gnu80 Manual

Contents

How to use this Manual

- If you come to a copy of `gnu80` already installed on a particular system, you may only need to browse through Chapter 1 and its Appendix and use the information in Chapter 2 and its Appendices in order to get useful work from the system.
- If you wish to install the program read Appendix F before proceeding (it is only one or two pages!).
- If you intend to install, use and modify the system, you will need to browse through most of the manual and ultimately through the code.

All users will benefit from browsing through the first two Chapters and their Appendices.

History

`gnu80` is a connected system of programs capable of performing *ab initio* Molecular Orbital (MO) calculations within the Linear Combination of Atomic Orbitals (LCAO) framework plus a restricted but extremely useful ability to calculate "post Hartree-Fock" (correlated) electron distributions and energies.

`gnu80` is a further development of the Gaussian 70, 76 and 80 systems already published. Gaussian 80 was originally implemented by J.A. Pople *et al.* and distributed (at cost via the Quantum Chemistry Program Exchange) for a DEC Vax-11/780. Gaussian 80 was subsequently implemented on the Amdhal V7b with IBM's operating system MVS-3.8 at the State University of Leiden, The Netherlands by P.N. van Kampen, G.F. Smits, F.A.A.M. de Leeuw and C. Altona.

Since the release of GAUSSIAN 80 all subsequent enhancements of the GAUSSIAN series of programs (82, 86, 88) have been sold under strict licence which, in particular, excludes their distribution to third parties. Since it is the aim of the work done here to make the software as freely available as possible, it is from the last *public domain* version of the GAUSSIAN series that `gnu80` has been developed.

This decision, of course, means that the enhancements to the system by the Carnegie-Mellon Team since GAUSSIAN 80 are not included, but the principle "workhorse" parts of the system are present (HF geometry optimisations, MP2 calculations etc.) and the method of interfacing user modules with `gnu80` is explicitly given.

In fact, this software is distributed on the strict understanding that it must be improved or passed on to a third party who will improve it.

Chapter 1

Introduction

gnu80 is a set of programs to perform the most commonly-required tasks of quantum chemistry. The numerical work involved in performing these tasks falls naturally into modeles and the programs of gnu80 reflect this underlying modularity.

The basic assumption involved in all the programs is the "Expansion Method" or "Algebraic Approximation" ; an approximate wavefunction for a molecular system (molecule, ion, radical, group of molecules) may be built up from (anti-symmetrised) products of Linear Combinations of Basis Functions. The basis functions themselves are linear combinations of Gaussian Functions. The approximate Molecular Hamiltonian embodies the non-relativistic Born-Oppenheimer model; the so-called elecetrostatic model, the only forces acting between the particles are those due to Coulomb's law.

With these two basic assumptions all calculations of molecular electronic structure fall into three stages:

1. The calculation of the energy integrals involving the one- and two-particle operators in the Hamiltonian and the basis functions; kinetic energy and nuclear attraction ("one-electron" integrals) and the far more numerous electron-electron repulsion integrals ("two-electron" integrals).

2. Use of these energy integrals and matrix techniques to solve the algebraic representation of the equations of the quantum mechanical model used.

3. Analysis of the results of the calculation

gnu80 provides implementations of these procedures with the provision of several different quantum mechanical models of the molecular electronic structure. Control programs are also provided for the automatic optimisation of molecular geometries by seeking turning points in the total energy of the molecule with respect to changes in those geometries.

There are two main areas of research in which gnu80 may be useful:

1. *Chemical research using quantum chemistry tools;* the calculation of the structure and properties of transient or otherwise experimentally inaccessible species to help in the resolution of chemical problems, elucidation of reaction paths etc.

2. *Quantum chemistry research;* calculations which are aimed at an understanding of the electronic structure of molecules and exploring the utility and limitations of the quantum mechanical models themselves.

gnu80 provides two distinct but related methods of working which, broadly speaking, are suitable for the two main uses of the system:

1. Use of Standard Methods

2. General Use

Both of these approaches must, of course, use the same capabilities of gnu80 and share the same ultimate limitations; types of quantum mechanical model and classes of Gaussian function available. Within these restrictions outlined earlier (electrostatic Hamiltonian and Gaussian Algebraic Approximation) the quantum mechanical models available are:

- "Hartree-Fock" , this term has come to mean a whole class of models related by the general idea of a one-term (single-determinant) approximation; the best single determinant (UHF), the best single determinant of doubly-occupied orbitals (RHF), the best single determinant of doubly- and singly-occupied orbitals for open-shell species (ROHF).

- Variational Methods including electron correlation; Configuration Interaction of double excitations from a Hartree-Fock determinant.
- Perturbation Methods of including electron correlation; Moller-Plesset perturbation theory to second and third order using the Hartree-Fock determinant as zero-order function.

When a quantum mechanical model has been chosen, considerations of the accuracy required and the available computational resources must be made in order to make a choice of the *numerical* limiations of the calculation. This choice dictates the number and type of Gaussian functions to be used. `gnu80` has libraries of standard Gaussian basis functions (particularly useful for Standard Methods) and the capability of accepting basis functions of the user's choice.

Thus the combination of the quantum mechanical *model* and the *numerical* choice of basis are primary items of data to `gnu80` which are independent of the particular molecular system under investigation. This information must be given in both Standard Methods and General Use; Chapter 2 describes the use of Standard Methods and Chapter 3 outlines General Use.

Appendix 1.A

Standard Nomenclature

1.A.1 The Hamiltonian

The molecular Hamiltonian used in the energy calculations of standard quantum chemistry is always the non-relativistic, Born-Oppenheimer "electrostatic" Hamiltonian, which is, in atomic units:

$$\hat{H}(\vec{r_1}, \vec{r_2}, \ldots, \vec{r_n}) = \sum_{i=1}^{n} \hat{h}(\vec{r_i}) + \sum_{i=1}^{n} \sum_{j<i} \frac{1}{r_{ij}} \tag{1.A.1.1}$$

Here, there are assumed to be n electrons in the molecule and their position vectors are $\vec{r_i}$. Each electron has a *One-electron Hamiltonian* of identical form:

$$\hat{h}(\vec{r_i}) = -\frac{1}{2}\nabla^2(\vec{r_i}) - \sum_{A=1}^{N} \frac{Z_A}{|\vec{r_i} - \vec{r_A}|} \tag{1.A.1.2}$$

where the position vectors of the nuclei are $\vec{r_A}$ and their charges are Z_A, and it issumed that there are N of them.

The electron repulsion terms are simply the Coulomb repulsions between unit like charges:

$$\frac{1}{r_{ij}} = \frac{1}{|\vec{r_i} - \vec{r_j}|}$$

the summation is over all *distinct* pairs and $i \neq j$ of course.

The associated Schrödinger equation:

$$\hat{H}(\vec{r_1}, \vec{r_2}, \ldots, \vec{r_n})\Psi(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}) = E\Psi(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}) \qquad (1.A.1.3)$$

in which the many-electron wavefunction $\Psi(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n})$ depends on the *spatial* and *spin* "co-ordinates" of the electrons. The collection of three spatial co-ordinates ($\vec{r_i}$) and one spin variable is written as $\vec{x_i}$. This equation cannot be solved and *ab initio* methods are designed to generate *approximate and model* solutions of 1.A.1.3 by a variety of variational and perturbation techniques.

1.A.2 Many-electron Wavefunctions

The many-electron wavefunction is approximated by a linear combination of **Slater Determinants** $\Phi_K(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n})$, each of which is the anti-symmetrised product of n **Spin-Orbitals** $\chi(\vec{x_i})$ depending on the space and spin variables of just one electron; for example a determinant constructed from the first n spin-orbitals $\chi_1 \ldots \chi_n$ is:

$$\Phi(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}) = \frac{1}{\sqrt{n!}} \begin{vmatrix} \chi_1(\vec{x_1}) & \chi_1(\vec{x_2}) & \ldots & \chi_1(\vec{x_n}) \\ \chi_2(\vec{x_1}) & \chi_2(\vec{x_2}) & \ldots & \chi_2(\vec{x_n}) \\ \chi_3(\vec{x_1}) & \chi_3(\vec{x_2}) & \ldots & \chi_3(\vec{x_n}) \\ \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots \\ \chi_{n-2}(\vec{x_1}) & \chi_{n-2}(\vec{x_2}) & \ldots & \chi_{n-2}(\vec{x_n}) \\ \chi_{n-1}(\vec{x_1}) & \chi_{n-1}(\vec{x_2}) & \ldots & \chi_{n-1}(\vec{x_n}) \\ \chi_n(\vec{x_1}) & \chi_n(\vec{x_2}) & \ldots & \chi_n(\vec{x_n}) \end{vmatrix}$$

The factor $1/\sqrt{n!}$ normalises Φ if the χ_i are an orthonormal set.

The number and construction of these determinants defines a **model** of molecular electronic structure and the accuracy with which the spin-orbitals may be computed is defined by practical factors in the system, in general

$$\Psi(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}) \approx \sum_{K=0}^{M} A_K \Phi_K(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}) \qquad (1.A.2.4)$$

The **Hartree-Fock** model throws all its effort into obtaining the best possible *one term* expansion; $A_0 = 1$, $A_K = 0$ for $K > 0$. The **Configuration Interaction** and **Moller-Plesset** methods improve on this single-term model by extending the expansion using the *virtual* orbitals generated by the Hartree-Fock variational procedure as a bye-product.

1.A.3 Spin-Orbitals

There are just two possible spin "functions" conventionally written α and β and usually the generation of spin-orbitals which are *mixtures* of these two functions is not considered. Thus the computational problem is the determination of the **spatial orbitals** which are the spatially dependent factors of the spin-orbitals χ_i:

$$\begin{aligned} \chi_i(\vec{x}) &= \psi_{i'}(\vec{r})\alpha \\ &\qquad\quad or \\ \chi_i(\vec{x}) &= \psi_{i'}(\vec{r})\beta \end{aligned}$$

In many applications it is useful to have a more compact notation for the relationship between the spin-orbitals and the spatial orbitals since the spin function is just a label; the "bar" and "no bar" notation is used:

$$\begin{aligned} \psi_i &= \psi_i\alpha \\ \overline{\psi}_i &= \psi_i\beta \end{aligned}$$

That is, the notation ψ_i may mean the spatial orbital *or* the α spin-orbital with spatial factor ψ_i according to context. Sometimes care must be taken to distinguish the two cases.

1.A.4 Linear Expansions for the Spatial Orbitals

Each spatial molecular orbital ψ_i, a function of ordinary three-dimensional space, is expanded as a linear combination of **Basis Functions** which are fixed for a particular calculation and are chosen on a variety of theoretical and (mostly) practical grounds. These basis functions ($\phi_k(\vec{r})$)

are key elements in the success of any calculation of molecular electronic structure:

$$\psi_i(\vec{r}) = \sum_{k=1}^{m} \phi_k(\vec{r}) C_{ki} \tag{1.A.4.5}$$

Where there are m basis functions with which to expand the n optimum molecular orbitals. In view of the relationship between spatial and spin orbitals, not all the n spatial molecular orbitals need be different; sometimes there will be *pairs* which are the same and the associated spin-orbitals only differ in spin factor. The Hartree-Fock variational method optimises the linear coefficients C_{ki} to ensure the best possible (lowest energy) description of the molecular system.

For reasons which are entirely practical, the basis functions must be **Gaussian Functions**, functions which have a factor

$$exp(-\alpha|\vec{r}|^2)$$

as part of their functional form. Since the "natural" atomic orbitals have a dependence like $exp(-\zeta|\vec{r}|)$ there has to be a much longer expansion in terms of Gaussians to ensure an accurate molecular orbital is computed, typically two or three times the length of a Slater orbital expansion.

The use of Gaussian functions *directly* in 1.A.4.5 would therefore make excessive requirements of storage for the electron-repulsion integrals and so a compromise is used whereby the length of the explicit expansion in 1.A.4.5 is restricted by taking the basis functions themselves to be *fixed* linear combination of so-called **Primitive** Gaussians:

$$\phi_k(\vec{r}) = \sum_{j=1}^{n_k} \eta_j(\vec{r}) d_{jk} \tag{1.A.4.6}$$

Here the length of the expansion may depend on the basis function in question, so that the **Degree of Contraction** n_k depends on k.

1.A.5 Primitive Gaussians

The general form of a primitive Gaussian function is usually chosen to be the product of a Cartesian Factor and an exponential:

$$\eta(\vec{r}) = N x^{\ell} y^m z^n exp(-\alpha r^2) \tag{1.A.5.7}$$

where $r = |\vec{r}|$ and ℓ, m, n are integers which characterise the *type* or *order* of the Gaussian function. N is a numerical factor chosen to *Normalise* the function to unity, clearly depending on α, ℓ, m, and n.

If a Gaussian primitive is expressed in terms of a global co-ordinate system, the components of the position vector of the centre on which it is based appear in an obvious way.

$$\eta_j(\vec{r}) = N(\ell_j, m_j, n_j; \alpha_j)(x - x_A)^{\ell_j}(y - y_A)_j^m(z - z_A)_j^n exp(\alpha_j|\vec{r} - \vec{r_A}|^2) \tag{1.A.5.8}$$

where the explicit dependence of the primitive on the position of its nucleus is given; $\vec{r_A} = (x_A, y_A, z_A)$ is the position vector of centre A. The subscript j serves to identify this particular η_j among the many.

The type of this primitive is given by

$$t = \ell_j + m_j + n_j$$

and a terminology related to the familiar atomic orbitals is used:

$\mathbf{t = 0}$ an s-type Gaussian or zeroth-order Gaussian

$\mathbf{t = 1}$ a p-type Gaussian or first-order Gaussian

$\mathbf{t = 2}$ a d-type Gaussian or second-order Gaussian

$\mathbf{t = 3}$ an f-type Gaussian or third-order Gaussian

In the first two cases (s and p) there is a direct correspondence between the Gaussians and the real Atomic Orbitals. For d,f and higher Gaussians there are more Cartesian factors of a given type than real Atomic Orbitals of the corresponding angular momentum (t is equal to the total angular momentum quantum number usually written as ℓ in atomic theory).

This technique of retaining some of the Gaussian primitives in *fixed* linear combinations is called **Contraction**. Of course, the calculations using all the primitives still have to be performed, but the advantage gained by using so-called contracted basis functions is that *storage* is saved. The price to be paid for the contraction technique is loss of variational flexibility; only the linear combination coefficients of the *basis functions* are optimised, not the coefficients of each primitive.

1.A.6 Single Determinant Energy Expression

The mean value of the energy of a single determinant of orthonormal (orthogonal and normalised) molecular orbitals

$$\int \chi_i(\vec{x})\chi_i(\vec{x})d\tau = \delta_{ij}$$

is

$$E[\Phi] = \int \Phi^*(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n})\hat{H}(\vec{r_1}, \vec{r_2}, \ldots, \vec{r_n})\Phi(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n})d\tau_1 d\tau_2 \ldots d\tau_n \tag{1.A.6.9}$$

Integration over spin *and* space variables is denoted by

$$\int \ldots d\tau \quad or \quad \int \ldots d\tau_1$$

while integration over space *only* is written

$$\int \ldots dV \quad or \quad \int \ldots dV_1$$

That is:

$$\int \ldots d\tau = \int\int \ldots dV ds$$

where s is the spin "variable" . That is, dV refers to the three spatial variables in $\vec{r}$ and $d\tau$ to the four variables in $\vec{x}$.

Completion of the integration, by separation and use of the specific form of $\hat{H}$ (Equation 1.A.1.1), together with the orthonormality relationships reduces this integral (over $3n$ spatial variables and n spin "variables") to:

$$E = \sum_{i=1}^{n} h_{ii} + \sum_{i=1}^{n}\sum_{j \le i}^{n}(J_{i,j} - K_{ij}) \tag{1.A.6.10}$$

Where

$$h_{ii} = \int \chi_i^*(\vec{x})\hat{h}(\vec{r})\chi_i(\vec{x})d\tau$$

$\hat{h}$ is the one-electron hamiltonian 1.A.1.2 and

$$J_{ij} =< ij|ij >= \int d\tau_1 \int d\tau_2 \chi_i^*(\vec{x_1})\chi_j^*(\vec{x_2}) \left(\frac{1}{r_{12}}\right) \chi_i(\vec{x_1})\chi_j(\vec{x_2}) \tag{1.A.6.11}$$

$$K_{ij} = \int d\tau_1 \int d\tau_2 \chi_i^*(\vec{x_1})\chi_j^*(\vec{x_2}) \left(\frac{1}{r_{12}}\right) \chi_j(\vec{x_1})\chi_i(\vec{x_2}) \tag{1.A.6.12}$$

The physical interpretation of the terms is straightforword. The integrals h_{ii} are the energy of an electron moving in the attractive field of the nuclei *alone* (i.e. in the absence of the other electrons). The integrals J_{ij} are the mean repulsions between electrons occupying χ_i and χ_j; they are called "Coulomb" terms for this reason. The integrals K_{ij} arise from the anti-symmetry of the wavefunction and have no srict classical analogue but their principle function in the energy expression is to cancel out the "self-repulsion" which would be included if they were not present (notice the summation *includes* the term $i = j$). The K_{ij} are called "Exchange Integrals" because of the way they arise in the mathematics of the expansion of the determinant.

1.A.7 Notation for Repulsion integrals

It is usual to write the electron repulsion terms in the single-determinant energy expression as special cases of a more general repulsion integral e.g:

$$K_{ij} =< ij|ji >= \int d\tau_1 \int d\tau_2 \chi_i^*(\vec{x_1})\chi_j^*(\vec{x_2}) \left(\frac{1}{r_{12}}\right) \chi_j(\vec{x_1})\chi_i(\vec{x_2}) \tag{1.A.7.13}$$

The notation $< ij|ij >$ and $< ij|ji >$ has been introduced in anticipation of a more general electron repulsion integral $< ij|k\ell >$:

$$< ij|k\ell >= \int d\tau_1 \int d\tau_2 \chi_i^*(\vec{x_1})\chi_j^*(\vec{x_2}) \left(\frac{1}{r_{12}}\right) \chi_k(\vec{x_1})\chi_\ell(\vec{x_2}) \tag{1.A.7.14}$$

Integrals of this type appear in the energy expression of multi-determinant wave functions in the "cross terms" between different determinants and are included here to define the notation.

The complex-conjugate notation has been given explicitly since, although the primitives and basis functions are obviously all *real*, the expansion coefficients C_{ki} may well be complex in some applications; leading to complex χ_i.

There are a variety of notations for the repulsion integrals either singly or in "standard combinations" , each has its own logic and there is no overwhelming reason to choose one or the other. It is sometimes more intuitively acceptable to use the physical interpretation of these integrals as the net repulsion between a charge distribution $\chi_i(\vec{x_1})\chi_k(\vec{x_1})$ and $\chi_j(\vec{x_2})\chi_\ell(\vec{x_2})$ as a justification for the "charge-cloud" notation:

$$(ik, j\ell) =< ij|k\ell > \tag{1.A.7.15}$$

where *round brackets* and *no* vertical bar distinguish between the two notations. The charge-cloud notation is particularly useful when the integrals are over *basis functions* not molecular orbitals since these are always real and the various possible permutations of i, j, k, ℓ which do not change the value of $(ik, j\ell)$ are easier to see, in fact, if the functions *are* real then interchanging i and k, or j and ℓ, or the *pairs* (i, k) and $(j\ell)$ do not change the value of $(ik, j\ell)$ as can be seen from the definition (equation 1.A.7.14).

Integrals like 1.A.7.14 usually occur in *pairs* in the evaluation of energy integrals from determinantal wavefunctions and it is often convenient to use a single symbol to mean a Coulomb integral and its corresponding Exchange integral although the difference between the two terms is no longer valid if $i \neq k$ and $j \neq \ell$:

$$< ij||k\ell >=< ij|k\ell > - < ij|\ell j > \tag{1.A.7.16}$$

The double bar serving to indicate that the "exchange" term has been included.

1.A.8 Spatial Orbital Repulsion Integrals

Throughout the previous two sections the electron-repulsion integrals have been expressed in terms of the *spin-orbitals* $\chi_i(\vec{x})$, but, since the "integration" over the spin is trivial, it is always possible to reduce integrals like 1.A.7.14 to a "genuine" integration over space (six dimensional since there are two particles involved). The electron-repulsion operator $1/r_{12}$ does not involve spin so that the spin integration can always be separated into factors which are zero or one depending of the spin factors involved:

$$\int \alpha^* \alpha ds = 1 \tag{1.A.8.17}$$

$$\int \beta^* \beta ds = 1 \tag{1.A.8.18}$$

$$\int \alpha^* \beta ds = 0 \tag{1.A.8.19}$$

$$\int \beta^* \alpha ds = 0 \tag{1.A.8.20}$$

so that any integral $< ij|k\ell >$ which contains spin-orbitals χ_i and χ_k which have *different* spin factors will be *zero* as will any integral with different spin factors in χ_j and χ_ℓ.

Thus, if the spin factors in the pairs χ_i, χ_k and χ_j, χ_ℓ are the *same* (and only then) the integral

$$< ij|k\ell >= \int d\tau_1 \int d\tau_2 \chi_i^*(\vec{x_1})\chi_j^*(\vec{x_2}) \left(\frac{1}{r_{12}}\right) \chi_k(\vec{x_1})\chi_\ell(\vec{x_2}) \tag{1.A.8.21}$$

becomes

$$< i'j'|k'\ell' >= \int dV_1 \int dV_2 \psi_{i'}^*(\vec{r_1})\psi_{j'}^*(\vec{r_2}) \left(\frac{1}{r_{12}}\right) \psi_{k'}(\vec{r_1})\psi_{\ell'}(\vec{r_2}) \tag{1.A.8.22}$$

where $\psi_{i'}$ is the spatial factor in the spin-orbital χ_i etc.

This fact reduces the number of repulsion integrals, particularly exchange integrals.

1.A.9 Basis Function Repulsion Integrals

In the calculation of molecular electronic structure by the basis function expansion method it is necessary to calculate the *molecular orbital* repulsion integrals by calculating the corresponding replusion integrals involving the *basis functions* and using the linear combination coefficients to generate the *molecular orbital* integrals. That is the integrals

$$< ij|k\ell >= \int dV_1 \int dV_2 \phi_i^*(\vec{x_1})\phi_j^*(\vec{x_2}) \left(\frac{1}{r_{12}}\right) \phi_k(\vec{x_1})\phi_\ell(\vec{x_2}) \tag{1.A.9.23}$$

Where the same symbol $< ij|k\ell >$ has been used to denote an electron repulsion integral over the basis functions as was used in the last section for these integrals over molecular orbitals. This is standard practice and it is usually clear from the context which integral is meant. If there is doubt then the more explicit notation

$$< \phi_i\phi_j|\phi_k\phi_\ell >= \int dV_1 \int dV_2 \phi_i^*(\vec{r_1})\phi_j^*(\vec{r_2}) \left(\frac{1}{r_{12}}\right) \phi_k(\vec{r_1})\phi_\ell(\vec{r_2}) \tag{1.A.9.24}$$

may be used for the basis-function integrals and

$$< \chi_i\chi_j|\chi_k\chi_\ell >= \int d\tau_1 \int d\tau_2 \chi_i^*(\vec{x_1})\chi_j^*(\vec{x_2}) \left(\frac{1}{r_{12}}\right) \chi_k(\vec{x_1})\chi_\ell(\vec{x_2}) \tag{1.A.9.25}$$

for the molecular orbital integrals. In practice, it is usually the basis-function integrals which are denoted by the simpler form $< ij|k\ell >$ and, if necessary, the molecular orbital integrals by the more explicit notation.

Notice that, since the basis functions are *spatial* functions the basis-function electron-repulsion integrals involve no spin integration; if one of them is zero it is because of symmetry or simply because the two charge-clouds are very remote.

Chapter 2

Standard Methods

2.1 Preview

The mnemonics available to access the facilities provided by Standard Methods provide a kind of terse command-shell to drive the whole of `gnu80`. The command themselves are placed on a single "Command Record" which is identified to `gnu80` by a hash (#) in its first column. Items on the Command Record are separated by comma (,) slash (/) minus (-) or space ( ). The whole Command Record, typically, specifies the quantum mechanical model, the choice of Gaussian basis and the use to which this combined approximation is to be put. There are standard defaults for unspecified commands. Before giving the definition of all the possible entries on the Command Record, some specific examples with a brief explanation will provide the flavour of this command shell.

# HF/STO-3G

This record requests the performance of a Hartree-Fock (single determinant) calculation using a standard **STO-3G** Gaussian basis for all the atoms of the molecule. By default, **RHF** is chosen if the system is closed-shell or **UHF** if the system is open-shell).

Throughout this manual the terms "Hartree-Fock" and "Self-Consistent Field (SCF)" are used interchangeably to mean the use of the Linear Expansion technique (LCAO) within the single-determinant model. See Appendix 2.B for possible pitfalls of this assumption.

# MP2/3-21G OPT

Here, a second-order Moller-Plesset calculation of the total energy is requested (necessarily preceded by a **UHF** or **RHF** calculation, as appropriate) using a split-valence standard basis for each atom in the molecule. The **OPT** command requests that the calculation be repeated for automatically-generated changes in the molecular geometry until a minimum is obtained at the OPTimum geometry.

This latter example illustrates another choice which must be made when using a particular quantum mechanical model; in many cases, for a combination of theoretical and practical reasons, one must choose a particular numerical algorithm and/or some limitations on the way the algorithm or the underlying quantum mechanical method is implemented.

In the particular case cited above, the optimisation of molecular geometry, it is sometimes necessary to specify the particular technique used to "drive" the quantum mechanical energy calculation method; either to be sure of generating scientifically meaningful results or to avoid excessive demands on computer resources. Thus in the case considered here, it is a matter of computational experience that the use of electron configurations in which the inner-shell electrons are excited in the perturbation expansion has minimal effect on the optimised geometrical parameters; as one might reasonably expect.

In point of fact, `gnu80` has only *one* optimisation algorithm which can be used with the **MP2** method (the Fletcher-Powell method, which does not require explicit derivatives of the energy with respect to the geometrical parameters: bond distances, bond angles etc.) but in the case of the **HF** model there are three algorithms to choose from.

Choices of this kind *within* a given model/numerical decision are called **OPTIONS** in `gnu80`. In addition to the "major" options of the type described above, there are many "minor" options available to govern the amount of output generated etc. etc. As a general rule the major options for a given choice of model/basis/use may be given in mnemonic form on the Command Record while many of the minor options must be specified by using a specific command **NONDEF** (for NON-DEFault options) on the Command Record followed by a list of such options.

The vast majority of cases of the use of Standard Methods do not require the use of **NONDEF**. Indeed, there are *default* options (chosen on the basis of much experience) for most of the types of calculation which have to be done and, in most cases of the use of Standard Methods the user need not even be aware of the existence of options.

In the case of **MP2** calculations `gnu80` automatically chooses the Fletcher-Powell optimisation algorithm and the "Frozen Core" option which excludes the use of core excitations in the perturbation expansion. However to illustrate how these options might be explicitly specified, the original Command Record is completely equivalent to the record

# MP2(FC)/3-21G OPT=FP

Where **(FC)** indicates the Frozen-Core option *within* the **MP2** method and **OPT=FP** explicitly choose the Fletcher-Powell optimisation technique. If the **MP2** calculation has to include *all* possible terms in the perturbation expansion then **(FC)** is replaced by **(FULL)** on the Command Record.

The full range of options accessed by this mnenonic method on the Command Record is given in the section 2.3.

Of course, the Command Record has no data describing the particular molecular system under investigation, this is provided by a separate section of the whole input data to `gnu80`.

The other pieces of data required to complete the description of the molecular electronic structure are the number of electrons and the spin multiplicity of the system. The number of electrons is determined from the atomic numbers of the atoms of the molecule which themselves are determined from the atomic chemical symbols which are read in as part of the molecular geometry.

If the molecule is charged this information must obviously be supplied and the spin multiplicity must also be supplied.

Because the Command Record specifies the sequence of calculations to be carried out and the order in which they must be performed i.e. the set of modules of `gnu80` to be executed, this sequence can be thought of as a *path* or *route* through `gnu80`. This ROUTE, once generated from the Command Record, is stored and used as control information during a `gnu80` run.

The principle difference between Standard Methods and General Use is the way in which this route is generated. Standard Methods involve the automatic generation of the route from the Command Record while General Use enables the route to be specified explicitly by the user. The **NONDEF** command provides a "intermediate" method of modifying a given route through `gnu80`.

This section has given a general overview of the command record and its use; later sections give detailed specifications of the possible entries on the Command Record.

2.2 Input Organisation

Input to `gnu80` consists of records of alphanumeric information usually free of (FORTRAN) format constraints. In the case of alphabetic information upper and lower case letters are completely interchangeable and may be used at the user's discretion to enhance the legibility of the input. Thus, the cobalt atom may be identified as CO, co, Co or even cO. The same applies to the commands on the Command Record; the example in the last section could well have been given as:

# hf/STO-3g

for example. The first thing that gnu80 does with an input record is to convert any alphabetics to upper case and it works internally entirely in upper case.

The input of a gnu80 run consists of several separate *sections* separated by blank records. The Input Sections themselves are not identified except by their *position* in the input file and the blank records separating them from their predessors. For example, the "Title" Input Section is not preceeded by the word TITLE, it is simply those records following the blank record after the Command Record. Similarly, the "Variables" Input Section is just those records (assuming them to be in the correct form) which follow the blank record after the Z-matrix; the Section is not announced by the word VARIABLES, for example.

Each section consists of one or more records (or lines) and is terminated by a blank record (or line). We shall use the word "record" from now on to mean record or line. The list of input sections, in the sequence in which they *must* appear if they appear at all, is:

1. **File Control Records (if any)**
2. **Command Record (and Non-Default options, if any)**
3. **Title**
4. **Molecule Specification**
5. **Variables Specification (if any)**
6. **Constants Specification (if any)**
7. **General Basis Specification (if any)**
8. **Alteration of Configuration** (α) **(if any)**
9. **Alteration of configuration** (β) **(if any)**

Each Section is described separately below. Only the sections 2 — 4 are **required** in every Job. If a particular section is not supplied, its terminating blank record should *not* be included in the input file.

The first section of this input structure, consisting of up to four *File Control* commands (identified by a percent sign, % , in column 1) can be used to preserve the details of the job for a re-start. These records are not essential for the preparation of successful jobs and a discussion is deferred until the **#RESTART** command is introduced.

2.3 The Command Record

This single record requires the most description since it defines the *nature* of the calculation to be performed; the quantum mechanical model, the numerical accuracy (Basis set) and the particular application. The commands on the Command Record are very terse but require non-terse specifications.

The Command Record itself is identified by a hash (#) in the first column. On initiation, `gnu80` reads records with no interpretation until it finds a Command Record. Immediately following the # on the Command Record may be one of N P or F which set certain "global" printing options for the whole run:

N Suppresses as much output as possible consistent with providing a useful summary of the job. Use **N** when there are no problems are anticipated with the job.

P Generates more verbose output from various links (particularly **HF**) and prints a summary of the passage through the Links. Useful for general information occasionally and if convergence difficulties are anticipated.

F Generates messages from the File I/O routines; only useful for debugging.

The # (or # N, # P or #F) *must* be followed by a space before the substance of the Command Record is given.

Using Standard Methods, the Command Record specifies three general pieces of information about the whole calculation:

- The Model of the molecular electronic structure to be used
- The accuracy of the expansion of the molecular orbitals (the particular basis used)
- The physical problem to be attacked by the use of this particular combination of model and accuracy

Obviously, some thought must be given to the first two of these items in order that the results are useful and applicable to the third. Some of the grosser errors are indicated in Appendix 2.B.

In addition to these three main commands the Command Record may be used to choose some commonly-required modifications of the way in which the calculation is to be performed; there are some Command Record mnemonics for certain common OPTIONS within the model/basis/use choice.

2.3.1 Models of Electronic Structure

One of the following commands must always be given (either explicitly or by default) for a `gnu80` run; each one specifies the overall approximation to be made of the molecular electronic structure of the system under investigation.

There are three classes of model:

1. Variants of the Molecular Orbital or Hartree-Fock model,
2. The (non-variational) Moller- Plesset perturbation method which includes some electron correlation,
3. The (variational) Configuration Interaction method using double substitutions which also includes electron correlation.

Factors influencing the choice of model are the obvious ones of accuracy against consumption of resources.

If the job involves the optimisation of molecular geometry, the Hartree-Fock methods are often the only practical choice, since they are inherently less expensive of computer resources particularly since it is only possible to use analytical gradient methods at the Hartree-Fock level in `gnu80`. In any case, geometry optimisations will usually *begin* by using the Hartree-Fock model and possibly be refined using the correlated methods.

The possible commands to choose a model of electronic structure, together with their major (command-record) options are given below:

HF (or RHF, UHF) These commands request Hartree-Fock calculations. If **HF** is given, the default based on the (supplied) multiplicity is applied. **RHF** (Closed-Shell Restricted Hartree-Fock) is used for singlets and **UHF** (Unrestricted Hartree-Fock, actually Different Orbitals for Different Spins: DODS) for higher multiplicities. In the latter case, separate α and β orbitals will be computed.

Similar operations are implied by specification of **U** or **R** with other procedures (**MP2** vs **RMP2** vs. **UMP2**, for instance); see below.

ROHF This command requests a Restricted Open-shell Hartree-Fock calculation for those non-singlet states which can be represented by a single determinant.

The resulting molecular orbitals will be either doubly or singly occupied and be spatially orthogonal to each other unlike the UHF spatial orbitals for different spins.

The single determinant function (and the component Molecular Orbitals generated by this procedure do not have many of the familiar formal properties of Hartree-Fock (Self Consistent) functions. Koopman's theorem is not valid and the Brillouin theorem is not obeyed, for example. These differences all arise from the fact that the ROHF wave function is a *constrained* single determinant.

SCFDM This command requests the use of a Direct Minimisation method to minimise the Hartree-Fock Energy Functional instead of the repeated-diagonalisations method used by **HF, UHF** and **ROHF**. It is more time-consuming than the standard procedure but its convergence properties are better. It should be used when **HF** or **UHF** do not converge (it is not available as a replacement for **ROHF**).

MP2 (or RMP2, UMP2) Requests second order Moller-Plesset perturbation calculation (restricted or unrestricted, depending upon **R** or **U**). Both **MP2** and **MP3** also generate a Hartree-Fock calculation as a necessary precursor to the perturbation calculation which uses the Self-Consistent Molecular Orbitals.

MP3 (or RMP3, UMP3) Requests second and third order Moller-Plesset perturbation calculation (restricted or unrestricted, depending upon **R** or **U**).

CID Requests a Configuration Interaction calculation with Double excitations. Again, this command requests a Hartree-Fock calculation (of the appropriate type: **RHF** or **UHF**) to generate the Molecular Orbitals for the CI.

Commands with Options Associated with the Model Commands

There are a few options available via mnemonics on the Command Record for the Hartree-Fock commands:

VSHIFT Does nothing unless given an option; (**VSHIFT=nnnn**). The occupied/virtual energy gap is increased by $nnnn/1000$. This is just an implementation of the "Level-Shifter" method to improve SCF convergence.

SCFCYC This is used to specify the maximum number of SCF cycles allowed. it is meaningless by itself, and is used by **SCFCYC=number**. The default for ' number' is 20 cycles for **SP, HF** jobs, and 32 cycles for **OPT** or post-SCF jobs. The use of the "Level Shifter" Command **VSHIFT** to improve convergence resets the default value of cycles to 50; this can be over-ridden as usual by the **SCFCYC** command.

COMPLEX This option allows the orbitals of the single determinant wave function to be complex. The option is only available for **RHF** i.e. not for **UHF**, **ROHF** or any post-SCF calculation.

The non-Hartree-Fock commands all have a similar set of OPTIONS which can be given on the Command Record; typically in parenthesis immediately following the command. The options describe the *extent* to which the perturbation or configuration interaction calculation will be carried out:

FC This is the default in the absence of any explicit option being given for **MP2, MP3, CID** commands. Its action is to limit the number of terms in the perturbation or CI expansion by excluding those arising from the excitation of the inner shell electrons.

FC is a mnemonic for Frozen Core.

FULL If the **FULL** option is given all possible terms in the perturbation or CI expansion are used even those involving the excitation of the atomic inner shells or cores.

Note that ONLY ONE of **UMP2, MP2, UMP3, MP3, CID** may be used on a given Command Record. Typical Command Records using the correlated models of electronic structure are:

# UMP2(FULL)/3-21G

# CID(FC)/6-31G**

#P MP3(FC)/STO-3G

where the command following the slash (/) requests a type of Gaussian basis detailed in the following section.

2.3.2 Basis Sets Internal to gnu80

When a *model* of electronic structure has been chosen, the most important *numerical* approximation within that model is the nature and length of the expansion used to approximate the Molecular Orbitals within that model. That is, one must make a *choice of (Gaussian) basis.*

In the same mnemonic spirit introduced for the quantum mechanical models there are a number of stored Gaussian basis sets available in `gnu80` which may be used via the following standard (descriptive) mnemonics. Thes mnemonics are also commands to `gnu80` and use of *one* of these on the Command Record will generate a basis set for the system described in the Molecule Input Section. This command chooses an overall molecular basis which has the same *type* of atomic Gaussian basis on every atom in the molecule.

- **STO-NG, STO-NG* (for N = 1,6) e.g STO-3G*, STO-4G**
- **3-21G, 3-21G*, 3-21G****
- **4-21G, 4-21G*, 4-21G****
- **6-21G, 6-21G*, 6-21G****
- **4-31G, 4-31G*, 4-31G****
- **6-31G, 6-31G*, 6-31G****
- **6-311G, 6-311G*, 6-311G****
- **LP-31G, LP-31G*, LP-31G****
- **LP-41G, LP-41G*, LP-41G****
- **LANL1MB, LANL1DZ (LP-31G is a synonym for LANL1DZ)**

The meanings of these mnemonics are assumed to be familiar.

By default, **STO-3G** is selected. The only option available is the specification of the type of *d* or *f* orbitals used (see below).

The "Local Potential" Basis Sets (e.g. **LP-31G**) will lead to calculations involving valence electrons only; that is, selection of any of these bases will **automatically** use an associated Local Potential on each atom in the molecule (except hydrogen and helium). The particular choice of Local Potential used is explained elsewhere. (see the description of the **PSEUDO** command.

Note that the use of the **PSEUDO** command simply uses the Local Potentials on the atoms; **it does not involve any choice of basis** so that a basis must be specified. Usually, this means the use of a user-supplied basis (see the **GEN** command) since the other bases are full bases including core orbitals and the use of the Local Potential basers also chooses an approriate basis so that **PSEUDO** is redundant there.

2.3.3 ECP Basis Set Types in gnu80 and g94

The Basis Set commands in `gnu80` have been changed to make them compatible with `g94`. They now are:

- **LANL1MB**; this is the same basis as g94.
  It uses the Minimal Los Alamos Bases for atoms from Na - Bi and an STO-3G set for atoms H-Ne.
  The Los Alamos Potentials are used for atoms Na-Bi and NO potential is used for first-row atoms; all electrons are treated explicitly. That is, for example, the Carbon atom has a full (5) basis (1s, 2s,2p) and atomic charge 6.

- **LANL1DZ**; again the same as g94.
  The Los Alamos Double Zeta Basis is used for atoms Na-Bi and a Dunning D95V ("Double Zeta for valence") for atoms H-Ne.
  The Los Alamos Potentials are used for atoms Na-Bi and NO potential is used for first-row atoms; all electrons are treated explicitly. That is, for example, the Carbon atom has a full (9) basis (1s, 2s, 2s',2p,2p') and atomic charge 6.

- **LAL1STV**; this is not available in g94.
  It uses the Minimal Los Alamos Bases for atoms from Na - Bi and an STO-3G set for atoms H-Ne.
  The Los Alamos Potentials are used for atoms Na-Bi and a CHF potential is used for first-row atoms. That is, for example, the

Carbon atom has a valence (4) basis (2s,2p) and atomic charge 4 plus the potential.

- **LAL1LP3**; this is not available in g94.
  The Los Alamos Double Zeta Basis is used for atoms Na-Bi and a LP-31G ("Double Zeta for valence") for atoms H-Ne.
  The Los Alamos Potentials are used for atoms Na-Bi and a CHF potential is used for first-row atoms. That is, for example, the Carbon atom has a split-valence (8) basis LP-31G (2s, 2s',2p,2p') and atomic charge 4 plus the potential.

  Notice that this means that previous gnu80 input files, which used LANL1MB for what is now LA1STV and LANL1DZ for what is now LA1LP3, must be changed if the same basis is required.

This is awkward but the non-compatibility with g94 is also awkward.

2.3.4 GEN: User-supplied Bases

Obviously, the user may wish to supply gnu80 with a basis set of his own choice. This is possible by use of the command **GEN** on the Command Record in place of a basis set mnemonic. The details of the input of a basis which is not internal to gnu80 are given below in Section 3.6.

2.3.5 nD, nF; Cartesians or Cubic Harmonics?

The only other basis-related command available once an overall basis set command has been given is in the choice of Cartesian or Cubic Harmonic functions for *d* and *f* functions.

The "Cartesian" Gaussian bases generate a set of 6 *"d"* functions which are equivalent to the usual set of 5 (Cubic Harmonic real) d functions plus an additional "s" -type Gaussian. Likewise, the Cartesian set of *"f"* 10 functions is equivalent to a set of 7 (Cubic Harmonic real) *f* functions plus a set of 3 *p* functions.

For some purposes the presence of these additional *s*-type or *p*-type functions produces an "unbalanced" basis and may lead to unforeseen results. It is possible to specify which case is required by the use of

5D or **6D** and **7F** or **10F** as appropriate; thus the above Command Record could be replaced by:
# MP2/6-31G* 5D or # MP2/6-31G* 6D

depending on individual requirements. The defaults are **5D**, **7F**.

The relative merits are obvious; **6D** introduces an additional degree of variational freedom for each *d*-set at the risk of unbalancing the *s*-type functions in the basis.

2.3.6 Model/Basis Summary

The quantum mechanical procedure and the basis-set commands may be conveniently combined as a specification of the theoretical model/numerical approximation scheme. Thus the Command Record:
# MP2/6-31G*

Requests an **MP2** (Frozen Core by default) calculation using the **6-31G*** basis set. The established convention is to use a slash (/) to separate the model command (in this case **MP2**) from the basis-set command (**6-31G*** here) to provide a convenient mnemonic for the "quality" of the calculation which is easily recognizable; although any of the Command Record command separators would be acceptable.

2.3.7 Uses of the Quantum Model

SP Requests a "single point" calculation; that is an energy calculation at the specified nuclear geometry. No optimisation of geometrical parameters is performed. This is the default type of job and so is not often explicitly specified.

OPT Requests that a geometry OPTimization is to be performed. The (geometrical) variables listed in program input Section 4 (see below) will be adjusted until a stationary point on the potential surface is found. By default the best available procedure is selected. thus, you can keep up with current developments by simply supplying **OPT** with no options. However, if conditions demand a particular type of optimization, it may be chosen by supplying the appropriate option.

FORCE This requests a single analytical calculation of the forces on the nuclei and that the energy derivatives with respect to the geometry parameters be evaluated. This is currently available only for HF energies.

NONDEF This command indicates that additional non-default options will be read in from the next recordset (see below).

TEST This command indicates that this is a test job. the route will be generated as normal. The job is aborted before the calculation of any integrals.

NONSTD This command indicates that this is not a standard job; the desired route will be explicitly supplied in the form of overlay numbers, options, and segment numbers. This is, in fact, the way in which the method of General Use is indicated to `gnu80`.

OPTIONS for the OPT Command; OPT=xxxx

The **OPT** command is, of course, the most widely used command specifying the use of a quantum mechanical model and there are a number of OPTIONS which may be specified by mnemonics on the Command Record:

GRAD Requests use of analytical gradients and is only available for HF (RHF or UHF) calculations.

MS Use Murtaugh-Sargent optimization [00]. It does use analytical gradients and is only available for **HF** (**RHF** and **UHF**) calculations. **OPT=MS** is generally slower but more reliable then **OPT=GRAD**.

FP Fletcher-Powell optimization [00]. This option does not require analytical derivates and may be used in conjunction with any procedure.

READFC For the first point in a gradient optimisation, read the force constants from the guess file. These must have been produced in a previous run, for example by the **STARONLY** option.

TS Optimize to a transition state - a turning point on the energy surface for which the Hessian has one and only one negative eigenvalue pictorially a saddle point on the surface.

STARONLY For **GRAD** optimisations, this option requests that the force constants associated with the movements in the degrees of freedom specified in the **Variables** input section will be estimated numerically. These force constants may be used as input to the **MS** optimisation program as starting points for the second derivative matrix. The second derivatives of the energy will be computed as directed on the variable definition records (see below).

Defaults: The default options for the command **OPT** are **GRAD** for Hartree-Fock (**RHF** or **UHF**) calculations and **FP** for post-Hartree-Fock.

2.3.8 Miscellaneous Commands

These command and associated options provide a variety of utilities in `gnu80`

SAVE saves selected data on the **GUESS** file. by default, this command saves both the Basis Set and the Molecular Orbital coefficients. (see also **#RESTART** and the File Control Commands)

Options for the **SAVE** command (**SAVE=xxxx**):

- **BASIS** saves the basis set.
- **MO** saves the Molecular Orbital coefficients.
- **FC** saves the Force Constants, if these are calculated. Note the unfortunate collision with the Frozen Core mnemonic available as an option for another command; there should be no danger of confusion.

GUESS specifies initial guess options. This does nothing unless one of the options is actually given. In the absence of the command, a default projected Huckel guess is used.

Options for the **GUESS** command (**GUESS=xxxx**):

- **READ** Read initial guess. This guess must have been created as output from a previous job and will be associated with a File Control Command to specify the file to read the guess from. (see also **RESTART**).
- **CORE** Diagonalize core Hamiltonian.
- **ALTER** Indicates that the orbitals selected for occupation in the HF wave function should not be those of lowest energy. If this option is used, information about the alteration of configuration will be expected in Sections 7 and 8 of 2.2. **ALTER** may be used itself as a command, being equivalent to **GUESS=ALTER**.
- **PRINT** Causes the initial guess to be printed.

ONLY Results in a route which goes only as far as L401 (the module which generates the guessed coefficients), and the guess is automatically printed when this option is selected. This is useful in preliminary runs to check if configuration alternation is necessary.

NOEXTRAP Issuing this command will cause all extrapolation in the SCF to be suppressed.

COORD Indicates that the geometry will be supplied in the form of Cartesian coordinates (not a Z-matrix). No optimization is possible in this case.

NORAFF This command demands that the "regular" integral format be used.

OPTCYC This sets the maximum number of optimization cycles. The format is **OPTCYC = number**, for instance. The default for ' number' is 10 plus one for each degree of freedom, with a maximum of 20 cycles. If this number is too small the chances are that there is something seriously wrong withe the model or the basis!

SYMM Specifies how symmetry is to be used in the calculation. By default, symmetry is used whenever possible, so this command means nothing unless an option is provided.

Options for the **SYMM** command:

INT Use symmetry in two-electron integral evaluation.

NOINT Do not use symmetry for these integrals.

GRAD Use symmetry in gradient evaluation.

NOGRAD Do not use symmetry in gradient evaluation.

NOSYMM This command supresses any use of symmetry in the calculation. By default, the symmetry of the molecule may be used to avoid calculation of some integrals either because they are zero by symmetry or because they are equivalent to other (calculated) integrals.

NOPOP Eliminates wavefunction and full population analysis.

MINPOP Causes a “minimal” population analysis. The orbital symmetries are printed, along with the “condensed to atoms” summary.

CNOE “Causes a Complete Neglect of Everything” calculation to be performed. This command does no calculation at all; it simply reads the data and generates the ROUTE. It is useful to generate ROUTEs for later modification.

ALTER Requests that the initial guess MO coefficients be rearranged. The program will read pairs of integers indicating orbitals which are to be interchanged in the initial guess. This input forms one input section, or two input sections if the calculation is **UHF** (one for α, one for β).

PSEUDO Requests that a model potential be substituted for the core electrons. This is automatically selected if one of the **LP-** bases or the **LANL1MB or LANL1DZ** bases have been specified on the Command Record.

UNITS Definition of the units used in the Z-matrix i.e. in the Molecule Input Section of the input data. Does nothing without an option:

Options for the **UNITS** command (**UNITS=xxxx**):

ANG Bond lengths are in Angstroms.

AU Bond lengths are in Bohrs.

DEG Angles are in degrees.

RAD Angles are in radians.

The default values are **(ANG, DEG)**.

2.3.9 Commands not yet Implemented

Since `gnu80` is, in one sense, a "snapshot" in the development of the GAUSSIAN series of programs, some parts of the code are behind others in the implementation. There are a number of commands which are recognised by the parser but the code to carry through the requested calculation is not present or not complete. In these cases a message is printed to that effect and the run is stopped. Clearly, the code to implement these commands are prime examples of additions to `gnu80` which could be carried through simply at the level of the numerical calculation with no changes necessary to the parse tables.

The unimplemented commands are:

GRAD for post-SCF models

MP4 fourth-order Moller-Plesset perturbation calculation of the correlation energy.

FREQ calculation of the vibration frequencies and normal co-ordinates (requires second derivative integrals)

CISD Configuration interaction with *single and* double excitations.

STABIL test the stability of (HF)SCF wavefunctions for stability with respect to certain classes of change.

FORCE force calculations for post-SCF models

2.3.10 System Defaults

By default the following commands and options are selected:

- **SP**
- **HF** (i.e. **RHF** for closed shells, **UHF** for open shells.)
- **STO-3G**
- **OPT = GRAD** for SCF optimisations, **OPT=FP** for post-SCF optimisations (**MP2, MP3, CID**)

- **FC** Frozen Core for all multi-determinant calculations (**MP2, MP3, CID**).

Note, however, that if an **HF/STO-3G** calculation is required (all defaults), a blank Command Record is *not* permitted.

2.3.11 Examples of Valid Command Records

Some examples of correct Command Records which generate valid routes may help make all this more clear:

- # RHF/STO-3G, SAVE=MO
- # HF/3-21G, VSHIFT=500
- # MP3(FULL)/6-311G** OPT
- # UHF-CI=FC/6-31G* OPT=FP
- # RHF/6-31G*, GUESS=READ
- # HF/STO-3G, SCFDM, ALTER, SAVE
- # RHF/STO-3G OPT NONDEF
  (blank record)
  3/34=1,35=4;3(2)/34=1,35=4;

This last Command Record is used to select a standard route, but the options 34 and 35 in the first and the second occurence of overlay 3 are set to 1 and 4 respectively. The meaning of the last example will be clear after the section on General Use has considered explicit routes and options.

Note that the possibility of separating commands and options by spaces, slashes or commas may be used to make the Command Record more intelligible to according to individual choice.

This completes the description of the commands and options which may be used on the Command Record input to `gnu80`. The Other sections of data usually constitute more data but require considerably less discussion.

2.4 Title Input Section

This section is required in the input but is not interpreted in any way by the gnu80 system. It appears in the output for purposes of identification and description. Typically, this might contain compound name, the symmetry, the electronic state and other relevant information. The title section cannot exceed five records. However, a single record is usually adequate. Remember that the title section must have a terminating blank record.

2.5 Molecule Input Section

This section is, of course, always required. It specifies the nuclear positions and the number of electrons of α and β spin. The input is free-field; the several items on each record may be separated by either blanks or commas.

The first record of the section specifies the net electric charge (signed integer) and the spin multiplicity (positive integer). Thus, for a neutral molecule in a singlet state, the entry ‘ 0 1’ is appropriate, while for a single charged anion radical, ‘ -1 2’ would be used.

The remaining records are used to specify the relative positions of the nuclei. Most of these will be real nuclei, used later in the molecular orbital computation. However, it is frequently useful to introduce “dummy nuclei” which help specify the geometry but are ignored subsequently; their use will become clear in examples given below.

Since the Molecule Input Section is the most demanding to prepare, it is worth a brief overview of the method used to specify the molecular geometry in gnu80.

2.5.1 The Z-Matrix, an Overview

Before giving the method used by gnu80 to read the geometrical information necessary for a job, it is useful to have an overview of the so-called Z matrix method. Some of the information in this outline is, of course, duplicated in the reference section of the manual below.

There are, at least, three obvious methods to supply the data specifying the relative positions of the atoms in a molecule to a program:

1. Use a "laboratory" Cartesian co-ordinate system and give all the atomic positions as absolute Cartesian co-ordinates.

2. Still using a laboratory co-ordinate system, give the molecular symmetry (say the point group symbol) and the absolute Cartesian co-ordinates of the symmetry-distinct atoms in the molecule; allowing the program to apply the molecular symmetry to generate the full Cartesian co-ordinates for the entire molecule.

3. Use an "internal" co-ordinate system and specify only the *relative* positions of the atoms and leave the program to provide an origin and orientation of the laboratory system and do the conversion to a Cartesian set.

Obviously, in cases 1 and 3 the symmetry group of the molecule may be *inferred* from the supplied co-ordinates.

The advantages and disadvantages of each approach are clear:

1. This choice is only really practical for small molecules or for very symmetrical molecules.

2. This method is most convenient for highly symmetrical systems; in most cases of low symmetry, it will degenerate into 1.

3. This is the most flexible approach, particularly since most experimental molecular data are presented as bond distances, bond angles etc. referred to a "natural" non-redundant system of internal co-ordinates.

However, when the question of the *automatic* generation of changes in molecular geometry during a structure optimisation arises, it is obvious that the use of the natural molecular parameters of choice 3 above is the only viable option. The variation of what are essentially arbitrary Cartesian co-ordinates is bound to be inefficient and redundant at best. At worst, the use of three independent co-ordinates for each atom in a molecule makes such automatic optimisations impossible.

That is, a description of molecular geometry in terms of the essential molecular geometrical parameters (bond distances and angles) is the only sensible way to both input data to a program and to organise such data internally to allow automatic structure variations which are likely to be chemically meaningful and numerically efficient.

The method used by `gnu80` (and by all the GAUSSIAN series and many other *ab initio* programs) is the so-called **Z-matrix** method.

The technique is to "walk " through the molecule, specifying the position of each atom as it is encountered by its distance from and orientation to the other atoms already encountered. In this way the natural geometric parameters used by chemists and spectroscopists, can be used directly to generate an overall molecular structure which is unique but not (as in the case of Cartesian co-ordinates) over-determined. The overall orientation of the molecule with respect to the laboratory co-ordinate system is not specified. This latter freedom is used by `gnu80` to position and orient the molecule in a way which co-incides with the standard group-theoretical choice of Cartesian axes if this is useful.

To completely specify an atom by this method, we need its atomic number and enough geometrical information to place it uniquely with respect to the other atoms in the system.

1. As a matter of convience, the atomic number is determined by the chemical symbol (H, LI, MN, etc.) and so this is the way the atomic number is supplied.

2. The "first" atom in the molecule needs no geometrical information, it is specified completely by its chemical symbol and all other atoms' positions are referred (either directly or indirectly) to it. As a convenience an atom' s label may have other characters following the chemical symbol in order to make the data more legible to humans; so, for example, an atom may be labelled C1 or LI4 etc. Let us assume that we have labelled our first atom C1. `gnu80` internally places this atom at the origin of the laboratory Cartesian co-ordinate system.
   The form of the input data (the line of the Z-matrix) is simply

   ```
       C1
   ```

3. The next atom, which typically will be an atom formally bonded to the first (i.e. a nearest neighbour) although, logically, it need not be, has its nature and position completely specified by its chemical symbol and its distance from the first atom; no angles are involved. Let us assume that it is called C2.

   The Z-matrix line of input consists of the label of the second atom, the label of the atom to which its distance is to be given and that distance:

   ```
   C2 C1 RC2C1
   ```

   where `RC2C1` is the distance between C1 and C2 and may be supplied (in Angstroms) as a number like 1.4 or may be left as an alphanumeric string whose numerical value is supplied later. Clearly, it is more convenient to call it `RC2C1` than `LUCIFER`, for example, but both are allowed. The *direction* from the first to the second atoms is arbitrary and is chosen by `gnu80` as the Cartesian z axis.

4. The third atom (C3 say) needs at least one distance and an angle to complete its specification. Since any triatomic is planar, `gnu80` places the third atom in the xz plane of the Cartesian system, which means that the first three atoms all lie in the xz plane. As before, C3 is specified by the three items `C3 C2 RC3C2` if the distance between C3 and C2 is the most convenient to supply. But this time the *angle* between the C3-C2 and C2-C1 bonds must be given. This information is supplied by giving the label of the third atom in the chain and the value of the angle (in degrees):

   ```
   C3 C2 RC3C2 C1 A
   ```

   where `A` is the C3C2C1 angle.

5. The fourth and (all subsequent atoms) requires more information to specify it uniquely since it may well be out of the plane defined by the first three atoms.

   The most convenient way to do this (with one important exception, which will be treated separately later) is to use a distance and a bond angle as in the previous case and to specify the *"out-of-plane angle"* formed by the new bond. Traditionally, chemists have used the Newman projection to define this *dihedral angle.*

If the four atoms in the chain are C1-C2-C3-C4, then, looking along the C3-C2 bond (i.e. from C3 towards C2), the C3-C4 bond and the C2-C1 bond do not eclipse each other (in general). The *clockwise* angle by which the C3-C4 bond must be rotated to bring it directly over the C2-C1 bond is the required dihedral angle (in degrees). The specification of the fourth (and subsequent) atoms is then given by:

```
C4 C3 RC4C3 C2 A2 C1 D
```

where `RC4C3` is the bond distance from `C4` to `C3`, `A2` is the angle `C4C3C2` and `D` is the dihedral angle.

The simplest case is perhaps the hydrogen peroxide molecule, H_2O_2. The dihedral angle is the angle between the two O-H bonds when viewed along the O-O bond. In this simplest case the dihedral angle may be taken as α or $360 - \alpha$ since there are no other atoms involved. In general, however, the *sense* of the angle must be preserved to ensure a correct orientation of subsequent atoms (either on the chain or branched)

This scheme is sufficient to specify completely the geometry of molecules of arbitrary complexity with the following provisos:

- When a molecule is not basically a chain but is more "compact" i.e. a central branched system like CH_3F, for example, the "dihedral angles" can still be defined by the above procedure and are acceptable to `gnu80` but they are not the sort of angle which the chemist normally calls dihedral angles. This is a small price to pay for a coherent system.

- Linear Molecules have dihedral angles zero. This is not a problem for actual linear molecules but it *is* a problem for larger molecules with *linear sections.* In these cases, passing along a linear chain causes the system to "lose its memory" of the original orientation of earlier non-linear pieces.

The second of these provisos can be cured (as can many other *practical* difficulties associated with the Z-matrix method) by the introduction of the concept of the **Dummy Centre**.

A dummy centre is just that; a centre introduced in order to make the Z-matrix specification of a particular molecule either:

- Possible, because it contains a linear sub-section or
- Easier, because some of the "dihedral angles" are awkward and artificial.

The dummy centre is specified in exactly the same way as "genuine" atomic centres and is recognised by `gnu80` by its own "chemical" symbol X. It is ignored in the calculation once the co-ordinate system has been set up.

The most characteristic use for a dummy centre is to resolve the linear sub-sections problem. Simple by putting a dummy centre off the linear axis, the dihedral angle between this dummy and subsequent off-axis atoms can be used. It is often found convenient to use dummy centres to generate Z-matrices for particular purposes, for example to define an angle in a molecule which is a "natural" parameter in an optimisation, but which is not defined by particular atoms; angles between the natural directions of functional groups or aromatic planes.

The the central concepts of the Z matrix are made more useful by the addition of a few technical aids.

- When giving the specification of an atom, the earlier atoms in the Z matrix may be referenced *by number* as well as by label. This facility is provided for compatibility with the earlier GAUSSIAN series and its use is not reccommended; the insertion of an atom in a chain will upset the numbering system but not reference by label.
- A dummy atom may be labelled by a `-` (minus or dash, not underscore) as well as by **X**.
- Any of the distances and angles in the Z matrix may be immediately preceeded by a minus with the obvious meaning. Clearly if the distance or angle is given explicitly as a number (like -1.4)

this is not news; it is included in the definition of "number". But the use of the minus is possible *even when the distance or angle is given symbolically* (like `RC2C1`).

This is very often useful to preserve the *symmetry* of a molecular arrangement during optimisations, or simply as an aid to getting the geometry expressed in an easily-recognisable form. Examples 2 and 3 in the compilation of input examples (Appendix 2.A.1) illustrates this point.

- It was remarked in the description of the dihedral angle that, in some cases, the actual physical interpretation of this angle is not that of a dihedral angle even when the geometrical recipe still works. It is often more sensible to define the position of an atomic centre by means of a distance and *two* bond angles. This facility is provided in `gnu80` and is described in the reference section below.

Obviously, at some point the symbols used to define distances and angles in the Z matrix must be given numerical values to enable `gnu80` to carry through an actual calculation. Either fixed numerical values must be supplied or (in the case of optimisations) initial values for `gnu80` to use as a starting point for automatically generated values.

This is done *after* the Z matrix is complete in the two input sections immediately following the blank record which terminates the Z matrix input section. The first of these sections is called the **Variables** section and is generally used to supply *initial value* assignments to the symbols during an optimisation. Obviously, if the **OPT** command has not been given on the command record, no optimisation will be performed and the assignments in the **Variables** section simply provide fixed values for the distances and angles.

The form of the **Variables** input section is simply a set of records of symbols followed by the initial numerical values of these symbols, separated by either a space or, more pictorially, by an equals symbol (=). In the example given above, the **Variables** section might include the records:

```
RC2C1=1.34
A 109.5
```

among other records.

If it required to have all the Z matrix data in the same general form — symbolic names for the distances and angles — and yet it is required to keep some of these geometrical parameters *fixed* (i.e. not take part in the optimisation) the the **Variables** input section is followed (after its terminating blank record) by the **Constants** input section which is of precisely the same form but gnu80 does not attempt to change these numerical values, keeping them fixed at the values given throughout the calculation.

2.5.2 The Z-matrix, Reference

Each nucleus (including dummies) is numbered sequentially and specified on a single record. This collection of data, determining the molecular geometry is referred to as the Z-matrix. Thus, the nature and location of the N'th nucleus is specified on the (N+1)'th record in the section in terms of the positions of the previously defined nuclei 1,2,. . . (N-1).

The information about the position of the N'th nucleus is contained in up to eight separate items on the record:

1. ELEMENT
2. N1
3. LENGTH
4. N2
5. ANGLE
6. N3
7. TWIST
8. J

each of these items is now discussed.

1. **ELEMENT** specifies the chemical nature of the nucleus. It may consist of just the chemical symbol such as **H** or **C** for hydrogen or carbon. Alternatively, it may be an alphanumeric string beginning with the chemical symbol, followed immediately by a secondary identifying integer. Thus, **C5** can be used to specify a carbon nucleus, identified as the fifth carbon in the molecule, for example. This is sometimes convenient in following conventional chemical numbering.

   Dummy nuclei are denoted by the symbols **X** or **-**. The item **ELEMENT** is required for every nucleus. For the first nucleus specified (N=1), it is the only item on the record.

2. **N1** specifies the (previously defined) nucleus for which the internuclear distance R(N,N1) will be given. This item may be either an integer (the value of N1 ¡ N) or an alphanumeric string. In the latter case, the string must (of course!) match the **ELEMENT** field of a previous Z-matrix record.

3. **LENGTH** is the internuclear distance R(N,N1). This may be either a positive floating point number giving the length in Angstroms (unless modified by the **UNIT** command) or an alphanumeric string (maximum 8 characters). In the latter case, the length is represented by a **VARIABLE** for which a value will be specified in Input Section 4.

   Use of variables in the Z-matrix is essential if optimization is to be carried out. However, they can also be used in single-point runs. The items **N1** and **LENGTH** are required for all nuclei after the first. For the second nucleus, only **ELEMENT**, **N1**, and **LENGTH** are required.

4. **N2** specifies the nucleus for which the internuclear angle $\theta(N, N1, N2)$ will be given. Again this may be an integer (the value of N2 ¡ N) or an alphanumeric string which matches a previous **ELEMENT** entry. Note that **N1** and **N2** must represent *different* nuclei.

5. **ANGLE** is the internuclear angle $\theta(N, N1, Nn2)$. This may be a floating point number giving the angle in degrees (unless mod-

ified by the **UNIT** command) or an alphanumeric string representing a variable. **N2** and **ANGLE** are required for all nuclei after the second. For the third nucleus, only **ELEMENT**, **N1**, **LENGTH**, **N2**, and **ANGLE** are required.

6. **N3** The significance of **N3** and **TWIST** depends on the value of the last item **J**. If J=0, or is omitted, **N3** specifies the nucleus for which the internuclear *dihedral* angle, $\phi(N, N1, N2, N3)$ will be given; as with **N1** and **N2**, this may be either an integer (N3¡N) or an alphanumeric string matching a previous **ELEMENT** entry.

7. **TWIST** (if J=0) is the internuclear dihedral angle $\phi(N, N1, N2, N3)$. Again, this may be a floating point number giving the angle in degrees (unless modified by **UNIT**) or an alphanumeric string representing a variable (or a variable preceeded by a negative sign). The dihedral angle is defined as the angle $(-180.0 < \phi <= 180.0)$ between the planes (N,N1,N2) and (N1,N2,N3). The sign is positive if the movement of the directed vector $R(N1 \rightarrow N)$ towards the directed vector $R(N2 \rightarrow N3)$ involves a righthanded screw motion.

8. **J** The above descriptions of **N3** and **TWIST** apply if the item **J** is zero or absent.

   Although it is usually possible to specify the nucleus N by a bond length, a bond angle, and a dihedral angle, it is sometimes simpler to replace the dihedral angle by a second bond angle. This possibility is called for by using J = +1 or -1.

9. Values of parameters for non-zero J:

   - **N3** If **J** is +1 or -1, **N3** specifies the nucleus for which the *second* internuclear angle $\chi(N, N1, N3)$ will be given. As usual, this may be either an integer (N3 ¡ N) or an alphanumeric string representing a previously defined **ELEMENT**.
   - **TWIST** If **J** is +1 or -1, then this item gives the value for the *second* internuclear angle $\chi(N, N1, N3)$.

As before, this may be either a floating point number (value in degrees, unless modified by **UNITS**) or an alphanumeric string representing a variable.

- **J** In the event of specification by two internuclear angles θ, χ, there will be two possible positions for the nucleus N. This is fixed by the sign of **J**. Thus J=+1 if the triple vector product:

  $$R(N1 \rightarrow N) \cdot (R(N1 \rightarrow N2) \times R(N1 \rightarrow N3))$$

  is positive, and J=-1 if the product is negative. Note that J=+1 corresponds to a clockwise rotation angle when looking from N1 to N.

The Z-matrix is terminated by the blank record which indicates the end of the molecule specification section.

If no variables have been introduced and if the command **ALTER** has not been invoked in the job-type section, the input is complete and `gnu80` will perform the requested computation.

2.6 Variable and Constant Input Sections

These sections contain values of parameters introduced into the Z-matrix as alphanumeric strings in the preceding molecule specification section. If there were no parameters (all lengths and angles having been specified by floating-point numbers), these sections are not required. If parameters are used, they are divided into two sets.

The first set contains the **VARIABLES** which are the parameters which are varied and optimized in the optimization run. Input Section 4 contains the *initial values* of the variables. If optimization is not called for, these values will be inserted for a single run as if they had been specified numerically in Input Section 3.

The second set of parameters contains **CONSTANTS** which are parameters which are not varied in optimization run. The values of these should be given in Input Section 5. The **CONSTANT** section is provided only for occasional convenience; it can be avoided completely if the values of the constants are entered numerically in program Input Section 3.

Each parameter in Input Sections 4 and 5 is given a value on a separate record, the value following the parameter name in free field. Blanks, commas or equal signs may be used as separators with perhaps equals signs being the most intuitively appealing. Each section is terminated by a blank record. Thus, if the variables (R1,R2,A) are given values (1.08, 1.36, 105.4), the full **VARIABLE** section (input section 4) consists of the four records:

```
R1 = 1.08
R2 = 1.36
A = 105.4
(blank record)
```

If the **CONSTANT** section is empty (i.e. if all of the parameters have been listed in the variable section), a second blank record is not needed.

If the job is a single point run (i.e. if the **OPT** command has not been issued in the Command Record), the program will substitute the values from Input Sections 4 and 5 into the Z-matrix and proceed.

If, on the other hand, optimization **(OPT)** has been requested, the program will adjust the **VARIABLES** (Input Section 4) but not the **CONSTANTS** (Input Section 5) until a stationary point has been located.

2.7 GEN Basis Set Input

This section is for specification of a basis set if a standard type is not used. If the basis has already been given in the Command Record (or the **GEN** command omitted) this section is not needed and no blank record is required. For fuller details, see Chapter 3.

2.8 Alteration of Configuration

If the command **GUESS=ALTER** or **ALTER** has been issued in the Command Record, information is expected in the alteration-of-configuration program input sections.

There will be one such section if a singlet state is involved but two for higher multiplicities since $\alpha-$ and $\beta-$ orbitals then have to be treated separately. Normally, the occupied orbitals are selected as those with lowest eigenvalues for the one-electron Hamiltonian used in the initial guess. These sections consist of a set of substitutions, indicating that one of these occupied orbitals is to be replaced by one of the other (virtual) orbitals. Each such substitution is on a separate record and has two integers N1, N2 (free field) indicating that orbital N1 is replaced by obital N2 and that orbital N2 is replaced by N1. The list of substitutions is terminated by the blank record at the end of the section.

Note that both sections are required for non-singlet states. Thus, if only $\alpha-$ substitutions are needed, the β section is expected even though it is empty and vice versa. The second blank record to indicate this omission has to be included.

2.9 Program Limitations

This section outlines the various limitations that exist within `gnu80`. These limitations occur throughout the system in the form of fixed dimension statements, algorithm design limitations, etc., and their overall effect is to restrict the "size" of calculation that can be performed in the sense of number of basis functions used or number of atoms in the molecule.

The limitations described here can be divided into two broad categories:

- Limitations associated with the Z-matrix specification
- Basis set restrictions.

This section is concluded by "quick-reference" Table which summarises the pertinent information.

2.9.1 Z-matrix Limitations

This class of restrictions involves: the size of the Z-matrix, the number of variables in a geometry optimization and the maximum number of atoms. Recall that a Z-matrix may have "dummy" atoms and therefore the number of atoms used in the calculation may be fewer than the number of records in the Z-matrix.

The total number of records in the Z-matrix may not exceed 50. This includes both dummies and genuine atoms.

Furthermore, the maximum number of variables which can be specified in an optimization must not exceed 50 (30 for a Fletcher-Powell (**FP**) optimization). These two restrictions are imposed by explicit DIMENSIONing which appears in the Links that process the Z-matrix and its variables (links 101, 102, 103, 105, 202 and 716).

The internal restrictions on the total number of atoms are not so strict. Within the interior of the program, the arrays which hold Cartesian co-ordinantes and atomic number information are DIMENSIONed to accommodate 100 atoms. This DIMENSIONing far exceeds the maximum size of the Z-matrix and certain integral evaluation limitations (described below). These expanded arrays were installed in the program the last time it was necessary to alter the COMMON-blocks containing them, and are intended for future expansion.

2.9.2 Basis Set Limitations

Throughout the Gaussian system, basis set limitations manifest themselves in two ways. Firstly, restrictions are imposed within the integral evaluation programs; limiting the number of primitive gaussian functions and how they are combined into atomic orbital basis functions. Secondly, DIMENSIONing requirements limit the total number of basis functions that can be used in each of the energy evaluation procedures (SCF, MP2, etc.)

General input data limitations

See 3.6 for a fuller discussion of non-default bases and 2.9 for a compendium of the various limitations on `gnu80` and the way that basis set limitations fit into the general scheme of limitations. Meanwhile, here is a summary of the limitations on basis sets which is most useful for the default bases; currently `MAXSHL` is set to 100.

`MAXSHL` shells; see definition of a shell below.
Types: s, p, d, f, sp, spd
Maximum of 6 Gaussian functions per shell but maximum of `3*MAXSHL` Gaussians totally.

Definitions of the concepts of *primitive* shells and *contracted* (or *full*) shells:

- A *primitive shell* is defined to be a set of basis functions up to and including functions of some specified effective maximum angular quantum number (ℓ) which share a common orbital exponent. Thus, an $\ell = 1$ shell would consist of the functions (s,px,py,pz) all with the same Gaussian exponent. Similarly, an $\ell = 2$ shell would contain (s,px,py,pz,xx,yy,zz,xy,xz,yz) where xx, etc. denote the normalized second-order Gaussian functions.

- A *full, or contracted* shell results from contracting the functions of several primitive shells together. In typical calculations, one normally uses contracted shells.

As an example, consider the carbon atom in the 6-31G basis. In this basis, the carbon will have 10 primitive shells (6+3+1). The first 6 primitive shells are s-shells ($\ell = 0$), and are contracted together to make a single s-type basis function. The next three shells are sp-shells ($\ell = 1$), each consisting of (s,px,py,pz) functions. These primitive shells are contracted together to make four atomic orbital basis functions: s, px, py and pz. The outermost primitive shell is also of sp type, and makes yet another 4 atomic orbital basis functions.

If the program were limited to this definition of shells, one would have to use a set of sp-functions whenever a set of d-functions was desired. Since it is frequently desired to have just a set of d or f type functions, some way must be devised to handle this. thus, we introduce the idea of a ‘ shell constraint’ . The shell constraint specifies which functions within a shell are actually employed. By appropriately setting the shell constraint, one can get just the d portion of an $\ell = 2$ shell.

2.9.3 Integral Evaluation Limitations

In order to fully understand the limitations in the integral programs, one must have some understanding of the concepts presented in 3.6

(input of non-standard bases). In the terminology introduced in 3.6, the limitations are as follows:

- The maximum number of primitive s- and p-type shells is `3*MAXSHL`,
- The maximum number of primitive d-shells is `MAXSHL`,
- The maximum number of contracted shells is `MAXSHL`, and the maximum degree of contraction is 8.

Note:

- That the `MAXSHL` contracted shell limit implicitly declares an `MAXSHL` atom limit if each atom has only a single shell of basis functions.
- The maximum degree of contraction limit of 8 is actually an input limitation imposed by routine GBASIS in L301; the integral evaluation programs are DIMENSIONed to handle expansions of up to 10 gaussians.

An important limitation in the integral programs concerns the types of shells which can be used. The present version of `gnu80` can handle s, sp, p and d shells. Re-DIMENSIONing of the array TQ in L314 to 40000 will enable the program to handle spd-shells at the SCF level. Implementation of this feature in the gradient program is not straightforward. Even though much of the code is present, f shells are not implementated in this version.

The last major restriction which appears in the integral programs is in the manner in which integral labels are packed. When the *standard* integral storage procedure is selected (in contrast to the *Raffenetti* storage modes), the suffixes I, J, K and L of the two-electron integral (IJ,KL) are packed into a 32 bit computer word as limited length quantities. This in effect limits the number of basis functions to 100. When the Rafenetti modes are selected, the two linearized suffixes IJ and KL (where e.g. IJ = (I*(I-1))/2 + J are packed into a standard integer. This imposes a theoretical limit of 361 basis functions which is well outside any real possibility.

Basis function limitations in SCF and post-SCF Links
Currently `MAXBAS` is set to 150

Link	Max. Basis
L401	`MAXBAS`
L501	`MAXBAS`
L502	80
L503	70
L505	70
L601	`MAXBAS`
L701	`MAXBAS`
L702	`MAXBAS`
L703	`MAXBAS`
L716	`MAXBAS`
L801	70
L802	70
L803	70
L901	70
L909	60
L910	60
L911	60
L912	60
L913	60

2.9.4 SCF and POST-SCF Limitations

The limitations present in the energy evaluation procedures are mainly dependent on DIMENSIONing requirements. These are summarized in the Table. These programs have only rarely been used at or near their respective dimension limits, and the user should exercise some caution when large calculations are attempted.

2.9.5 Basis Set Ranges

The atoms for which the standard basis sets apply are:

Basis Function Ranges

STO-3G	**H - Xe**
3-21G	**H - Ar**
6-21G	**H - Ar**
4-31G	**H - Ne**
6-31G	**H - Ne**
6-311G	**H - Ne**
LP-31G	**H - Cl**
LANL1MB	**Na - U**
LANL1DZ	**Na - U**

Note:

- All these basis ranges **exclude** the Lanthanides for which there are no bases or potentials (yet).
- **LANL1DZ** and **LP-31G** are synonyms; use of either command uses the **LP-31G** basis for the first-row atoms (and related potentials) and the **LANL1DZ** basis and potentials for all other atoms.
- Use of **LANL1MB** results in the use of the **STO-3G** valence set (and related potentials) for atoms of the first row and the actual **LANL1MB** basis for all other atoms.

These ranges are not affected by the addition of either p or d polarization functions.

But note that the addition of polarisation functions to an STO-3G basis for atoms of the first row of the periodic table (H —Ne) is forbidden because it is an "unbalanced" basis which may give poor or meaningless results. Thus specifying STO-3G* for a calculation on the water molecule will result in a warning message and an STO-3G calculation being performed.

2.9.6 Concluding Remarks

The sections above briefly describe the most limitations built into the gnu80 system. The restrictions which are, by far, the most likely to be encountered are those associated with the size of the Z-matrix, number of geometrical variables or the total number of basis functions. In practice, the integral program limitations almost never cause problems.

It is important to point out that underlying all of the above restrictions is the question of practicality. The amount of computation in the SCF step increases very rapidly with the number of basis functions. In the correlation procedures, MP2, MP3 and CID, the computation is also strongly dependent on the number of electrons.

Appendix 2.A

Standard Methods: Input/Output Examples

2.A.1 Introduction

By far the best method of becoming familiar with the use of `gnu80` is by studying examples. In this section there are a number of “typical” runs with some comments and in the next section there are selected listings of the output generated by these runs. Most of the examples use Standard Methods.

Throughout these examples the records are numbered, this is simply for convenience of reference, the record numbers *do not* form part of the input to `gnu80`. Again, throughout the examples a blank record is indicated by ‘ ‘(blank record)” in order to be free of any quirks of line-spacing in the manual; of course “(blank record)” must be replaced in the actual input by a blank record!

Only a *few* examples of the output generated by these input examples are given later as the output quickly becomes unwieldy in a manual!

2.A.1.1 Example 1 HF calculation at fixed Geometry

```
1. # HF/STO-3G

2. (blank record)

3. WATER AT EXPERIMENTAL GEOMETRY

4. (blank record)

5. 0 1

6. O

7. H 1 0.956

8. H 1 0.956 2 104.5

9. (blank record)
```

This is a straightforward (single-point) closed-shell (RHF) run on the (singlet) ground state of the water molecule using the STO-3G basis which is also the default. No **VARIABLES** are introduced, so only Input Sections 1, 2 and 3 are needed. Record 1 is the command record and the blank record terminates the job type input section. Records 3 and 4 are the title input section. The remaining records (5, 6, 7, 8, 9) make up the Molecule Specification Input Section.

Alternative, Using the Element names for atom reference

```
1. # HF/STO-3G

2. (blank record)

3. Water at Experimental Geometry

4. (blank record)

5. 0 1

6. O
```

```
7. H O 0.956

8. H O 0.956 2 104.5

9. (blank record)
```

This dataset will generate exactly the same run as the one above, the only difference is that the hydrogen atoms' positions are referred to the oxygen atom *by its symbol in the Z-matrix* rather than by its number in the records of the Z-matrix. Similar replacements may be made in all the datasets in the examples, if this is a more attractive method of presentation.

The method of "reference by symbol" does have the advantage of "Z-matrix position invariance' ; adding or subtracting atoms to the Z-matrix does not involve changes to existing records since they are internally consistent. Notice that this method is best used together with the precaution of giving all atoms in a given run a different symbol; in the above example the two hydrogens have the same label. This will not cause trouble in this particular case, of course.

2.A.1.2 Example 2. Simple Geometry Optimisation

```
 1. # HF/STO-3G OPT

 2. (blank record)

 3. WATER STO-3G STRUCTURE

 4. (blank record)

 5. 0 1

 6. O

 7. H 1 P

 8. H 1 H 2 A

 9. (blank record)

10. R=1.0

11. A=105.0

12. (blank record)
```

This is an optimization where the O-H bond length R and the H-O-H bond angle A are adjusted to minimize the HF/STO-3G energy. Note that the **VARIABLE** R appears in *two* places in the Z-matrix (both O-H bonds). This means that the optimization is *implicitly constrained* and that the two bondlengths are not allowed to take on separate (different) values. This use of **VARIABLES** permits symmetry constraints to be imposed *and maintained* (C_{2v} in this case). This example has a **VARIABLES** input section (records 10, 11 and 12).

2.A.1.3 Example 3 A General Optimisation

```
# HF/STO-3G OPT

WATER STO-3G STRUCTURE UNCONSTRAINED

0 1
O
H 1 R1
H 1 R2 2 A

R1=0.9
R2=1.1
A=105.0

```

This optimization is not constrained to retain C_{2v} symmetry during the optimisation because the two bond lengths are represented by different **VARIABLES** which are given different initial values.

2.A.1.4 Example 4 Use of CONSTANTS

1. `# HF/STO-3G OPT`
2. `(blank record)`
3. `WATER STO-3G STRUCTURE`
4. `(blank record)`
5. `0 1`
6. `C`
7. `H1 O R`
8. `H2 O R H1 A`
9. `(blank record)`
10. `R=1.0`
11. `(blank record)`
12. `A=105.0`
13. `(blank record)`

This is similar to Example 2, except that (1) the Hydrogen nuclei are numbered directly as H1 and H2 and (2) only the bond length is varied, the angle A being treated as a constant. The final section is an example of a **CONSTANTS** Input Section.

2.A.1.5 Example 5 A more General Z-matrix

```
 1. # HF/3-21G OPT
 2. (blank record)
 3. FLUOROMETHANE C3V 3-21G OPTIMIZATION
 4. (blank record)
 5. 0 1
 6. C
 7. F 1 CF
 8. H 1 CH 2 HCF
 9. H 1 CH 2 HCF 3 120.
10. H 1 CH 2 HCF 3 -120.
11. (blank record)
12. CF=1.38
13. CH=1.09
14. HCF=110.6
15. (blank record)
```

This HF/3-21G optimization of Fluoromethane in C_{3v} symmetry illustrates the use of the *dihedral angle* for the second and third hydrogen atoms. Note that mnemonic choice of names of **VARIABLES**.

2.A.1.6 Example 6 Use of a Dummy Centre

```
 1. # HF/3-21G OPT

 2. (blank record)

 3. AMMONIA C3V OPTIMIZATION

 4. (blank record)

 5. 0 1

 6. N

 7. - 1 1.

 8. H 1 NH 2 HNX

 9. H 1 NH 2 HNX 3 120.

10. H 1 NH 2 HNX 3 -120.

11. (blank record)

12. NH=1.0

13. HNX=70.

14. (blank record)
```

This example illustrates the use of a Dummy centre (number 2) to fix the three-fold axis in C_{3v} ammonia. Note that the “radial” position of the dummy on the axis is irrelevant and that the distance 1.0 used could have been replaced by any other positive number. HNX is the angle between an NH bond and the threefold axis.

2.A.1.7 Example 7 Direct use of HNH angle

```
# HF/3-21G OPT

AMMONIA C3V 3-21G OPTIMIZATION

0 1
N
H 1 NH
H 1 NH 2 HNH
H 1 NH 2 HNH 3 HNH 1

N H=1.0
HNH=106.0

```

This is similar to Example 6 except that the HNH bond angle is used as a **VARIABLE**. J=+1 for the third hydrogen permits it to be defined by two bond angles.

2.A.1.8 Example 8 Use of a Dummy atom to maintain "symmetry"

```
 1. # HF/3-21G OPT
 2. (blank record)
 3. OXIRANE C2V 3-21G OPTIMIZATION
 4. (blank record)
 5. 0 1
 6. X
 7. C1 X HALFCC
 8. C X UX C1 90.
 9. C2 X HALFCC C 90.  C1 180.
10. H1 C1 CH X HCC 0 HCCO
11. H2 C1 CH X HCC 0 -HCCU
12. H3 C2 CH X HCC 0 HCCO
13. H4 C2 CH X HCC 0 -HCCO
14. (blank record)
15. HALFCC=0.75
16. OX=1.0
17. CH=1.08
18. HCC=130.
19. HCCO=130.
20. (blank record)
```

This Example illustrates two points. First, a dummy centre is placed at the center of the C-C bond to help *constrain* the CCO triangle to be isoceles. OX is then the perpendicular distance from 0 to the C-C bond and the angles OXC are held at 90.0 . Second, note that some of the entries in the Z-matrix are represented by the *negative* of the dihedral angle variable HCCO. This is useful in symmetry-constrained situations like this.

Examples 9 and 10 Bond Angle Limitations

If **FORCE** or **OPT** runs are carried out, gnu80 is unable to handle bond angles of 180 which occur in linear molecular fragments. Examples are linear acetylene and the C4 chain in butatriene. Difficulties may also be encountered in nearly linear situations such as ethynyl groups in unsymmetrical molecules. These sutuations can be avoided by introducing dummy centres along the angle bisector and using the half-angle as the **VARIABLE** or **CONSTANT**. This is illustrated in example 9 and 10.

2.A.1.9 Example 9 Linear HCN

```
 1. # HF/6-31G* OPT

 2. (blank record)

 3. HYDROGEN CYANIDE LINEAR 6-31G* OPTIMIZATION

 4. (blank record)

 5. 0 1

 6. N

 7. C 1 CN

 8. - 2 1.  1 90.

 9. H 2 CH 3 90.  1 180.

10. (blank record)

11. CN=1.20

12. CH=1.06

13. (blank record)
```

2.A.1.10 Example 10 Use of a Half-Angle

```
 1. # HF/3-21G OPT

 2. (blank record)

 3. NCOH 3-21G OPTIMIZATION

 4. (blank record)

 5. 0 1

 6. N

 7. C 1 CN

 8. - 2 1.  1 HALF

 9. O 2 CO 3 HALF 1 180.

10. H 4 OH 2 COH 3 0.

11. (blank record)

12. CN=1.2

13. CO=1.3

14. OH=1.0

15. HALF=80.

16. COH=105.

17. (blank record)
```

In this optimization, HALF represents half of the NCO angle which is expected to be close to linear. Note that the position of the dummy centre is chosen so that a value of HALF less than 90 corresponds to a (slightly) Cis arrangement.

2.A.1.11 Example 11 A preparation run for an Open-Shell species

```
 1. # GUESS=ONLY
 2. (blank record)
 3. AMINO RADICAL TEST OF INITIAL GUESS
 4. (blank record)
 5. 0 2
 6. N
 7. H 1 NH
 8. H 1 NH 2 HNH
 9. (blank record)
10. NH=1.03
11. HNH=120.0
12. (blank record)
```

This is a short test of the NH_2 radical to find whether any **ALTER** commands are needed to generate a desired electronic state. Note that **HF/STO-3G** is used by default since the command record is not null.

2.A.1.12 Example 12 Use of Example 11 in a Production Run

```
 1. # HF/STO-3G ALTER OPT
 2. (blank record)
 3. AMINO RADICAL STO-3G STRUCTURE OF 2-A1 STATE
 4. (blank record)
 5. 0 2
 6. N
 7. H 1 NH
 8. H 1 NH 2 HNH
 9. (blank record)
10. NH=1.03
11. HNH=120.
12. (blank record)
13. (blank record)
14. 4 5
15. (blank record)
```

This example finds the **UHF/STO-3G** structure of the $^2A_1$ excited state of the amino radical. Examination of the output from the previous example indicates that a β-electron has to move from orbital 4 to orbital 5 to obtain the correct initial configuration. Note than an extra blank record is necessary to indicate an empty α **ALTER** section.

2.A.1.13 Example 13 Moller-Plesset Run

1. `# MP3/6-31G**`
2. `(blank record)`
3. `WATER MP3/6-31G** AT HF/6-31G* GEOMETRY`
4. `(blank record)`
5. `0 1`
6. `O`
7. `H 1 P`
8. `H 1 F 2 A`
9. `(blank record)`
10. `P=0.947`
11. `A=105.5`
12. `(blank record)`

2.A.1.14 Example 14 Configuration Interaction Run

```
# CID/6-31G** NOPOP

WATER CID/6-31G** AT HF/6-31G* GEOMETRY

0 1
O
H 1 R
H 1 R 2 A

R=0.947
A=105.5

```

This example does a single point run at the CID/6-31G*//HF/6-31G* level of theory.

2.A.2 Output Examples

It is, for obvious reasons, undesirable to give complete output listing for all the input examples listed in Section 2.A.1. However, it *is* useful to have, for reference purposes, a record of what the results of each example are. So, for each example there is, in this Section, *one page* of severely edited output which gives the essence of the calculation. The exception to this type of display is Example 2 which is, perhaps the most common type of `gnu80` run; geometry optimisation at the HF/STO-3G level. In this case the full output is listed together with some commentary.

2.A.2.1 Example 1 and 3—14

Example 1

```
SCF DONE:  E(RHF) =  -74.9627570354     A.U. AFTER    6 CYCLES
            CONVG  =    0.4234d-04            -V/T =   2.004974
 COMPONENT                          A.U.                   KCAL/MOL
 ---------------------------------------------------------------
 TOTAL                        -74.962757035408          -47040.629
 ELECTRONIC                   -84.169308351904          -52817.924
 NUCLEAR REPULSION              9.206551316496            5777.295
 KINETIC                       74.591723452750           46807.798
 POTENTIAL                   -149.554480488158          -93848.428
 ELECTRONIC POTENTIAL        -158.761031804654          -99625.723
 ONE-ELECTRON POTENTIAL      -196.983005149312         -123610.775
 TWO-ELECTRON POTENTIAL        38.221973344659           23985.053
 ORBITAL SYMMETRIES.
       OCCUPIED: (A1) (A1) (B2) (A1) (B1)
       VIRTUAL:  (A1) (B2)
  THE ELECTRONIC STATE IS 1-A1.
      MOLECULAR ORBITAL COEFFICIENTS
                           1         2         3         4         5
                         (A1)      (A1)      (B2)      (A1)      (B1)
     EIGENVALUES --   -20.24160  -1.26904  -0.61852  -0.45319  -0.39132
   1 1   O   1S        -0.99413   0.23274   0.00000  -0.10325   0.00000
   2         2S        -0.02659  -0.83304   0.00000   0.53728   0.00000
   3         2PX        0.00000   0.00000   0.00000   0.00000   1.00000
   4         2PY        0.00000   0.00000   0.60650   0.00000   0.00000
   5         2PZ       -0.00436  -0.13005   0.00000  -0.77698   0.00000
   6 2   H   1S         0.00599  -0.15880   0.44495  -0.27733   0.00000
   7 3   H   1S         0.00599  -0.15880  -0.44495  -0.27733   0.00000
          TOTAL ATOMIC CHARGES.
              1
  1   O   8.367459
  2   H   0.816270
  3   H   0.816270
 DIPOLE MOMENT (DEBYE): X= 0.0000   Y= 0.0000   Z= 1.7276   TOTAL= 1.7276
```

Example 3

```
                      ---------------------------
                      !  OPTIMIZED PARAMETERS   !
                      ! (ANGSTROMS AND DEGREES) !
 ----------------------                           ----------------------
 !      NAME         VALUE   DERIVATIVE INFORMATION (ATOMIC UNITS)     !
 -----------------------------------------------------------------------
 !       R1          0.9893   -DE/DX =    0.000052                     !
 !       R2          0.9894   -DE/DX =   -0.000009                     !
 !        A        100.0135   -DE/DX =    0.00007                      !
 -----------------------------------------------------------------------
 GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
 STANDARD BASIS: STO-3G     (S, S=P, 5D, 7F)
   7 BASIS FUNCTIONS       21 PRIMITIVE GAUSSIANS
   5 ALPHA ELECTRONS        5 BETA ELECTRONS
     NUCLEAR REPULSION ENERGY    8.9065689700 HARTREES
 RAFFENETTI 1 INTEGRAL FORMAT.
 TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED OFF.
 ORBITAL SYMMETRIES.
       OCCUPIED: (A') (A') (A') (A') (A")
       VIRTUAL:  (A') (A')
  THE ELECTRONIC STATE IS 1-A'.
      MOLECULAR ORBITAL COEFFICIENTS
                               1         2         3         4         5
                             (A')      (A')      (A')      (A')      (A")
     EIGENVALUES --    -20.25159  -1.25757  -0.59383  -0.45976  -0.39263
   1 1   O  1S          -0.99422  -0.23376   0.00001  -0.10404   0.00000
   2        2S          -0.02585   0.84444  -0.00005   0.53821   0.00000
   3        2PX         -0.00319   0.09411  -0.39377  -0.57908   0.00000
   4        2PY          0.00000   0.00000   0.00000   0.00000   1.00000
   5        2PZ         -0.00268   0.07897   0.46944  -0.48576   0.00000
   6 2   H  1S           0.00558   0.15560   0.44923  -0.29509   0.00000
   7 3   H  1S           0.00558   0.15558  -0.44922  -0.29513   0.00000
```

Example 4

```
                       ---------------------------
                       !  OPTIMIZED PARAMETERS   !
                       ! (ANGSTROMS AND DEGREES) !
 ----------------------                           ----------------------
 !      NAME          VALUE   DERIVATIVE INFORMATION (ATOMIC UNITS)     !
 ------------------------------------------------------------------------
 !        R          0.9867   -DE/DX =    0.000072                      !
 ------------------------------------------------------------------------
 GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
 STANDARD BASIS: STO-3G     (S, S=P, 5D, 7F)
   7 BASIS FUNCTIONS       21 PRIMITIVE GAUSSIANS
   5 ALPHA ELECTRONS        5 BETA ELECTRONS
     NUCLEAR REPULSION ENERGY    8.9188268795 HARTREES
 RAFFENETTI 1 INTEGRAL FORMAT.
 TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
 ORBITAL SYMMETRIES.
       OCCUPIED: (A1) (A1) (B2) (A1) (B1)
       VIRTUAL:  (A1) (B2)
  THE ELECTRONIC STATE IS 1-A1.
      MOLECULAR ORBITAL COEFFICIENTS
                             1         2         3         4         5
                           (A1)      (A1)      (B2)      (A1)      (B1)
     EIGENVALUES --    -20.24477  -1.25333  -0.60334  -0.44828  -0.38917
   1 1   O  1S           0.99420   0.23402   0.00000   0.10125   0.00000
   2        2S           0.02597  -0.84450   0.00000  -0.52539   0.00000
   3        2PX          0.00000   0.00000   0.00000   0.00000   1.00000
   4        2PY          0.00000   0.00000   0.60484   0.00000   0.00000
   5        2PZ          0.00405  -0.11908   0.00000   0.77187   0.00000
   6 2   H  1S          -0.00565  -0.15655   0.44665   0.28889   0.00000
   7 3   H  1S          -0.00565  -0.15655  -0.44665   0.28889   0.00000
```

Example 5

```
                       ---------------------------
                       !  OPTIMIZED PARAMETERS   !
                       ! (ANGSTROMS AND DEGREES) !
 ----------------------                           -----------------------
 !      NAME          VALUE   DERIVATIVE INFORMATION (ATOMIC UNITS)     !
 ------------------------------------------------------------------------
 !       CF          1.404    -DE/DX =    0.000001                      !
 !       CH           1.0794  -DE/DX =    0.000021                      !
 !       HCF       109.3882   -DE/DX =    0.000004                      !
 ------------------------------------------------------------------------
 GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
 STANDARD BASIS: 3-21G     (S, S=P, 5D, 7F)
  24 BASIS FUNCTIONS       39 PRIMITIVE GAUSSIANS
   9 ALPHA ELECTRONS        9 BETA ELECTRONS
     NUCLEAR REPULSION ENERGY   37.0948856107 HARTREES
 RAFFENETTI 1 INTEGRAL FORMAT.
 TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
 ORBITAL SYMMETRIES.
       OCCUPIED: (A1) (A1) (A1) (A1) (E) (E) (A1) (E) (E)
       VIRTUAL:  (A1) (E) (E) (A1) (E) (E) (A1) (A1) (E) (E) (A1)
                 (E) (E) (A1) (A1)
 UNABLE TO DETERMINE ELECTRONIC STATE.
                           1          2          3          4          5
                         (A1)       (A1)       (A1)       (A1)        (E)
     EIGENVALUES --   -26.11975 -11.24559  -1.54808   -0.95383  -0.67357
                           6          7          8          9         10
                          (E)       (A1)        (E)        (E)      (A1)
     EIGENVALUES --    -0.67357  -0.63411  -0.51726   -0.51726   0.27861
```

Example 6

```
                       ---------------------------
                       !  OPTIMIZED PARAMETERS   !
                       ! (ANGSTROMS AND DEGREES) !
 ----------------------                           ----------------------
 !      NAME          VALUE   DERIVATIVE INFORMATION (ATOMIC UNITS)     !
 -----------------------------------------------------------------------
 !       NH          1.0026   -DE/DX =   -0.000036                      !
 !       HNX        73.6317   -DE/DX =    0.000042                      !
 -----------------------------------------------------------------------
 GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
 STANDARD BASIS: 3-21G     (S, S=P, 5D, 7F)
  15 BASIS FUNCTIONS       24 PRIMITIVE GAUSSIANS
   5 ALPHA ELECTRONS        5 BETA ELECTRONS
     NUCLEAR REPULSION ENERGY   12.0363428366 HARTREES
 RAFFENETTI 1 INTEGRAL FORMAT.
 TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
 ORBITAL SYMMETRIES.
       OCCUPIED: (A1) (A1) (E) (E) (A1)
       VIRTUAL:  (A1) (E) (E) (E) (E) (A1) (A1) (E) (E) (A1)
 UNABLE TO DETERMINE ELECTRONIC STATE.
                          1         2         3         4         5
                        (A1)      (A1)       (E)       (E)      (A1)
     EIGENVALUES --   -15.43152  -1.12003  -0.62031  -0.62031  -0.38894
                          6         7         8         9        10
                        (A1)       (E)       (E)       (E)       (E)
     EIGENVALUES --     0.27525   0.37737   0.37737   1.18654   1.18654
  15        1S  (O)     0.24962
        CONDENSED TO ATOMS (ALL ELECTRONS)
             1          2          3          4
  1   N   6.867228   0.336337   0.336337   0.336337
  2   H   0.336337   0.436131  -0.032274  -0.032274
  3   H   0.336337  -0.032274   0.436131  -0.032274
  4   H   0.336337  -0.032274  -0.032274   0.436131
        TOTAL ATOMIC CHARGES.
             1
  1   N   7.876238
  2   H   0.707921
  3   H   0.707921
  4   H   0.707921
 DIPOLE MOMENT (DEBYE): X= 0.0000   Y= 0.0000   Z= 1.7521   TOTAL= 1.7521
```

Example 7

```
                       ---------------------------
                       !  OPTIMIZED PARAMETERS   !
                       ! (ANGSTROMS AND DEGREES) !
----------------------                           ----------------------
!      NAME          VALUE   DERIVATIVE INFORMATION (ATOMIC UNITS)     !
------------------------------------------------------------------------
!       NH          1.0026   -DE/DX =   -0.00005                       !
!       HNH       112.3966   -DE/DX =   -0.000004                      !
------------------------------------------------------------------------
GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
STANDARD BASIS: 3-21G     (S, S=P, 5D, 7F)
 15 BASIS FUNCTIONS       24 PRIMITIVE GAUSSIANS
  5 ALPHA ELECTRONS        5 BETA ELECTRONS
    NUCLEAR REPULSION ENERGY   12.0364013291 HARTREES
RAFFENETTI 1 INTEGRAL FORMAT.
TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
ORBITAL SYMMETRIES.
      OCCUPIED: (A1) (A1) (E) (E) (A1)
      VIRTUAL:  (A1) (E) (E) (E) (E) (A1) (A1) (E) (E) (A1)
UNABLE TO DETERMINE ELECTRONIC STATE.
                          1          2          3          4          5
                        (A1)       (A1)        (E)        (E)       (A1)
    EIGENVALUES --   -15.43150  -1.12001  -0.62032  -0.62032  -0.38892
                          6          7          8          9         10
                        (A1)        (E)        (E)        (E)        (E)
    EIGENVALUES --     0.27526   0.37738   0.37738   1.18660   1.18660
         CONDENSED TO ATOMS (ALL ELECTRONS)
             1          2          3          4
 1   N   6.867255   0.336349   0.336349   0.336349
 2   H   0.336349   0.436084  -0.032267  -0.032267
 3   H   0.336349  -0.032267   0.436084  -0.032267
 4   H   0.336349  -0.032267  -0.032267   0.436084
         TOTAL ATOMIC CHARGES.
             1
 1   N   7.876301
 2   H   0.707900
 3   H   0.707900
```

```
  4   H   0.707900
 DIPOLE MOMENT (DEBYE): X= 0.0000   Y= 0.0000   Z= 1.7513   TOTAL= 1.7513
```

Example 8

```
                         --------------------------
                         !  OPTIMIZED PARAMETERS   !
                         ! (ANGSTROMS AND DEGREES) !
 ----------------------                           ----------------------
 !      NAME          VALUE   DERIVATIVE INFORMATION (ATOMIC UNITS)     !
 ------------------------------------------------------------------------
 !     HALFCC        0.7371   -DE/DX =   -0.00006                       !
 !       OX          1.2713   -DE/DX =   -0.000051                      !
 !       CH          1.0707   -DE/DX =    0.000014                      !
 !       HCC       119.2413   -DE/DX =    0.000031                      !
 !      HCCO       103.2904   -DE/DX =    0.000055                      !
 ------------------------------------------------------------------------
 GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
 STANDARD BASIS: 3-21G     (S, S=P, 5D, 7F)
  35 BASIS FUNCTIONS       57 PRIMITIVE GAUSSIANS
  12 ALPHA ELECTRONS       12 BETA ELECTRONS
     NUCLEAR REPULSION ENERGY   74.3281278409 HARTREES
 RAFFENETTI 1 INTEGRAL FORMAT.
 TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
 ORBITAL SYMMETRIES.
       OCCUPIED: (A1) (B2) (A1) (A1) (A1) (B2) (B1) (A1) (A2) (B2)
                 (A1) (B1)
       VIRTUAL:  (A1) (B2) (B1) (A1) (B2) (A2) (B2) (A1) (B1) (B2)
  THE ELECTRONIC STATE IS 1-A1.
                           1          2          3          4          5
                         (A1)       (B2)       (A1)       (A1)       (A1)
     EIGENVALUES --   -20.46808 -11.23219 -11.23184  -1.39571  -0.94125
                           6          7          8          9         10
                         (B2)       (B1)       (A1)       (A2)       (B2)
     EIGENVALUES --    -0.86460  -0.70946  -0.64316  -0.55335  -0.52035
                          11         12         13         14         15
                         (A1)       (B1)       (A1)       (B2)       (B1)
     EIGENVALUES --    -0.44599  -0.44550   0.27678   0.27965   0.31400
          TOTAL ATOMIC CHARGES.
              1
  1   C   6.191493
  2   O   8.551892
  3   C   6.191493
  4   H   0.766280
  5   H   0.766280
  6   H   0.766280
  7   H   0.766280
 DIPOLE MOMENT (DEBYE): X= 0.0000   Y= 0.0000   Z=-2.7845   TOTAL= 2.7845
```

Example 9

```
                      ---------------------------
                      !  OPTIMIZED PARAMETERS   !
                      ! (ANGSTROMS AND DEGREES) !
 ----------------------                           -----------------------
 !      NAME          VALUE   DERIVATIVE INFORMATION (ATOMIC UNITS)     !
 ------------------------------------------------------------------------
 !       CN          1.1325   -DE/DX =   -0.000135                      !
 !       CH          1.0588   -DE/DX =    0.000142                      !
 ------------------------------------------------------------------------
 GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
 STANDARD BASIS: 6-31G*      (S, S=P, 6D, 7F)
  32 BASIS FUNCTIONS       60 PRIMITIVE GAUSSIANS
   7 ALPHA ELECTRONS        7 BETA ELECTRONS
     NUCLEAR REPULSION ENERGY   24.3124728053 HARTREES
 RAFFENETTI 1 INTEGRAL FORMAT.
 TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED OFF.
 ORBITAL SYMMETRIES.
       OCCUPIED: (SG) (SG) (SG) (SG) (SG) (PI) (PI)
       VIRTUAL:  (PI) (PI) (SG) (SG) (PI) (PI) (SG) (SG) (PI) (PI)
                 (SG) (SG) (SG) (DLTA) (DLTA) (PI) (PI) (DLTA)
                 (DLTA) (PI) (PI) (SG) (SG) (SG) (SG)
 UNABLE TO DETERMINE ELECTRONIC STATE.
                        1         2         3         4         5
                      (SG)      (SG)      (SG)      (SG)      (SG)
     EIGENVALUES --   -15.59707 -11.28843  -1.24284  -0.80838  -0.57381
                        6         7         8         9        10
                      (PI)      (PI)      (PI)      (PI)      (SG)
     EIGENVALUES --    -0.49705  -0.49705   0.20433   0.20433   0.23318
        CONDENSED TO ATOMS (ALL ELECTRONS)
            1          2          3
  1   N   6.475039   0.928252  -0.024403
  2   C   0.928252   4.654130   0.351305
  3   H  -0.024403   0.351305   0.360523
        TOTAL ATOMIC CHARGES.
            1
  1   N   7.378888
  2   C   5.933687
  3   H   0.687425
DIPOLE MOMENT (DEBYE): X= 0.0000   Y= 0.0000   Z= 3.2086   TOTAL= 3.2086
```

Example 10

```
                      ---------------------------
                      !  OPTIMIZED PARAMETERS   !
                      ! (ANGSTROMS AND DEGREES) !
----------------------                           ----------------------
!      NAME          VALUE   DERIVATIVE INFORMATION (ATOMIC UNITS)     !
------------------------------------------------------------------------
!       CN          1.1398   -DE/DX =   -0.000449                      !
!       CO          1.3084   -DE/DX =   -0.000121                      !
!       OH          0.9703   -DE/DX =    0.000022                      !
!      HALF        90.6316   -DE/DX =    0.000051                      !
!       COH       114.1532   -DE/DX =   -0.00011                       !
------------------------------------------------------------------------
GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
STANDARD BASIS: 3-21G     (S, S=P, 5D, 7F)
 29 BASIS FUNCTIONS       48 PRIMITIVE GAUSSIANS
 11 ALPHA ELECTRONS       11 BETA ELECTRONS
    NUCLEAR REPULSION ENERGY   58.2705799512 HARTREES
RAFFENETTI 1 INTEGRAL FORMAT.
TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED OFF.
ORBITAL SYMMETRIES.
      OCCUPIED: (A') (A') (A') (A') (A') (A') (A') (A") (A') (A')
      VIRTUAL:  (A') (A') (A") (A') (A') (A") (A') (A') (A') (A')
 THE ELECTRONIC STATE IS 1-A'.
                         1         2         3         4         5
                       (A')      (A')      (A')      (A')      (A')
    EIGENVALUES --   -20.57074 -15.50999 -11.29438  -1.48307  -1.26122
                         6         7         8         9        10
                       (A')      (A')      (A")      (A')      (A')
    EIGENVALUES --    -0.85979  -0.68335  -0.65765  -0.55829  -0.47341
                        11        12        13        14        15
                       (A")      (A')      (A')      (A")      (A')
    EIGENVALUES --    -0.45424   0.18647   0.24987   0.26428   0.36665
         TOTAL ATOMIC CHARGES.
              1
 1   N   7.428592
 2   C   5.317444
 3   O   8.703003
```

```
  4   H   0.550962
 DIPOLE MOMENT (DEBYE): X= 2.1123   Y= 0.0000   Z= 3.2717   TOTAL= 3.8943
```

Example 11

```
NUMBER OF SYMMETRY OPERATIONS:  2
STANDARD BASIS: STO-3G     (S, S=P, 5D, 7F)
  7 BASIS FUNCTIONS       21 PRIMITIVE GAUSSIANS
  5 ALPHA ELECTRONS        4 BETA ELECTRONS
    NUCLEAR REPULSION ENERGY    7.4893197651 HARTREES
RAFFENETTI 2 INTEGRAL FORMAT.
TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
  INITIAL GUESS WAVE FUNCTION.
REAL ALPHA MO COEFFICIENTS.
                         1          2          3          4          5
  1 1   N  1S        -1.00079  -0.19876   0.00000  -0.02648   0.00000
  2        2S         0.00128   0.76494   0.00000   0.14715   0.00000
  3        2PX        0.00000   0.00000   0.00000   0.00000   1.00000
  4        2PY        0.00000   0.00000  -0.52797   0.00000   0.00000
  5        2PZ       -0.00195  -0.04939   0.00000  -0.96265   0.00000
  6 2   H  1S         0.00462   0.23018  -0.46197  -0.07274   0.00000
  7 3   H  1S         0.00462   0.23018   0.46197  -0.07274   0.00000
REAL BETA MO COEFFICIENTS.
                         1          2          3          4          5
  1 1   N  1S        -1.00079  -0.19876   0.00000  -0.02648   0.00000
  2        2S         0.00128   0.76494   0.00000   0.14715   0.00000
  3        2PX        0.00000   0.00000   0.00000   0.00000   1.00000
  4        2PY        0.00000   0.00000  -0.52797   0.00000   0.00000
  5        2PZ       -0.00195  -0.04939   0.00000  -0.96265   0.00000
  6 2   H  1S         0.00462   0.23018  -0.46197  -0.07274   0.00000
  7 3   H  1S         0.00462   0.23018   0.46197  -0.07274   0.00000
INITIAL GUESS ORBITAL SYMMETRIES.
  ALPHA ORBITALS
      OCCUPIED: (A1) (A1) (B2) (A1) (B1)
      VIRTUAL:  (B2) (A1)
  BETA ORBITALS
      OCCUPIED: (A1) (A1) (B2) (A1)
      VIRTUAL:  (B1) (B2) (A1)
```

Example 12

```
                       ---------------------------
                       !  OPTIMIZED PARAMETERS   !
                       ! (ANGSTROMS AND DEGREES) !
 ----------------------                           ----------------------
 !      NAME          VALUE   DERIVATIVE INFORMATION (ATOMIC UNITS)     !
 -----------------------------------------------------------------------
 !       NH           1.015    -DE/DX =   -0.00006                      !
 !       HNH        131.292    -DE/DX =    0.000012                     !
 -----------------------------------------------------------------------
 GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
 STANDARD BASIS: STO-3G     (S, S=P, 5D, 7F)
   7 BASIS FUNCTIONS       21 PRIMITIVE GAUSSIANS
   5 ALPHA ELECTRONS        4 BETA ELECTRONS
     NUCLEAR REPULSION ENERGY    7.5846617923 HARTREES
 RAFFENETTI 2 INTEGRAL FORMAT.
 TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
 UHF OPEN SHELL SCF.
 ORBITAL SYMMETRIES.
   ALPHA ORBITALS
       OCCUPIED: (A1) (A1) (B2) (A1) (B1)
       VIRTUAL:  (A1) (B2)
   BETA ORBITALS
       OCCUPIED: (A1) (A1) (B2) (B1)
       VIRTUAL:  (A1) (A1) (B2)
  THE ELECTRONIC STATE IS 2-A1.
                           1         2         3         4         5
                         (A1)      (A1)      (B2)      (A1)      (B1)
     EIGENVALUES --   -15.26364  -1.05077  -0.61200  -0.45160  -0.33075
                           1         2         3         4         5
                         (A1)      (A1)      (B2)      (B1)      (A1)
     EIGENVALUES --   -15.23574  -0.93643  -0.57761  -0.27520   0.28110
          ATOMIC SPIN DENSITIES.
              1           2           3
  1   N   1.030351  -0.020328  -0.020328
  2   H  -0.020328   0.017559   0.007920
  3   H  -0.020328   0.007920   0.017559
 DIPOLE MOMENT (DEBYE): X= 0.0000   Y= 0.0000   Z= 0.7624   TOTAL= 0.7624
```

```
FERMI CONTACT ANALYSIS (ATOMIC UNITS).
            1
1   N   0.515487
2   H   0.009898
3   H   0.009898
```

Example 13

```
SCF DONE:  E(RHF) =  -76.0235780622     A.U. AFTER   14 CYCLES
            CONVG  =    0.1016d-07             -V/T =   2.002297
 ORBITAL SYMMETRIES.
       OCCUPIED: (A1) (A1) (B2) (A1) (B1)
       VIRTUAL:  (A1) (B2) (B2) (A1) (A1) (B1) (B2) (A1) (A2) (A1)
                 (B1) (B2) (A1) (B2) (B1) (A2) (A1) (A1) (B2) (A1)
  THE ELECTRONIC STATE IS 1-A1.
      MOLECULAR ORBITAL COEFFICIENTS
                            1         2         3         4         5
                          (A1)      (A1)      (B2)      (A1)      (B1)
     EIGENVALUES --   -20.55824  -1.34479  -0.71084  -0.56854  -0.49750
 DIPOLE MOMENT (DEBYE): X= 0.0000   Y= 0.0000   Z= 2.1591   TOTAL= 2.1591
 RANGE OF M.O.'S USED FOR CORRELATION:   2  25
 RHF INTEGRAL TRANSFORMATION:
                                 5395 M.O.-INTEGRALS CREATED
 NORM(A1)=  0.10246d+01
 E2=       -0.19576446d+00        EUMP2=      -0.76219342518d+02
 MOLLER-PLESSET THIRD ORDER PERTURBATION THEORY
 ***********************************************
 E3=       -0.64696469d-02        EUMP3=      -0.76225812165d+02
 VARIATIONAL ENERGIES WITH THE FIRST-ORDER WAVEFUNCTION:
 E(VAR1)=  -0.76216222061d+02     E(CID,4)=   -0.76219291969d+02
```

Example 14

```
SCF DONE:  E(RHF) =  -76.0235780622     A.U. AFTER   14 CYCLES
            CONVG  =    0.1016d-07             -V/T =   2.002297
 RANGE OF M.O.'S USED FOR CORRELATION:   2  25
 RHF INTEGRAL TRANSFORMATION:
                                 5395 M.O.-INTEGRALS CREATED
 NORM(A1)=  0.10246d+01
 E2=       -0.19576446d+00        EUMP2=      -0.76219342518d+02
 CONFIGURATION INTERACTION WITH DOUBLE SUBSTITUTIONS
 ***************************************************
 ITERATIONS= 30   CONVERGENCE= 0.200d-05
 ITERATION STEP#  1
 ******************
 E3=       -0.64696469d-02        EUMP3=      -0.76225812165d+02
 DE(CI)=   -0.19264400d+00        E(CI)=      -0.76216222061d+02
 NORM(A)=   0.10237356d+01
 SIZE-CONSISTENCY CORRECTION:
 S.C.C.=   -0.71077439d-02        E(CI,SIZE)= -0.76223329804d+02
 ITERATION STEP#  2
 ******************
 DE(CI)=   -0.19599910d+00        E(CI)=      -0.76219577165d+02
 NORM(A)=   0.10246551d+01
 SIZE-CONSISTENCY CORRECTION:
 S.C.C.=   -0.75220564d-02        E(CI,SIZE)= -0.76227099221d+02
 ITERATION STEP#  3
 ******************
  ....
  ....
 ITERATION STEP#  6
 ******************
 EXTRAPOLATION PERFORMED
 DE(CI)=   -0.19629698d+00        E(CI)=      -0.76219875043d+02
 NORM(A)=   0.10250212d+01
 SIZE-CONSISTENCY CORRECTION:
 S.C.C.=   -0.76495816d-02        E(CI,SIZE)= -0.76227524625d+02
 ITERATION STEP#  7
 ******************
 DE(CI)=   -0.19629722d+00        E(CI)=      -0.76219875282d+02
```

```
NORM(A)=   0.10250232d+01
SIZE-CONSISTENCY CORRECTION:
S.C.C.=   -0.76502237d-02        E(CI,SIZE)= -0.76227525506d+02
*************************************************************
```

2.A.2.2 Example 2

Here is an annotated copy of the output generated by Example 2 of Section 2.A.1; perhaps the most common type of `gnu80` run. The geometry of the water molecule is optimised using the (closed-shell) HF model and a simple STO-3G orbital basis; the geometry is constrained to remain at C_{2v} throughout the optimisation by the form of the Molecule input section.

This is typical output from a job which requests **OPT**imization of a Hartree-Fock structure. The following points are of general interest:

1. The command record is reproduced.

2. The detailed route through the links in the program is given. This is a standard route generated from the command record. For detailed interpretation of the route information, see B.

3. The title input section is reproduced.

4. The charge, multiplicity and input Z-matrix are reproduced together with initial values of the parameters. Any inconsistencies (e.g. incompatible multiplicity and electron number, parameter names not defined in Z-matrix, etc.) would be noted here and the run aborted.

5. The string GRADGRAD... delimit output from the Berny optimization program, link 103. On the initialization pass through this link, a table giving the initial values of the variables to be optimized is printed. Diagonal second derivatives of the energy with respect to the variables are needed by the optimization program initial values are estimated via an empirical procedure.

6. The initial Z-matrix is listed with the numerical values of all parameters substituted. This is followed by a set of Cartesian coordinates generated from this information and a matrix of internuclear distance. The last two of these are useful for debugging Z-matrix input. Note that, by default, the run is aborted at this point if an interatomic distance of less than 0.5A is computed or if an invalid Z-matrix parameter (such as a negative length) is encountered.

7. The next part of the output refers to symmetry aspects of the molecule. The stoichiometry, framework group and the corresponding number of degrees of freedom are listed. (Some fairly evident shorthand symbols such as SGV for σ_v are used in the framework notation.) If the number of variables is equal to the number of degrees of freedom, the optimization is *full* within the symmetry constraints imposed by the framework group (implicit in the Z-matrix). If the number of variables is less than the number of degrees of freedom, the optimization is *partial.* After the framework group has been determined, axes are rotated to a standard orientation in which symmetry elements coincide with Cartesian elements. (Full explanation of the standard orientation is in `SUBROUTINE ORDOC` in link 202.) This does not take place if the symmetry is so low that if leads to no useful gain in efficiency. The new Cartesian coordinates are listed and then the number of symmetry operations actually used by the program is given. (`gnu80` only makes use of twofold operations, so this number may be less than total number of symmetry operations.)

8. Quantum mechanical data such as basis sets, number of electrons and number and nature of two-electron integrals are printed next. (See Section D.6 for a description of integral formats.) Note that the information following the basis set in parentheses is in error but it is only the *printing* which is in error!

9. The initial guess program prints the symmetries of the orbitals occupied in the initial wave function. If the orbital symmetry assignment fails, this part of the output will contain some question marks, but the job will proceed. If the GUESS=PRINT command had been given, the full initial-guess wave function would have been printed here.

10. The result of the SCF calculation (RHF in this example) is given next. A convergance failure would have been noted had it occurred and would have been followed by abortion of the run. If this occurs, the initial guess should be checked and the job rebsummitted with the SCFDM keyword command. CONVG is the

RMS difference between the density matrix elements in consecutive cycles of the classical SCF procedure. -V/T is the virial ratio.

11. The orbital symmetries of the wavefunction are then given and this is followed by the full function and the one-electron eigenvalues. The latter give the Koopmans' approximation to the corresponding ionization potentials (in Hartrees). In general only the first five virtual molecular orbitals are printed. Note that the printed symmetry of the wave function is wrong if there are any degenerate occupied molecular orbitals.

12. A Mulliken population analysis is next carried out; the dipole moment is also given. All of this output uses the standard orientation. If the optimization (OPT) command had not been used, the output would have ended here. As the population analysis matrix is symmetric it is necessary to double the values of the off-diagonal elements to get an overlap population.

13. The axes are next restored to the original set (as noted in the output) This is followed by a calculation of the Cartesian forces (in Hartrees/ Bohr) and the corresponding derivatives with respect to the internal coordinates (lengths and angles used in the Z-matrix). If the FORCE command had been used instead of OPT, the output would have ended here.

14. The program then proceeds to the next step in the optimization. Note that all of the output from the optimization program is in atomic units. Using the current second-derivative (Hessian) matrix and the calculated first derivative vector, projection is made to a new set of values for the variables. (NEW X). In general, each step is composed of a linear search along the line connecting the current and previous points and a Newton-Raphson (quadratic) step. Under certain conditions one of these searches may be omitted in which case an appropriate explanation is printed.

15. There are four conditions which must be met before it is deemed that a stationary point has been located. A table is printed showing the current status of each of the conditions.

16. A new Hartree-Fock calculation is now initiated and carried through in the same way as before (except that the previous wave function is used for the new initial guess). This leads to further optimization steps.

17. The optimization is terminated when all four conditions are simultaneously met, as indicated by message OPTIMIZATION COMPLETED. (If the optimization procedure fails then resubmission with OPT=MS is recommended.) A table of the values of the variables at the stationary point is then printed along with the final forces. Note that the variable values are printed an Angstroms and degrees but that the derivatives are given in Hartree/Bohr or Hartree/radian.

18. The final wave function and Mulliken analysis is printed.

```
*******************************************
 gnu80: PORTABLE GAUSSIAN SYSTEM
********************************************
  ---------------
  # HF/STO-3G OPT
  ---------------
 1//1,3;
 2//2;
 3/6=3,11=9,25=11,30=1/1,2,3,11;
 4/16=1/1;
 5/6=7/1,2;
 6//1;
 7/27=1/1,2,16;
 1//3;
 2//2;
 3/6=3,11=9,25=11,30=1/1,2,11;
 5/6=7/1,2;
 7/27=1/1,2,16;
 1//3(-4);
 3/6=3,11=9,25=11,30=1/1,3;
 5/6=7,12=1/1,2;
 6//1;
 99//99;
  ------------------------------------
  WATER STRUCTURE OPTIMISATION STO-3G
  ------------------------------------
 SYMBOLIC Z-MATRIX
    CHARGE = 0 MULTIPLICITY = 1
 O
 H     1     R
 H     1     R         2     A
      VARIABLES
  R                    1.
  A                  105.
 GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
BERNY OPTIMIZATION
INITIALIZATION PASS
                       ---------------------------
                       !   INITIAL PARAMETERS    !
                       ! (ANGSTROMS AND DEGREES) !
 ----------------------                            ----------------------
 !      NAME          VALUE   DERIVATIVE INFORMATION (ATOMIC UNITS)     !
 ------------------------------------------------------------------------
 !       R          1.       ESTIMATE D2E/DX2                           !
 !       A        105.       ESTIMATE D2E/DX2                           !
 ------------------------------------------------------------------------
 GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
 ------------------------------------------------------------------------
                         Z-MATRIX (ANGSTROMS AND DEGREES)
 CD CENT ATOM  N1      LENGTH      N2     ALPHA      N3      BETA       J
 ------------------------------------------------------------------------
  1   1   O
  2   2   H     1   1.000000 (  1)
  3   3   H     1   1.000000 (  2)  2  105.000 (  3)
 ------------------------------------------------------------------------
                  Z-MATRIX ORIENTATION:
 ------------------------------------------------------------------------
 CENTER       ATOMIC                  COORDINATES (ANGSTROMS)
 NUMBER       NUMBER              X             Y             Z
 ------------------------------------------------------------------------
    1            8              0.00000000    0.00000000    0.00000000
    2            1              0.00000000    0.00000000    1.00000000
    3            1              0.96592583    0.00000000   -0.25881905
 ------------------------------------------------------------------------
                    DISTANCE MATRIX (ANGSTROMS)
              1          2          3
  1   O   0.000000
  2   H   1.000000   0.000000
  3   H   1.000000   1.586707   0.000000
 STOICHIOMETRY    H2O
 FRAMEWORK GROUP  C2V<C2(O),SGV(H2)>
 DEG. OF FREEDOM    2
                   STANDARD ORIENTATION:
 ------------------------------------------------------------------------
 CENTER       ATOMIC                  COORDINATES (ANGSTROMS)
 NUMBER       NUMBER              X             Y             Z
 ------------------------------------------------------------------------
    1            8              0.00000000    0.00000000   -0.12175229
    2           1               0.00000000    0.79335334    0.48700914
    3           1               0.00000000   -0.79335334    0.48700914
 -----------------------------------------------------------------------
 NUMBER OF SYMMETRY OPERATIONS:  2
 STANDARD BASIS: STO-3G     (S, S=P, 5D, 7F)
   7 BASIS FUNCTIONS       21 PRIMITIVE GAUSSIANS
   5 ALPHA ELECTRONS        5 BETA ELECTRONS
     NUCLEAR REPULSION ENERGY    8.8003395031 HARTREES
 RAFFENETTI 1 INTEGRAL FORMAT.
 TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
      144 INTEGRALS PRODUCED FOR A TOTAL OF       144
 INITIAL GUESS ORBITAL SYMMETRIES.
       OCCUPIED: (A1) (A1) (B2) (A1) (B1)
       VIRTUAL:  (B2) (A1)
 RHF CLOSED SHELL SCF.
 REQUESTED CONVERGENCE ON DENSITY MATRIX=  0.1000d-06 WITHIN  32 CYCLES.
 ITER  ELECTRONIC-ENERGY       CONVERGENCE    EXTRAPOLATION
 ----  -----------------       -----------    -------------
   1  -0.836400480512479d+02
   2  -0.837468019504395d+02    0.4717d-01
   3  -0.837614333581677d+02    0.2005d-01
   4  -0.837641343159253d+02    0.8648d-02       4-POINT.
   5        (NON-VARIATIONAL)
   6  -0.837647845061776d+02    0.5484d-04
   7  -0.837647845272043d+02    0.2365d-04
   8  -0.837647845313501d+02    0.1060d-04       4-POINT.
   9        (NON-VARIATIONAL)
  10  -0.837647845323884d+02

 SCF DONE:  E(RHF) =   -74.9644450293     A.U. AFTER   10 CYCLES
            CONVG  =    0.2320d-07             -V/T =   2.006285
 COMPONENT                       A.U.                   KCAL/MOL
 ---------------------------------------------------------------
 TOTAL                      -74.964445029260          -47041.689
 ELECTRONIC                 -83.764784532388          -52564.078
 NUCLEAR REPULSION            8.800339503129            5522.389
 KINETIC                     74.496267425257           46747.898
 POTENTIAL                 -149.460712454517          -93789.586
 ELECTRONIC POTENTIAL      -158.261051957645          -99311.975
```

```
ONE-ELECTRON POTENTIAL    -196.174545269398         -123103.451
TWO-ELECTRON POTENTIAL      37.913493311753           23791.475
ORBITAL SYMMETRIES.
      OCCUPIED: (A1) (A1) (B2) (A1) (B1)
      VIRTUAL:  (A1) (B2)
 THE ELECTRONIC STATE IS 1-A1.
     MOLECULAR ORBITAL COEFFICIENTS
                        1         2         3         4         5
                      (A1)      (A1)      (B2)      (A1)      (B1)
    EIGENVALUES --   -20.24657  -1.24722  -0.59661  -0.44669  -0.38857
  1 1   O  1S          0.99423   0.23458   0.00000   0.10045   0.00000
  2        2S          0.02571  -0.84944   0.00000  -0.52050   0.00000
  3        2PX         0.00000   0.00000   0.00000   0.00000   1.00000
  4        2PY         0.00000   0.00000   0.60459   0.00000   0.00000
  5        2PZ         0.00393  -0.11474   0.00000   0.76909   0.00000
  6 2   H  1S         -0.00552  -0.15538   0.44747   0.29394   0.00000
  7 3   H  1S         -0.00552  -0.15538  -0.44747   0.29394   0.00000
                        6         7
                      (A1)      (B2)
    EIGENVALUES --     0.56320   0.69443
  1 1   O  1S         -0.12648   0.00000
  2        2S          0.81425   0.00000
  3        2PX         0.00000   0.00000
  4        2PY         0.00000  -0.96843
  5        2PZ         0.73747   0.00000
  6 2   H  1S         -0.76814   0.80141
  7 3   H  1S         -0.76814  -0.80141
   FULL MULLIKEN POPULATION ANALYSIS.
                        1         2         3         4         5
  1 1   O  1S          2.10722
  2        2S         -0.10698   1.98624
  3        2PX         0.00000   0.00000   2.00000
  4        2PY         0.00000   0.00000   0.00000   0.73107
  5        2PZ         0.00000   0.00000   0.00000   0.00000   1.20938
  6 2   H  1S         -0.00121  -0.01892   0.00000   0.16235   0.11230
  7 3   H  1S         -0.00121  -0.01892   0.00000   0.16235   0.11230
                        6         7
  6 2   H  1S          0.62160
  7 3   H  1S         -0.04061   0.62160
```

```
    GROSS ORBITAL CHARGES.
                            1
  1 1   O  1S          1.99782
  2        2S          1.84143
  3        2PX         2.00000
  4        2PY         1.05576
  5        2PZ         1.43397
  6 2   H  1S          0.83551
  7 3   H  1S          0.83551
         CONDENSED TO ATOMS (ALL ELECTRONS)
             1          2          3
 1   O   7.819938   0.254520   0.254520
 2   H   0.254520   0.621600  -0.040609
 3   H   0.254520  -0.040609   0.621600
         TOTAL ATOMIC CHARGES.
             1
 1   O   8.328978
 2   H   0.835511
 3   H   0.835511
DIPOLE MOMENT (DEBYE): X= 0.0000   Y= 0.0000   Z= 1.6661   TOTAL= 1.6661
***** AXES RESTORED TO ORIGINAL SET *****
          MAX     0.017249     RMS     0.009712
----------------------------------------------------------
CENTER     ATOMIC              FORCES (HARTREES/BOHR)
NUMBER     NUMBER              X           Y           Z
----------------------------------------------------------
   1          8          -0.002843    0.000000   -0.002181
   2          1           0.013820    0.000000   -0.015068
   3          1          -0.010978    0.000000    0.017249
----------------------------------------------------------
------------------------------------------------------------------------
                   INTERNAL COORDINATE FORCES (HARTREES/BOHR OR /RADIAN)
CENT ATOM N1      LENGTH      N2      ALPHA       N3       BETA        J
------------------------------------------------------------------------
 1   O
 2   H     1  -0.015068 (  1)
 3   H     1  -0.015068 (  2)  2  -0.026117 (  3)
------------------------------------------------------------------------
                 MAX     0.026117     RMS     0.019461
```

```
GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
BERNY OPTIMIZATION
SEARCH FOR A LOCAL MINIMUM.
STEP NUMBER   1 OUT OF A MAXIMUM OF  12
ALL QUANTITIES PRINTED IN INTERNAL UNITS (HARTREES-BOHRS-RADIANS)
SECOND DERIVATIVE MATRIX NOT UPDATED -- FIRST STEP
THE SECOND DERIVATIVE MATRIX:
                          R         A
          R           1.28137
          A           0.00000   0.30965
    EIGENVALUES ---    0.30965   1.28137
LINEAR SEARCH NOT ATTEMPTED -- FIRST POINT
VARIABLE       OLD X    -DE/DX   DELTA X   DELTA X   DELTA X     NEW X
                                 (LINEAR)    (QUAD)   (TOTAL)

   R         1.88973  -0.03014   0.00000  -0.02352  -0.02352   1.86621
   A         1.83260  -0.02612   0.00000  -0.08434  -0.08434   1.74825
        ITEM              VALUE     THRESHOLD  CONVERGED
MAXIMUM FORCE           0.030137     0.000450     NO
RMS     FORCE           0.028198     0.000300     NO
MAXIMUM DISPLACEMENT    0.084344     0.001800     NO
RMS     DISPLACEMEMT    0.061915     0.001200     NO
PREDICTED CHANGE IN ENERGY -0.001456
GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
------------------------------------------------------------------------
                        Z-MATRIX (ANGSTROMS AND DEGREES)
CD CENT ATOM  N1      LENGTH      N2     ALPHA      N3      BETA       J
------------------------------------------------------------------------
 1   1   O
 2   2   H     1   0.987554 (   1)
 3   3   H     1   0.987554 (   2)  2  100.167 (   3)
------------------------------------------------------------------------
                 Z-MATRIX ORIENTATION:
------------------------------------------------------------------------
CENTER        ATOMIC                  COORDINATES (ANGSTROMS)
NUMBER        NUMBER              X             Y             Z
------------------------------------------------------------------------
   1             8            0.00000000    0.00000000    0.00000000
   2             1            0.00000000    0.00000000    0.98755422
   3            1                0.97204571    0.00000000   -0.17432866
-----------------------------------------------------------------------
                  DISTANCE MATRIX (ANGSTROMS)
             1          2          3
 1   O   0.000000
 2   H   0.987554   0.000000
 3   H   0.987554   1.514874   0.000000
STOICHIOMETRY    H2O
FRAMEWORK GROUP  C2V<C2(O),SGV(H2)>
DEG. OF FREEDOM    2
                  STANDARD ORIENTATION:
-----------------------------------------------------------------------
CENTER        ATOMIC                  COORDINATES (ANGSTROMS)
NUMBER        NUMBER               X             Y             Z
-----------------------------------------------------------------------
   1            8                0.00000000    0.00000000   -0.12673629
   2            1                0.00000000    0.75743724    0.50694515
   3            1                0.00000000   -0.75743724    0.50694515
-----------------------------------------------------------------------
NUMBER OF SYMMETRY OPERATIONS:  2
STANDARD BASIS: STO-3G     (S, S=P, 5D, 7F)
  7 BASIS FUNCTIONS       21 PRIMITIVE GAUSSIANS
  5 ALPHA ELECTRONS        5 BETA ELECTRONS
    NUCLEAR REPULSION ENERGY    8.9228580565 HARTREES
RAFFENETTI 1 INTEGRAL FORMAT.
TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
     144 INTEGRALS PRODUCED FOR A TOTAL OF       144
RHF CLOSED SHELL SCF.
REQUESTED CONVERGENCE ON DENSITY MATRIX=  0.1000d-06 WITHIN  32 CYCLES.
ITER  ELECTRONIC-ENERGY       CONVERGENCE    EXTRAPOLATION
----  -----------------       -----------    -------------
  1  -0.838959653161141d+02
  2  -0.838887179983350d+02    0.1835d-02
  3  -0.838887450742661d+02    0.8238d-03
  4  -0.838887500919666d+02    0.3599d-03       4-POINT.
  5        (NON-VARIATIONAL)
  6  -0.838887513098814d+02    0.2931d-06
  7  -0.838887513098826d+02    0.1083d-06
  8  -0.838887513098828d+02
```

```
SCF DONE:  E(RHF) =   -74.9658932533     A.U. AFTER    8 CYCLES
           CONVG  =    0.4705d-07             -V/T =   2.005951
COMPONENT                       A.U.                   KCAL/MOL
----------------------------------------------------------------
TOTAL                      -74.965893253341          -47042.597
ELECTRONIC                 -83.888751309883          -52641.869
NUCLEAR REPULSION            8.922858056542            5599.272
KINETIC                     74.522446994007           46764.326
POTENTIAL                 -149.488340247348          -93806.923
ELECTRONIC POTENTIAL      -158.411198303890          -99406.195
ONE-ELECTRON POTENTIAL    -196.386504202976         -123236.459
TWO-ELECTRON POTENTIAL      37.975305899087           23830.264
***** AXES RESTORED TO ORIGINAL SET *****
          MAX     0.002369     RMS     0.001436
-----------------------------------------------------------
CENTER     ATOMIC              FORCES (HARTREES/BOHR)
NUMBER     NUMBER            X            Y           Z
-----------------------------------------------------------
   1          8          -0.002369    0.000000   -0.001982
   2          1           0.000248    0.000000    0.002110
   3          1           0.002121    0.000000   -0.000129
-----------------------------------------------------------
-------------------------------------------------------------------------
                   INTERNAL COORDINATE FORCES (HARTREES/BOHR OR /RADIAN)
CENT ATOM N1       LENGTH      N2      ALPHA       N3       BETA        J
-------------------------------------------------------------------------
 1   O
 2   H     1   0.002110 (  1)
 3   H     1   0.002110 (  2)  2  -0.000463 (  3)
-------------------------------------------------------------------------
                 MAX     0.002110     RMS     0.001744
GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
BERNY OPTIMIZATION
SEARCH FOR A LOCAL MINIMUM.
STEP NUMBER   2 OUT OF A MAXIMUM OF  12
ALL QUANTITIES PRINTED IN INTERNAL UNITS (HARTREES-BOHRS-RADIANS)
UPDATE SECOND DERIVATIVES USING INFORMATION FROM POINTS:  1  2
THE SECOND DERIVATIVE MATRIX:
                          R         A
          R           1.28193
          A           0.00203   0.31694
    EIGENVALUES ---   0.31693   1.28194
LINEAR SEARCH NOT ATTEMPTED -- RMS FORCE IS LESS THAN FSWTCH ( 0.01000)
VARIABLE       OLD X    -DE/DX   DELTA X   DELTA X   DELTA X     NEW X
                                 (LINEAR)    (QUAD)   (TOTAL)

   R         1.86621   0.00422   0.00000   0.00329   0.00329   1.86950
   A         1.74825  -0.00046   0.00000  -0.00148  -0.00148   1.74677
        ITEM               VALUE     THRESHOLD  CONVERGED
MAXIMUM FORCE            0.004221     0.000450     NO
RMS     FORCE            0.003002     0.000300     NO
MAXIMUM DISPLACEMENT     0.003295     0.001800     NO
RMS     DISPLACEMEMT     0.002554     0.001200     NO
PREDICTED CHANGE IN ENERGY -0.000007
GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
------------------------------------------------------------------------
                       Z-MATRIX (ANGSTROMS AND DEGREES)
CD CENT ATOM  N1      LENGTH      N2     ALPHA      N3      BETA       J
------------------------------------------------------------------------
 1   1   O
 2   2   H     1   0.989298 (  1)
 3   3   H     1   0.989298 (  2)  2  100.083 (  3)
------------------------------------------------------------------------
                 Z-MATRIX ORIENTATION:
------------------------------------------------------------------------
CENTER        ATOMIC                    COORDINATES (ANGSTROMS)
NUMBER        NUMBER                 X             Y             Z
------------------------------------------------------------------------
   1             8                0.00000000    0.00000000    0.00000000
   2             1                0.00000000    0.00000000    0.98929774
   3             1                0.97401935    0.00000000   -0.17319445
------------------------------------------------------------------------
                   DISTANCE MATRIX (ANGSTROMS)
             1          2          3
 1   O   0.000000
 2   H   0.989298   0.000000
 3   H   0.989298   1.516609   0.000000
STOICHIOMETRY    H2O
```

```
FRAMEWORK GROUP  C2V<C2(O),SGV(H2)>
DEG. OF FREEDOM    2
                  STANDARD ORIENTATION:
------------------------------------------------------------------------
CENTER        ATOMIC                 COORDINATES (ANGSTROMS)
NUMBER        NUMBER              X             Y             Z
------------------------------------------------------------------------
   1            8               0.00000000    0.00000000   -0.12707235
   2            1               0.00000000    0.75830432    0.50828941
   3            1               0.00000000   -0.75830432    0.50828941
------------------------------------------------------------------------
NUMBER OF SYMMETRY OPERATIONS:  2
STANDARD BASIS: STO-3G     (S, S=P, 5D, 7F)
  7 BASIS FUNCTIONS       21 PRIMITIVE GAUSSIANS
  5 ALPHA ELECTRONS        5 BETA ELECTRONS
    NUCLEAR REPULSION ENERGY    8.9073487817 HARTREES
RAFFENETTI 1 INTEGRAL FORMAT.
TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
     144 INTEGRALS PRODUCED FOR A TOTAL OF       144
RHF CLOSED SHELL SCF.
REQUESTED CONVERGENCE ON DENSITY MATRIX=  0.1000d-06 WITHIN  32 CYCLES.
ITER  ELECTRONIC-ENERGY       CONVERGENCE    EXTRAPOLATION
----  -----------------       -----------    -------------
  1  -0.838717588320286d+02
  2  -0.838732496849179d+02    0.1102d-03
  3  -0.838732497881338d+02    0.5013d-04
  4  -0.838732498077915d+02    0.2224d-04      4-POINT.
  5        (NON-VARIATIONAL)
  6  -0.838732498125776d+02    0.1023d-06
  7  -0.838732498125777d+02
SCF DONE:  E(RHF) =   -74.9659010308     A.U. AFTER    7 CYCLES
           CONVG  =    0.4113d-07            -V/T =   2.006000
COMPONENT                       A.U.                   KCAL/MOL
---------------------------------------------------------------
TOTAL                     -74.965901030833          -47042.602
ELECTRONIC                -83.873249812578          -52632.142
NUCLEAR REPULSION           8.907348781745            5589.540
KINETIC                    74.518807994534           46762.042
POTENTIAL                -149.484709025367          -93804.645
```

```
ELECTRONIC POTENTIAL      -158.392057807111          -99394.184
ONE-ELECTRON POTENTIAL    -196.354814562426         -123216.573
TWO-ELECTRON POTENTIAL      37.962756755315           23822.389
***** AXES RESTORED TO ORIGINAL SET *****
          MAX     0.000271     RMS     0.000145
-----------------------------------------------------------
CENTER     ATOMIC              FORCES (HARTREES/BOHR)
NUMBER     NUMBER             X            Y            Z
-----------------------------------------------------------
   1          8          -0.000271    0.000000   -0.000227
   2          1           0.000146    0.000000    0.000101
   3          1           0.000125    0.000000    0.000126
-----------------------------------------------------------
------------------------------------------------------------------------
                   INTERNAL COORDINATE FORCES (HARTREES/BOHR OR /RADIAN)
CENT ATOM N1      LENGTH      N2      ALPHA       N3      BETA        J
------------------------------------------------------------------------
 1   O
 2   H     1   0.000101 (  1)
 3   H     1   0.000101 (  2)  2  -0.000273 (  3)
------------------------------------------------------------------------
                 MAX     0.000273     RMS     0.000178
GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
BERNY OPTIMIZATION
SEARCH FOR A LOCAL MINIMUM.
STEP NUMBER   3 OUT OF A MAXIMUM OF  12
ALL QUANTITIES PRINTED IN INTERNAL UNITS (HARTREES-BOHRS-RADIANS)
UPDATE SECOND DERIVATIVES USING INFORMATION FROM POINTS:  1  2  3
THE SECOND DERIVATIVE MATRIX:
                          R         A
          R           1.25023
          A           0.06846   0.28070
    EIGENVALUES ---    0.27589   1.25504
LINEAR SEARCH NOT ATTEMPTED -- RMS FORCE IS LESS THAN FSWTCH ( 0.01000)
VARIABLE       OLD X    -DE/DX   DELTA X   DELTA X   DELTA X     NEW X
                                 (LINEAR)    (QUAD)   (TOTAL)

   R         1.86950   0.00020   0.00000   0.00022   0.00022   1.86972
   A         1.74677  -0.00027   0.00000  -0.00102  -0.00102   1.74575
```

```
        ITEM              VALUE     THRESHOLD  CONVERGED
MAXIMUM FORCE           0.000273     0.000450     YES
RMS     FORCE           0.000240     0.000300     YES
MAXIMUM DISPLACEMENT    0.001024     0.001800     YES
RMS     DISPLACEMEMT    0.000740     0.001200     YES
PREDICTED CHANGE IN ENERGY  0.000000
OPTIMIZATION COMPLETED.
   -- STATIONARY POINT FOUND.
   -- LAST STEP NOT IMPLEMENTED.
                      ---------------------------
                      !  OPTIMIZED PARAMETERS   !
                      ! (ANGSTROMS AND DEGREES) !
----------------------                           ----------------------
!      NAME          VALUE   DERIVATIVE INFORMATION (ATOMIC UNITS)     !
------------------------------------------------------------------------
!        R          0.9893   -DE/DX =    0.000202                      !
!        A        100.0826   -DE/DX =   -0.000272                      !
------------------------------------------------------------------------
GRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRADGRAD
STANDARD BASIS: STO-3G     (S, S=P, 5D, 7F)
  7 BASIS FUNCTIONS       21 PRIMITIVE GAUSSIANS
  5 ALPHA ELECTRONS        5 BETA ELECTRONS
    NUCLEAR REPULSION ENERGY    8.9073487817 HARTREES
RAFFENETTI 1 INTEGRAL FORMAT.
TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
ORBITAL SYMMETRIES.
      OCCUPIED: (A1) (A1) (B2) (A1) (B1)
      VIRTUAL:  (A1) (B2)
 THE ELECTRONIC STATE IS 1-A1.
     MOLECULAR ORBITAL COEFFICIENTS
                            1          2          3          4          5
                          (A1)       (A1)       (B2)       (A1)       (B1)
    EIGENVALUES --     -20.25150  -1.25754  -0.59401  -0.45961  -0.39259
  1 1   O  1S            0.99422  -0.23377   0.00000  -0.10401   0.00000
  2        2S            0.02585   0.84442   0.00000   0.53806   0.00000
  3        2PX           0.00000   0.00000   0.00000   0.00000   1.00000
  4        2PY           0.00000   0.00000   0.61260   0.00000   0.00000
  5        2PZ           0.00416   0.12283   0.00000  -0.75607   0.00000
  6 2   H  1S           -0.00559   0.15561   0.44919  -0.29500   0.00000
  7 3   H  1S         -0.00559   0.15561  -0.44919  -0.29500   0.00000
                         6         7
                       (A1)      (B2)
    EIGENVALUES --     0.58182   0.69295
  1 1   O  1S          0.12586   0.00000
  2        2S         -0.82045   0.00000
  3        2PX         0.00000   0.00000
  4        2PY         0.00000  -0.96003
  5        2PZ        -0.76328   0.00000
  6 2   H  1S          0.76931   0.81468
  7 3   H  1S          0.76931  -0.81468
   FULL MULLIKEN POPULATION ANALYSIS.
                         1         2         3         4         5
  1 1   O  1S          2.10786
  2        2S         -0.10778   2.00645
  3        2PX         0.00000   0.00000   2.00000
  4        2PY         0.00000   0.00000   0.00000   0.75056
  5        2PZ         0.00000   0.00000   0.00000   0.00000   1.17350
  6 2   H  1S         -0.00112  -0.02495   0.00000   0.16118   0.11883
  7 3   H  1S         -0.00112  -0.02495   0.00000   0.16118   0.11883
                         6         7
  6 2   H  1S          0.62609
  7 3   H  1S         -0.04538   0.62609
    GROSS ORBITAL CHARGES.
                         1
  1 1   O  1S          1.99783
  2        2S          1.84878
  3        2PX         2.00000
  4        2PY         1.07292
  5        2PZ         1.41116
  6 2   H  1S          0.83465
  7 3   H  1S          0.83465
         CONDENSED TO ATOMS (ALL ELECTRONS)
              1          2          3
 1    O   7.822817   0.253942   0.253942
 2    H   0.253942   0.626090  -0.045383
 3    H   0.253942  -0.045383   0.626090
         TOTAL ATOMIC CHARGES.
              1
  1   O   8.330702
  2   H   0.834649
  3   H   0.834649
 DIPOLE MOMENT (DEBYE): X= 0.0000   Y= 0.0000   Z= 1.7091   TOTAL= 1.7091

 **** gnu80 STOPS EXECUTING *****
```

Appendix 2.B

Banana Skins

2.B.3 Non-existent Functions

It is relatively easy to list the limitations of the gnu80 system; maximum number and type of basis functions, maximum amount of integral storage, type of quantum mechanical model ate. etc. What are not quite so obvious are the limitations of some of the underlying *theoretical methods.*

Tecnically, for example, the basic **HF** models are the equations resulting from using the *parametric* variation method on the Hartree-Fock Energy functional; the full (functional) variation method in which *arbitrary* variations are allowed in the form of the MOs has been replaced by the variation of a finite set of linear parameters, the MO coefficients. This is done for entirely practical reasons and in most cases the errors arising from this limitation of the variation method are entirely *quantitative*; the minimum energy of the Hartree-Fock limit is not reached.

However, the parametric variation principle makes the all-important *assumption* of the *existence* of the solutions. A closed-shell calculation of the electronic structure of the hydride anion proceeds smoothly to a minimum which is, unfortunately, an artifact of the linear expansion method and the closed-shell model. The energy of this anion is higher than that of the hydrogen atom and so the HF solution of lowest energy for the hydride anion should be a hydrogen atom and a free electron. Using the Different Orbitals for Different Spins (DODS or UHF) model will generate a solution which is as close to this latter situation as the basis allows but, *unless very diffuse orbitals have been included in the basis* to simulate a free-electron function, the true minimum is not accessible to the linear expansion method; this is a *qualitative* error.

The General rule is: do not attempt to use the facilities of gnu80 to calculate "Approximations to functions which do not exist"
.

2.B.4 Anions

The largest class of system for which the Hartree-Fock solutions do not exist (their energies are higher than that of the neutral molecule) are anions. Very, very few anions carrying a single negative charge have bound-state Hartree-Fock solutions and *no* multiply-charged anions have bound solutions. This is in the Hartree-Fock limit, any approximations due to finite bases will, of course, make this result stronger.

As in the case of the hydride anion mentioned above, use of the **HF** model with a standard basis will often generate a bogus approximation to the non-existent bound state.

2.B.5 Dissociation Limits

It is well-known that the closed-shell MO method, with its insistence on sets of *doubly-occupied* Molecular Orbitals will often give the wrong description of a molecule as the bond lengths increase. Typically, the closed-shell **HF** model will predict that a bond dissociate to *ionic* fragments rather than neutral species. Often the use of the **UHF** model in these cases will solve the problem, provided that the program is flexible enough to allow the use of the **UHF** model with an even (nominally paired) number of electrons.

2.B.6 Spin Eigenfunction Constraints

There is a wealth of practical/computational experience which shows that the imposition of a spin eigenfunction constraint on a single-determinant wavefunction often constrains the action of the variation principle so badly that completely spurious results are obtained. The classic case in the literature is that of the optimisation of the structure of the allyl radical which has a single unpaired electron in the conventional picture. If a single determinant wave function has this constraint *imposed* throughout the structure optimisation, there is an artificial cusped maximum in the energy at the observed structure and a pair of equivalent minima having one long and one short C-C bond.

If the constraint is removed (i.e. a UHF function is used) the optimisation generates the correct qualitative structure and the quantitative result depends, as usual, on the quality of the basis.

The recommendation is:

Always use the UHF model if there is any suspicion of qualitative errors in the structure.

Chapter 3

General Use of gnu80

3.1 Beyond ' Standard Methods'

The reasons why one might want to use the more fundamental structures of gnu80 are, in general, one or more of the following:

1. The basis sets provided internally by gnu80 may not be adequate or appropriate for a particular problem. In section 2.9.2, the limitations of the basis sets are given; acceptable quality internal basis sets are only available for atoms up to Argon. Although gnu80 recognises the chemical symbols of the atoms of the third row of the periodic table (first transition series), there are *no* basis sets provided for these atoms.

2. The calculations may be required in a sequence which is not generated by the command record parser.

3. Much the most common requirement is to recover from some kind of disaster in a Standard gnu80 run; a convergence failure or a machine failure during a long run.

In point of fact, many of these requirement can be met by using a "standard method" command record including the **NONDEF** command and a specific set of non-default options.

3.2 Program Structure

3.2.1 Introduction

In order to be able to use the full flexibility of gnu80 it is necessary to have some knowledge of the way the system is structured and the way in which the Commands and Options of the last Chapter relate to the more basic Overlays, Links and Options. Basically, General Use consists of replacing the work of the module which parses the command record and generates a ROUTE through gnu80. If the output of one of the sample data sets is examined, the ROUTE is the set of records which consists of (small) integers separated by slashes, commas and equals signs at the start of the job. It is this sequence or some replacement for it which must be generated "by hand" if gnu80 is to be of General Use.

However, except for the simplest ROUTEs with the most common options (which could, in any case, be more easily generated by Standard Methods!) it is almost impossible to generate a ROUTE from first principles. What is done in practice, if the Standard Methods command record will not generate the ROUTE required, is to use the Standard Methods parser to generate the nearest ROUTE and then *modify* this ROUTE manually by adding links and options.

In fact, this can be done in a two-stage process by starting with a Standard Method command record, adding the **NONDEF** command and suppliying some non-default options then, finally, changing the ROUTE and any remaining options. The beginner is strongly advised to get some familiarity with the **NONDEF** command and its consequences before experimenting with General Use.

In fact, it is just about as feasible to write a useful, non-standard, ROUTE from first principles as it is to learn to ride a bicycle or to swim by reading the manual.

3.2.2 Overview of gnu80

gnu80 consists of a series of programs which may be called in arbitrary sequence and which communicate with each other through disc files. Each such program is called a **LINK**; the links are grouped into

OVERLAYS. The numbering of the links is such that each overlay may contain up to 99 links. Thus overlay 1 may have links 101, 102, ... 199, overlay 2 may have links 201,202, ... 299 and so forth. Links with closely related functions (from the quantum chemistry point of view) are grouped together in the same overlay. Note that this terminology is essentially historical and is only used to designate the different *parts* of the program, and has little coincidence with the physical overlay structure of the load module if, indeed, the load module is explicitly overlaid; it may be all loaded or the "overlays" managed by the virtual memory manager.

Overlay 0 reads the command record, forms the detailed route, and sets up files and some general tables.

Overlay 1 reads Input Sections 2—5 and control optimization procedures.

Overlay 2 computes Cartesian coordinates of the nuclei and derives symmetry related information.

Overlay 3 sets up the basis set and evaluates one- and two-electron integrals.

Overlay 4 forms the initial guess needed for Hartree-Fock wavefunctions.

Overlay 5 determines the Hartree-Fock wavefunction (SCF procedures).

Overlay 6 performs analysis and output of Hartree-Fock wavefunctions and energies.

Overlay 7 evaluates integral derivatives (with respect to nuclear coordinates) and uses them to evaluate Hartree-Fock forces on the nuclei.

Overlay 8 transforms the two-electron integrals to the basis of molecular orbitals.

Overlay 9 calculates the correlation energy either by Pertubation Theory or by Configuration Interaction.

Overlay 99 is the termination Overlay.

The complete set of links, together with a short functional description, is listed in Appendix A at the end of this chapter.

3.3 Program Options

The execution of each Overlay is controlled by a number of OPTIONS (numbered from 1 to 50). Each OPTION may be assigned positive integer values, 0 being the default.

Note that the value of an OPTION is held unchanged throughout execution of *all the links in one Overlay.* Thus the significance of a particular OPTION applies to all the component links in one pass through the Overlay. An exception is Overlay 5, where the OPTIONs in Link 503 differ from those appropriate to 501 or 502. However, these are independent SCF programs and execution of 503 immediately after 501 or 502 is never required.

The full list of OPTIONs in gnu80 is given in Appendix B which follows Appendix A at the end of the chapter.

3.4 Non-Default OPTIONs

OPTIONs set by the route generator can be reset to any particular value without having to specify a complete non-standard route (see 3.5 for non-standard routes). This is accomplished by using the **NONDEF** command in the command record. A separate input section (1b) should be inserted *after the blank record following the command record* which specifies the non-default OPTIONs. For most jobs, all of the non-default OPTIONs can be placed on one record, but overflow to other records is permissable. The section must be terminated by a blank record.

Non-default OPTIONs can be set for any record generated by the route generator. They are set by specifying the Overlay for which the OPTIONs are to be changed, followed by a separator (space , : /) followed by a list of the OPTIONs and their new values. Thus:
3/34=1,35=4

will set OPTIONs 34 and 35 in Overlay 3 to 1 and 4 respectively. If OPTIONs have to be reset in different Overlays the specifications must be separated by a semi-colon (;). Thus:
2/34=1;3/34=5
will set OPTION 34 to 1 in Overlay 2 and to 5 in Overlay 3. It is sometimes necessary to set OPTIONs in one or more occurences of an Overlay. This is done by specifying the occurence number of the link:
3(2)/34=5;3(3)/34=5
will set OPTION 34 to 5 in the second and the third occurence of link 3 and
3(*)/34=1
will set OPTION 34 to 1 in all occurences of Overlay 3.

3.5 Non-Standard Routes

Non-Standard routes may be specified by requesting a particular sequence of Overlays and links together with associated OPTIONs.

The command record is now replaced by a series of job-descriptions records beginning with the record:
# NONSTD

This is followed by one record for each Overlay in the sequence. This record gives the Overlay number, a slash (/) symbol, the OPTIONs for that Overlay, another slash (/) symbol, the list of links within that Overlay to be executed and finally a semi-colon (;) symbol.
Thus:
7/5=3,7=4/2,3,16;

commands a pass through links 702, 703, 716 (in this order) with OPTION 5 set equal to 3 and OPTION 7 equal to 4. If all OPTIONs have their default (0) value, the record would be:
7//2,3,16;

One further feature in the route specification is the **JUMP** number. This is given in parentheses at the end of the link list, before the semi-colon. It indicates where the execution jumps to after completion of the

current Overlay. If jump is omitted, the default value is +1, indicating that the program will proceed to the next record in the list. if JUMP = -4, on the other hand, as in:
7//2,3,16(-4);

then execution will next take place four route records back. The **JUMP** feature permits loops to be built into the route and is useful for optimization runs. JUMP is altered to +1 in Overlay 1 when optimization are complete by an appropriate argument in the call to the chaining routine.

3.6 General Basis Sets: non-STO

A set of "standard" Basis sets (STO-3G, 3-21G, etc.) is stored internally in `gnu80`. When one of these bases is selected (either with a basis command in the command record or by setting the appropriate Overlay 3 OPTIONs), no basis set input is required.

A "non-standard" basis can be used in a `gnu80` calculation, either by specifying **GEN** (for general basis) in the command record, or by setting the appropriate overlay 3 OPTIONs. In this case, the basis set description must be provided as *input* in a separate basis input section.

In this context, any basis which is not *internal* to `gnu80` (see Section 2.3.2) is called a general basis. This includes both what might be called "genuine" general bases *and* "standard" bases *which are not provided by* `gnu80`. For example, the STO-1G basisis a general basis, not because it is in any way unusual but simply because STO-1G bases provided internally by `gnu80`. In practice, this is the most common reason for the use of the **GEN** command; to enable a standard (i.e. non-Pseudopotential) calculation to be performed on a molecule containing an atom of atomic number greater than Argon.

Note that the internally-stored "minimal" bases (STO-3G) are used in the calculation of the default GUESS of SCF starting point; the Huckel or Projected Huckel GUESS. Thus, if an atom of atomic number greater than that that of the heaviest atom for which this basis is available is to be included in a molecular calculation, the default GUESS cannot be provided. The only sensible option in this case is to use the eigenvectors of the one-electron Hamiltonian i.e. GUESS=CORE. This is done and a message printed.

3.6.1 Shells

Before describing the format of the basis section, a discussion of *shells* is useful.

A *shell* is a set of basis functions $\{\phi_\mu\}$ with common shared exponents. gnu80 supports *four* kinds of shells:

- s shells; An s-shell contains a single s-type basis function.
- p shells; A p-shell contains the three basis functions $p_x,\ p_y,$ and p_z.
- d shells; A d-shell may be defined to contain either the six "second-order" basis functions $d_{x^2},\ d_{y^2},\ d_{z^2},\ d_{xy},\ d_{yz},\ d_{zx}$ or the five "pure d" (Cubic Harmonic) basis functions $d_{z^2-r^2},\ d_{x^2-y^2},\ d_{xy},\ d_{yz},\ d_{zx}$
- sp shells; An sp-shell contains four basis functions with common gaussian exponents: one s-type function and the three p-functions comprising a p-shell.

Usually, a single basis function is a linear combination of more than one primitive gaussian function. Thus, for an s-type function, one may have for the basis function $\phi_\mu(r)$:

$$\phi_\mu(\vec{r}) = \sum_{i=1}^{N} d_{i\mu} exp(-\alpha_i f^2 r^2)$$

Here N is the number of primitive functions composing the basis function and is called the *degree of contraction* of the basis function. The coefficients $d_{i\mu}$ are called contraction coefficients. The quantities α_i are the *exponents* and f is the *scale factor* for the basis function.

3.6.2 Exponents and Scale Factors

The STO-NG Gaussian expansions of Slater Type Orbitals are provided for the first 36 atoms of the periodic table (H—Ar) and assume the following STO orbital exponents which are either the optimum isolated-atom values for inner STOs or molecule-optimised values for the valence shells. The exponents are constrained by the condition that they are for *shells* rather than for individual STOs i.e.

$$\zeta_{2s} = \zeta_{2p}\zeta_{3s} = \zeta_{3p}$$

There is some confusing nomenclature in the literature arising from the way in which the Gaussian expansion of STOs is carried out. A given expansion of a STO of a given type (e.g. 2p) is a set of linear expansion coefficients and Gaussian exponents as above.

The expansion coefficients ($d_{i\mu}$) are *independent* of the STO exponent (ζ, say) and the α_i depend on the *square* of ζ. Thus the expansion coefficients and exponents (α_i are usually quoted for an STO exponent $\zeta = 1$ and the corresponding scaled vales for all other STOs can be easily obtained.

This "scaling" of the Gaussian exponents has led to the use of the term "scale factors" being used for the STO *exponents.* However, it is an empirical fact that, in a molecular environment, the STO optimum STO exponents are not quite the same as the ones for the separate atoms and so they are often "scaled" to allow for this molecular electron re-arrangment. The numbers which do this scaling (usually numbers around unity, but typically greater than 1) are also call "scale factors" ! Thses are the f_i in the above expansion. In this manual the STO orbital exponents will be called exponents ($\zeta's$) and any factors which multiply the orbital exponents to accommodate a molecular environment will be called scale factors.

With this in mind the standard STO *exponents* for the STO-NG expansions are:

Atom	1s	2sp	3sp
H	1.24	-	-
He	1.69	-	-
Li	2.69	0.80	-
Be	3.68	1.15	-
B	4.68	1.50	-
C	5.67	1.72	-
N	6.67	1.95	-
O	7.66	2.25	-
F	8.65	2.55	-
Ne	9.64	2.88	-
Na	10.61	3.48	1.75
Mg	11.59	3.90	1.70
Al	12.56	4.36	1.70
Si	13.53	4.83	1.75
P	14.50	5.31	1.90
S	15.47	5.79	2.05
Cl	16.43	6.26	2.10
Ar	17.40	6.74	2.33

The N-31G, N-31G* and N-31G** have the same inner-shell STO orbital exponent (the one expanded in terms of N Gaussians) and a "split-valence" set of two STOs in the valence shell (one expanded in terms of 3 Gaussians and one expanded as a single term). The relevant valence STO exponents are obtained from the above *single* exponents by multiplying each single valence-shell STO exponent by each of the two *scale factors* given below in turn.

Hydrogen	1s	1s'
H	1.20	1.15

Atom	1s	2sp	2sp'	3sp	3sp'
B	1.00	1.03	1.12	-	-
C	1.00	1.00	1.04	-	-
N	1.00	0.99	0.98	-	-
O	1.00	0.99	0.98	-	-
F	1.00	1.00	1.00	-	-
P	1.00	1.00	-	0.98	1.02
S	1.00	1.00	-	0.98	1.01
Cl	1.00	1.00	-	1.00	1.01

In the case of the LP-N1G (Local potential optimised) bases no scale factors are used i.e. the scale factor is 1 for both inner and outer orbitals.

The N-31G* and N-31G** basis sets have polarisation functions added with the following standard polarization *exponents*.

Atom	Value
H	1.1
Li	0.2
Be	0.4
B	0.6
C-Ne	0.8

The standard polarization exponents for STO-NG* basis are:

Atom	Value
Na, Mg	0.09
Al-Cl	0.39

The maximum degree-of-contraction (N) permitted in `gnu80` is eight.

Just as the basis function here is formed from a linear combination of primitive functions; a shell, in general, is composed of primitive shells. A p-shell, for instance, is a set of three p-type basis functions with common exponents, contraction coefficients and scale factors. In a sp-shell there are four basis functions (s, and a p-shell) and four primitive functions in each primitive shell. In each primitive shell, the exponents and scale factors are the same in each of the four functions. However, the Contraction coefficients for the s-type function are usually be different than for the three p-type functions.

To illustrate further the features described above, consider the series of basis sets STO-3G, 4-31G and 6-31G* for the carbon atom. The STO-3G basis on any first row atom consists of only 2 shells. One shell is an s-shell consisting of a set of 3 primitive gaussian functions least-squares fitted to a Slater Type Orbital (STO) 1s orbital with an appropriate scale-factor; for the carbon atom, the ls orbital scale-factor is 5.67. The other sp-shell is a least-squares fit of 3 gaussians to Slater 2s and 2p orbitals with the constraint that the s and p functions have equal exponents.

For carbon atom the 2sp-shell has a scale factor of 1.72. The 4-31G basis on a first row atom has three shells. One shell is a contraction of four primitive s-type gaussians with a scale factor of unity. The second shell is a combination of three primitive sp-shells, again with a scale factor of unity. The third shell consists of a single sp-function with a scale factor of 1.04. The 6-31G* basis has four shells and differs qualitatively from the 4-31G basis in only two respects. First, the innermost shell is a contraction of six gaussian instead of four; and secondly the last shell is a d-shell.

External basis sets may be read in to `gnu80` by setting IOP(5)=7 and ICP(6)=0 or by specifying **GEN** (for general basis) in the command record. Further specification of the desired external basis can be achieved by using IOP(8) (refer to Appendix B for a listing of the overlay 3 OPTIONs). All external basis input is handled by routine GBASIS. A schematic illustration of this input is given below.

- Number of Gaussian functions (degree of contraction) in each shell (80I1).
- Center assignments (zero-center terminates reading of input). 35I2 format.
- Shell descriptor record (A4,A6,A4,I2,F12.6)
  Field 1 (A4) ' STO' use STO routines.
  ' ' read in records defining functions.
  '****' step to next set of centers.

Field 2 (A6) name used in printing and if field 1 = ' STO', this field defines the routine from which the STO-NG functions are taken.
Field 3 (A4) type of shell.
'   S' s-shell.
'   P' p-shell.
'   D' d-shell.
'   F' f-shell.
'   SP' sp-shell.
'   SPD' spd-shell.
Field 4 (I2) number of Gaussians (degree of contraction) for the current shell.
Field 5 (F12.6) scale-factor for current shell.
Primitive Gaussian record (4E20.10)
Field 1 exponent.
Field 2 s-coefficient.
Field 3 p-coefficient.
Field 4 d-coefficient.
In the case of an f-shell, the f-coefficient is taken from field one. the number of primitive function records read in is determined by the degree of contraction specified on the preceeding shell descriptor record.

The basic unit of information that routine GBASIS deals with is the *shell definition block*. A shell definition block, together with IOP(8), contains all necessary information to define a shell of functions. It consists of a *shell descriptor record* and from one to eight *primitive Gaussian records*. The shell descriptor record has 5 fields with

```
FORMAT(4X, A6, A4, I2, F10.4)
```
or
```
FORMAT(A4, A6, A4, I2, F10.4)
```

If the first four characters of the record are ' STO' , then the data may be abbreviated and the routines provided will generate the exponents and coefficients for 1s 2sp 3sp 3d and 4sp shells, this option will be described later.

If the first four characters in the record are blank, the rest of the record contains IORB, ITYPE, NGAUSS and SC.

IORB is simply a four-character label used to identify this shell, e.g. ‘ 2SPI’ for the larger-exponent (“inner”) shell of a split-valence pair of shells.

ITYPE defines the shell type and shell constraint and may take on the values ‘ bbbS’ ‘ bbbP’ and ‘ bbSP’ (b denotes a blank space) denoting, respectively, an s-shell, p-shell, or an sp-shell.

NGAUSS specifies the number of primitive Gaussian shells in the contraction for the shell being defined and must be between 1 and 8.

SC is the shell scale-factor

Subsequent records define the exponents α_k and contraction coefficients, $d_{k\mu}$ for the NGAUSS primitives composing the shell.

The FORMAT of these records is (4E20,10).

field 1 contains the exponent α_k

fields 2 contains the coefficient of the s-type function in this shell,

field 3 contains the coefficient of the p-type function if this shell requires one (i.e. is an sp shell)

field 4 contains the coefficient of the d-type function in this shell if one is required.

NGAUSS such records are required.

One customarily places at least one, and quite often several shells on any given nuclear centre. A centre definition block consists of a centre identifier record, and one shell definition block for each shell desired on the centre(s) specified, and is terminated by a record with **** in columns 1-4. The centre identifier record has 35I2 FORMAT and simply gives the number of the centres on which the basis functions are to be placed.

The decription of input to GBASIS is now complete except for the first record and the last record. The first record (80I1) contains the degree of contraction for each shell in the calculation. Centres are delimited by a contraction of 0 and the list is terminated by a 9. Overall input to GBASIS is terminated by a blank record.

Example

In this example, which is Example 16 of the input examples, a 6-31G** SCF calculation is performed on the water molecule. However, instead of using the internally stored basis, the basis set specification is provided as input (in the Basis Section). The Basis Section is called for by the presence of the command **GEN** in the command record. The command **6D** indicates that if d-type functions are included in the basis, then the six Cartesian Gaussian functions should be used. The Title and Geometry Sections are as described earlier, and the remaining records constitute the Basis Section.

```
 1. #N RHF/GEN, 6D

 2. (blank record)

 3. gnu80 EXAMPLE JOB 16

 4. WATER MOLECULE: RHF/6-31G** READ IN, EXPERIMENTAL GEOMETRY

 5. (blank record)

 6. 0 1

 7. O

 8. H 1 0.957

 9. H 1 0.957 2 104.5

10. (blank record)

11. 6311031103119

12. 01

13. 1S S 6 1.00

14. 5484.67166 0.00183107443

15. 825.234946 0.0139501722

16. 188.046958 0.0684450781

17. 52.9645 0.232714336
```

```
18. 16.8975704 0.4701928980

19. 5.79963534 0.358520853

20. 2SPI SP 3 0.99

21. 15.8551334 -0.110777549 0.0708742682

22. 3.67302682 -0.148026262 0.339752839

23. 1.03434522 1.13076701 0.727158577

24. 2SPO SP 1 0.98

25. 0.281138924 1.0 1.0

26. D D 1 1.

27. 0.8 1.

28. ****

29. 0203

30. 1S S 3 1.20

31. 13.007734 0.03349460434

32. 1.96207942 0.2347269535

33. 0.444528953 0.813757326

34. 1S S 1 1.15

35. 0.121949156 1.0

36. P P 1 1.0

37. 1.1 0.0 1.0

38. ****

39. (blank record)
```

Record 11 specifies the number of shells on each of the three centres, and the number of primitives in each shell. Column one contains a 6, indicating that the first shell on centre 1 (Oxygen) is a contraction of six primitive Gaussians. The 3 in column two indicates that the second shell on centre 1 is a contraction of three primitives. The next two columns contain 1's; the next two shells on this centre each contain only one primitive shell. The 0 in column five terminates the list of shells for this centre. Thus, the Oxygen atom (centre 1) has four shells centred on it. These shells are contracted shells with degrees of contraction of 6, 3, 1 and 1, respectively. The specification for centre two is provided in column 5-7. This centre has three shells, with degrees of contraction of 3, 1 and 1, respectively. Centre three has the same specification as centre two. Finally, the 9 in column 13 terminates the list.

Records 12—28 and 29—39 constitute two centre definition blocks. Consider the second of these two blocks, records 29—39. The first line in the block provides a list of the centres to which the shells are to be attached. In this case centres two and three are specified. Thus, the two hydrogen atoms will each have the same types of basis functions. records 30—33 provide the first of three shell definition blocks for the hydrogens. The shell descriptor record, record 30, contains four fields. The first field is just a string used for output. The second field is ' bbbS', indicating that the shell being defined is an s-shell. The next field specifies the degree of contraction for the shell, in the case three. The final field contains the scale factor for the shell, 1.20. Since the degree of contraction here is three, the next three lines must specify the exponents and contraction coefficients for the three primitives. In each of records 31—33, the first field contains the gaussian exponent, and the second field provides the contraction coefficient.

The next shell definition block, records 34 and 35, illustrates nothing new. It merely specifies an s-shell with scale factor 1.15 and degree of concentration 1. Records 26-37 specify a p-shell with degree of contraction 1 and a scale factor of 1.0. The gaussian exponent is 1.1, the s-coefficient field is zero (for a p-shell) and the p-coefficient is 1.0. record 38 terminates the centre definition block, and record 39 (a blank line) terminates the Basis Input Section.

The specification of an sp-shell is illustrated by records 24 and 25. The second of the shell descriptor record (record 24) is ' bbSP' , specifying an sp-shell. The degree of contraction here is one, and record 25 provides the exponent, s-coefficient, and p-coefficient for the single primitive shell. Records 20—23 describe an sp-shell with degree of contraction three. Finally, the specification of a d-shell is shown in records 26 and 27. The second field of record 26 is ‘ bbbD’ , specifying a d-shell. On the following line, the exponent is provided, the s- and p-coefficients are zero, and the d-coefficient is 1.0

Example 16, Output summary

```
  25 BASIS FUNCTIONS       42 PRIMITIVE GAUSSIANS
   5 ALPHA ELECTRONS        5 BETA ELECTRONS
     NUCLEAR REPULSION ENERGY    9.1969310957 HARTREES
 RAFFENETTI 1 INTEGRAL FORMAT.
 TWO-ELECTRON INTEGRAL SYMMETRY IS TURNED ON.
     5235 INTEGRALS PRODUCED FOR A TOTAL OF      5235
     9148 INTEGRALS PRODUCED FOR A TOTAL OF     14383
 PROJECTED HUCKEL GUESS.
 INITIAL GUESS ORBITAL SYMMETRIES.
       OCCUPIED: (A1) (A1) (B2) (A1) (B1)
       VIRTUAL:  (B2) (A1)
 RHF CLOSED SHELL SCF.
 REQUESTED CONVERGENCE ON DENSITY MATRIX=  0.5000d-04 WITHIN  32 CYCLES.

SCF DONE:  E(RHF) =  -76.0231735680     A.U. AFTER    7 CYCLES
            CONVG  =    0.2557d-04            -V/T =   2.002801
 ORBITAL SYMMETRIES.
       OCCUPIED: (A1) (A1) (B2) (A1) (B1)
       VIRTUAL:  (A1) (B2) (B2) (A1) (A1) (B1) (B2) (A1) (A2) (A1)
                 (B1) (A1) (B2) (B2) (A2) (B1) (A1) (A1) (B2) (A1)
  THE ELECTRONIC STATE IS 1-A1.
                          1         2         3         4         5
                        (A1)      (A1)      (B2)      (A1)      (B1)
     EIGENVALUES --   -20.56041  -1.34040  -0.70352  -0.56872  -0.49704
          CONDENSED TO ATOMS (ALL ELECTRONS)
              1          2          3
  1   O   8.056815   0.308390   0.308390
  2   H   0.308390   0.379646  -0.024833
  3   H   0.308390  -0.024833   0.379646
          TOTAL ATOMIC CHARGES.
              1
  1   O   8.673594
  2   H   0.663203
  3   H   0.663203
 DIPOLE MOMENT (DEBYE): X= 0.0000   Y= 0.0000   Z= 2.1846   TOTAL= 2.1846
```

3.7 General Basis Sets: STO type

The explicit input of an STO-NG basis is only necessary in order to be able to carry out a calculation on a molecule using STO-NG basis functions which are not provided internally or non-minimal STO-NG bases.

As outlined above the shell descriptor record has 5 fields with

```
FORMAT(A4, A6, A4, I2, F10.4)
```

If the first item supplied as data is ‘ bSTO’ (b is a blank) then (for shells up to 4sp) the input of non-standard basis data is assumed to be that of the Gaussian expansion of an STO and does not require the exponents and coefficients to be supplied explicitly, although they may be, if required.

Thus in this case only the string ‘ bSTO’ is supplied and the rest of this record contains *for each shell*: contains IORB, ITYPE, NGAUSS and SC.

IORB is now interpreted and must contain one or other of ‘ bbnS’ , ‘ bbnP’ , ‘ bnSP’ , ‘ bbnD’ where N is the principle quantum number of the shell.

ITYPE defines the shell type and shell constraint and may take on the values ‘ bbbS’ ‘ bbbP’ ‘ bbbD’ and ‘ bbSP’ (b denotes a blank space) denoting, respectively, an s-shell, p-shell, d-shell or an sp-shell.

NGAUSS specifies the number of primitive Gaussian shells in the contraction for the shell being defined and must be between 1 and 8. Here it is the N of STO-NG

SC is the shell scale-factor; in this case the STO shell exponent.

An Example: Zinc Dimethyl

The rules in this section are best understood by an example, here is the STO data for the Zinc dimethyl molecule:

```
# HF/GEN 5D

Zinc dimethyl HF/STO-3G (input Basis)

0 1
ZN
C1 ZN RZNC
H1 C1 RCH ZN AN
H2 C1 RCH ZN AN H1 120.0
H3 C1 RCH ZN AN H1 -120.0
X  ZN 1.0 C1 90.0 H1 0.0
C2 ZN RZNC X 90.0 C1 180.0
H4 C2 RCH ZN AN H1 180.0
H5 C2 RCH ZN AN H2 180.0
H6 C2 RCH ZN AN H3 180.0

RZNC=1.682
RCH=1.09
AN=108.0

3333303330303030330303039
 1
 STO   1S   S 3   29.43
 STO  2SP  SP 3   12.52
 STO  3SP  SP 3   5.19
 STO   3D   D 3   4.90
 STO  4SP  SP 3   1.90
****
 2 6
 STO   1S   S 3   5.67
 STO  2SP  SP 3   1.72
****
 3 4 5 7 8 9
 STO    1S   S 3   1.24
****
(blank record)
```

There are a few points to be made about this dataset:

1. Even though the STO bases are provided internally for the Zinc, Carbon and Hydrogen atoms, the use of **GEN** over-rides their use and they must be provided explicitly for this example.

2. The data is slightly redundant in the sense that the contraction lengths are given *twice*; once in the initial record of degrees of contraction (since the system cannot know that some STO records are to follow) and once on each STO record.

3. The N of STO-NG is taken from the second input item (1S, 4SP etc.)

4. The assumed numbering of the atoms, used both in the degree-of- contraction record and on the record preceeding each set of shells, *omits the dummy atoms*, thus the atoms in Zinc dimethyl are numbered 1 to 9, omitting the dummy X, used to avoid an 180 bond angle.

3.8 Non-Standard Optimization Input

In general, for a "Berny" optimization or a Murtaugh-Sargent optimization, the more accurate the initial guess of the second derivative matrix, the fewer steps that will be needed to reach the stationary point. Because of this feature, provision has been made for users to *provide* diagonal second derivative or (for Berny optimization) to request numerical computation (by finite difference) of elements of the force constant matrix.

A ' 1' following a variable value in program input section 4 indicates that the number following on the same record should be used as the initial diagonal second derivative for that variable. The number must be in atomic units (Hartree-Bohr-radian). To convert from spectroscopic units (mdynes-Angstroms-radian) multiply by 0.064229 for stretches, 0.22937 for bends and 0.121376 for strech-bond interactions.

A ' 2' following a variable value instruct the program to compute that diagonal second derivative before proceding with the optimization.

A ‘ 3’ following a variable value causes the diagonal and off- diagonal second derivatives for that variable to be estimated by finite differences.

3.9 RESTARTing a gnu80 Run

Usually, when a gnu80 job exits (either normally or abnormally), it leaves potentially useful data on the disk in the form of read-write files, two-electron integrals, or post-SCF buckets.

The **RESTART** feature allows a subsequent gnu80 job to recover and use these data, provided they have been placed in non-scratch files. In the normal course of events, gnu80 uses scratch files for both electron-repulsion integral storage and general file use. It is possible, by the use of *file control records* to make these files permanent. There are two files, the “integral” file (for repulsion integrals) and the “read-write” file (for general data storage, inter-link communication etc.etc.).

The files are made permanent by placing one or two file control records **before** the Job Control Record in a gnu80 job input. They both have the ‘ % ’ symbol in the first column of the record and have the form:

%int=filename1

%rwf=filename2

where “filename1” and “filename2” are two (different!) filenames which satisfy the naming conventions of the local operating system (this is not checked by gnu80, of course). Note that, unusually in gnu80, case **is** significant and both files will be generated with **names as entered** if this is significant to the operating system. This may well cause some surprises if a lower-case name was intended (in Unix, for example).

Both records have all blanks ignored (even *within* filenames, and examples are:

%int = repulsion.int

%RWF=dump.rwf

If you need blanks in your filenames, you must change the code in `SUBROUTINE LOCMND` to omit blank stripping.

If a job is to be restarted using these saved files, the restart job must have the same file control records before the Job Command Record. Sometimes, if files are to be updated during a sequence of restarted runs, **it is advisable to copy the saved files and restart with the copies.**

The ROUTE is stored on the read-write file, so a job can most easily be *re*-started, rather than submitted as a continuation. It may be restarted at any point up to and including the point at which it finished a complete Link. For example, if a job has actually run to completion it may be restarted anywhere in the original ROUTE.

Thus it must be indicated **where** the job is to be restarted. This is done with a record of the form:

#RESTART Lxxx(N)

The ‘ #’ must be in column 1, and ‘ Lxxx(N)’ indicates that the job is to restart at the N’th occurrence of link Lxxx. N is optional, and defaults to 1. In fact, ‘ Lxxx’ is optional also, and if not given, the job restarts at the link in which the previous job terminated. Since the original input file is lost, you must supply any input to the job that the remaining links will need.

Note that the RESTART Command starts immediately after the #, there is no blank as in the Job Command Record

It is also possible to supply a new route while restarting. In this case, the route information in the read-write file is updated while the rest of the data remains intact. this is accomplished with the **RESTART** command:

#RESTART L1
# new route goes here....

This causes execution to resume with Link1, which will read a new route. One must, of course, supply any data input required by the new route. This is a method of avoiding the re-computation of the repulsion integrals in a job related to an earlier one. Of course, the new ROUTE must explicitly omit the calculation which is being avoided (this usually means a **NONSTD** command).

Thus a job originally submitted as:

%int = repulsion.int
%RWF=dump.rwf
# HF/6-31g
(blank record)
Data Records

can be restarted at the SCF stage by:

%int = repulsion.int
%RWF=dump.rwf
#RESTART L501
(blank record)

always provided that the original job got as far as generating the data for Link 501.

3.9.1 Restarting; An Example

It often happens that it is difficult to get a Geometry Optimisation to *start* in the sense that it is difficult to get a converged set of MO coefficients for the initial geometry. Usually, since the changes in geometry are small during the geometry optimisation, once a set of converged MO coefficients has been obtained for one point these coefficients are good enough to ensure that, when used as a starting point for other geometries close by, things will go smoothly.

This is particularly galling if one has done a huge calculation of molecular integrals and the SCF does not converge. It is useful to be able to try the SCF again with some different convergence aids. Here is a pair of jobs which illustrate this practice.

The first job is an SCF calculation on the water molecule with a standard basis:

```
%INT=example.int
%RWF=example.rwf
# HF/3-21G,VSHIFT=500

  Test of the use of RESTART

0 1
O
H 1 0.956
H 1 0.956 2 104.5
```

In point of fact this run does go to completion with the convergence aid of “Level Shifter” set to 0.5 (`VSHIFT = 500`; actual value 500/1000) but such a job may have failed to converge.

Notice that the two files **must** be saved to use for a `RESTART`.

Now we may restart the job from where it finished with a different Level Shifter:

```
%INT=example.int
%RWF=example.rwf
%SAVE=H2O.SAVE
#RESTART L1
# NONSTD
5/9=3500,32=1/1,2;
99//99;
```

This time the Route has to be given explicitly by use of the `NONSTD` command; it was constructed by setting up a test Command Record anticipating success and therefore saving the converged MO coefficients:

```
# HF/3-21G,VSHIFT=3500,SAVE=MO
```

and editing the output Route from `gnu80`.
The meaning of the items are:

- The Leading `5` is the Overlay to be run (Overlay 5 is the SCF overlay)
- `9=3500` is just `VSHIFT=3500` in Overlay 5
- `32=1` is just `SAVE=MO` in Overlay 5
- `1,2` are the Links to be run in Overlay 5 (`L501, L502`

This job, when successfully run will generate a converged set of MO coefficients and save them for future use in an optimisation, say, on file `H2O.SAVE` suitable for use in a job like:

```
%GUESS=H2O.SAVE
# HF/3-21G,VSHIFT=500,GUESS=READ,OPT

    Re-using the Converged MO coefficients

0 1
O
H1 O ROH
H2 O ROH H1 104.5

ROH=0.956
```

None of the working files are saved, again anticipating success.

3.9.2 Portability of RESTART Files

For obvious reasons of speed of data transfer the two main gnu80 files (RWF and INT) are binary files and therefore not portable between machines of different types. However the SAVE and GUESS files are written and read as FORMATTED files and so can be moved from machine to machine at will.

3.9.3 Disaster Recovery

There is a utility program which will read the standard ("printed") output of gnu80 and generate a SAVE file from the printed MO coefficients (assuming that these occur in the standard output). If you have output from a job which ran but generated no SAVE file and you wish to RESTART a similar job, this can be done.

Error Messages

If a Link is specified on the **RESTART** record which is not in the original ROUTE, an error message is given and the job terminates.

Appendix A

List of links in gnu80

The following is a list of the **Links** of `gnu80` and a brief summary of their function.

MAIN initialization, controls overlaying.

L001 reads route, initializes disc files, fills error-function interpolation table.

L101 Reads:

1. Title
2. Z-matrix
3. Variables (if any)
4. Constants (if any)

L102 controls “Fletcher-Powell” optimization.

L103 controls “Berny” optimization.

L105 controls “Murtaugh-Sargent” optimization.

L202 calculates coordinates from Z-matrix and determines:

1. Stoichiometry
2. Framework group

3. Symmetry Information.
4. Rotates molecule to standard (centre of charge) orientation.

L301 Generates basis set information.

L302 Computes overlap, kinetic, and potential integrals.

L303 Computes x-, y- and z-dipole integrals.

L305 Setup for pseudo-potential integrals; obselete, it is now a dummy and its removal is overdue.

L306 Computes pseudo-potential integrals.

L310 Primitive two-electron integral program (spdf). for testing purposes only.

L311 sp two-electron integral program.

L314 (sp)d two-electron integral program.

L401 Generates initial guess at density matrix.

L501 RHF closed shell SCF.

L502 UHF open shell SCF.

L503 Direct Minimization SCF (does RHF/UHF, real/complex).

L505 Restricted open-shell SCF program.

L601 Mulliken population analysis; Fermi contact analysis for open shell systems; computes dipole moment.

L602 Provides output for interfacing with the **RPAC** suite of programs for Random-Phase Approximation Calculations of molecular properties.

L701 Calculates one-electron integral first derivatives.

L702 Calculates two-electron integral first derivatives for sp functions.

L703 Calculates two-electron integral first derivatives for spd functions.

L705 Calculates pseudo-potential first derivative integrals for sp bases.

L716 Converts forces to internal coordinate forces and communicates with optimization control programs.

L801 Setup program for transformation of two-electron integrals; Generates molecular orbital coefficient matrix and eigenvalues removing the orbitals which are not used in the correlation study.

L802 RHF closed-shell transformation of two-electron integrals.

L803 UHF open-shell transformation of two-electron integrals.

L901 Computes anti-symmetrized two-electron integrals; computes MP2 energy and Moller-Plesset first-order wave-function.

L909 Initialization for CID and higher-order energy perturbation calculations.

L910-L912 Carry out higher-order perturbation calculations, or one CID iteration.

L913 Calculates various energies. in the case of a CID calculation, L913 tests for convergence, and if necessary returns to L910 for the start of the next iteration.

L9999 Terminates the run normally.

Appendix B

Link Descriptions and Options

B.1 OVERLAY 1

Gaussian system input and optimization control In the `gnu80` system, OVERLAY 1 contains those programs which read in geometry and optimization input and those which control optimization calculations. Currently, the following Links are implemented:

Link 101 Basic input. this Link reads in the title, Z-matrix, variables, and constants sections of the input.

Link 102 Fletcher-Powell optimization program. this Link implements the algorithm of Fletcher and Powell as modified by Binkley and Pople. It, along with the interface code present in other Links, is capable of driving geometry optimizations. Derivative information is gained by numerical differentiation of the energy with respect to the geometrical variables.

Link 103 Gradient optimization program. This program is used in conjunction with those Links which produce analytical energy derivatives to perform geometry optimizations.

Link 105 Murtaugh-Sargent optimization program. This program also uses analytically determined first derivatives to optimize geometry with respect to energy; in general, it is not as efficient as Link 103, it has a more stable algorithm and is less prone to aimless wandering on the potential surface. Note that most of the options for Link 103 are only examined in the first call to the Link in any given run.

OPTIONS for OVERLAY 1

IOP(5) L103 mode of optimization
0 find local minimum
1 find a saddle point
N find a stationary point on the energy surface with N negative eigenvalues of the 2nd derivative matrix

IOP(6) L103 and L105 maximum number of steps **(OPTCYC=N)**
0 `NSTEP = MIN(20,NVAR+10)`
N `NSTEP=N`

IOP(7) L103 and L105 convergence on the first derivative and estimated displacement for the optimization:
(RMS first derivative) < `CONFV`,
(RMS estimated displacement) < `CONVX=4*CONFV`
0 `CONFV = 0.0003` Hartree/bohr or radian
N `CONFV = 0.001 / N`

IOP(8) L103 maximum step size allowed during optimization
0 `DXMAXT = 0.2` bohr or radian
N `DMAXT = 0.01 * N`

IOP(10) L103 input of initial second derivative matrix all values must be in atomic units (Hartree, bohr, and radians).
0 NO
1 `READ ((FC(I,J),J=1,I),I=1,NVAR) (8F10.6)`
2 `READ I,J, FC(I,J) (5I3,F20.0)` end with a blank record.
3 read from guess file

IOP(12) L103 optimization control parameters
0 use default values
1 read in new values for all parameters (see `INITBS`)

IOP(14) L103 minimum RMS force for which a linear search will be attempted
0 `FSWTCH = 0.01` Hartrees/bohr or /radian
N `FSWTCH = 0.001 * N`

IOP(15) L103 abort if derivatives too large
This has been disabled due to the fact taht large detivatives do genuinely appear in molecules of high symmetry (when one variable may change much geometry) and molecules containing heavy atoms. The Original usage is given below:
0 `FMAXT = 1.0` Hartree / bohr or radian
N `FMAXT = 0.1 * N`

IOP(16) L103 maximum allowable magnitude of the eigenvalues of the second derivative matrix. If the limit is exceeded, the size of the eigenvalue is reduce to the maximum, and processing continues.
0 `EIGMAX = 25.0` Hartree / bohr**2 or radian**2
N `EIGMAX = 0.1 * N`

IOP(17) L103 minimum allowable magnitude of the eigenvalues of the second derivative matrix. similar to IOP(16)
0 `EIGMIN = 0.0001`
N `EIGMIN = 1.0 / N`

IOP(18) L103 star only option **(OPT=STARONLY)**
0 proceed normally
1 second derivatives will be computed as directed on the variable definition records. NO optimization will occur.

IOP(19) L103 skip linear search.
0 NO
1 YES.

IOP(20) L101 input units **(UNITS=)**
0 angstroms degrees
1 bohrs degrees
2 angstroms radians

3 bohrs radians

IOP(29) L101 Specification of nuclear centers
Note that this option is usually set in conjuntion with IOP(29) in Link 202.
0 by Z-matrix
1 by direct coordinate input

IOP(30) L101 Nuclear Charges
0 nuclear charge equals atomic number
1 read in center name or number `I`, read in charge `CHG` (floating point) nuclear charge for `I`-th nucleus is replaced by `CHG`. recordset must be ended by blank record.

IOP(32) L103 writing of second derivatives to the punch unit (GUESS file) at the conclusion of the optimization.
0 NO
1 YES

IOP(33) L101 L102 L103 debug print
0 off
1 on

IOP(34) L101 L102 L103 debug + dump print
0 off
1 on

B.2 OVERLAY 2 (Link 202 only)

Procedure to determine the coordinates, given the Z-matrix, and to analyze the molecular symmetry, if requested. This Link receives the

Z-matrix or coordinates from the RW-files, and determines the framework group of the molecule and produces the standard (3 by 3) transformation matrices. When input consists of a Z-matrix, routine `ZTOC` is called upon to obtain the coordinates.

B.2.1 OPTIONS for OVERLAY 2

IOP(9) printing of distance matrix.
0 print distance matrix (only if there are more than two atoms in the molecule).
1 do not print the distance matrix.

IOP(10) Tetrahedral angle fixing.
0 angles within 0.001 degree of 109.471 will be set to `acos(-1/3)`.
1 do not test for such angles.

IOP(11) printing of Z-matrix and resultant coordinates.
0 print.
1 do not print.

IOP(12) Crowding abort control
0 if two atoms are less than 0.5 angstroms apart, abort the run.
1 do not abort the run for small interatomic distances.

IOP(15) Symmetry control.
0 leave symmetry in whatever state it is presently in.
1 unconditionally turn symmetry off. Note that `SYMM` is still called, and will determine the framework group. However, the molecule is not oriented.

IOP(29) direct coordinate input Note that this option is usually set in conjunction with IOP(29) in L101.
0 coordinates were input via the Z-matrix.
1 coordinates were input directly.

B.3 OVERLAY 3

Gaussian system integral package.
Overlay 3 consists of the necessary programs to evaluate the one- and two-electron integrals required for an SCF calculation.
This package consists of the following Links:

L301 Constructs basis set, either through internally stored data or through input and performs other initialization chores for the integral programs.

L302 Calculates the Overlap **(S)**, Kinetic Energy **(T)** and Core-Hamiltonian **(H)** one-electron integrals.

L303 Calculates the x-, y- and z-dipole integrals (one-electron).

L305 Formula generator for the pseudo-potential program. OBSELETE

L306 Evaluates one-electron pseudo-potential integrals.

L310 Evaluates s-, p-, d- and f-type two-electron integrals by use of general formula; used only for testing.

L311 Evaluates two-electron integrals for those shell combinations that contain s- and p-functions.

L314 Evaluates s-, p-, d- and f-type two-electron integrals using the method of Rys polynomials.

Note that there is some overlap in functionality between the two-electron integral programs listed above. In standard calculations, one would normally use Links 301, 302, 303, 311 and 314 to obtain all the necessary integrals for basis functions up (and including) d-functions. If d-functions are not present in the route, L314 can be omitted.

B.3.1 OPTIONS for OVERLAY 3

IOP(5) Type of basis set
0 Minimal STO-2G to STO-6G

1 Extended 4-31G,5-31G,6-31G
2 Minimal STO-NG (valence functions only)
3 Extended LP-N1G (valence basis for coreless Hartree-Fock pseudopotentials)
The Los Alamos Split Valence Basis is equivalent to LP-31G. 4 Extended 6-311G (UMP2 frozen core optimized) basis.
and ** effected by IOP(21) as usual.
use IOP(8) to select 5d/6d.
5 Split Valence N-21G (or NN-21G) basis for first or second row atoms. (various implementations may omit second row atoms.) see IOP(6) for determination of the number of Gaussians in the inner shell.
7 GENERAL; see routine `GBASIS` for input instructions.

IOP(6) Number of Gaussian functions
N STO-NG, N-31G, LP-N1G, STO-NG-valence, N-21G.
Note: if IOP(5)=3 and IOP(6)=8; LP-31G for Li, Be, B, Na, Mg, Al LP-41G for other row one and two atoms.
Default Options:
IOP(6)=0
if IOP(5)=0 N=3 STO-3G
if IOP(5)=1 N=4 4-31G
if IOP(5)=2 N=3 STO-3G (valence)
if IOP(5)=3 N=3
if IOP(5)=5 N=3
When IOP(5)=7 (general bases), this option is used to control where the basis is taken from:
0 read general basis from the input stream.
1 read the general basis from the rw-files and merge with the coordinates in blank common to produce the current basis.
This option is useful when doing general basis geometry optimizations.

IOP(7) Polarization type.
0 none.
1 Add a set of second-order Gaussians **(6D)** to first row n-31G

atoms (N-31G* basis), or,
if IOP(5)=0, add a set of d-functions **(5D)** to second row STO-NG atoms (STO-NG* basis). or,
if IOP(5)=3, add a set of 5d to rows 1 and 2
2 Does the same as IOP(7)=1, but additionally adds a set of p-functions to N-31G hydrogen atoms (N-31G** basis).

IOP(8) Selection of second-order Gaussians/true d-functions.
0 selection determined by the basis:
N-31G* 6d
N-31G** 6d
N-21G* 5d
STO-NG* 5d
LP-N1G* 5d
LP-N1G** 5d
General Basis 5d.
1 use 5d throughout. (**5d**)
2 use 6d throughout. (**6d**)

IOP(9) selection of third-order/true f-functions.
0 reserved for when f-functions are part of standard bases.
1 use 7f throughout.
2 use 10f throughout.
Note f-functions only partly implemented.

IOP(10) Modification of internally stored bases.
0 none.
1 Read in replacement scale-factors. Standard scale-factors are listed below.
2 Read in replacement polarization exponents for N-31G*, N-31G** and STO-NG* bases. standard values are listed after the standard scale-factors.
3 combination of 1 and 2 above (ie. read in both scale-factors and polarization exponents).
For general basis runs, this option has the following definitions:
0 Not a scale-factor run.

1 Continuation entry for scale-factor optimization.
2 Initialization entry for a scale-factor optimization run.
See routine `GBASIS` for further details on scale-factor optimizations.

IOP(11) Control of two-electron integral storage format.
0 Standard integral format is used.
1 Raffenetti 1 integral format is used. Can only be used with the closed shell SCF.
2 Raffenetti 2 integral format. Suitable for use with the open shell (UHF) SCF.
3 Raffenetti 3 integral format. Suitable for use with open shell RHF SCF and the post-SCF procedures.

IOP(16) Check for pseudopotential run. See IOP(17) through IOP(19) for more details.
0 NO
1 YES
NOTE IOP(17)-IOP(19) apply only if IOP(16)=1

IOP(17) Specification of pseudopotentials
0 Use internally stored 'Coreless Hartree-Fock'
7 Read in from input stream (see `PINPUT` for details)

IOP(18) Printing of pseudopotentials
0 Print only when these are read
1 Print
2 Do not print

IOP(19) Specification of substitution potential type
0 Do not use any substitution potentials
N Replace the standard potential of this run (eg.CHF), with a substitution potential of type N wherever such a substitution potential exists.

IOP(23) Definition of two-electron integral scale factor. (for a discussion of how two-electron integrals are stored, see the program documentation).
0 default, 10**8.
N (10**8)*(10**N).

IOP(24) Printing of Gaussian function table.
0 Table is printed only if non-standard features are used.
1 Print table.
2 Do not print table.
10 Print out the basis in a form suitable for GEN input

IOP(25) Number of last two-electron integral Link.

IOP(26) Test option **(TEST)**
0 proceed as normal
1 print Gaussian function table and abort job after L301

IOP(27) Handling of small two-electron integrals.
0 Discard integrals with magnitude less than 10**-6.
N Discard integrals with magnitude less than 10**-N.
Beware of underflow when N is made large.

IOP(28) Polarization option
0 NO special features invoked.
10 Compute all two-electron integrals in L310
Note: L310 should be included in the route by the use of the **NONSTD** command.
Note: option 25 should be set to 10.
14 Compute all two-electron integrals in L314.

IOP(29) Rotation of coordinates.
0 Coordinates are not rotated.
1 Read 1 record in `3E20.10` format giving the three Euler angles ϕ, θ and χ.

IOP(30) Control of two-electron integral symmetry.
0 Two-electron integral symmetry is turned off.
1 Two-electron integral symmetry is turned on. Note, however, the `SUBROUTINE SET2E` will interrogate `ILSW` to see if the symmetry rw-files exist. If they don't, symmetry has been turned off elsewhere, and `SET2E` will also turn it off here.

IOP(32) Punching of `COMMON/B/` in compressed form.
0 NO punching.
1 punch `COMMON/B/` in compressed form. The data is written to the GUESS file for possible use by the initial guess in a subsequent job.

IOP(33) Integral package printing.
0 NO integrals are printed.
1 Print one-electron integrals.
3 Print two-electron integrals in standard format.
4 Print two-electron integrals in debug format.
5 Combination of 1 and 3.
6 Combination of 1 and 4.

IOP(34) Dump option.
0 NO dump.
1 Control words printed (as usual).
2 Additionally, `COMMON/B/` is dumped at the beginning of each integral Link.
3 Additionally, the integrals are printed (standard format).

B.4 OVERLAY 4 (Link 401 only)

This is a program which produces an initial guess to the solution of the SCF equations. This guess is in the form of molecular orbital coefficients and/or density matrices which are stored on the appropriate read-write files. The steepest descents procedure (Link 503) requires

MO coefficients as an initial guess, while the classical SCF procedures (Links 501 and 502) require density matrices. Since a density matrix can be produced from the MO coefficients, but not vice versa, the former is a more constraining requirement. There are several ways in which this guess may be produced. One easy way is to diagonalize the core hamiltonian. In general, this is not a very good guess, but it is applicable to any basis set, and is available as an option.

Another type of guess is called the Hückel guess, which is modeled after extended Huckel MO theory. Essentially, the initial guess is formed from internally stored constants (for more details see `SUBROUTINE HUCKEL`). These constants were determined from studies on internal minimal and split valence basis sets (STO-3G, 4-31G, 6-31G), so the use of this type of guess with bases other than these is not recommended.

This Huckel guess can be applied to other bases in the following way: the guess MO coefficients are formed from internal data as if there were an STO-3G basis set on the molecule. The guess MO vectors in the desired basis are then formed by choosing the vectors which give the best least-squares fit to those described in the STO-3G basis. Since this will usually produce fewer than N basis vectors,the MO coefficient matrix must be completed with orthonormal vectors of the proper symmetry if the full matrix is required (Link 503). This procedure is called a projected Hückel guess, and is applicable to any basis set.

A still better type of guess, usually, is to read the coefficient or density matrix from the input stream. If the matrix read in is for a basis other than the one used in the current run, the matrix can be projected (by a least-squares fit) into the desired basis. Since the projected MO vectors can be normalized and orthogonalized, and this is not possible for a projected density matrix, projection of MO coefficients usually produces a better guess than projection of the density matrix.

B.4.1 OPTIONS for OVERLAY 4

IOP(5) Type of guess.

0 Default. This gives a Huckel guess for minimal bases, or a projected huckel guess otherwise.

1 Read guess from GUESS file.

2 Guess from core Hamiltonian.
3 Blocked Huckel guess.
4 Projected Huckel guess.
5 Renormalize and orthogonalize the coefficients which are currently on the read-write files.

IOP(6) Forced projection when guess is read in
0 Do not force projection.
1 Force projected guess, even when bases are identical.
2 Suppress projection.

IOP(7) SCF constraints.
0 use `ILSW` to determine.
1 real RHF.
2 real UHF.
3 complex RHF.
4 complex UHF.
5 complex, but use `ILSW` to decide whether RHF/UHF.

IOP(8) Alteration of configuration.
0 Do not alter configuration.
1 Read in pairs of integers (2I3) indicating which pairs of MOs are to be interchanged. Pairs are read until a blank record is encountered.
Note:
If the configuration is altered on an open shell system, two sets of data as described above will be expected first for α, second for β.

IOP(10) Orbitals to mix with complex.
0 Mix the HOMO with the LUMO.
1 Read from records (2I3) pairs of integers indicating which pairs of orbitals are to be mixed. Reading is terminated by a blank record.
Note:

The same considerations for open shell systems which applied in IOP(8) apply here, also.

IOP(12) Off-diagonal scale factor for Hückel guess.
0 Default. (K/2=.875)
N (K/2=N*.4375)

IOP(16) Completion of coefficient matrix after projection.
0 Complete the coefficient matrix after projection.
1 Do not complete.

IOP(22) Type of basis set.
0 Use `ILSW` to determine.
1 Minimal basis.
2 Extended basis.
7 General basis.

IOP(23) five/six d, seven/ten f.
0 use `ILSW` to determine.
1 five d, seven f.
2 six d, seven f.
3 five d, ten f.
4 six d, ten f.

IOP(24) Polarization functions on hydrogen.
0 Use `ILSW` to determine.
1 NO polarization functions on hydrogen.
2 A set of p functions on each hydrogen.

IOP(25) Polarization functions on first row atoms.
0 Use `ILSW` to determine.
1 NO polarization functions.
2 A set of d functions on each first row atom.

IOP(26) Polarization functions on second row atoms.
0 Use `ILSW` to determine.
1 NO polarization functions.
2 A set of d functions on second row atoms.
Note that whenever `ILSW` is over-ridden, it is also over-written.

IOP(33) Printing of guess.
0 NO printing.
1 Print the MO coefficients.
2 Print everything.

IOP(34) Dump option. 0 NO dump.
1 Turn on all possible printing.

B.5 OVERLAY 5

Link 501 Perform Roothaan SCF procedure using the method of repeated diagonalizations.

Link 502 solution of the Pople-Nesbet equations by the method of repeated diagonalizations. J. Chem. Phys. 22, 571 (1954)

Link 505 Solves the Binkley, Pople and Dobosh equations for a spin-restricted open shell system. The wavefunction produced is not compatible with the `gnu80` post-SCF procedures (CI, MP2, MP3).

B.5.1 OPTIONS for OVERLAY 5 (501,502,505 but not 503)

IOP(5) L501,L502 location of input density matrix.
0 density matrix is taken from the rw-files.
1 the density matrix is read in via routine `BINRD`.

IOP(6) L501,L502 requested convergence on the density matrix.
0 iterations are performed until the RMS convergence on the density matrix is $<$ 10**(-5) or `MAXCYC` is reached.

N (1 < N < 8) requested convergence is 10**(-N).

IOP(6) L505 convergence on the density matrix.
0 5.0D-7 (Note that this is less than in the closed shell RHF program because here we are converging on three matrices.
N 1 < N < 8 final convergence=10**(-2*N).

IOP(7) L501,L502 maximum number of SCF iterations. **(SCFCYC=)**
0 up to 32 iterations will be performed.
N up to N iterations.

IOP(7) L505 maximum number of SCF cycles.
0 20 cycles. (Note that each cycle involves the formation of three Fock matrices; appropriate time should be allowed.
N 1 < N < 7 number of cycles is 2**(IOP(7)-1). a value of one will permit only a single cycle.
7 `MAXCYC`=64*IOP(22)+8*IOP(23)+IOP(24).

IOP(8) L501,L502 energy convergence.
0 convergence on the density matrix. see option 6 for details.
N (0 < N < 7) the SCF is assumed to have converged when the change in the energy is .le. 10**(-3-N).
7 input desired value via `INCRD`, see below.
Note that if this option is set, the density matrix criterion is not used at all.

IOP(11) L501,L502 extrapolation control.
0 both three-point and four-point extrapolation are performed when applicable.
1 three-point extrapolation is inhibited, but the program will still perform four-point extrapolation when possible.
2 both three-point and four-point extrapolation schemes are 'locked out' (ie. disabled).

IOP(12) L501,L502 entry mode.
0 normal entry mode, regular SCF is performed.
1 control is passed immediately to the punch/print code (IOP(32)). this is useful at the termination of an optimization run.

IOP(13) L501,L502,L505 action on convergence failure.
0 the run is terminated in error mode (via `LNK1E`) if the SCF fails to converge.
1 the run is allowed to continue, but the convergence failure bit in `ILSW` is set.

IOP(14) L501 UHF test option.
0 no.
1 YES, turn the current run into a UHF run at the end of this Link.

IOP(14) L502 control of annihilation of spin contaminants.
0 calculation is performed (provided of course that enough space exists in the rw-files).
1 calculation is bypassed.
2 calculation is performed, contingent on space, and the system rw-files for the appropriate density matrices are updated (useful if one wants a population analysis).

IOP(16) L505 control of use of convergence routine.
0 use convergence routine.
1 lock-out convergence routine.

IOP(32) L501,L505 punch (via `BINWT`) option. **Obselete**
0 NO punching is performed.
1 the molecular orbital coefficients are written to the guess file at the end of the job. These may provided as an initial guess to a subsequent job (see Link 401).
2 the MO coefficients and the density matrix are punched at the

end of the SCF.
3 the MO coefficients and the density matrix are punched at the end of each iteration of the SCF.

IOP(32) L502 whether to save the MO coefficients or density matrices on the guess file.
0 don't save.
1 save final MO coefficients.
2 save final density matrices.
3 save both.
4 save both each cycle of the SCF.

IOP(33) L501,L502,L505 print option.
0 only summary results are printed (with possible control from the 'no-print' option).
1 the eigenvalues and the MO coefficients are printed at the end of the SCF.
2 same as IOP(33)=1, but additionally the density matrix is printed.
3 same as IOP(33)=2, but at the end of each iteration.
4 same as IOP(33)=3, but all matrix transactions are printed (BEWARE: much output even on small molecules.)

IOP(34) L501,L502,L505 dump option.
Standard system defaults apply here.
Input via routine incrd: (L501,L502)
If IOP(8) = 7 or IOP(6)=8, program will read one record in free-field format to obtain the user supplied values for density matrix convergence and energy convergence.
This one record has two fields:
Field one: floating point density matrix convergence criterion
Field two: floating point energy convergence criterion.
The appropriate field is only used if the associated option is set to 7.

B.6 Link 503 SCFDM

Solution of the Pople-Nesbet equations by means of a direct minimization method involving a sequence of univariate searches.

B.6.1 OPTIONS for Link 503

IOP(6) convergence on density matrix
0 5.*10**(-5)
N 10**(-N)

IOP(7) maximum number of univariate searches
0 32
N N cycles.

IOP(8) selection of the procedure of direct minimization
0 steepest descent with search parameters default
1 steepest descent with search parameters read (see below)
2 classical SCF (Roothaan's method of repeated diagonalization
4 conjugate gradients with search parameters default
5 conjugate gradients with search parameters read (see below) the search parameters are: `MAX`: number of search points (I1) `MIN`: number of search points (I1) initial stepsize, `TAU` (G18.5) scaling factor for subsequent `TAU` (G20.5) and `Q` (G20.5)

IOP(9) switch to classical SCF after density matrix has achieved a certain convergency
0 NO
1 YES, criterion default: 10(**-3)
2 YES, criterion read in (format G16.10)

IOP(11) apply extrapolation procedures for classical SCF 0 four-point only
1 four-point only

2 none

IOP(12) null entry for final saving of data.
0 normal entry.
1 null entry (zero cycles). this is for saving final MO coefs in optimizations. see IOP(32).

IOP(14) reordering of the orbitals (maintaining continuity of the wavefunction along the search path)
0 on: Bessel criterion
1 on: stronger individual-overlap criterion
2 off

IOP(15) controls the auto-adjustment of `TAU` in `INTOPN`
0 done
1 `TAU` is kept fixed

IOP(16) inhibit performance of minimization of alternate wavefunction provided by second order procedures
0 NO
1 YES

IOP(17) condition the off-diagonal terms of the MO-Fock matrix:
-set to zero if `GABS(F(I,J)).LE.FUZZY`
-delete coupling terms between almost degenerate
`(DELTA E .LE. DEGEN)` MO vectors
0 `FUZZY`=1.d-10, `DEGEN`=2.d-5
1 fuzzy and degen read in (2d20.14)

IOP(18) cutoff criteria in symmetry determination of MOs.
-symmetry is determined if largest off-diagonal MO
Fock-matrix element `GABS(F(I,J)).GE.STHRS`
-elements `GABS(F(I,J)).LE.L8AN` are considered to be zero

0 `STHRS`=1.d-4, `SPAN`=5.d-7
1 `STHRS` and `SPAN` read in (2d20.14)

IOP(19) print `F(1),T`. (read one record with `START`, `END` in 2i2)
0 NO
1 YES

IOP(20) max-time exit (in order to dump for restart. see DOUBAR)
0 NO
1 YES
to obtain a max-time dump, proceed as follows: set this option to 1, the next Link to be performed should be L901 (`DOUBAR`), set IOP(15) there to 6.
Note: set IOP(14) on the integral route record (write integrals on tape). of course, tape 'c' has to be assigned
To restart from such a dump: the Link preceeding SCF should be `DOUBAR`. set the following options there: IOP(15)=7, IOP(19)=1 after that blank common, the integrals and the rwf are loaded

IOP(21) action if `OTEST` detects problems:
0 abort run via `LNK1E`.
1 continue run.

IOP(31) override print-save option
0 NO (use ILSW)
1 force print-save on
2 force print-save off

IOP(32) save the MO coefficients and/or the density matrix on the guess file.
0 none
1 MO coefficients only
2 density matrix only
3 both

IOP(33) printing
0 NO printing
1 print MO coefficients at end
2 print everything at end
3 print everything each cycle ... and at end

B.7 OVERLAY 6 (Link 601 only)

This program performs a Mulliken population analysis for a computed wave function.

B.7.1 OPTIONS for OVERLAY 6

IOP(5) open or closed shell.
0 use `ILSW` to determine.
1 forced open shell.
2 forced closed shell.
The remaining options are print/no-print options. if the value of the option is zero, the default value (given below) is assumed. if the option is set to 1, the information is printed, and if it is 2, the printing is suppressed.
0 default.
1 print.
2 do not print.

IOP(6) distance matrix. default: no-print.

IOP(7) molecular orbital coefficients. default: print.

IOP(8) density matrix. default: no-print.

IOP(9) full population analysis. default: print.

IOP(10) gross orbital charges. default: print.

IOP(11) gross orbital type charges. default: no-print.

IOP(12) condensed to atoms. default: print.

B.8 OVERLAY 7

OVERLAY 7 is concerned with calculation of first and second derivatives of the energy with respect to nuclear coordinates.

Link 701 calculates and uses one-electron integral derivatives to get first energy derivatives.

Link 702 calculates and uses two-electron integral (sp only) derivatives to get first energy derivatives.

Link 703 calculates and uses two-electron integral (spd) derivatives to get first energy derivatives.

Link 705 calculates and uses one-electron pseudopotential integrals.

Link 716 completes evaluation of energy derivatives and transforms results to internal coordinates.

B.8.1 OPTIONS for Link 701

IOP(33) print option.
0 NO printing.
1 print atomic derivative contributions at end.

IOP(34) dump option.
0 NO dumping.
1 usual system stuff.
2 dump derivative contributions from within shell loops.

B.8.2 OPTIONS for Link 702

IOP(18) establish critical cut-offs within shell loops.
0 use standard values.

N `VTOL`=10**(-IOP(18)-3)
Note: this is a 'use at own risk' option, and hence is not documented fully. briefly, setting this option may speed things up, but it can also sometimes give unpredictable results.

IOP(27) file initialization control.
1 read in previous derivative contributions from file `IRWFX` before computing anything.

IOP(28) skip option to defer integral evaluation to L703.
0 compute as normal.
1 do all gradient integrals in L703.

IOP(34) dump option. 0 NO dumping.
1 usual system stuff.
2 dump derivative contributions from within shell loops.

B.8.3 OPTIONS for Link 703

IOP(27) file initialization control.
1 the contributions computed in dphnix are added to previous information contained in read/write file IRWFX.

IOP(28) integral evaluation option.
0 compute as normal. (sp done in L702, spd done here.)
1 do all gradient integrals in L703.

IOP(33) print option.
0 NO printing.
1 print final contributions to `FXYZ`.

IOP(34) dump option.
0 NO dumping.
1 usual system stuff.
2 dump derivative contributions from within shell loops.
options for Link 716

IOP(7) use of internal coordinates
0 YES
1 NO

IOP(27) . does L702 read previous force information
0 NO
1 YES

IOP(30) use of symmetry in OVERLAY 7
0 use (subject to availability).
1 don't use

B.9 OVERLAY 8

The purpose of this OVERLAY is to do the transformation of the two-electron integrals from the AO to MO basis.

Link 801 this Link prepares the MO coefficients and the one-electron energies for the post-SCF programs as follows: a selected set of MOs (selected using IOP(10)) is written onto read/write file number ISPECT. Some quantities commonly used in post-SCF routines are evaluated and written on read/write file number INFORB.

Link 802 this Link does the 2-electron integral transformation for RHF systems.

Link 803 this Link does the 2-electron integral transformation for UHF systems.

B.9.1 OPTIONS for OVERLAY 8

IOP(5) RHF or UHF
0 read in from ilsw
1 RHF
2 UHF

IOP(6) specifies which single-bar integrals are to be computed.
0 (ia—jb)
1 (ia—jb), (ij—ab)
2 (ia—jb), (ij—ab), (ij—kl)
3 (ia—jb), (ij—ab), (ij—kl), (ij—ka)
4 (ia—jb), (ij—ab), (ij—kl), (ij—ka), (ia—bc)
5 (ia—jb), (ij—ab), (ij—kl), (ij—ka), (ia—bc), (ab—cd)
in terms of what can be done with what integrals:
0 MP2
2 MP3, CI

IOP(7) test SCF convergence flag
0 YES
1 NO

IOP(8) option to change the core available
0 `MDV` is set to default value.(see `DATA` statement)
1 read in `MDV` (i6 format)

IOP(9) use `TRCL80` always.(mainly a debugging option)
0 NO
1 YES.

IOP(10) window is selected as follows:
0 all molecular orbitals are taken
1 the core MOs are frozen
2 a record is read in (2i3) indicating the start and the end

IOP(30) the molecular orbitals outside the window are set to zero, thus simulating the window without changing anything else (for test purposes).
0 NO
1 YES

IOP(31) perform primitive post-SCF operations
0 none
1 CI

B.10 OVERLAY 9

Link 901 Conversion of the set of single-bar integrals provided by the AO-to-MO transformation routine to the packed set of double-bar integrals and a-coefficients. this program also calculates the second order Moller-Plesset perturbation energy `E(MP2)`.

Link 909 Iterative solution of the CI equations involving all single and double substitutions and Moller-Plesset perturbation theory at second and third orders. partitioned into 5 Links: `CIS1`,...,`CIS5` `CIS1` sets up the information needed for the matrix multiplication `W=V*A` , `CIS2` to `CIS4` then perform this matrix multiplication, and `CIS5` finally evaluates all sorts of correlation energies.

B.10.1 OPTIONS for OVERLAY 9

IOP(5) method
1 CID. CI with all double substitutions.
2 MP3. third order perturbation theory.

IOP(6) criteria for termination of the iteration
0 default convergence criterion and maxcycle
N (N=1,...,6) perform max. N cycles use default convergence criterion

7 read in maxcycles and convergence criterion (i2,d18.13)

IOP(7) update the energy in `COMMON/GEN/`
0 YES, with the correlation energy, ECI in CI and EUMP3 in MP3 calculations
1 YES, with EUMP3.
7 NO

IOP(18) iteration scheme:
Formation of DE in

$$A(S) = \frac{W(S)}{(DE - \Delta(S))}$$

i.e. in the formation of a new wave function.
0 use DE depending on the method used. (IOP(5)).
For method = 0 or 1 DE = W(0)/A0
for method greater than 1 DE = 0
Note that for perturbation methods (method=2,3,4,5) DE is not really needed since the wave function formed is never used.
1 W(0)/A0 always.
2 0. always.

IOP(19) inhibit extrapolation
0 NO
1 YES

IOP(20) `CUTOFF` for AO integrals
0 standard `CUTOFF` (10**(-6))
1 `CUTOFF` read in (format(G20.14))

IOP(25) print pair contribution and weight to correlation energy
0 NO
1 YES, at the end of CI

2 YES, at each cycle
3 YES, at one cycle given by input (i3)
4 YES, at first cycle and at end

IOP(26) normalization of the wavefunction
0 normalized to `A(0)=1`.
1 $\sum_s A(s)^2 = 1$ (all s)
Note: perturbation theoretical results are valid with NORM=0 only

IOP(28) printing of dominant configurations.
0 do not scan the 'A' vector at the end of CI.
1 scan the 'A' vector at the end of CI and print the dominant configurations.

IOP(30) calculation of the CI density matrix.
0 do not calculate the density matrix.
1 calculate the density matrix at the end of CI.

Appendix D

Modifications to gnu80

D.1 Warning

It must be said at the outset that any decision to make significant changes to `gnu80` should not be taken lightly. In common with all the GAUSSIAN series of programs, `gnu80` is **fragile**; it is:

- Fragile in *design*; the whole system has grown up empirically and has the "bottom up" structure of Baroque complexity associated with such developments.

- Fragile in *data communication*; changes in the code may have unpredictable effects in remote parts of the system.

- Fragile in *"data structures"* ; this point is treated in detail below in the discussion of `TREAD` and `TWRITE`.

- Fragile in *detailed coding*; there is no consistent nomenclature convention and much of the code is not FORTRAN 77 and is FORTRAN 8x deprecated.

However, it does work and work very well. It can be considerably improved from the point of view of maintenance and development be *systematically* removing the worst of the fragility; in a word to make the system more *robust*.

In this respect the most rewarding excercise would be to make the system more modular; involving:

- The rationalisation of the inter-link interfaces, the rationalisation of the COMMON blocks in names, lengths and internal structure and the conversion of the file numbers in which the COMMON blocks are stored (and their lengths) into Global objects.
- Systematic elimination of the riot of EQUIVALENCE statements.

These changes (supported by improvements in documentation) [1] would make the links into large-scale "Objects" (in modern terminology) which would recognise and act on the COMMON block messages.

However, from a quantum chemistry point of view, these enhancements are non-productive work and most efforts are likely to be concentrated on *functional* enhancements to the system.

D.2 Preview

There are two very broad categories of change which may be made to `gnu80`:

- Improvement or replacement of existing functionality (e.g. speeding up the code or implementing new integral evaluation techniques or diagonalisation methods).
- Introduction of completely new facilities (e.g. the addition of a Random Phase Approximation link)

The second of these requires everything that the first requires *plus* changes to the parser tables and route generator in order that the new command may be recognised and an appropriate route generated.

For the time being, attention is concentrated on the former enhancement. Within this framework there are again two broad areas where changes might be made:

- The calculation of molecular integrals over the basis functions; one-electron integrals or electron-repulsion integrals and the use of these new integrals with the existing quantum-mechanical models.

[1] Here, as elsewhere, it is important to distinguish between *program documentation* and mere *operating instructions*

- The use of existing integral generation facilities to implement additional quantum-mechanical techniques.

In fact, the latter is easier to implement for obvious reasons; all that is neede is the location of the one-electron integrals (the COMMON block) and the file containing the repulsion integrals. In the former case the data structures describing the atoms in the molecule and the orbital basis etc. must be located *and the detailed structure of the information known.*

Before any particular case can be discussed, it is necessary to have an "overview" of the salient parts of gnu80 (and, by implication GAUSSISAN) organisation.

D.3 Data Communication in gnu80

The numerical work invlved in *ab initio* quantum chemical calculations fall naturally into a series of independent tasks. The overall structure of gnu80 reflects this underlying modularity. The large size of gnu80 means that it must sometimes be overlaid and, more important, the demands made for computrer resources (time and file store) are such that the system should be re-startable after a machine event.

This raises the question of data communication between the modules of gnu80. The communication must be done via non-volatile (disk) files which have to contain the essential data structures input to the system or generated by the system which are essential for later modules.

Broadly speaking the data structures are of two types:

1. Electron-repulsion integrals

2. Other, more heterogeneous, data input or generated by the system; one-electron integrals, orbital and symmetry specifcations etc. etc.

The latter sets of data are grouped together in gnu80 as named COMMON blocks, copies of which are output as they are updated or as a link is changed so that the contents of the COMMON blocks may be restored at a later stage in the run.

This method of data communication makes use of one of the peculiarities of the FORTRAN language which enables a heterogeneous set of data to be treated as a single unit, providing a crude kind of “DATA STRUCTURE” in FORTRAN.

The COMMON facility of FORTRAN declares a specific set of variables to be *global*; to be available to all program segments in which the COMMON declaration

```
COMMON /CNAME/ list
```

is made (where `CNAME` is the literal name of the COMMON block and `list` is a list of variables of arbitrary type). A side-effect of this facility is that the items in the `list` are forced to be *contiguous* in storage. That is the whole COMMON block can be addressed by (e.g.) the address of the first member of `list`. Now, recall that FORTRAN passes all parameters to SUBROUTINEs “by address” . Thus, knowing the name of the first member of `list` and the *length* of `list` (in some convenient units like the length of an integer or a double precision real), we may pass this information through an inter-segment interface and *treat it as a unit* since FORTRAN forgets everything except the address in such circumstances. Again, in modern terminology, the arguments of SUBROUTINES in FORTRAN are “Overloaded” .

In particular, we may write the whole of `list` and read it back *as a unit* in one FORTRAN statement; providing an efficient, if crude, method of inter-link data communication. An example will make this clear, particularly to the non-FORTRAN programmer:

```
DOUBLE PRECISION A
INTEGER FILE, FLEN, I, J, K, L, ID
COMMON /DUMMY/ I, J, A(200), K, L(50), ID
DATA FILE, FLEN/10,227/
.
.
.
CALL WRITE(FILE,FLEN,I)
.
.
END
SUBROUTINE WRITE(UNIT,LENGTH,START)
INTEGER UNIT, LENGTH
DOUBLE PRECISION START(1)
INTEGER I
WRITE(UNIT) (START(I),I=1,LENGTH)
RETURN
END
```

The FORTRAN channel number of the file is `FILE` (i.e. 10) and its length is `FLEN` (i.e. 227) assuming that one double precision real is twice as long as an integer. Notice that `WRITE` is called with an `INTEGER` argument (`I`) whereas the type of the third argument of `WRITE` is `DOUBLE PRECISION`, FORTRAN allows this but it will often be flagged as an error if one calls a routine with arguments of different type in the same segment.

This is the method used in `gnu80`, the routines `TWRITE` and `TREAD` are the ones which perform these tasks; manual pages are given below.

NAME TWRITE

Standard file output routine

SYNOPSIS

```
subroutine TWRITE(FILE,X,M,N,MM,NN,K)
double precision X(1)
integer FILE, M, N, MM, NN, K
```

DESCRIPTION

TWRITE writes data to the `gnu80` internal file number `FILE` from the double precision array `X`. In the calling segment X must be `DIMENSION`ed `MM` by `NN`. The data written is taken from this array as far as `M` by `N` consistent with FORTRAN matrix storage rules. `K` indicates whether or not the matrix has been stored in a compressed mode (for symmetric matrices) i.e. contains only M(M+1)/2 elements, not M**2. The routine is called to write actual matrices to files *and* to write COMMON blocks in which case usually `N=NN=1` and `M=MM=` length of file (in units of double precision reals, padded out if necessary)

ARGUMENTS:

FILE The `gnu80` internal file number

X Array containing the data to be written.

M Actual number of rows in the matrix X.

N Actual number of columns in the matrix X.

MM Number of rows in the DIMENSION statement of calling program.

NN Number of columns in the DIMENSION statement of calling program.

K `K = 0` means that all the matrix is write, `K = 1` means only "half" was write.

SEE ALSO

TREAD, TQUERY, NTRAN, FILEIO

DIAGNOSTICS None; but NTRAN tracks errors

NAME TREAD

Standard file input routine

SYNOPSIS

```
subroutine TREAD(FILE,X,M,N,MM,NN,K)
double precision X(1)
integer FILE, M, N, MM, NN, K
```

DESCRIPTION

TREAD reads data from the `gnu80` internal file number `FILE` into the double precision array `X`. In the calling segment X must be `DIMENSION`ed `MM` by `NN`. The data read is stored in this array as far as `M` by `N` consistent with FORTRAN matrix storage rules. `K` indicates whether or not the matrix has been stored in a compressed mode (for symmetric matrices) i.e. contains only M(M+1)/2 elements, not M**2. The routine is called to read actual matrices from files *and* to read COMMON blocks in which case usually `N=NN=1` and `M=MM=` length of file (in units of double precision reals, padded out if necessary)

ARGUMENTS:

FILE The `gnu80` internal file number

X Array to receive the data read.

M Actual number of rows in read matrix.

N Actual number of columns in read matrix.

MM Number of rows in the DIMENSION statement of calling program.

NN Number of columns in the DIMENSION statement of calling program.

K `K = 0` means that all the matrix is read, `K = 1` means only "half" was read.

SEE ALSO

TWRITE, TQUERY, NTRAN, FILEIO

DIAGNOSTICS None; but NTRAN tracks errors

The advantages of simplicity using this method are obvious but it has contributed to the fragility of `gnu80` by requiring both the name of the first member of the COMMON block *and its length.* If, for example, one makes an addition to a COMMON block which is required to be communicated to other links then the CALL to `TWRITE` and `TREAD` must be changed to allow for the new length. Unfortunately, many of these lengths are effectively "hard-wired" into `gnu80` as absolute constants. They must be made Global variables if changes are to be made securely.

D.4 gnu80 Read/Write Files

Here is a list of the `gnu80` internal file numbers currently in use and their contents.

501 COMMON /GEN/

502 Title and Atomic Orbital labels

503 Error Function interpolation table

504 Coordinate portion of blank COMMON

505 AO scaling factors

506 COMMON/B/; Basis set information

507 COMMON /ZMAT/, the Z-matrix

508 COMMON /IBF/ integral buffer format

509 incomplete integral buffer

510 COMMON /FPINFO/ Fletcher-Powell optimization program data

511 COMMON /GRDNT/ energy, first and second derivatives over NVAR

512 pseudo-potential information

513 COMMON /DIBF/ integral derivative buffer format

514 Overlap matrix

515 Core-hamiltonian

516 Kinetic energy integrals

517 Fermi contact integrals

518 x-dipole integrals

519 y-dipole integrals

520 z-dipole integrals

521 Cartesian first and second derivatives of the energy

522 eigenvalues, α and if necessary, β

523 symmetry assignments

524 MO coefficients, real α

525 MO coefficients, imaginary α

526 MO coefficients, real β

527 MO coefficients, imaginary β

528 density matrix, real α

529 density matrix, imaginary α

530 density matrix, real β

531 density matrix, imaginary β

532 density matrix, real total

533 density matrix, imaginary total

534 density matrix, real spin

535 density matrix, imaginary spin

536 Fock matrix, real α

537 Fock matrix, imaginary α

538 Fock matrix, real β

539 Fock matrix, imaginary β

540 molecular α-β overlap (u), real

541 molecular α-β overlap (u), imaginary

542 pseudo-potential information

543 pseudo-potential information

544 pseudo-potential information

545 COMMON /ORB/ - window information

546 bucket entry points

547 eigenvalues (double precision with window: always α and β, even in RHF case)

548 MO coefficients (double precision with window, α and if necessary, β)

549 molecular orbital α-β overlap, double precision with window

550 holds virgin copy of COMMON /B/ during GBASIS optimizations.

551 symmetry operation info (permutations, transformation matrices, etc.)

552 character strings containing the stoichiometric formula and framework group designation.

553 temporary storage of COMMON /GEN/ during **FP** optimizations.

554 alternate starting MO coefficients, from L918 L503, real α.

555 alternate starting MO coefficients, from L918 L503, imaginary α.

556 alternate starting MO coefficients, from L918 L503, real β.

557 alternate starting MO coefficients, from L918 to L503, imaginary β.

558 computed harmonic frequencies, in wavenumbers.

559 COMMON /MAP/.

562 symmetry operations for orbital symmetry assignments.

563 integer symmetry assignments (α).

564 integer symmetry assignments (β).

565 lists of symmetry equivalent shells and basis functions

566 symmetry partitions for l506

567 GVB pair information

568 occupation numbers and J and K coefficients (L506)

569 label COMMON for L506

570 COMMON /ZSUBST/

571 energy weighted density matrix.

572 coordinate array with dummies intact.

573 not used

574 COMMON /MSINFO/ Murtaugh-Sargent program data

575 COMMON /OPTGRD/ gradient optimization program data

576 COMMON /TESTS/ control constants in L105

577 non default atomic charges

992 COMMON /NTR/

993 COMMON /INFO/

994 COMMON /PHYCON/

995 COMMON /MUNITS/

996 COMMON /IOP/

997 COMMON /MOL/

998 COMMON /ILSW/

999 Overlay data

D.5 Basic I/O Routines

There are two types of disk I/O performed by `gnu80`:

1. Inter-link data communication and scratch I/O

2. Electron-repulsion integral storage and retrieval

The first of these use **TREAD** and **TWRITE** as we have seen above. These routines call **FILEIO** which itself calls the most basic routine **NTRAN**. The repulsion integral I/O is performed by calls to the various entry points of a general integral I/O routine **INTEGI** which calls **NTRAN**.

Manual pages for **FILEIO** and **NTRAN** are given below.

NAME FILEIO

Random access Input/Output

SYNOPSIS

```
subroutine FILEIO(IOPER, IFILNO, LEN, Q, IPOS)
double precision Q(1)
integer IOPER, IFILNO, LEN, IPOS
```

DESCRIPTION

FILEIO performs all the functions dealing with the logically *random-access* I/O of gnu80.

Although **FILEIO** may appear to maintain a large number of files (up to 200), all of this data is actually stored into a *single* disk file in the usual FORTRAN sense. This (FORTRAN random access) file is divided up by **FILEIO** and portions of it are allocated to each of the logical files (buckets and read-write files). Thus "files " in **FILEIO** terminology are not the same as "files" in standard FORTRAN terminology. It is therefore necessary to use one or the other consistently in a program. gnu80 uses "files" always to mean **FILEIO**-compatible files. Thus data *must* always be read or written by **FILEIO** and *not* by standard FORTRAN READ or WRITE statements. Of course, **FILEIO** itself uses standard FORTRAN statements to do the actual I/O!

The two types of file "buckets" and "read-write files" only differ conceptually not essentially; roughly speaking, read-write files are inter-link communication and buckets are scratch space (particularly during integral transformations).

For each file (bucket or read-write file), **FILEIO** maintains both read and write pointers. A write operation on a file, for instance, starts at the position of the write pointer for that file. For each value written, the write pointer is incremented. The read pointer behaves the same way on read operations.

ARGUMENTS:

IOPER type of I/O operation to perform:

- **0** define file `IFILNO` to have length `LEN`
- **+1** synchronous write operation. Control returns to the calling routine when the write is completed. (this operation defines a file with length `LEN` if the file was not previously defined). Note that in `gnu80` *portable* FORTRAN I/O is used so that *all* I/O is synchronous notwithstanding information to the contrary.
- **+2** synchronous read operation.
- **4** define subfile.
- **5** delete the file indicated by `IFILNO`.
- **6** delete all routine-volatile files (those with numbers larger that 2999).
- **7** delete all the link-volatile files. This should not be done by a user directly, but rather let `NEXTOV` do this for you when overlaying.
- **8** delete all overlay-volatile files. Once again, don't do this unless you're sure you know what you're doing.
- **9** **FILEIO** open. This is done in chain at the beginning of the program, and should not be used elsewhere.
- **10** **FILEIO** close. This is done automatically by `NEXTOV` on its way out, and should not be used elsewhere.
- **11** this returns the length of the specified file (in `LEN`). if the file is undefined, then 0 is returned. Thus, you can check for the existence of a file before trying to read from it.
- **12** **FILEIO** initialization. This is done by `MAIN` at the beginning of the run, and should not be done elsewhere.

IFILNO The absolute value of `IFILNO` is the number of the file which is to be read, written, defined, or deleted. Any non-zero value in permitted, but the following conventions are to be observed in `gnu80`:

- **1-499** permanent buckets.
- **500-999** permanent read-write files.

1000-1499 overlay volatile buckets. These are deleted automatically before each new overlay.

1500-1999 overlay volatile read-write files.

2000-2499 link volatile buckets. These are automatically deleted before each new link.

2500-2999 link volatile read-write files.

3000-3499 routine volatile buckets. These are deleted when **FILEIO** is called with `IOPER=6`, and also before each new link.

3500-3999 routine volatile read-write files.

For read or write operations, supplying the *negative* of the file number causes the read or write pointer to be rewound (reset to the base of the file) before the operation. Note that this is done before the argument `IPOS` is processed. For file deletion or file definition operations, the sign of the file number is ignored.

LEN This quantity is the number of double precision values (quadruples of bytes, usually) to be transferred in a read or a write operation, or the number of these values to be allocated in a define file operation.

Q a double precision array for the data read or written.

IPOS Used to help specify the position in the file at which an I/O operation will commence. The value of the read or write pointer is incremented by `IPOS` before the read or write operation is performed. The possible rewind of the pointer (as specified by the *sign* of `IFILNO`) is done before `IPOS` is considered. Thus, a *positive* file number means that `IPOS` specifies the new position relative to the current position; a *negative* `IFILNO` means `IPOS` specifies the new position relative to the begining of the file.

SEE ALSO

NTRAN, TREAD, TWRITE

DIAGNOSTICS

None directly, but **NTRAN** provides some.

NAME NTRAN

Lowest level I/O routine.

SYNOPSIS

```
subroutine NTRAN(UNIT, OP, NWRDS, X, L)
double precision X(1)
integer UNIT, OP, NWRDS, L
```

DESCRIPTION

This routine is developed to handle the I/O requests of **FILEIO** and to perform all asynchronous I/O. (two electron storage). It provides `gnu80` with a *word addressable* disc I/O capability. It is called by **FILEIO** and the routines **IREAD, IWRITE, IWIND** etc. which are ENTRY points to **INTEGI**

NTRAN can nominally handle synchronous and asynchronous I/O requests. In the `gnu80` implementation all I/O is portable FORTRAN 77 and so there is no asynchronous I/O but the code has been left in (as comments) in case this facility is ever supported by future FORTRANs.

Synchronous I/O is random access (on 4 byte word level), **NTRAN** is designed to allow the FORTRAN programmer to store and fetch arbitrary data from disc. Arbitrary means that any number of 4 byte words may be transferred to or from any location on a disc file. Asynchronous I/O is basically sequential, and more restricted than synchronous I/O, in the sense that only an *integer* number of records can be written or read. This implies that the same number of words must be read back as were written out on that record previously. All operations or options are performed only on the specified unit. The synchronous I/O is performed with the use of a direct access unit. At the moment this is fixed (through a `DEFINE FILE` statement). Because of this **NTRAN** can only handle one direct access unit (`DEFINE`d to FORTRAN unit 18), but the code could be extended to handle other direct access without much effort. Currently gnu80 only uses one asynchronous unit (3) but **NTRAN** is coded to handle 2 more (just define them through an appropriate **NTRAN** call).

Note that the code in **NTRAN** traps most of the common errors *before* the FORTRAN I/O does, so that, for example, attempts to read an unwritten record is trapped and an error message given. This policy has only one possible bad side-effect; the *length* of the total file is specified by the program as LEN18*RECL where RECL is the record length specified in the OPEN statement for unit 18 (see Chapter F). This number need not have any relationship to the *actual* disk space available on a particular system. So it is possible, by setting LEN18 too small, to get a message that there is no file space left when there is physical space available. The solution is obvious; set LEN18 as a very large integer and risk a system-generated storage crisis or actually *calculate* LEN18 for your filestore.

Usage notes:

1. UNIT must always be equal to one of the logical unit numbers stored in the array UNITS, even for calls which are not device specific.
2. Don't rewind before a reposition. If necessary, **NTRAN** will do this automatically. To reposition **NTRAN** will either:
   Read records until the specified position is reached, and if necessary write records to extend the file.
   Backspace the unit until the specified location is reached
   Rewind the unit and skip records to the specified location.
3. Only use OP=23 (wait) if you want to use the data read or redefine the array used in a write operation. All other waits are performed automatically if needed. Use the negative form of the op parameter if you want to simulate synchronous I/O on an otherwise asynchronous unit.
4. The correct sequence of operation is (synchronous)
   a) define unit (26)

b) reopen unit if it is an old file (21)
c) reposition unit (6)
d) read/write (2,1)
e) go to c
asynchronous:
a) define unit (27)
b) rewind unit (10)
c) read or write
d) a rewind (10) should be issued between read and write operations

ARGUMENTS:

UNIT the fortran logical unit number of the file to be accessed. `UNIT` maps onto a local unit number via the array `UNITS`. All calls to **NTRAN** require a valid unit. Note that the value of `IUNIT` is used as a local event flag number to signal I/O completion of asynchronous I/O.

OP specifies the operation to carry out. A *negative* value indicates that **NTRAN** is not to return control to the calling program until the requested I/O operation is complete. (only for 1 and 2)

+/- 1 write

+/- 2 read

6 reposition

10 rewind

21 reopen old unit, `NWRDS` = number of records in old file.

22 close unit (only for synchronous I/O)

23 wait

26 define synchronous unit (always done first)

27 define asynchronous unit (always done first)

29 print switch, `NWRDS`=0 means no printing, `NWRDS`$\neq$ 0 means print.

NWRDS the number of words involved in the operation. A radix (number of 8 bit bytes per user datum) of 4 is assumed, thus nwrds specifies the number of longwords to transfer.

X data to be transferred.

L status word:
-1 indicates operation in progress
-3 indicates end of file was encountered during synchronous read
-2 indicates error
$>$ 0 indicates successful completion of the request, `L` containing the number of words transferred.
Note that as currently coded **NTRAN** will abort the job if an error occurs, therefore L=-3 should not occur.

SEE ALSO
FILEIO

DIAGNOSTICS
Provides error trapping *before* the FORTRAN diagnostics.

D.6 Two-electron Integral Storage

D.6.1 Introduction

The efficient storage and retrieval of the huge numbers of electron-repulsion integrals generated during any *ab initio* caculation is a major design problem in quantum chemical calculations. Fortunately, the access of the basis-function integrals is essentially *sequential.* The number of basis-function repulsion integrals is proportional to the fourth power of the number of basis functions and the time required to compute any one of them depends on the *type* of the basis function and the number of primitive Gaussian functions which make up the four basis functions involved in the integral; if each basis function is of the same degree of contraction then the time taken to compute an integral depends on the fourth power of this degree of contraction.

During the generation of the electron repulsion integrals they are placed into a *Buffer* which is written out to mass storage when it is full. The process continues until a complete (sequential) file of all the integrals has been generated. The way in which the integrals and their *labels* (the indices i, j, k, ℓ) are stored in the buffer (and therefore in the final file) depends on a number of tactical considerations.

D.6.2 Storage Formats

At the moment, there are *four* different storage formats for two electron integrals. In the discussion below the **charge cloud** notation will be used for the electron repulsion integrals over *real* basis functions:

$$(ij, k\ell) =< \phi_i\phi_k|\phi_j\phi_\ell >= \int dV_1 \int dV_2\phi_i(\vec{r_1})\phi_j(\vec{r_1}) \left(\frac{1}{r_{12}}\right) \phi_k(\vec{r_2})\phi_\ell(\vec{r_2})$$

Format 0 each integral value $(ij, k\ell)$ is stored together with its four indices i, j, k, ℓ. One of these Format 0 integrals can be further classified according to whether or not any indices are coincident. that is, the integral (43,21) is stored differently than say (53,22). Integrals having no coincidences are classified as **type 1** integrals; those involving coincidences are termed **type 2** integrals. It turns out to be advantageous in the SCF if type 1 and type 2 integrals are separated. In order to achieve this separation, type 1 integrals are loaded into the buffer starting at the beginning and type 2 integrals starting at the end of the buffer. When the two types meet, the buffer is full, and then written out. each label-integral combination that is inserted into the buffer has the storage requirements of one (long) integer and one double-precision real; that is 12 bytes long (one byte = 8 bits) on many systems. The label portion occupies the integer portion, and the remaining storage contains $(ij, k\ell)$ stored in full double precision. The storage of the *four* indices of the integral in the space of *one* long integer is a device to save file space; clearly the *values* of the indices are *small integers*, typically not greater than 100. In addition to being in different parts of the buffer, type 1 and type 2 integrals have different label construction. A type 1 label consists of four areas containing i, j, k, ℓ. A type 2 label consists of three areas containing up to three unique indices, and a smaller code containing an index indentifying the type of coincidences involved.

Format 1 It is possible to achieve substantial savings in the steps that process the two-electron integrals if the integrals are pre-processed suitably in the integral generation Links. This technique was originally suggested by R. Raffenetti *(Chem. Phys. Lett.(1973),* **20***, p335)* who advocates sorting the integrals to achieve time savings. Specifically, if

$$(ij, k\ell) = (ij, k\ell) - 0.25[(i\ell, jk) + (ik, j\ell)] \tag{D.6.1}$$

is computed and stored, the SCF processing time can be reduced by about a factor of three for standard closed shell computations.

Using this storage mode, the buffer is divided into two partitions. The first partition contains all the labels, and the second partition contains the integrals. In this case it is not necessary to store all four indices i, j, k, ℓ, but only the *linearized* indices (ij) and $(k\ell)$,

$$(ij) = (i(i-1))/2 + j$$

$$(k\ell) = (k(k-1))/2 + \ell$$

thus a label consists of two short integer quantities containing (ij) and $(k\ell)$. The integral is again stored as a full double precision value.

Format 2 Two other quantities related to (1) can de identified which are necessary for open shell calculations and complex SCF calculations:

$$(ij| + |k\ell) = (i\ell, jk) + (ik, j\ell) \tag{D.6.2}$$

$$(ij| - |k\ell) = (i\ell, jk) - (ik, j\ell) \tag{D.6.3}$$

Format 2 is identical to format 1 except that *two* floating point numbers are stored for each label: integrals D.6.3 and D.6.3.

Format 3 This format is identical to format 2 except now we store all three combinations.

D.6.3 Buffer construction

As the integrals are computed, they are loaded into a buffer (working array in memory) according to one of the above prescriptions. When the buffer is filled, a control word characterizing the buffer is appended, and the buffer is written out. The *buffer control word* contains a code indicating whether or not the current buffer is the last buffer. It also contains the count of the number of integrals within the buffer. Note that for format 1 buffers, two counts are required, one for each type of integral. This control word is currently placed at the beginning of the buffer.

The size of a buffer is currently set to 4760 double precision words, which is 19040 bytes. This is entirely historical; it is the size of a track on a now-obselete IBM disk pack. This size can be changed by modifiying the parameters located in `SUBROUTINE SET2E` (L301), but this must only be done with extreme caution as the size of this buffer appears *explicitly* in many places in the system.

D.6.4 I/O Considerations

The necessary routines to read and write the integral buffers are available as entry points in the `SUBROUTINE INTEGI`. The relevant `ENTRY` pointa are:

`IREAD` reads a buffer.

`IWRITE` writes a buffer.

`IWAIT` waits for the previous I/O request.
Not implemented in `gnu80` - see NTRAN manual page

`IWIND` rewinds the integral file.

D.6.5 Implementation Considerations

FORTRAN language routines are available to pack and unpack the three different label types (format 0, types 1 and 2; formats 1,2 and 3). these routines are:

`PACK1, UNPCK1` format 0, type 1

`PACK2, UNPCK2` format 0, type 2

`PACK3, UNPCK3` formats 1,2,3

It is not necessary to have a detailed description of these routines, simply that each *pair* is complementary.

D.7 Parsing the Command Record

In order to be able to make significant changes to `gnu80` (adding new links and the possibility of new Routes) it is necessary to understand how the Route is generated from the command record. The command record must be *parsed* for the meaning of the symbols it contains, checked for errors and meaningless symbols and finally the correct command translated into a valid Route.

These tasks are performed by the `INTEGER FUNCTION QPARSE` and its associated data structure a *Parse Table.* The information in the following sections, together with a study of the use of `QPARSE` in `gnu80`, should enable the user to master the parsing of the command record.

D.7.1 QPARSE use

QPARSE is a table-driven line parser. It is called with a *string of characters* to be parsed, a parse table, and a *Result Vector.* The input string is just a string of characters which, in line with the design of gnu80, is contained in an INTEGER array. The parse table tells the FUNCTION which sub-strings are meaningful in the input string (a number, a certain keyword, etc.), and what to do if a specified meaningful sub-string is found in the input string.

The RESULT Vector contains the output from QPARSE. The specification of QPARSE is:

```
INTEGER FUNCTION QPARSE(RESULT,TABLE,LINE,LENGTH)
INTEGER RESULT(1), TABLE(1), LINE(1), LENGTH
```

LINE contains the (ASCII) string to be parsed stored four characters to a (default) integer.

TABLE is a DATA structure containing the information necessary to parse the string in LINE This structure is discussed in detail below.

RESULT is an INTEGER array in which the results of the parse are stored. The description of the TABLE structure gives the meaning of the RESULT array.

LENGTH The number of *characters* in the string stored in LINE.

QPARSE The INTEGER value returned by QPARSE gives the status of the parse. The possibilities are:

- 0 Success; a transition has occured to EXI
- -1 Failure; a transition has occured to FAI. This indicates some syntax error in the input string.
- -2 Failure: an ambigous keyword was detected in the input string.
- -3 Error; an error was detected in the parse table.
- 1 Return; the parse is not yet completed, but control has returned to the caller because of a parse-table request.

Before any calls to QPARSE are made, the routine must be initialized by a call to QPINIT which sets certain options to QPARSE.

```
SUBROUTINE QPINIT(TABLE,BLNKS,CAPS,ABVRS)
INTEGER TABLE(1), BLNKS, CAPS, ABVRS
```

TABLE the parse table used as input to QPARSE. The remaining arguments set options in the parser.

BLNKS By default, any number of (contiguous) blanks and tabs in the input string are treated as invisible delimiters. A non-zero value for BLNKS means that blanks will be seen by QPARSE.

CAPS By default, the difference between upper and lower case letters is ignored. A non-zero value of CAPS indicates that only exact matches are acceptable.

ABRVS By default, any unambiguous abbreviation is allowed for a keyword. A non-zero value of ABRVS makes abbreviations non-acceptable.

The following material describes how to construct a parse table with some examples.

D.8 The QPARSE Table

The QPARSE table, which descibes how the input record is to be parsed, can be built with FORTRAN DATA statements.

The table consists of one or more *states*, and the general structure of the table is illustrated below:

```
<name-of-state>             (1 default integer)
<transition definition>     (variable length)
<transition definition>         "       "
          "
          "
          "
<end-of-state>              (1 default integer)
<name-of-state>             (1 word)
<transition definition>
          "
          "
          "
<end-of-state>
<end-of-table>              (1 default integer)
```

Thus, the table consists of one or more states, and each state consists of one or more *transition definitions*. Generally, the states provide the overall control structure for the parse, and the transition definitions provide the details of what to look for in the input record, and what to do if it's found.

The <name-of-state> field is just a string of up to four characters which labels the state. The parser has three pre-defined state names which should not be used as <name-of-state> fields in the table. These are `RET`, `EXI` and `FAI`. The use and importance of these special state names are descibed below in the detailed description of the transition definition. The <name-of-state> field may be left zero in some cases, but one default integer must always be allocated in the table.

The <transition definition> field is of variable length and is discussed in detail below. Generally, a transition definition tells the parser what kind of object to look for in the input record (keyword, character, integer, etc.), and what to do if it's found. If the object specified by the transition definition matches the next character(s) in the input record, then the character(s) from the input record are accepted, and a state transition occurs. This transition may be to a new state, or just back to the top of the transition definition. Other subfields of the transition definition can specify one of a set of simple operations which can be performed during the transition:

- Store the object accepted from the input record into a specified storage location.
- Add a given value into a specified storage location.

The <end-of-state> mark is just aa integer with a particular value (`EOS`) which terminates the state. If none of the transition definitions in the current state succeeds, then this mark will be encountered and a transistion will occur to the special state `FAI` and the parse fails. In this case, a failure status will be returned by `QPARSE`.

The <end-of-table> mark is just a <name-of-state> field containing the value $\widehat{\text{E}}$ND.

D.8.1 Transition Definitions

A *transition definition* consists of a number of subfields, one of which is of variable length:

<alphabet token>, <destination>,<index>,<value>

Each of these subfields is discussed individually below.

<token> This field describes what to look for in the input record. If this token matches the next data in the input record, then the data in the record is *accepted*, and the transition succeeds. For instance, one can ask:

> "if the next characters in the record form a decimal integer, then accept them and make the transition."

Characters which are accepted are removed from the front of the input record, and subsequent transition definitions will try to match whatever characters come after these. If the transition succeeds, then the rest of the transition definition is used to determine the details of what actions are to be taken. If there is no match, then the transition fails, and the next transition definition in the current state is checked against the same characters. If no transition definitions remain, then a transition to the state `FAI` occurs, and the parse fails. The various tokens, some of which are longer than one integer, are detailed below in the section on the `QPARSE` alphabet.

<destination> This field is a string of characters which names the state to which the transition is to be made. If this field is zero, then a transition is made to the state immediately following the current state. Otherwise, this string should match either one of the specified state names, or one of the <name-of-state> fields in the table. The special states, and the effect of a transition to each is detailed below:

`EXI` Causes the parser to halt, and to return an zero status (return value of `QPARSE`). A transition to `EXI` indicates successfull completion of the parse.

`FAI` causes the parser to halt, and return a failure status. A transition to `FAI` indicates a syntax error in the input record.

`RET` causes a transition to the state immediately following the current one in the table. However, control is returned to the calling routine before the first transition definition in this new state is examined. This is useful when the calling routine needs to perform some other action before the parse can continue.

<index> and <value> These fields are used to request some simple operations which can be performed during the transition. If both of these values are non-zero, then <value> is added into `RESULT(<index>)`. `RESULT` is one of the calling arguments to the parser. If only the <index> field is non-zero, then the object accepted from the input record (character, integer, keyword, etc.) is stored into `RESULT(<index>)`. If the <index> field is zero, then nothing is done during the transition.

D.8.2 QPARSE Alphabet

1. Tokens which match single characters.
   If it is requested that this token be stored into `RESULT` (a zero value for <value>), then this character is justified in the indicated word as `A1`.

   `NUM` matches any numeric character.

`ALP` matches any alphabetic character.

`ALN` matches any alphanumeric character.

`CHR` matches any character.

2. Tokens which match numbers. If it is requested that one of these be stored in `RESULT`, the corresponding numeric value is stored there. Note the difference, then, between the tokens `NUM` and `D10` (below). Each of these matches one numeric character in the input record, but there is a difference if the object accepted is to be stored into `RESULT`. `NUM` causes the *character* to be stored there, while `D10` causes the corresponding integer value to be stored. The following tokens are recognised:

   `D10` matches a base 10 digit.

   `D16` matches a hexadecimal digit.

   `D8` matches an octal digit.

   `D2` matches a binary digit.

   `I10` matches a decimal integer. this must be terminated in the input record by a non-alphanumeric character (or by the end of the input record).

   `I16` matches a hexadecimal integer.

   `D8` matches an octal integer.

   `I2` matches a binary integer.

   `FP` matches a floating point number.

   `DP` matches a double precision floating point number. the only difference between this and `FP` is that this takes two locations in `RESULT`. (starting with `RESULT(<index>)`.

3. Tokens which match strings. If one of these is to be stored into `RESULT`, then a variable number of words in `RESULT` will be used, depending unpon the length of the string. The integer actually indicated by <index> will contain a count of characters in the string, and subsequent words will contain the ASCII string. The following tokens are recognised:

`N,<keyword>` This token matches a keyword. `N` is a positive integer which indicates the number of characters in the keyword which follows. For instance, the token describing the keyword "HELLO" takes three integers;

5, `'HELL','O'`

By default, any abbreviation is accepted for a keyword, as long as it is unambiguous. A keyword is defined as an alphanumeric string terminated by a non-alphanumeric character. Thus, if the characters 'HELLOAB' were at the front of the input record, the above token ('HELLO') would not match, since the string 'HELLO' in the input record is not terminated by a *non-alphanumeric* character.

`-N,<string>` This variable-length token matches a specific string of characters. `-N` means that the character(s) to be matched follow in the next *integer(s).* For instance, the token:

-5,'HELL','O'

matches the string 'HELLO'. note the difference between the token for the string 'HELLO' and that for the keyword 'HELLO'. If the input record contains the characters 'HELLOAB', the string 'HELLO' (-5,'HELL','O') will match. The keyword 'HELLO' (5,'HELL','O') will not match, however, since it is not terminated by a non-alphanumeric character in the input record.

`WRD` This token matches any alphanumeric string (terminated by a non-alphanumeric character).

`STR,N,<delims>` Matches a user-defined string. This is similar to `WRD`, except that the characters which delimit the desired string are supplied in the token. `N` is the number of such delimiters, and the delimiting characters are supplied in the subsequent words.

4. Tokens which remove no characters from the input string. The following are recognised:

`NUL` Always matches. This results in no characters being removed from the front of the input record, but the supplied transition occurs. It is useful as a “go to” -type command.

`EOL` Matches end-of-line. If there are no more characters in the input record, then this token matches.

D.9 Character Manipulation in gnu80

The design of the character manipulation system in `gnu80` is necessarily based on that of the GAUSSIAN series and it *predates* the introduction of the `CHARACTER` data type in FORTRAN; all character strings are therefore stored and manipulated in other data types, principally `INTEGER`.

This has two unfortunate effects from the point of view of FORTRAN 77 portability:

1. It renders the system difficult to change to make use of the `CHARACTER` data type since many of the data structures of type `INTEGER` contain actual integer values as well as character strings; that is, there is not the possibility of simply changing the *type* of the variables used to store character strings.

2. It is a potential source of disaster, since the storage of character strings in non-`CHARACTER` data type is likely to be discontinued in FORTRAN. Many current compilers do not allow this extension to the ANSI standard and all will flag it as a warning.

The use of `'A'` as a character constant pre-dates the introduction of the `CHARACTER` data type so that `gnu80` contains character strings stored in `INTEGER` data types *and* genuine FORTRAN 77 character constants as well as the occasional Hollerith string.

The introduction of the `CHARACTER` data type *together with an implicit length* causes a complication in FORTRAN `SUBROUTINE` argument transmission, since somehow, the *length* of a character string must be passed through an interface. Since all FORTRAN argument passing is done *by address* this means that an additional argument (hidden from the user) must be passed if a `SUBROUTINE` has an argument of type

CHARACTER. Specifically, if a SUBROUTINE has an argument I, then the following combination of code is *not allowed*:

```
      SUBROUTINE NAME(I)
      INTEGER I
      .
      .
      RETURN
      END

      PROGRAM MAIN
      INTEGER K
      CHARACTER*4 J
      .
      .
      CALL NAME(K)
      .
      CALL NAME(J)
      .
      CALL NAME('ABCD')
      .
      END
```

The compiler may not allow it or it may crash on execution since the second and third `CALL`s actually generate *two* arguments in the passage to `NAME`; the address of the string and its length. It is therefore necessary to have *two* routines to perform some character string manipulations; using the above example the routines `NAMEC` and `NAMEI` are required to make the following code acceptable.

```
INTEGER K
CHARACTER*4 J
DATA K/'WXYZ'/
DATA J/'ABCD'/
.
.
.
CALL NAMEI(K)
.
.
CALL NAMEC(J)
.
.
END

SUBROUTINE NAMEC(I)
CHARACTER*(*) I
.
.
RETURN
END

SUBROUTINE NAMEI(I)
INTEGER I
.
.
RETURN
END
```

and the routine `NAMEC` *must* be used in statements like

```
CALL NAMEC('FGHJ')
```

i.e. with character constant arguments.

This compromise has been adopted in gnu80 and all the character manipulation routines are prime candidates for systematic replacement by ANSI FORTRAN 77 equivalents. A very brief specification of the routines is given below. However, the use of `INTEGER`s to store character strings has the effect of making the routines potentially compiler-dependent in the sense that a charcter input to an `INTEGER` variable by reading in A1 format or by a `DATA` statement may be stored at *either end* of the storage area assigned to the `INTEGER` since `INTEGER` type occupies more storage than `CHARACTER*1` type. There are only two rational possibilities, storage at the most or least significant end of the `INTEGER` and both are used by different compilers. In `gnu80` this potential non-portability has been concentrated into the `FUNCTION IORD` which *tests* which type of storage has been used before manipulating the characters (assuming that the characters are stored as ASCII codes) see D.10. The only remaining and highly unlikely source of disaster is if a compiler uses a different allocation for the result of reading an `INTEGER` in A1 format from the allocation generated by a statement like

```
DATA INT /'A'/
```

D.10 Main Character Manipulation Routines

Since the specification of these small `SUBROUTINES` and `FUNCTIONS` is rather brief, only a concise description of each is given; that is, the full manual page format is not used.

The three central string manipulation routines are `GETCHR`, `PUTCHR` `LORD` and `IORD`.

The two arguments to the `INTEGER FUNCTION GETCHR` are a string and a cursor. The value of the cursor is incremented by one, then the "cursor-th" character in the string, padded with blanks, is returned as the `INTEGER` value of the function. This returned character is justified in the word as A1 in `FORTRAN`. For instance, the `FORTRAN` statements:

```
INTEGER GETCHR
I = 12
J = GETCHR('A CHARACTER STRING, I)
```

results in `J` being assigned the value of `'S'` in the sense defined above; the actual value of `J` as an `INTEGER` type depends on the machine/compiler but will be consistent with other of the routines described below.

The following statement is not allowed in FORTRAN 77:

```
IF ( J .EQ. 'S') .....
```

Therefore all logical comparisons are made by the use of `DATA` statements and `INTEGER` comparisons:

```
DATA CHS/'S'/
IF (J .EQ. CHS) ...
```

or, (in general) by the use of the `FUNCTION IORD`:

```
IF (J .EQ .IORD('S'))  ...
```

Both the `DATA` statement and the `FUNCTION IORD` justify the character as in A1 format.

INTEGER FUNCTION GETCHR

```
INTEGER FUNCTION GETCHR(STRING, CURSOR)
INTEGER STRING(1), CURSOR
```

This integer function increments `CURSOR` by one, and returns as its value the character in `STRING` pointed to by the new value of `CURSOR`. The returned value will be justified in the `INTEGER` as if it had been read in A1 format, and the rest of the character positions will be padded with blanks (ASCII 32).

SUBROUTINE PUTCHR

```
SUBROUTINE PUTCHR(CHR,STRING,CURSOR)}
INTEGER STRING(1), CURSOR
CHARACTER*1 CHR
```

This routine first increments `CURSOR` by one, then puts the character in `CHR` into `STRING` at the location pointed by `CURSOR`.
This routine is the one to use when characters which are stored in variables of type `CHARACTER` are to be added to a `STRING`.

SUBROUTINE PUTICR

```
SUBROUTINE PUTICR(CHR,STRING,CURSOR)}
INTEGER STRING(1), CHR, CURSOR
```

This routine first increments `CURSOR` by one, then puts the character in `CHR` into `STRING` at the location pointed by `CURSOR`. **This routine is the one to use when characters which are stored in variables of type `INTEGER` are to be added to a `STRING`.**

To illustrate the difference between `PUTCHR` and `PUTICR`, the following pieces of code have the same effect on `STRING`:

```
CURSOR=3
CALL PUTCHR('A', STRING, CURSOR)
```

and

```
INTEGER IA
DATA IA/'A'/
CURSOR = 3
CALL PUTICR(IA, STRING, CURSOR)
```

INTEGER FUNCTION IORD

```
INTEGER FUNCTION IORD(CHR)
CHARACTER*(*) CHR
```

This function returns the (machine/compiler dependent) `INTEGER` value of the character string `CHR` (up to four characters). It should be called by `IORD('A')` or by `IORD('ABCD')` i.e. a `CHARACTER` constant or variable of length up to 4. It will return the same value as a comparable `DATA` statement would. As supplied, the function uses a simple test to discover how the machine/compiler stores characters in a variable of type `INTEGER` before performing the manipulation. The test is simply whether or not the `INTEGER` variables `IA` and `IB` generated by

```
INTEGER IA, IB
DATA IA, IB/'A','B'/
```

differ by one or not. The assumption is that, if they differ by one, then the character is stored at the *least* significant end of the `INTEGER` if not, then the character is stored at the *most* significant end of the integer. Inspection of the actual code shows that this test simply chooses an integer offset which is 1 or 4. If it is *known* which mode of storage is used in a particular machine then this constant may be unilaterally set to 1 or 4 to avoid these repeated tests. This is *not* reccommended since the function is a very minor consumer of computer time!

INTEGER FUNCTION LORD

```
      INTEGER FUNCTION LORD(CHR)
      CHARACTER*1 CHR
```

This routine returns the entry number of the `CHARACTER` variable `CHR` in the ASCII character table. It is implemented by a call to the FORTRAN 77 standard function `ICHAR`.

INTEGER FUNCTION ILORD

```
      INTEGER FUNCTION ILORD(CHR)
      INTEGER CHR
```

This routine returns the entry number of the first character contained in the `INTEGER` variable `CHR` in the ASCII character table. It is the same as `LORD` for an `INTEGER` argument. The reasons for its existence are the same as for the `PUTCHR/PUTICR` pair.

LOGICAL FUNCTION IFALPH

```
      LOGICAL FUNCTION IFALPH(CHR)
      INTEGER CHR
```

This logical function returns `.TRUE.` if the 'first' character in the `INTEGER` `CHR` is an alphabetic character, `.FALSE.` if otherwise.

INTEGER FUNCTION CROP

```
      INTEGER FUNCTION CROP(CHR)
      INTEGER CHR
```

This function returns as its value the character in CHR with all character locations except the first replaced by blanks.

INTEGER FUNCTION INTCHR

```
      INTEGER FUNCTION INTCHR(CHR,BASE)
      INTEGER CHR, BASE
```

This function returns the numeric value of the digit which is in the 'first' character in the INTEGER CHR. If this character is not a digit in base BASE, then a value of -1 is returned. (thus 0..9, A..F returns 0..15 if BASE=16).

LOGICAL FUNCTION STREQ

```
      LOGICAL FUNCTION STREQ(STR1,STR2,LEN)
      INTEGER STR1(1), STR2(1), LEN
```

This function returns .TRUE. if the two strings have LEN equal characters, or .FALSE. otherwise.

SUBROUTINE STROUT

```
      SUBROUTINE STROUT(IOUT,STR,LEN,CRLF)
      INTEGER IOUT, STR(1), LEN, CRLF
```

This routine just outputs a character string to FORTRAN unit IOUT. The characters to be output are in STR, and there are LEN of these characters. CRLF indicates whether a "newline" is desired at the end of the string (0 means no, 1 means yes). CRLF=-1 indicates that the current STR should be printed on the same line as the previous one. Note that the CRLF specifications are only compatible with other STROUT calls, not with the standard FORMAT I/O calls.

SUBROUTINE LOCSTR

```
      SUBROUTINE LOCSTR(SUBSTR,LENSUB,STRING,LENSTR,CURSOR)
      INTEGER SUBSTR(1), LENSUB, STRING(1), LENSTR, CURSOR
```

This routine searches through `STRING` for the `SUBSTR`. The search starts at the current location of the `CURSOR` in `STRING`. `LENSUB` is the length of the substring, and `LENSTR` is the length of the string to be searched. The value of `CURSOR` is updated so that `GETCHR` will return the first character of the substring. If no match is found, `CURSOR` is returned -1.

SUBROUTINE APPFP

```
      SUBROUTINE APPFP(X,N,BB,NBB)
      DOUBLE PRECISION X, BB(1)
      INTEGER N, NBB
```

This routine puts the characters representing the `DOUBLE PRECISION` value `X` into the buffer `BB`, updating the cursor `NBB` as it does. The argument `N` is the number of desired figures after the decimal point (maximum 12).

SUBROUTINE APPFFD

```
      SUBROUTINE APPFFD(X,NDECIM,BB,NBB)
      DOUBLE PRECISION X, BB(1)
```

Same routine as `APPFP` but now the representation is in FORTRAN `D-FORMAT`.

INTEGER FUNCTION DELTYP

```
      INTEGER FUNCTION DELTYP(CHR)
      INTEGER CHR
```

This function checks to see what kind of delimiter the character in `CHR` is. possible values of the function are:

0 the character is not a delimiter.

1 the character is a word delimiter.

2 the character is a statement delimiter.

4 the character is a record delimiter.

INTEGER FUNCTION NCHRPW

```
INTEGER FUNCTION NCHRPW(IDUMMY)
INTEGER IDUMMY
```

This function returns the number of characters which can be represented in a single precision `REAL` or an `INTEGER` (4 in most cases).

Appendix 4.A

gnu80 COMMON BLOCKS

4.A.1 COMMON /MOL/

```
      INTEGER NATOMS, ICHARG, MULTIP, NAE, NBE, NE,
     & NBASIS, IAN
      DOUBLE PRECISION ATMCHG. C
      COMMON/mol/ NATOMS,ICHARG,MULTIP,NAE,NBE,NE,NBASIS,
     & IAN(101),ATMCHG(100),C(300)
```

Here is the basic information concerning the molecule for which the energy calculation is being carried out.

NATOMS the number of atoms in the molecule.

ICHARG the total electric charge on the molecule, for neutral, 1 for cations, etc.

MULTIP the molecule's spin multiplicity (1 for singlets, 2 for doublets, etc.)

NAE the number of electrons of α spin.

NBE the number of electrons of β spin.

NE the total number of electrons (= NAE + NBE).

NBASIS the number of basis functions.

IAN(I) the atomic number of atom I. This array is dimensioned to 101 so that the number of double-words in the COMMON block is even.
Note that this is strictly the Atomic Number, not the nuclear charge which may be different due to the use of Effective Potentials for the inner shells.

ATMCHG(I) the nuclear charge of atom I. In non-effective potential runs this is the same as the atomic number IAN(I).

C(I) the x, y, and z, coordinates of the atoms in atomic units. The coordinates are stored in the order x1, y1, z1, x2, y2, z2, x3, etc.

4.A.2 COMMONs /ZMAT/ and /ZSUBST/

/ZMAT/

```
      INTEGER IANZ, IZ, LBL, LALPHA, LBETA, NZ, NVAR
      DOUBLE PRECISION BL, ALPHA, BETA
      COMMON /ZMAT/   IANZ(50),IZ(50,4),BL(50),ALPHA(50),BETA(50),
      $                LBL(50),LALPHA(50),LBETA(50),NZ,NVAR
```

/ZMAT/ consists of eight arrays dimensioned to hold the data from up to 50 Z-matrix records and two variables containing counters.

IANZ atomic numbers

IZ connectivity data

BL bond lengths

ALPHA valence angles

BETA dihedral or second valence angle

LBL an array used to map from the array of variable values in /ZSUBST/ to the array BL in /ZMAT/.

- For LBL(N)=0, no substitution is required to get BL(N): it already contains the proper value.
- For LBL(N) = I; BL(N) = VALUES(I).
- For LBL(N) = -I; BL(N) = -VALUES(I).

LALPHA analagous to LBL for ALPHA.

LBETA analagous to LBL for BETA.

NZ the number of records in the Z-matrix.

NVAR the number of variables, equal to the number of symbols defined in the VARIABLES section of the input.

/ZSUBST/

```
      INTEGER  INTVEC
      DOUBLE PRECISION VALUES, FPVEC, ANAMES
      COMMON /ZSUBST/ ANAMES(50),VALUES(50),INTVEC(50),FPVEC(50)
```

/ZSUBST/ contains the data in the VARIABLES section of the input.

ANAMES alphanumeric names of the variables. (limited to 8 characters in the current version).

VALUES the corresponding numeric values. These will be altered in the course of geometry optimizations and potential surface scans.

INTVEC an array of the integer values following the symbol and value on a line in the VARIABLES section.

FPVEC a corresponding array containing the floating point number. The use of these two arrays depends upon the route.

4.A.3 COMMON /B/

/B/ is a common block which contains the necessary arrays to define and describe the basis set.

```
      INTEGER SHELLA,SHELLN,SHELLT,SHELLC,SHLADF,AOS,AON
      DIMENSION SHLADF(80), C4(80)
      COMMON/B/EXX(240),C1(240),C2(240),C3(240),X(80),Y(80),Z(80),
     $         JAN(80),SHELLA(80),SHELLN(80),SHELLT(80),SHELLC(80),
     $         AOS(80),AON(80),NSHELL,MAXTYP
      EQUIVALENCE(SHLADF(1),C3(161)),(C4(1),C3(81))
```

This common block is organized in such a manner so as to facilitate the calculation of integrals over Gaussian functions.

Before describing the various arrays, it will be useful to define the concepts of primitive shells and contracted (or full) shells.

A *primitive* shell is defined to be a set of basis functions up to and including functions of some maximum angular quantum number which share a common exponent. thus, an $\ell = 1$ shell would consist of the functions (s,px,py,pz) all with the same Gaussian exponent. Similarly, an $\ell = 2$ shell would contain (s,px,py,pz,xx,yy,zz,xy,xz,yz) where xx, etc. denote the normalized second-order Gaussian functions.

A *full, or contracted* shell results from contracting the functions of several primitive shells together. in typical calculations, one normally uses contracted shells.

As an example, consider the carbon atom in the 6-31G basis. in this basis, the carbon will have 10 primitive shells (6+3+1). The first 6 primitive shells are s-shells ($\ell = 1$), and are contracted together to make a single s-type basis function. The next three shells are sp-shells ($\ell = 1$), each consisting of (s,px,py,pz) functions. These primitive shells are contracted together to make four atomic orbital basis functions: s, px, py and pz. The outer-most primitive shell is also of sp type, and makes yet another 4 atomic orbital basis functions.

If the program were limited to this definition of shells, one would have to use a set of sp-functions whenever a set of d-functions was desired. Since it is frequently desired to have just a set of d or f type functions, some way must be devised to handle this.

Thus, the idea of a ‘ shell constraint’ is introduced. the shell constraint specifies which functions within a shell are actually employed.

By appropriately setting the shell constraint, one can use only the d portion of an $\ell = 2$ shell.

This general information enables the arrays which appear in `/B/` to be described.

EXX contains the Gaussian exponents for all the the primitive shells. The array `SHELLA` contains pointers into `EXX` for the various primitive shells.

C1 contains the s coefficients for all the primitive shells; indexed by `SHELLA`.

C2 contains the p coefficients for the primitive shells; indexed by `SHELLA`.

C3 this array really consists of 3 sub-arrays (each 80 long) which contain the d and f coefficients plus the d and f pointer table for d and f type shells. these arrays are `C3, C4` and `SHLADF`, respectively.

X the x-Cartesian coordinate for each of the primitive shells.

Y the y-Cartesian coordinate for each of the primitive shells.

Z the z-Cartesian coordinate for each of the primitive shells.

JAN **reserved for future expansion**

SHELLA `SHELLA(I)` contains the starting location within `(EXX,C1,C2)` of the data (exponents, s-coefficients, p-coefficients) for the I'th primitive shell.

SHELLN `SHELLN(I)` contains the number of primitive Gaussians in the I'th primitive shell.

SHELLT `SHELLT(I)` contains the maximum angular quantum number of the I'th shell.

SHELLC `SHELLC(I)` contains the shell constraint for the I'th shell (see table below).

AOS `AOS(I)` gives the starting atomic orbital basis function number (ie number within the list of atomic orbital basis functions) of the I'th shell. note that `AOS` is always filled as though the shell contained all possible lower angular momentum functions.
See `SUBROUTINE GBASIS` for details on how `AOS` is filled.

AON **reserved for future expansion**

NSHELL the number of primitive shells.

MAXTYP the highest angular quantum number present in the basis.

The following table summarizes the relationship between SHELLT AND SHELLC.

TYPE	FUNCTIONS	SHELLT	SHELLC
S	S	0	0
SP	S PX PY PZ	1	0
SPD	S PX PY PZ	2	0
	XX YY ZZ XY XZ YZ		
P	PX PY PZ	1	1
D	XX YY ZZ XY XZ YZ	2	2
F	XXX YYY ZZZ XYY	3	2
	XXY XXZ XZZ YZZ		
	YYZ XYZ		

4.A.4 COMMON /INFO/

```
INTEGER INFO
COMMON /INFO/ INFO(10)
```

This is an area reserved for the communication of miscellaneous information between links. It is little used at the moment.

INFO(1)..(3) not used at present

INFO(4) summary word for the calculation (see `SUBROUTINE ARCSET` in link 1)

INFO(5)..(10) not used at present

4.A.5 COMMON /PHYCON/

```
DOUBLE PRECISION PHYCON
COMMON /PHYCON/ PHYCON(30)
```

This common block holds the values of the physical constants which are used in the program. It is initialized by `SUBROUTINE PHYFIL` in link1 which also documents the sources for the values used.

PHYCON(1) `TOANG`, angstroms per bohr

PHYCON(2) `TOKG`, kilograms per amu

PHYCON(3) `TOE`, esu per electron charge

PHYCON(4) `PLANCK`, Planck's constant

PHYCON(5) `AVOG`, Avogadro number

PHYCON(6) `JPCAL`, joules per calorie

PHYCON(7) `TOMET`, metres per bohr

PHYCON(8) `HARTREE`, joules per Hartree

PHYCON(9) `SLIGHT`, speed of light

PHYCON(10) `BOLTZ`, Boltzman constant

PHYCON(11)..PHYCON(30) not used at present

4.A.6 COMMON /MUNIT/

```
INTEGER IUNIT
COMMON/munit/ iunit(20)
```

Defines the unit numbers for all external data files needed by `gnu80`. `IUNIT` is initialized by `SUBROUTINE DEFUNT` in link 1. The purpose of this COMMON is to centralize the definitions of the FORTRAN logical units required in the Gaussian system. Thus, if it is ever necessary to change to different logical units on another machine, one should be able to merely change the definitions in `DEFUNT`.

Note that in `gnu80`, unit number 18 (munit 7) is not initialized by `DEFUNT`. This unit is "hard wired" by a specific `OPEN` satatement in the `MAIN` segment.

IUNIT(1) not used.

IUNIT(2) primary input (usually channel 5).

IUNIT(3) primary printed output (usually channel 6).

IUNIT(4) primary punched output (usually channel 7 but now obselete).

IUNIT(5) `BINWT` unit (obselete).

IUNIT(6) `BINRD` unit (obselete).

IUNIT(7) Direct Access files and buckets (usually channel 18).

IUNIT(12) Repulsion integrals (usually channel 3).

4.A.7 COMMON /IOP/

```
INTEGER IOP
COMMON/IOP/ IOP(50)
```

The system **OPTIONS** are stored in this COMMON. The definitions of the 50 options differ from overlay to overlay. This COMMON is updated in `FUNCTION NEXTOV`. See B for a full description of the contents of this COMMON block.

4.A.8 COMMON /ILSW1/

```
INTEGER ILSW
COMMON/ILSW1/ ILSW1(2)
```

This is the so-called the Inter-Link Status ‘ Word’ . The inter-link status word is an area on read/write file 998 which is used as a storage place for information required by *several* overlays. The information is packed into the area bit-wise, and `SUBROUTINE ILSW` is designed to facilitate the reading and updating of this information. The use of each bit (or group of bits) is described below.

```
SUBROUTINE ILSW(IOPER,WHERE,WHAT)
INTEGER IOPER, WHERE, WHAT
```

Each of the arguments is described below, first `IOPER`:

IOPER = 1 update the bits indicated by `WHERE` with the value of `WHAT`.

IOPER = 2 . determine the status of the bits indicated by `WHERE` and store the result in `WHAT`.

`WHERE` is the sequence number (in `COMMON/CWD/`) of the information which is to be read or updated.

`WHAT` this argument is used to update the indicated bits if `IOPER`=1, or is returned with the status of these bits if `IOPER`=2. The current possibilities for `WHERE` are:

```
1     CONST    0-1    SCF constraints.
                      0 ... real RHF.
                      1 ... real UHF.
                      2 ... complex RHF.
                      3 ... complex UHF.

2     IPURD    2      number of d functions.
                      0 ... five d.
                      1 ... six d.

3     BASIS    3-5    type of basis set.
```

```
                                 0 ... minimal.
                                 1 ... extended.
                                 2 ... general.

4       POLAR       6-8          polarization functions (p, d).
                                 bit 6=0 ... no p functions on hydrogen.
                                 =1 ... p functions on hydrogen.
                                 bit 7=0 ... no d functions on first row.
                                 =1 ... d on first row.
                                 bit 8=0 ... no d on second row.
                                 =1 ... d on second row.

                                 note that these bits can be addressed
                                 individually (see below).

5       CONVER      9            SCF convergence.
                                 0 ... the SCF has not gone max cycles.
                                 1 ... the SCF has gone max cycles.
                                 6      stabil   10     2nd order stability of wave func
                                 0 ... not tested or not stable.
                                 1 ... tested and found stable.

7       SYM         11           symmetry of univariate search in second.
                                 0 ... search is symmetric for tau pos/neg
                                 1 ... search is not symmetric.

8       PUESS       12           type of guess for {\tt TAU} (see {\tt SCFDM}).
                                 0 ... tau not available.
                                 1 ... tau available.

9       BNREAD      14           whether binrd has tried and failed to
                                 read a file of records.
                                 0 ... no.
                                 1 ... yes.

10      IFPONH      6            polarization on hydrogen (see above).

11      IFDON1      7            polarization on first row (see above).
```

12	IFDON2	8	polarization on second row (see above).
13	IFFON1	15	f functions on first row. 0 ... no. 1 ... yes.
14	IFFON2	16	f functions on second row. 0 ... no. 1 ... yes.
15	IFFON3	17	f functions on third row. 0 ... no. 1 ... yes.
16	F7F10	18	number of f functions. 0 ... seven f. 1 ... ten f.
17	IFPOL	15-17	used to address these bits collectively. (see description above.)
18	IFAU	13	if coordinates in blank COMMON are in atomic units. 0 ... no, angstroms instead. 1 ... yes.
19	IFFP	19	if this is a Fletcher-Powell optimization run. 0 ... no. 1 ... yes.
20	PRTOFF	20	control print in nextov and chain 0 ... print turned on 1 ... print turned off
21	PSAVE	21	paper save master switch. 0 ... (off) paper used as normal. 1 ... (on) paper usage drastically reduced.

```
22    OPCLO   22     closed/open selector.
                      0 ... closed shell.
                      1 ... open shell.
                      note: this bit is used for RHF open
                      shell.

23    IFGRD   23    if this is a gradient
                      optimization run
                      0 ... no.
                      1 ... yes.

24    IFRC    24    if energy derivatives are to be
                      calculated
                      0 ... yes.
                      1 ... no.

25    IFARCH  25   if a summary of the job is to be placed
                      into the archive file.
                      0 ... no.
                      1 ... yes.

26    NOSYM   26-27 should the point group symmetry of the
                      molecule be used to eliminate
                      redundant computation
                      0 ... yes.
                      1 ... no, the point group will be
                      computed and printed only.
                      2 ... no, a change in the point group
                      has been detected during an
                      optimization.
```

4.A.9 COMMON /GEN/

```
DOUBLE PRECISION DGEN
COMMON/GEN/DGEN(47)
```

This is a COMMON area where many of the ‘ single number’ results of the current calculation are stored. The value of `DGEN(I)` for the various values of I are given below:

1 electronic configuration of SCF wave-function.

2-20 not used.

21 root-mean-squared force of optimized parameters.

22 dipole moment.

23 rms error in density matrix.

24-30 not used.

31 `TAU` from `SCFDM`

32 SCF energy.

33 UMP2 energy.

34 UMP3 energy.

35 UMP4(sdtq) energy. (reserved for future use)

36 CID energy, or `E(VAR1)`. (reserved for future use)

37 CISD energy. (reserved for future use)

38 MP4DQ energy. (reserved for future use)

39 MP4SDQ energy (reserved for future use)

40 CCD energy. (reserved for future use)

41 nuclear repulsion energy.

42 T (length of correction of reference determinant).

43 updated energy for Fletcher-Powell optimizations.

44 $< \hat{S}^2 >$ of SCF wave function.

45 $< \hat{S}^2 >$ corrected to first order (after `DOUBAR`).

46 $< \hat{S}^2 >$ corrected for doubles (not implemented).

47 a0.

Appendix E

Stubs, Ghosts and Error Messages

E.1 Stubs and Ghosts

Stubs and Ghosts are either the remains of earlier stages in the development of a program or unfulfilled promises for the future development of the system.

Stub A Stub is a program segment (`SUBROUTINE` or `FUNCTION` which has a correct interface to be called but actually does nothing (except possibly output a message that it is a Stub) and returns control to the caller immediately. Its existence may be evidence of the intention to provide a new facility or that a redundant facility has been removed.

Ghost A Ghost is a piece of code which is *in practice* unreachable. Again, it may be evidence of good intentions or of debugging. Most compilers will object to the presence of code which is *in principle* unreachable (un-numbered statements following a transfer of control, for example) but they cannot detect situations like

```
      I = 0
      IF(I .EQ. 0) GO TO 999
C     Unreachable
      A = B
```

`gnu80` has many Stubs and Ghosts which exist for both types of reason. In particular, there are Stubs which are generated by the removal of machine-specific (Assembler) segments. They could be removed by systematically deleting the (useless) calls.

E.2 Error Messages

There is no explicit list of error messages generated by `gnu80` because they are just that; error *messages* not error *codes*. They are mostly completely self-explanatory.

Appendix F

Installation of gnu80

Insofar as the source code of `gnu80` is designed to be portable FORTRAN 77, there are no installation instructions except "Generate the Fortran source from the FWEB files, Compile the FORTRAN and link it to form an executable module" .
However, if you need a simpler route, go to the directory `gnu80web` and change the first line to read: `PREFIX="directory where you are"` and then say:

```
./build_gnu80 > & build_gnu80.log &
```

After some time you should find the file `a.out` has been created in that directory together with a (large) `build_gnu80.log` file with all the gory details of what has happened during the build.
If you do not have (or do not like) FWEB then you will have to go into the `src` and `NBO` directories and edit off all the FWEB details from the Fortran source which will be *very* tedious.

However, there are one or two points where the FORTRAN 77 standard is not precise enough to ensure complete portability and it is wise to be aware of these:

- The *units* of the `RECL` (record length) parameter in a direct access

`OPEN` statement

```
        OPEN(UNIT=file,ACCESS='DIRECT',RECL=record_length, ...)
```

is not defined by the standard; typically it may be bytes or quadruples of bytes.
In `gnu80` as supplied a unit of *bytes* is assumed and `RECL` is set to 16380 (=4*4095).
To use the existing file I/O routines (via `NTRAN`) the record length must be enough to store 4095 default-length `INTEGERS`. If a different buffer is required, then `SUBROUTINE NTRAN` must have the occurences of 4095 changed and the `RECL` parameter changed accordingly.

- The mode of storage of (particularly) characters in variables of type `INTEGER` is not defined by the standard (see `IORD` D.10).

- The coding used to store characters is not defined although the ASCII code is almost universally implemented. `gnu80` asumes that the ASCII code is used but it is hoped that the use of EBCDIC will not cause difficulties!

In addition to these points *within* the standard, there is an important and systematic departure from the standard in this release of `gnu80`. Characters are stored in variables of non-`CHARACTER` type either by reading in A-format or by `DATA` initialisation statements. This is endemic in the code and cannot be eradicated from this release!

If the storage of `CHARACTER` constants in variables of non-`CHARACTER` type is a compilation *error* on the target machine/compiler, not simply a *warning*, then this release of `gnu80` cannot be installed on that system.