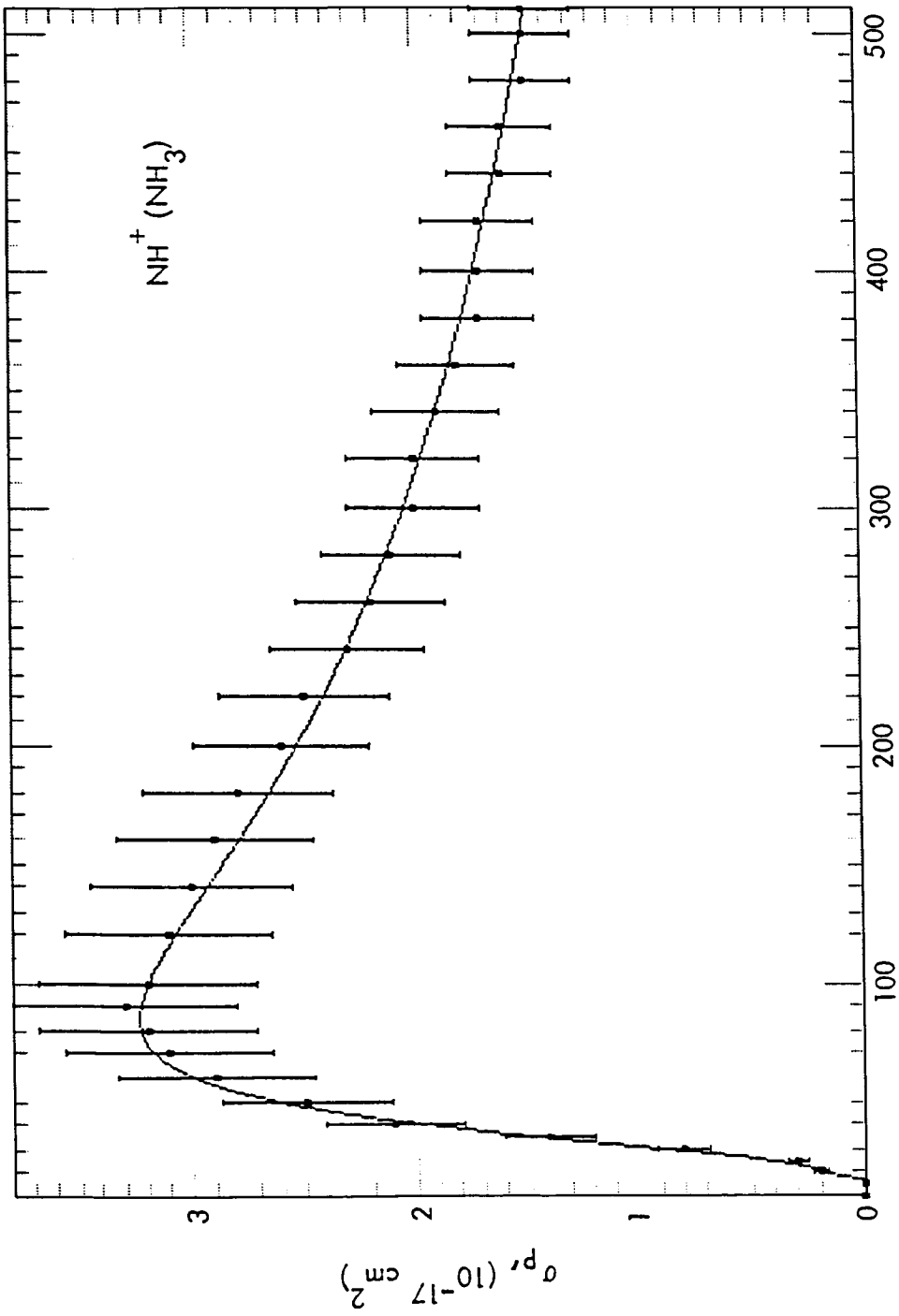

Figure 22. Electron impact ionization cross section for the production of NH^+ from NH_3

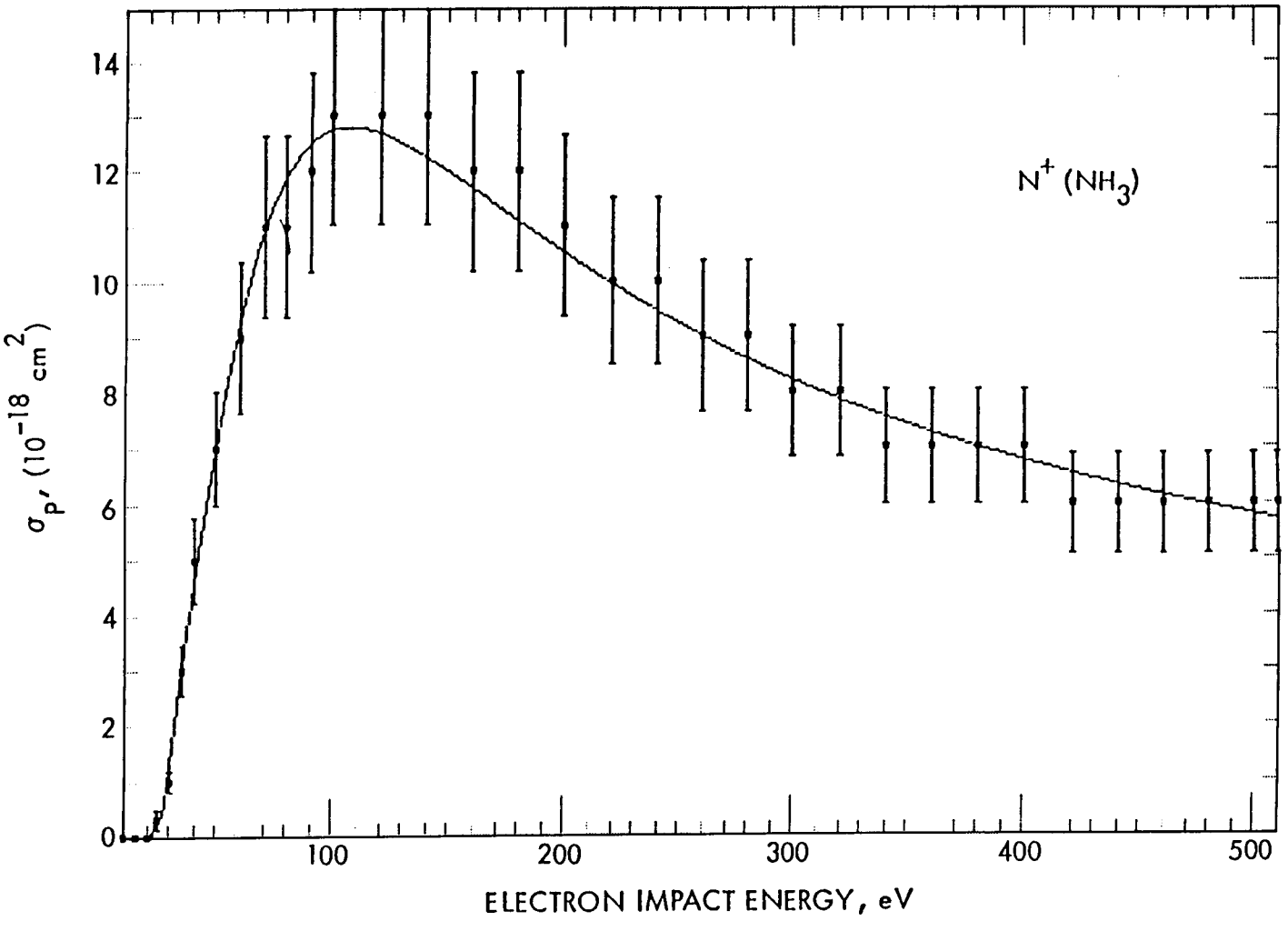

Figure 23. Electron impact ionization cross section for the production of N^+ from NH_3

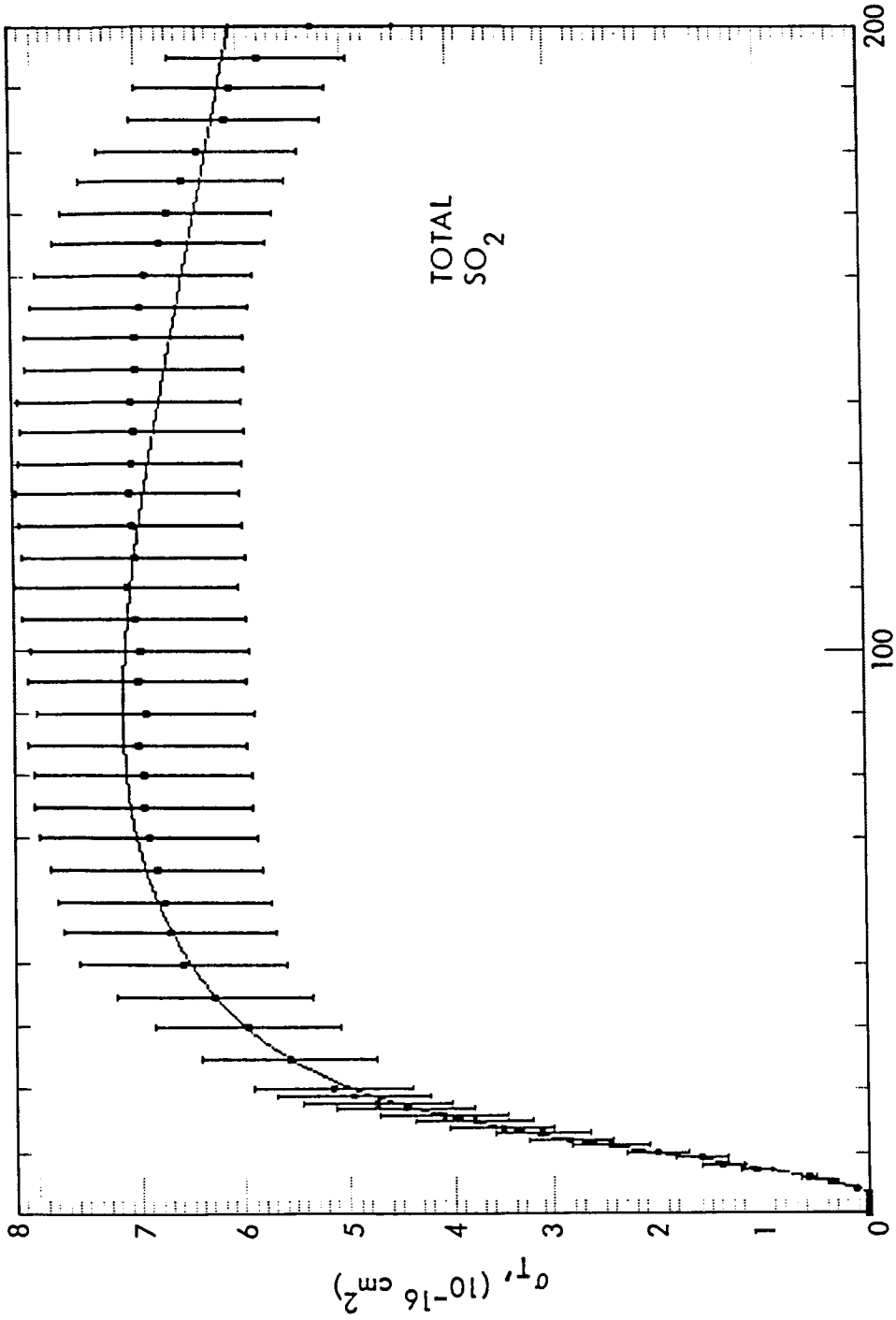

Figure 24. Total electron impact ionization cross section for SO_2

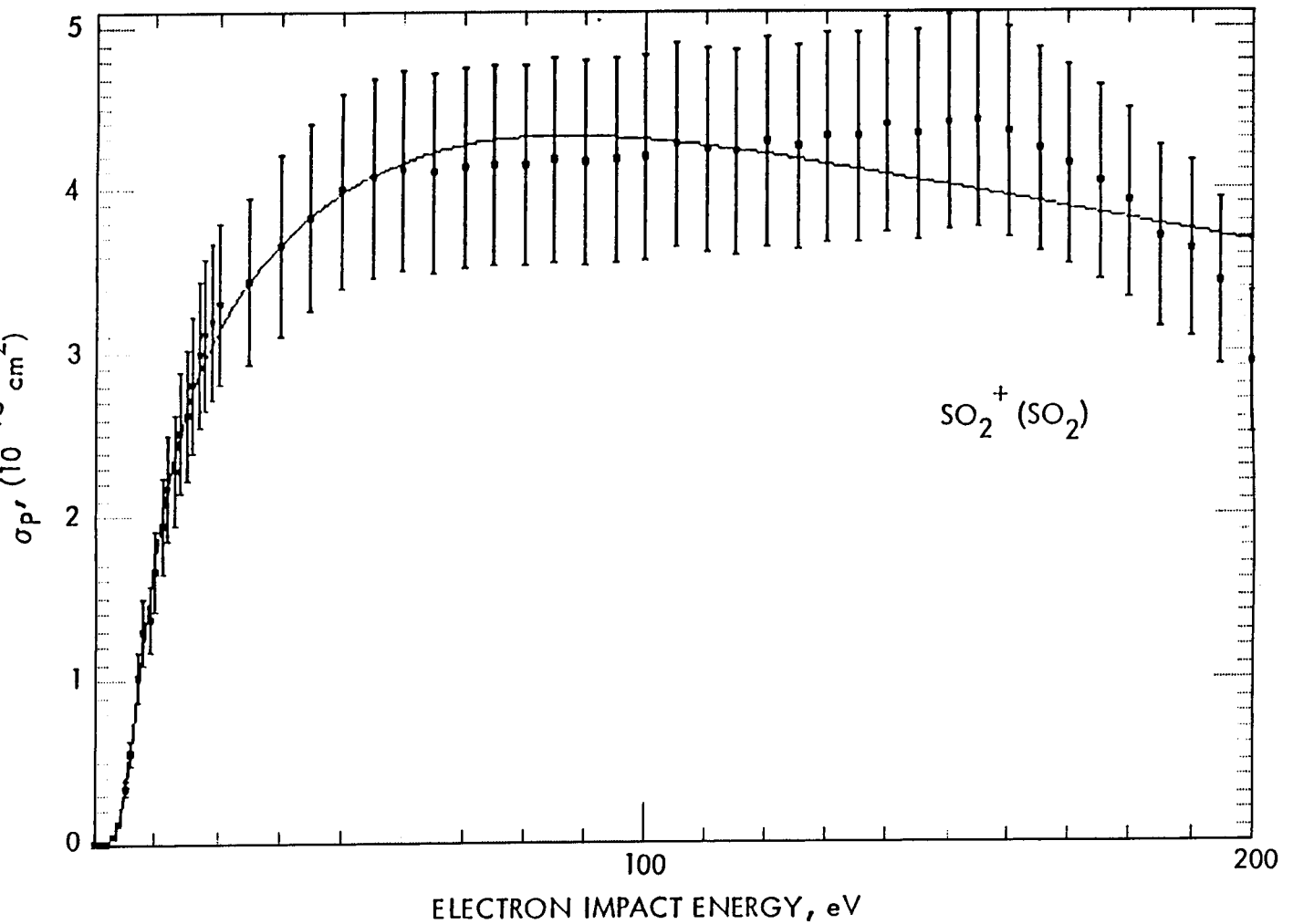

Figure 25. Electron impact ionization cross section for the production of SO_2^+ from SO_2

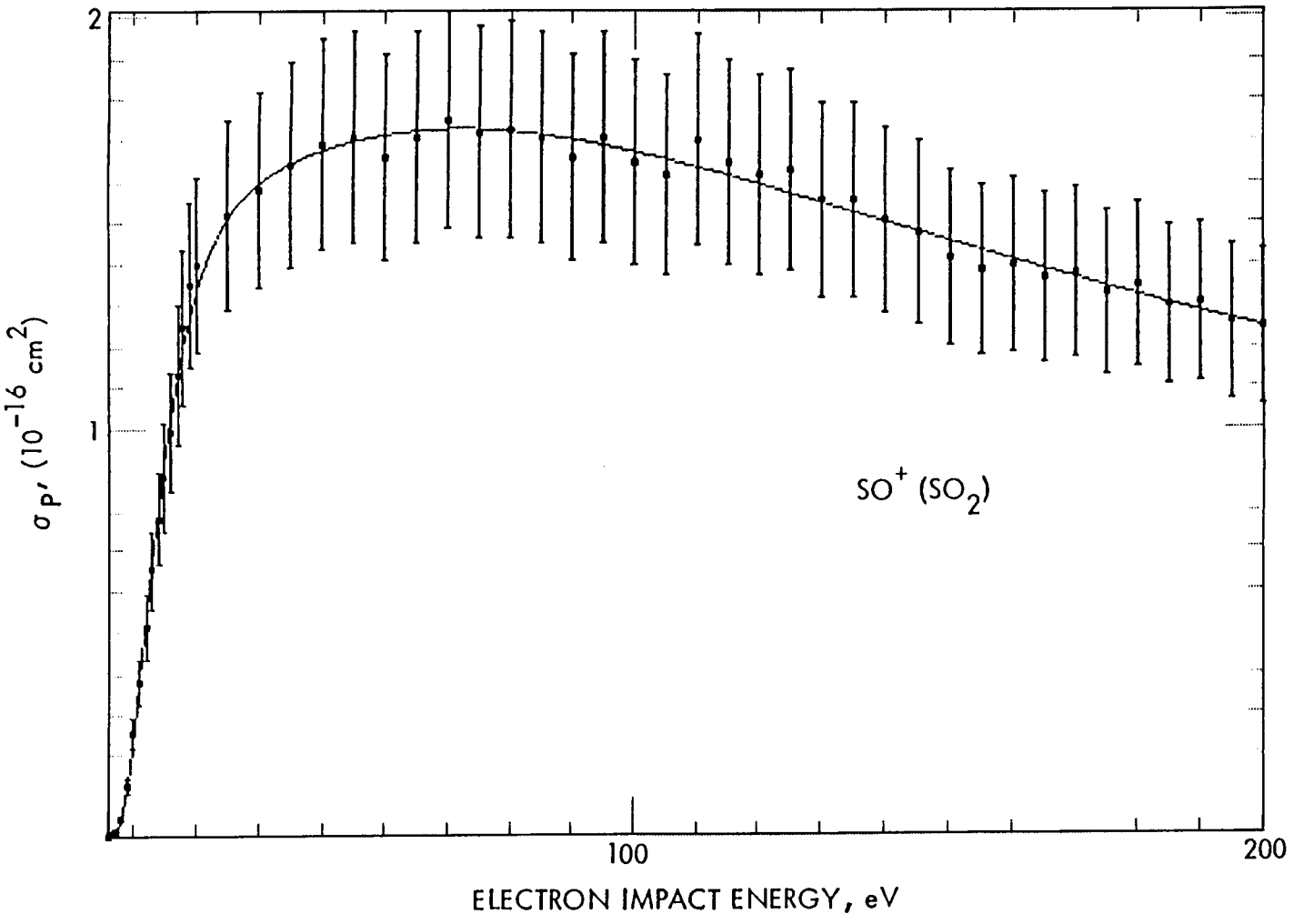

Figure 26. Electron impact ionization cross section for the production of SO^+ from SO_2

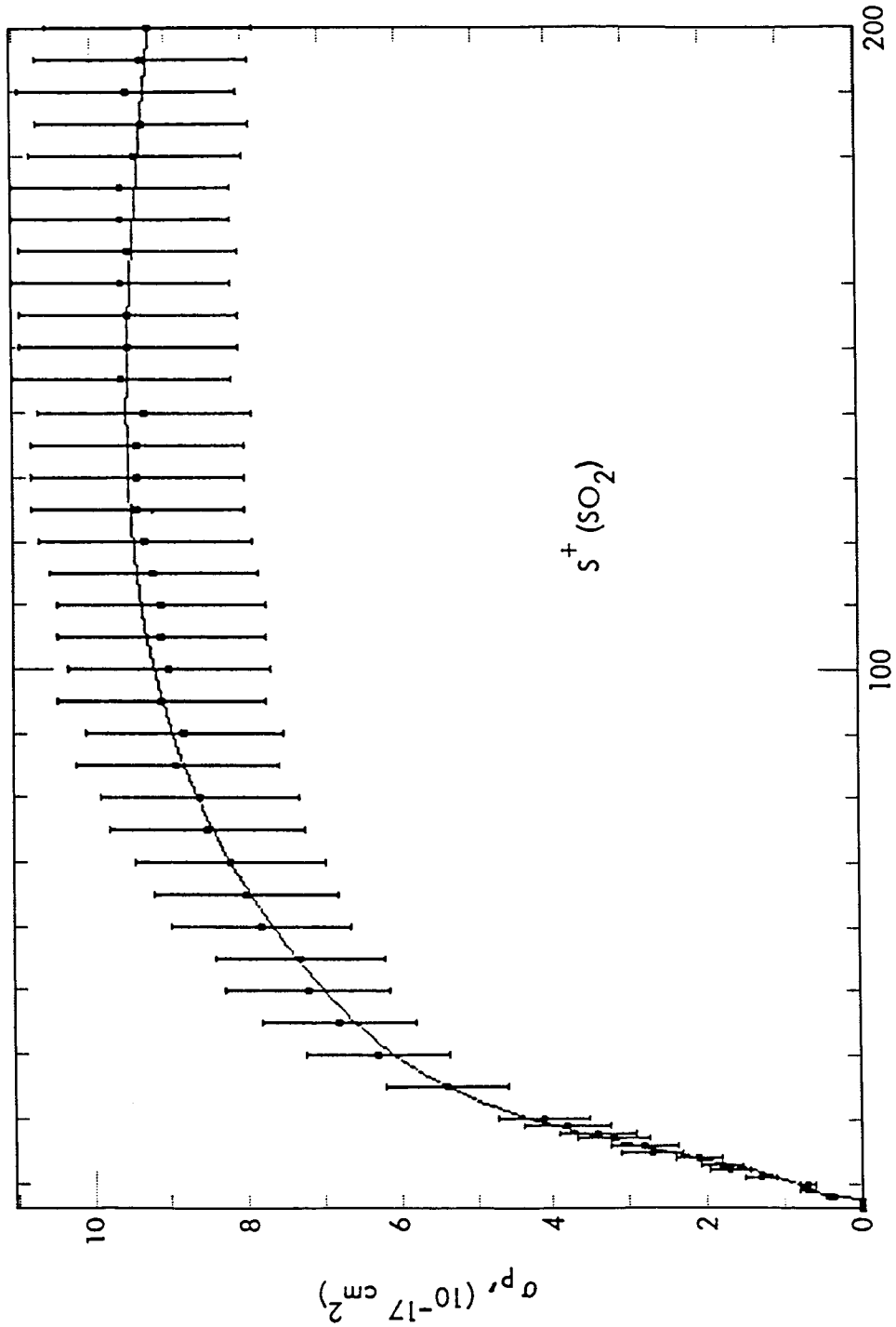

Figure 27. Electron impact ionization cross section for the production of S^+ from SO_2

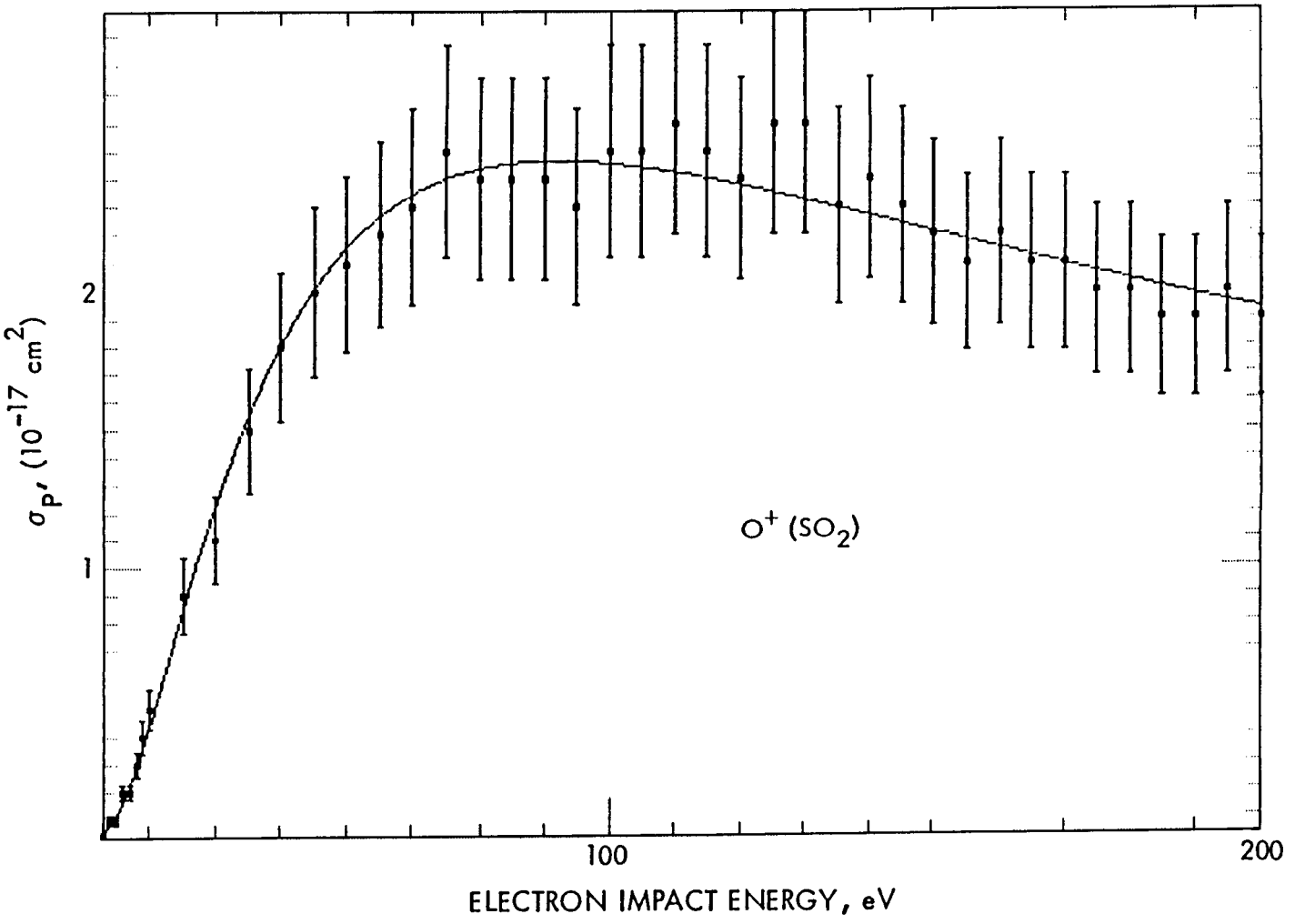

Figure 28. Electron impact ionization cross section for the production of O^+ from SO_2

APPENDIX I
EXPERIMENTAL APPARATUS AND METHOD

A schematic diagram of the apparatus is shown in Fig.3. It utilizes a crossed electron beam—molecular/atomic beam collision geometry. In the case of gases, the beam of atoms or molecules is produced by flowing the gas through a capillary array. Alternatively, in the case of species which are solids or liquids at room temperature, an electron bombarded oven or a resistance heated oven is utilized to produce the target beam. The beam of electrons is generated by heating a pure tungsten filament. The electrons are first extracted from the filament and are then accelerated or decelerated by three cylindrical lenses. This beam is collimated by the help of an axial B field which is produced by a solenoid within which the electron gun and a Faraday cup are housed. The solenoid produces the B field of the order 200 G.

The present beam of electrons is energy unselected. An energy profile of this beam was obtained by utilizing the retarding potential on the Faraday cup. It is found that the full width at half maximum (FWHM) is approximately 300 meV. The energy of the electrons is varied by changing the bias on the filament with respect to the last electrode of the electron gun. This electrode is kept at ground (earth) potential. It was found that the beam current, as measured by the Faraday cup, remained constant as the energy

of the beam was changed from 0.5 to about 10 eV which is the range of the present interest. Although the energy of the electron beam can be obtained by recording the filament bias voltage, the contact potentials at various surfaces tend to change it from its actual value. In the present work, the energy of the beam was calibrated by utilizing the accurately known values of ionization potentials of the rare gases.

The positive ions produced by collision of electrons with the target molecules are extracted out of the B field by two parallel molybdenum wire meshes between which a voltage is applied. This voltage produces a homogeneous electric field with a gradient of 3 to 10 V/cm at the target. The direction of the field is normal to both electron beam and molecular beam. One grid is biased negative with respect to the ground and the other positive. The molecular beam is kept at ground potential. It was found that this arrangement did not disturb the electron beam. The efficiency of extraction of ions was measured by changing the extracting electric field from 0 up to 10 V/cm. It was found that by increasing the electric field strength, the detected ion intensity increased rapidly in the beginning. However, at about 3 eV and above the ion intensity became almost constant as a function of the field strength. This indicated that the measured ion current did not depend on the initial energy and angular distribution of the ions. All our measurements were performed in this region of extracting voltage. The extracted ions are accelerated from 0 to about 200 V/cm and focused at the entrance aperture of a quadrupole mass spectrometer by an ion lens (Fig.1). This mass spectrometer has a resolution of approximately 1 amu. The mass analyzed ions are accelerated by a 3.2 kV potential and are detected by a spiraltron

multiplier. Each ion is counted as an event by a multichannel scaler.

A vacuum of about 10^{-8} Torr was obtained when the gas forming the molecular beam was not flowed into the vacuum chamber. However, the pressure rose to about 10^{-7} Torr when the molecular beam was on.

In order to obtain the absolute values of the cross sections, the relative flow[25] technique developed in our laboratory for collision cross section measurement was utilized. The method employs a measurement of the ratio of the intensity of the positive ions of the unknown species (for example, O^+/SO_2, S^+/SO_2, SO^+/SO_2, SO_2^+/SO_2) to that of a known species (for example, He^+/He, Ne^+/Ne, Ar^+/Ar, or Kr^+/Kr). At the same time, the flow rates and pressure behind the capillary array[25] are measured. First, the gas [AB of Eq.(23) below] whose ionization cross section has to be measured is flowed through the capillary array and a beam is formed. The positive ion intensity $I(B^+)$ is then measured. Subsequently, the gas AB is turned off and Ar is flowed through the capillary array. The positive ion current $I(Ar^+)$ is again recorded. Providing that the measurement is performed under the conditions of molecular flow through the capillary array, the following relation is used to obtain the cross section:

$$\sigma\left(\frac{B^+}{AB}\right) = \sigma\left(\frac{Ar^+}{Ar}\right) \cdot \frac{I(B^+)}{I(Ar^+)} \cdot \left[\frac{m(Ar)}{m(AB)}\right]^{1/2} \cdot \left(\frac{N(Ar)}{N(AB)}\right) \cdot K, \qquad (26)$$

where $m(AB)$ and $m(Ar)$ are molecular weight of respective gases, $N(Ar)$ and $N(AB)$ are the flow rates of the two gases through the capillary array, and K is a calibration constant which determines the transmission efficiency of the

ion optics, quadrupole mass spectrometer, and charged particle detector for B^+ and Ar^+.

The calibration constant K for the various masses was experimentally obtained. We chose gases whose ionization cross sections are well known. These are (H^+/H_2)[26], (He^+/He)[27], (O^+, O_2)[26], (Ne^+/Ne)[27], (Ar^+/Ar)[27], and $Kr^+/Kr)$[27]. Since all the quantities in Eq.(23) are either known or can be obtained experimentally for the two gases out of the ones mentioned above except K, the values of K for various mass numbers ranging from H to Kr can be calculated. We followed this method for calibrating our instrument. The relative efficiency K as a function of the mass number is a bell shaped curve. It is increasing with the mass number up to about mass number 45, then it is decreasing at higher mass numbers. Our results are in agreement with Ehlert's measurements[28]. More detail about the relative efficiency measurement can be found in Orient and Srivastava's paper[29].

The contribution of the background scattering (both direct beam contribution and scattering by the background gas) to the scattering from the target gas beam is measured by providing an alternate leak to the vacuum chambers. The flow to the chamber is switched from the capillary array to the alternate gas inlet and the proper background pressure for the desired gas is established. The mass selected beam intensity is then measured as a function of the electron beam energy. It is found that the maximum value of the background scattering is about 5%.

APPENDIX II

Program Listing for HP9836C Computer

```
program surf1 (input,output);
import mylib;                                    { graphic library C.I.T.      }
const
   top    = 350;       pi     = 3.14159268;       maxdata= 200;
   bottom = 20;        maxpar = 30;               termx  = 9999;
   left   = 20;        long   = 480;              maxfunc= 100;
   right  = 500;       height = 330;              maxstack= 20;

type
   chars     = '#'..'z';
   charset   = set of chars;
   elem_type = (cons,coef,oper,vars,ends);       { type of function elements   }
   oper_type = (oadd,osub,omul,odiv,opwr,oexp,olog,ocos,osin,oatn,
                otan,ocsh,osnh,otnh);            { defined basic operators     }
   func_rec  = record                            { function elements           }
                  case datum : elem_type of
                     vars      : (valr : real);
                     coef,cons : (vali : integer);
                     oper      : (valo : oper_type);
                  end;
   func_arr  = array [0..maxfunc] of func_rec;   { function array              }
   stck_arr  = array [1..maxstack] of real;
   vhtype    = (vertical,horizontal);
   mode_type = (instr,none,stat);                { uncertainty type            }
   theograf  = array [left..right] of real;      { array of graph              }
   coeff_rec = record                            { record of a coefficient     }
                  a,siga,dela : real;
                  end;
   coeff_type= array [1..maxpar] of coeff_rec;   { array of coefficients       }
   cons_type = array [1..maxpar] of real;        { array of function constants }
   data_rec  = record                            { record of data              }
                  x,y,sigy : real;
                  end;
   expdata   = array [1..maxdata] of data_rec;   { array of data               }
   filedata  = file of data_rec;
   filefuncrec=record                            { record of a saved function  }
                  sort,sub : integer;
                  end;
   filefunc  = file of filefuncrec;
   mattype   = array [1..maxpar,1..maxpar] of real;
   oneddata  = array [1..maxdata] of real;

var
   dmaxy, dminy, maxx, maxy, minx, miny : real;   mode    : mode_type;
   lamda,x2,x1,corrx, corry,cutoff : real;        funca : func_arr;
   datg, dat  : theograf;                         coeff : coeff_type;
   indat, indatg : expdata;                       name : string[12];
   filea : filedata;                              fileb : filefunc;
   cfterms,connum,npts,nfree,color : integer;     cona   : cons_type;
```

```
   mch  : char;                                    exitprog : boolean;

{-------------------------------------------------------------------------}

function ff(fop : oper_type;xi,xi_1 :real):real; { elementary operations      }
begin                                            { between two elements       }
case fop of                                      { select operation           }
   osub : ff := xi_1 - xi;
   oadd : ff := xi_1 + xi;
   omul : ff := xi_1 * xi;
   odiv : ff := xi_1 / xi;
   opwr : ff := exp(xi*ln(xi_1));
   oexp : if abs(xi) < 700 then ff := exp(xi) else
             if xi > 0 then ff := 1e300 else ff := 1e-300;
   olog : if xi > 1e-300 then ff := ln(xi) else ff := -1e300;
   ocos : ff := cos(xi);
   osin : ff := sin(xi);
   oatn : ff := arctan(xi);
   otan : ff := sin(xi)/cos(xi);
   osnh : if abs(xi) < 700 then ff := 0.5*(exp(xi) - exp(-xi)) else
             if xi > 0 then ff := 1e300 else ff := -1e300;
   ocsh : if abs(xi) < 700 then ff := 0.5*(exp(xi) + exp(-xi)) else
             ff := 1e300;
   otnh : if abs(xi) > 700 then ff := (exp(xi) - exp(-xi))/(exp(xi) + exp(-xi))
             else if xi > 0 then ff := 1 else ff := -1;
   end;
end;

{-------------------------------------------------------------------------}

function f(ffunca:func_arr;fx,fcutoff: real;fcoeff: coeff_type;
           fcona : cons_type): real;             { evaluate function ffunca at }
var i,st : integer; fstck : stck_arr;            { fx                          }
begin
st := 0;                                         { stack count                 }
i := 1;
if fx <= fcutoff then  f := 0 else               { f = 0 below cut off point   }
   begin
      while ffunca[i].datum <> ends do           { unstack & evaluate function }
         begin
            if ffunca[i].datum <> oper then st := st + 1;{stack unless operator}
            with ffunca[i] do
               case datum of
                  vars : fstck[st] := fx;        { load values onto stack      }
                  cons : fstck[st] := fcona[vali];
                  coef : fstck[st] := fcoeff[vali].a;
                  oper : case valo of            { evaluate w/ given operator  }
                            oadd,osub,odiv,omul,opwr : begin
                               fstck[st-1] := ff(valo,fstck[st],fstck[st-1]);
```

```
                        st := st -1;
                     end;
                     oexp,olog,ocos,osin,oatn,otan,ocsh,osnh,otnh :
                        fstck[st] := ff(valo,fstck[st],fstck[st]);
                     end;
               end;
            i := i + 1;                          { increment stack count       }
         end;
      f := fstck[1];                             { f is first element of stack }
   end;                                          { after evaluation            }
end;

{-------------------------------------------------------------------------------}

procedure maxmindata(mindat : expdata);          { find maximum and minimum    }
var i : integer;                                 { of both x and y of data     }
begin
maxx  := mindat[1].x;                            { initial max. min.           }
maxy  := mindat[1].y + mindat[1].sigy;
minx  := mindat[1].x;
miny  := mindat[1].y - mindat[1].sigy;
for i := 1 to npts do                            { search for max. min.        }
   with mindat[i] do
      begin
         if x > maxx then maxx := x;
         if x < minx then minx := x;
         if y + sigy > maxy then maxy := y + sigy;
         if y - sigy < miny then miny := y - sigy;
      end;
dmaxy := maxy;                                   { save dmaxy, dminy for graph }
dminy := miny;
corrx := long/(maxx - minx);                     {correction factors for screen}
corry := height/(maxy - miny);
end;

{-------------------------------------------------------------------------------}

procedure maxmin;                                { calculate max and min for   }
var i : integer;                                 { both the graph              }
begin                                            { and input data              }
maxy := dmaxy;
miny := dminy;
for i := left to right do
   begin
      if dat[i] > maxy then maxy := dat[i];
      if dat[i] < miny then miny := dat[i];
   end;
corry := height/(maxy - miny);                   { recalulate y correction     }
end;
```

```
{-------------------------------------------------------------------------}

function theodata(tfunca : func_arr;tcoeff : coeff_type;
                  tcona : cons_type): theograf;  { caluculate values for graph }
var i : integer;  tcorx : real;
begin
tcorx := long/(maxx - minx);
for i := left to right do                        { for every pixel points      }
   theodata[i] := f(tfunca,minx + (i-left)/tcorx,cutoff,tcoeff,tcona);
end;

{-------------------------------------------------------------------------}

function theonorm(tdat : theograf): theograf;    { normalize values for graph  }
var i : integer;
begin
for i := left to right do  theonorm[i] := (tdat[i] - miny)*corry + bottom;
end;

{-------------------------------------------------------------------------}

function expnorm(eindat : expdata): expdata;
var i : integer;                                 { normalize data for graphics }
begin
for i := 1 to npts do
   begin
      expnorm[i].x    := (eindat[i].x - minx)*corrx + left;
      expnorm[i].y    := (eindat[i].y - miny)*corry + bottom;
      expnorm[i].sigy := eindat[i].sigy*corry;
   end;
end;

{-------------------------------------------------------------------------}

procedure drawtheo(ddatg : theograf);            { draw the graph function     }
var i : integer;
begin
m_move(left,round(ddatg[left]));                 { first point                 }
for i := left to right do
   m_draw(i,round(ddatg[i]));
end;

{-------------------------------------------------------------------------}

procedure drawexp(dindatg : expdata;dnpts : integer);{ draw data points       }
var i,dx,dy,dsigy : integer;
begin
for i := 1 to dnpts do
```

```
   begin
      dx := round(dindatg[i].x);                    { rounding normalized values  }
      dy := round(dindatg[i].y);
      dsigy := round(dindatg[i].sigy);
      m_drawrect(dx-1, dy-1, dx+1, dy+1);           { draw small box for point    }
      m_move(dx-1, dy+dsigy);                       { draw end point for error bar}
      m_draw(dx+1, dy+dsigy);
      m_move(dx, dy+dsigy);                         { error bar                   }
      m_draw(dx, dy-dsigy);
      m_move(dx-1, dy-dsigy);                       { another end point           }
      m_draw(dx+1, dy-dsigy);
   end;
end;

{-------------------------------------------------------------------------------}

procedure readdata(var rindat : expdata;var rnpts : integer;strt:integer);
var i : integer;                                    { user input data             }
begin
write('Condition of sigma-y:   1-Statistical   2-No sigma-y   3-Instrumental ');
readln(i);                                          { kind of uncertainty         }
rnpts := strt;                                      { strt = 0 for new data       }
repeat
   rnpts := rnpts+ 1;
   case i of
      1,2 : begin
              rindat[rnpts].sigy := 0;
              if i = 1 then mode := stat   else mode := none;
              write('(',rnpts:1,') x, y? (x = 9999 to end)  ');
              readln(rindat[rnpts].x, rindat[rnpts].y);
            end;
      3 : begin
              mode := instr;
              write('(',rnpts:1,') x, y, sigma-y? (x = 9999 to end)  ');
              readln(rindat[rnpts].x, rindat[rnpts].y, rindat[rnpts].sigy);
            end;
      otherwise;
   end;
until (rindat[rnpts].x = termx) or (rnpts = maxdata) or (i<1) or (i>3);
if rindat[rnpts].x = termx then rnpts := rnpts-1;{ do not store 9999          }
end;

{-------------------------------------------------------------------------------}

procedure savefile(sindat : expdata;snpts : integer);
var i : integer;                                    { save input data             }
begin
write('name of file ');
readln(name);
```

```
rewrite(filea,name);
for i := 1 to snpts do
   begin
      filea^ := sindat[i];
      put(filea);
   end;
close(filea,'save');
end;

{-------------------------------------------------------------------------}

procedure readfile(var rindat : expdata;var rnpts : integer;strt : integer);
var i : integer;                                  { read from file input-data   }
begin
write('read from ');
readln(name);
reset(filea,name);
rnpts := strt;
while not eof(filea) do
   begin
      rnpts := rnpts+1;
      rindat[rnpts] := filea^;
      get(filea);
   end;
close(filea);
write('Is sigma-y  1-Statistical    2-None   3-Instrumental ');
readln(i);
case i of
   1 : mode := stat;   2 : mode := none;    3 : mode := instr; otherwise;  end;
end;

{-------------------------------------------------------------------------}

function weight(windat: expdata;i: integer): real;{ calculate weights for      }
begin                                             { each data points          }
with windat[i] do
   case mode of
      stat : if y <> 0 then weight := 1/abs(y)
             else weight := 1;
      none : weight := 1;
      instr: if sigy <> 0 then weight := 1/sqr(sigy) else weight := 1e10;
      end;
end;

{-------------------------------------------------------------------------}

function chisqr(cindat : expdata; cfunc : func_arr; ccoeff: coeff_type): real;
var cx2 : real;                                   { calculate chi square        }
    i : integer;
```

```
begin
cx2 := 0;
if nfree <= 0 then chisqr := 0 else
   begin
      for i:= 1 to npts do  cx2 := cx2 + weight(cindat,i)*sqr(abs(cindat[i].y -
                            f(cfunc,cindat[i].x,cutoff,ccoeff,cona)));
      chisqr := cx2/nfree;                          { reduced chi squared        }
   end;
end;

{-------------------------------------------------------------------------------}

function deriv(dindat : expdata;i : integer):oneddata;
var j : integer;                                    { calculate derivatives      }
    coj,yfit,dumder : real;                         { with respect to all coeff. }
begin
for j := 1 to cfterms do
   begin
      coj := coeff[j].a;
      coeff[j].a := coj + coeff[j].dela;
      yfit := f(funca,dindat[i].x,cutoff,coeff,cona);
      coeff[j].a := coj - coeff[j].dela;
      dumder := (yfit-f(funca,dindat[i].x,cutoff,coeff,cona))/(2*coeff[j].dela);
      if dumder <> 0 then deriv[j] := dumder else deriv[j] := 1e-10;
      coeff[j].a := coj;
   end;
end;

{-------------------------------------------------------------------------------}

procedure swap(var s1,s2 : real);                   { swap two numbers s1 and s2 }
var dummy : real;
begin
dummy := s1;
s1 := s2;
s2 := dummy;
end;

{-------------------------------------------------------------------------------}

procedure matinv(var marray : mattype;var merr : boolean);
var i,j,k,l : integer;                              { invert matrice, calc. det. }
    arrmax,dummy : real;  ik,jk : array [1..maxpar] of integer;
begin
merr := false;
for k := 1 to cfterms do
   begin
      arrmax := 0;
      for i := k to cfterms do                      { find largest element       }
```

```
        for j := k to cfterms do
           if abs(arrmax) <= abs(marray[i,j]) then
              begin
                 arrmax := marray[i,j];
                 ik[k] := i;
                 jk[k] := j;
              end;
     if arrmax = 0 then merr := true else       { swap column and row        }
        begin                                   { matrices to insure         }
           i := ik[k];                          { computational accuracy     }
           if i>k then
              for j := 1 to cfterms do  swap(marray[k,j],marray[i,j]);
           j := jk[k];
           if j>k then
              for i := 1 to cfterms do  swap(marray[i,k],marray[i,j]);
        end;
     if not merr then                           { invert matrix              }
        begin
           for i := 1 to cfterms do
              if i<>k then marray[i,k] := -marray[i,k]/arrmax;
           for i := 1 to cfterms do
              for j := 1 to cfterms do
                 if (i<>k) and (j<>k) then
                    marray[i,j] := marray[i,j] + marray[i,k]*marray[k,j];
           for j := 1 to cfterms do
              if j<>k then marray[k,j] := marray[k,j]/arrmax;
           marray[k,k] := 1/arrmax;
        end;
  end;
if not merr then                                { return matrice to its      }
  for l := 1 to cfterms do                      { prearranged state          }
     begin
        k := cfterms - l + 1;
        j := ik[k];
        if j>k then
           for i := 1 to cfterms do  swap(marray[i,k],marray[i,j]);
        i := jk[k];
        if i>k then
           for j := 1 to cfterms do  swap(marray[k,j],marray[i,j]);
     end;
end;

{-------------------------------------------------------------------------}

procedure curfit(cindat : expdata;var ccoeff : coeff_type;
                 var cx2 : real;cwt : oneddata); { calculate grad. & new coeff.}
var xsq1 : real;                        carray, alpha : mattype;
    cderiv : oneddata;                  i,j,k,l : integer;
    b : coeff_type;                     beta : array[1..maxpar] of real;
```

```
    cerr : boolean;
begin
for j := 1 to cfterms do                          { initialize matrice          }
   begin
      beta[j] := 0;
      for k := 1 to j do   alpha[j,k]:= 0;
   end;
for i := 1 to npts do                             { calculate beta and alpha    }
   begin
      cderiv := deriv(cindat,i);
      for j := 1 to cfterms do
         begin
            beta[j] := beta[j] + cwt[i]*
               (cindat[i].y-f(funca,cindat[i].x,cutoff,ccoeff,cona))*cderiv[j];
            for k := 1 to j do
               alpha[j,k] := alpha[j,k] + cwt[i]*cderiv[j]*cderiv[k];
         end;
   end;
for j := 1 to cfterms do
   for k :=1 to j do  alpha[k,j] := alpha[j,k];
xsq1 := cx2;
repeat                                            { begin calculating direction }
   for j := 1 to cfterms do                       { of minimal chi square       }
      begin
         for k := 1 to cfterms do                 { normalize alpha for accuracy}
            carray[j,k] := alpha[j,k]/sqrt(alpha[j,j]*alpha[k,k]);
         carray[j,j] := 1 + lamda;
      end;
   matinv(carray,cerr);                           { invert mtrix                }
   if cerr then writeln('Non-unique system of equation found ');
   b := ccoeff;
   for j := 1 to cfterms do                       { new coefficients?           }
      for k := 1 to cfterms do
         b[j].a := b[j].a +
                   beta[k]*carray[j,k]/sqrt(alpha[j,j]*alpha[k,k]);
   cx2 := chisqr(cindat,funca,b);
   if cx2 > xsq1 then  lamda := lamda*10;         { gradient search?            }
until cx2 <= xsq1;                                { keep going to minimum pt    }
ccoeff := b;                                      { new coefficients            }
for j := 1 to cfterms do  ccoeff[j].siga := sqrt(carray[j,j]/alpha[j,j]);
lamda := lamda/10;                                { linearization?              }
end;

{-------------------------------------------------------------------------------}

procedure drawscale(dmin,dmax:real;dcon,dsc:integer;dvh:vhtype);
var dfac : real;                                  { draw scale on graph         }
    pwcor, pwmax, pwmin, pwdif, gmin, gmax, i, k : integer;
begin
```

```
if dmax > dmin then                                  { compare magnitudes of max.  }
   begin                                             { min. and calculate steps    }
      if abs(dmax) < 1 then pwcor := -1  else pwcor := 0;  { for scales            }
      if dmax <> 0 then pwmax := trunc(ln(abs(dmax))/ln(10)) + pwcor
         else pwmax := -400;
      if dmin <> 0 then pwmin := trunc(ln(abs(dmin))/ln(10)) + pwcor
         else pwmin := -400;
      pwdif := trunc(ln(dmax-dmin)/ln(10)) + pwcor;
      if pwmax > pwmin then dfac := exp((-1 + pwmax)*ln(10))
      else if pwmin > pwmax then dfac := exp((-1 + pwmin)*ln(10))
      else dfac := exp((-1 + pwdif)*ln(10));
      gmax := trunc(dmax/dfac);
      gmin := round(dmin/dfac + 0.5);
      for i := gmin to gmax do
         begin
            case dvh of
               vertical : begin                      { left and right verticals    }
                             k := round(corry*(i*dfac - dmin));
                             m_move(dcon,k + bottom);
                             if (i=0) and (dcon = left) then
                                m_draw(right,k + bottom) else
                             if i mod 10 = 0 then m_draw(dcon+3*dsc,k+bottom)
                                else m_draw(dcon+dsc,k + bottom);
                          end;
               horizontal:begin                      { top and bottom horizontals  }
                             k := round(corrx*(i*dfac - dmin));
                             m_move(k + left,dcon);
                             if (i=0) and (dcon = bottom) then
                                m_draw(k + left,top) else
                             if i mod 10 = 0 then m_draw(k + left,dcon+3*dsc)
                                else m_draw(k + left,dcon+dsc);
                          end;
            end;
         end;
   end;
end;

{-------------------------------------------------------------------------}

procedure drawframe;                                 { draw rectangle w/ scale     }
begin
m_color(m_yellow);
m_drawrect(left,bottom,right,top);                   { draw box                    }
writeln('maxy = ',maxy,'    miny = ',miny);          { draw scales                 }
drawscale(miny,maxy,left,5,vertical);
drawscale(miny,maxy,right,-5,vertical);
drawscale(minx,maxx,top,-5,horizontal);
writeln('maxx = ',maxx,'    minx = ',minx);
drawscale(minx,maxx,bottom,5,horizontal);
```

```
m_color(m_red);
m_move(left,10);
m_displaytext('Data file: ');
m_displaytext(name);
end;

{-------------------------------------------------------------------------------}

procedure draweverything(dindat : expdata;var ddat : theograf);
var dindatg : expdata;   ddatg : theograf;       { draw both data and function }
begin
maxmindata(dindat);                              { calculating...              }
ddat := theodata(funca,coeff,cona);
maxmin;
dindatg := expnorm(dindat);
ddatg   := theonorm(ddat);
m_clear;                                         { and drawing                 }
drawframe;
m_color(color);
drawtheo(ddatg);
m_color(m_red);
drawexp(dindatg,npts);
end;

{-------------------------------------------------------------------------------}

procedure wrtfunc(wfunca : func_arr);            { write function in reverse   }
var i : integer;                                 { polish notation             }
begin
writeln; write('f = ');
i := 1;
if wfunca[1].datum = ends then writeln(' undefined');{ undefined functions    }
while wfunca[i].datum <> ends do                 { changing into human         }
   begin                                         { readable forms              }
      with wfunca[i] do
         case datum of
            vars : write('x');
            coef : write('(C',vali:1,')');
            cons : write('(A',vali:1,')');
            oper : case valo of
                      oadd : write('+');        oexp : write('exp');
                      osub : write('-');        olog : write('log');
                      odiv : write('/');        ocos : write('cos');
                      omul : write('*');        osin : write('sin');
                      opwr : write('^');        oatn : write('arctan');
                      otan : write('tan');      ocsh : write('cosh');
                      osnh : write('arctan');   otnh : write('tanh');
                   end;
            end;
```

```
      i := i + 1;
   end;
end;

{-------------------------------------------------------------------------}

procedure readfunc(var rfunc : func_arr;var rcfterms,rconnum : integer);
var ch : char;                                        { read function in R.P.N.     }
    i : integer;                          notaset : charset;
begin
page;
rcfterms := 0;                                        { reset function counts       }
rconnum := 0;
i := 0;                                               { initializations and         }
rfunc[1].datum := ends;                               { instructions                }
notaset := ['#','@','+','-','*','/','^','e','l',
            's','c','a','t','C','S','T','x',',',';'];
writeln('to enter a constant, type @ then the constant # and <RETURN>');
writeln('to enter a coefficient, type # then the coeff.# and <RETURN>');
writeln('other accepted notations :');
writeln('+, -, *, /, ^, e(xp), l(og), s(in), c(os), x(var), ,(undo), ;(end');
writeln('t(an), C(osh), S(inh), T(anh)');
writeln; writeln('Useful equations for statistical study');
writeln('Gaussian = (C1)(C2)*(C3)/(2)(0.5)^/(-0.5)x(C4)-(C3)/(2)^*exp*');
writeln('Lorentz  = (C1)(C2)(2)/(2)^*x(C3)-(2)^x(C2)(2)/-(2)^+/');
writeln('Resonance= (C1)(2)^x(2)^(C1)(2)^-(2)^x(C2)*(2)^+(0.5)^/');
writeln; writeln('enter function in REVERSE POLISH NOTATION');
writeln; write('f = ');
repeat
   read(ch);
   if not (ch in notaset) then                        { input error                 }
      begin
         write(' input error ');
         wrtfunc(rfunc);
      end
   else                                               { interpreting inputs         }
      begin
         i := i + 1;
         rfunc[i+1].datum := ends;
         with rfunc[i] do
            case ch of
               'x' : datum := vars;
               '@' : begin
                        datum := cons;
                        write('?');
                        readln(vali);
                        if vali > rconnum then rconnum := vali;
                        wrtfunc(rfunc);
                     end;
```

```
            '#' : begin
                     datum := coef;
                     write('?');
                     readln(vali);
                     if vali > rcfterms then rcfterms := vali;
                     wrtfunc(rfunc);
                  end;
            ',' : i := i - 2;
            '+','-','*','/','^','e','l','s','c','a','t','C','S','T'
                : begin
                     datum := oper;
                     case ch of
                        '+' : valo := oadd;        'e' : valo := oexp;
                        '-' : valo := osub;        'l' : valo := olog;
                        '*' : valo := omul;        'c' : valo := ocos;
                        '/' : valo := odiv;        's' : valo := osin;
                        '^' : valo := opwr;        'a' : valo := oatn;
                        't' : valo := otan;        'C' : valo := ocsh;
                        'S' : valo := osnh;        'T' : valo := otnh;
                     end;
                  end;
            ';' : datum := ends;
            otherwise;
            end;
      end;
until ch = ';';
end;

{-------------------------------------------------------------------------}

procedure currentfit(x2 : real);                { display infomation of fit   }
var i : integer;
begin
page;
write('Data file : ',name);
case mode of
   stat : writeln('   Sigma Y : Statistical');
   none : writeln('   No sigma y');
   instr: writeln('   Sigma Y : Instrumental');
end;
write('Function of graph is   ');               { write function, its         }
wrtfunc(funca);                                 { constants and coefficients  }
writeln;
for i := 1 to connum do
      writeln('A',i:1,' = ',cona[i]);
for i := 1 to cfterms do
      writeln('C',i:1,' = ',coeff[i].a,'    Sigma C',i:1,' = ',coeff[i].siga);
writeln('Chi square = ',x2);
writeln('Minimum y = ',miny,'      Maximum y = ',maxy);
```

```
writeln('Minimum x = ',minx,'      Maximum x = ',maxx);
write('hit any key to return ');
read(mch);
end;

{-------------------------------------------------------------------------}

procedure fit(var x2 : real);                    { controls fitting          }
var m : integer; x1,chidif : real;  wt : oneddata;
begin
write('color?   ');                              { input conditions for fitting}
readln(color);
write('cut off point? ');
readln(cutoff);
write('enter maximal absolute difference of successive chi square (0.0001) ');
readln(chidif);
write('display graph as function being fitted? ');
readln(mch);
lamda := 0.001;
nfree := npts - cfterms;
for m := 1 to cfterms do                         { initially guessed coeff.   }
   begin
      write('C',m:1,'?, delta? ');
      readln(coeff[m].a,coeff[m].dela);
   end;
for m:= 1 to npts do   wt[m] := weight(indat,m);
x2 := chisqr(indat,funca,coeff);
writeln('chi square = ',x2);
if mch = 'y' then draweverything(indat,dat);
repeat
   writeln('lamda = ',lamda,'       ....fitting....');
   x1 := x2;
   if nfree > 0 then curfit(indat,coeff,x2,wt);
   writeln; writeln; writeln;
   for m := 1 to cfterms do                      { display temporary coeff.   }
      writeln('C',m:1,' = ',coeff[m].a,
              '    Sigma C',m:1,' = ',coeff[m].siga);
   writeln('chi square = ',x2);
   if mch = 'y' then draweverything(indat,dat);
until abs(x1-x2) <= chidif;                      { stop fitting?             }
writeln('fitting completed');
draweverything(indat,dat);
currentfit(x2);
end;

{-------------------------------------------------------------------------}

procedure drawfunc;                              { draw graph between x1 & x2 }
var m : integer;
```

```
begin
m_clear;
write('enter minimum x and maximum x  ');
readln(minx,maxx);
dmaxy := -9.999e99;                                  { normalizing...          }
dminy := 9.999e99;
write('New coefficients? ');
readln(mch);
if mch = 'y' then
   for m := 1 to cfterms do
      begin
         write('C',m:1,' = ');
         readln(coeff[m].a);
      end;
corrx := long/(maxx - minx);
dat := theodata(funca,coeff,cona);
maxmin;
datg := theonorm(dat);
write('color of graph? ');                           { drawing...              }
readln(color);
drawframe;
m_color(color);
drawtheo(datg);
end;

{-------------------------------------------------------------------------------}

procedure displaydata(dindat:expdata;dnpts: integer);
var i : integer;
begin                                                { display current data file  }
page;
writeln('Data file: ',name);
if  dnpts > 0 then
   for i := 1 to dnpts do
      with dindat[i] do
         begin
            write('X(',i:1,') = ',x,'     ');
            write('Y(',i:1,') = ',y,'     ');
            writeln('SIGMA-Y(',i:1,') = ',sigy);
            if i mod 20 = 0 then                     { end of screen           }
               begin
                  write('hit any key to continue ');
                  read(mch);
                  page;
               end;
         end;
write('hit any key to return');                      { end of data             }
read(mch);
end;
```

```
{-------------------------------------------------------------------------}

procedure drawdata(dindat:expdata);              { draw data alone          }
var  dindatg : expdata;
begin
maxmindata(dindat);                              { normalizing...           }
dindatg := expnorm(dindat);
m_clear;                                         { drawing...               }
drawframe;
m_color(m_red);
drawexp(dindatg,npts);
end;

{-------------------------------------------------------------------------}

procedure deffunc(var dfunca : func_arr;var dcfterms,dconnum:integer);
var m : integer;
begin                                            { input a function         }
repeat
   readfunc(dfunca,dcfterms,dconnum);            { read function            }
   wrtfunc(dfunca);
   writeln;
   write('correct? ');
   readln(mch);
   dfunca[0].datum := cons;
   dfunca[0].vali := 0;
   if (dconnum > 0) and (mch = 'y') then         { get constants if appropriate}
      for m := 1 to dconnum do
         begin
            write('A',m:1,'= ');
            readln(cona[m]);
         end;
until mch = 'y';
end;

{-------------------------------------------------------------------------}

procedure transfer(var tindat: expdata;var tarray: oneddata;dir,subrec:integer);
var i : integer;                                 { subprocedure of modify   }
begin                                            { procedure to transfer data }
for i := 1 to npts do                            { from one array to another  }
   case dir of
      1 : case subrec of
             1 : tarray[i] := tindat[i].x;
             2 : tarray[i] := tindat[i].y;
             3 : tarray[i] := tindat[i].sigy;
          end;
      2 : case subrec of
```

```
            1 : tindat[i].x := tarray[i];
            2 : tindat[i].y := tarray[i];
            3 : tindat[i].sigy := tarray[i];
          end;
   end;
end;

{-------------------------------------------------------------------------}

procedure modify(var mindat:expdata);          { make modifications to data  }
var i,tcom : integer;                          { as a whole group            }
    uarray,varray : oneddata;
begin
repeat
   page;                                       { instructions                }
   writeln('1 --> (X -> Uarray)               2 --> (X -> Varray)');
   writeln('3 --> (Y -> Uarray)               4 --> (Y -> Varray)');
   writeln('5 --> (SIGMA Y -> Uarray)         6 --> (SIGMA Y -> Varray)');
   writeln('7 --> (Uarray -> X)               8 --> (Varray -> X)');
   writeln('9 --> (Uarray -> Y)               10--> (Varray -> Y)');
   writeln('11--> (Uarray -> SIGMA Y)         12--> (Varray -> SIGMA Y)');
   writeln('13--> (U = f(U))                  14--> (V = f(V))');
   writeln('15--> (U = f(V))                  16--> Display data');
   writeln('other numbers--> end');
   readln(tcom);
   case tcom of
      1  : transfer(mindat,uarray,1,1);        { transfering...              }
      2  : transfer(mindat,varray,1,1);
      3  : transfer(mindat,uarray,1,2);
      4  : transfer(mindat,varray,1,2);
      5  : transfer(mindat,uarray,1,3);
      6  : transfer(mindat,varray,1,3);
      7  : transfer(mindat,uarray,2,1);
      8  : transfer(mindat,varray,2,1);
      9  : transfer(mindat,uarray,2,2);
      10 : transfer(mindat,varray,2,2);
      11 : transfer(mindat,uarray,2,3);
      12 : transfer(mindat,varray,2,3);
      13,14,15: begin                          { transforming data           }
           deffunc(funca,cfterms,connum);
           for i := 1 to npts do
              case tcom of
                 13 : uarray[i] :=  f(funca,uarray[i],-1e300,coeff,cona);
                 14 : varray[i] :=  f(funca,varray[i],-1e300,coeff,cona);
                 15 : uarray[i] :=  f(funca,varray[i],-1e300,coeff,cona);
              end;
           end;
      16 : displaydata(mindat,npts);
      otherwise;
```

```
   end;
until (tcom < 1) or (tcom > 16);
end;

{-------------------------------------------------------------------------------}

procedure update(var uindat : expdata;var unpts : integer);
var i,j ,k: integer;    newval : real;          { make changes and add data   }
begin
repeat
   writeln('1-Remove data point   2-Add data   3-Change data  4-Display data ');
   write('other numbers to exit ');             { updating options            }
   readln(i);
   case i of
      1 : repeat                                { removing a single data point}
            write('remove item #? (9999 to terminate) ');
            readln(j);
            if (j<unpts) and (j>0) then
               begin
                  for k := j to unpts -1 do
                     uindat[k] := uindat[k+1];
                  unpts := unpts - 1;
               end
            else if j = unpts then unpts := unpts - 1;
          until j = termx;
      2 : readdata(uindat,unpts,unpts);         { add data points             }
      3 : repeat                                { change data                 }
            write('item #? (9999 to terminate) ');
            readln(j);
            if j <= unpts then
               with uindat[j] do
                  begin
                     write('X(',j:1,') = ',x,'     ');
                     write('Y(',j:1,') = ',y,'     ');
                     writeln('SIGMA-Y(',j:1,') = ',sigy);
                     write('Change   1-X       2-Y      3-SIGMA-Y ? ');
                     readln(k);
                     write('new value? ');
                     readln(newval);
                     case k of
                        1 : x := newval;
                        2 : y := newval;
                        3 : sigy := newval;
                        otherwise;
                     end;
                  end;
          until j = termx;
      4 : displaydata(uindat,unpts);            { display data                }
      otherwise;
```

```
   end;
until (i<1) or (i>4);
end;

{-------------------------------------------------------------------------------}

function diffdata(dindat : expdata;ddat : theograf):expdata;
var i : integer;                                    { get difference plot       }
begin
diffdata := dindat;
for i := 1 to npts do
   diffdata[i].y := dindat[i].y - ddat[round((dindat[i].x-minx)*corrx + left)];
end;

{-------------------------------------------------------------------------------}

procedure getfunc(var gfunc : func_arr;var gcfterms,gconnum : integer);
var i : integer;                                    { read function in R.P.N.   }
    funcname : string[12];              dumfunc : filefuncrec; { from file      }
begin
page;
write('Read function from file? ');
readln(funcname);
reset(fileb,funcname);
gcfterms := 0;                                      { initializations           }
gconnum := 0;
i := 0;
gfunc[0].datum := cons;
gfunc[0].vali  := 0;
repeat
   i := i + 1;
   gfunc[i+1].datum := ends;
   dumfunc := fileb^;
   with gfunc[i] do
    with dumfunc do                                 { intepreting saved function }
      case sort of
         1 : datum := vars;
         2 : begin
                datum := cons;
                vali  := sub;
                if vali > gconnum then gconnum := vali;
             end;
         3 : begin
                datum := coef;
                vali  := sub;
                if vali > gcfterms then gcfterms := vali;
             end;
         4 : begin
                datum := oper;
```

```
                case sub of
                   1 : valo := oadd;        6 : valo := oexp;
                   2 : valo := osub;        7 : valo := olog;
                   3 : valo := omul;        8 : valo := ocos;
                   4 : valo := odiv;        9 : valo := osin;
                   5 : valo := opwr;        11: valo := oatn;
                   10: valo := otan;        12: valo := ocsh;
                   13: valo := osnh;        14: valo := otnh;
                end;
             end;
          otherwise;
          end;
   get(fileb);
until eof(fileb);
close(fileb);
wrtfunc(gfunc);  writeln;
if gconnum > 0 then                                  { get constants if appropriate}
   for i := 1 to gconnum do
      begin
         write('A',i:1,'= ');
         readln(cona[i]);
      end;
end;

{-------------------------------------------------------------------------------}

procedure savefunc(sfunc : func_arr);                { save function in R.P.N.     }
var i : integer;
    funcname : string[12];              dumfunc : filefuncrec;
begin
page;
write('Save function to file? ');
readln(funcname);
rewrite(fileb,funcname);
i := 1;
repeat                                               {change function into textfile}
   with sfunc[i] do
      case datum of
         vars : dumfunc.sort := 1;
         cons : begin
                   dumfunc.sort := 2;
                   dumfunc.sub  := vali;
                end;
         coef : begin
                   dumfunc.sort := 3;
                   dumfunc.sub  := vali;
                end;
         oper : begin
                   dumfunc.sort := 4;
```

```
                  with dumfunc do
                     case valo of
                        oadd : sub := 1;        oexp : sub := 6;
                        osub : sub := 2;        olog : sub := 7;
                        omul : sub := 3;        ocos : sub := 8;
                        odiv : sub := 4;        osin : sub := 9;
                        opwr : sub := 5;        oatn : sub := 11;
                        otan : sub := 10;       ocsh : sub := 12;
                        osnh : sub := 13;       otnh : sub := 14;
                     end;
               end;
         otherwise;
         end;
   fileb^ := dumfunc;
   put(fileb);
   i := i + 1;
until sfunc[i].datum = ends;
close(fileb,'save');
end;

{-------------------------------------------------------------------------}

procedure extrapolate;                          { extrapolate known function  }
var ex,ey : real;
begin
page;
writeln('To terminate, enter 9999 for x');
repeat
   write('x = ');
   readln(ex);
   ey := f(funca,ex,cutoff,coeff,cona);
   writeln('x = ',ex,'     f(x) = ',ey);
until ex = termx;
end;

{-------------------------------------------------------------------------}

procedure wait_for_pen(var wpen : m_tablet_info;wcorx,wcory,wx,wy : real);
begin                                           { wait til pen is depressed   }
repeat                                          { and return coordinate       }
   repeat
      m_readpen(wpen);
   until wpen.moving or wpen.dn;
   page;
   writeln(wpen.x*wcorx + wx,'  ',wpen.y*wcory + wy);
until wpen.dn;
end;

{-------------------------------------------------------------------------}
```

```
procedure readgraf(var rindat : expdata;var rnpts : integer);
var   mypen : m_tablet_info;                        { read data from graph        }
      rmaxx, rmaxy, rminx, rminy, rcorx, rcory : real; { on graphics tablet     }
      amaxx, amaxy, aminx, aminy, scorx, scory : real;
begin
page;
m_move(10,360);
m_displaytext('place graph on graphics tablet');
m_move(10,350);
m_displaytext('move pen to the lower left corner of your graph and press');
wait_for_pen(mypen,1,1,0,0);
rminx := mypen.x;
rminy := mypen.y;
m_clear;
m_move(10,360);
m_displaytext('move pen to the upper right corner of your graph and press');
wait_for_pen(mypen,1,1,0,0);
rmaxx := mypen.x;
rmaxy := mypen.y;
write('enter actual minx, maxx, miny, maxy ');   { get actual values on graph  }
readln(aminx,amaxx,aminy,amaxy);
rcorx := (amaxx - aminx)/(rmaxx - rminx);        { normalizing...              }
rcory := (amaxy - aminy)/(rmaxy - rminy);
scorx := long/(rmaxx - rminx);
scory := height/(rmaxy - rminy);
m_clear;
m_move(10,360);
m_displaytext('move to point to be collected and depress, box 1 to end');
rnpts := 0;
m_color(m_yellow);
m_drawrect(left,bottom,right,top);
m_color(m_red);
mch := 'n';                                      { flag to stop data entry     }
repeat                                           { collecting data points      }
    rnpts := rnpts + 1;
    wait_for_pen(mypen,rcorx,rcory,
                -rminx*rcorx + aminx,-rminy*rcory + aminy);
    rindat[rnpts].x    := (mypen.x-rminx)*rcorx + aminx;
    rindat[rnpts].y    := (mypen.y-rminy)*rcory + aminy;
    rindat[rnpts].sigy := 0;
    m_circle(round(scorx*(mypen.x-rminx)+left),
             round(scory*(mypen.y-rminy)+bottom),1);
    if ((mypen.x>0) and (mypen.x<22)) and
       ((mypen.y>393) and (mypen.y<417)) then begin
       write('data entry complete? ');
       readln(mch);
       end;
until (mch = 'y') or (rnpts = 100);
```

```
rnpts := rnpts - 1;                                  { do not enter menu value     }
end;

{-------------------------------------------------------------------------------}

procedure menu;                                      { display menu and            }
var com : integer;    difdat : expdata;              { interact with user          }
begin
writeln('1-Get a file     2-Save current file       3-Define a function');
writeln('4-Input data     5-Fit                     6-Plot data');
writeln('7-Plot function  8-Display data            9-Modify data');
writeln('10-Update data   11-Difference Plot        12-Function & Data');
writeln('13-Get function  14-Save function          15-Extra-(Inter)-polate');
writeln('16-Chi square    17-Digitize graph         18-Combine files');
writeln('19-Exit program');
readln(com);
case com of                                          { executing chosen option     }
   1 : begin                                         { get data file               }
          readfile(indat,npts,0);
          nfree := npts - cfterms;
       end;
   2 : if npts > 0 then savefile(indat,npts);        { save datafile               }
   3 : deffunc(funca,cfterms,connum);                { user input function         }
   4 : begin                                         { user input data             }
          name := 'not named';
          readdata(indat,npts,0);
          nfree := npts - cfterms;
       end;
   5 : if npts > 0 then fit(x2);                     { fitting                     }
   6 : if npts > 1 then drawdata(indat);             { plotting data               }
   7 : drawfunc;                                     { plotting function           }
   8 : displaydata(indat,npts);                      { display data                }
   9 : if npts > 0 then modify(indat);               { modify group of data        }
   10: if npts > 0 then update(indat,npts);          { update one data point       }
   11: begin                                         { difference plot             }
          writeln('enter c for difference plot IFF data was fitted');
          readln(mch);
          if mch = 'c' then
             begin
                difdat := diffdata(indat,dat);
                maxmindata(difdat);
                if npts > 1 then drawdata(difdat);
                currentfit(x2);
             end;
       end;
   12: begin                                         { plot both data and graph    }
          writeln('enter c for plot IFF data was fitted');
          readln(mch);
          x2 := chisqr(indat,funca,coeff);
```

```
            if mch = 'c' then  begin draweverything(indat,dat);
                 currentfit(x2);   end;
         end;
      13: getfunc(funca,cfterms,connum);              { read function from file     }
      14: if funca[1].datum <> ends then savefunc(funca); { save defined function }
      15: extrapolate;                                { extrapolating from known fcn}
      16: begin                                       { given a function and a set  }
            draweverything(indat,dat);                { of data point, plot and     }
            x2 := chisqr(indat,funca,coeff);          { determine chi square        }
            currentfit(x2);
         end;
      17: readgraf(indat,npts);                       { read from graph on tablet   }
      18: if npts > 0 then readfile(indat,npts,npts);{ Combine two files           }
      19: exitprog := true;                           { exiting program             }
      otherwise;
      end;
end;

{-------------------------------------------------------------------------------}

procedure currentinfo;                                { display status of data etc. }
var i : integer;
begin
page;
write('Current data file is ',name);                  { name of data and function   }
case mode of
   stat : writeln('   Sigma Y : Statistical');
   none : writeln('   No sigma y');
   instr: writeln('   Sigma Y : Instrumental');
end;
write('Current function is   ');
wrtfunc(funca);
writeln;
if connum > 0 then for i := 1 to connum do            { constants                   }
      writeln('A',i:1,' = ',cona[i]);
writeln;
end;

{-------------------------------------------------------------------------------}

begin
page;                                                 { initialization              }
funca[1].datum := ends;
name := 'none';
exitprog := false;
cfterms := 2;
mode := none;
m_init_graphics;
repeat                                                { begin interaction with user }
```

```
   currentinfo;
   menu;
until exitprog;
end.
```