NUMERICAL INTEGRATION OF THE ELECTRON DENSITY

AISHA EL-SHERBINY

Your file Votre référence
ISBN: 0-612-84000-X
Our file Notre référence
ISBN: 0-612-84000-X

NUMERICAL INTEGRATION OF THE ELECTRON DENSITY

by

© Aisha El-Sherbiny
B.Sc. (Ain Shams University), M.Sc. (University of Cairo)

A thesis submitted to the
School of Graduate Studies
in partial fulfillment of the
requirements for the degree of
Master of Science.

Department of Chemistry
Memorial University of Newfoundland

February, 2002

ST. JOHN'S NEWFOUNDLAND

To my father's soul and my mother.

Abstract

In quantum chemistry three-dimensional integrals of the type

$$I = \int F(\mathbf{r})\, d\mathbf{r},$$

are common. In general, the integrand $F(\mathbf{r})$ extends over all the molecular space. Sometimes, as occurs in density functional theory, the integrals can not be solved analytically and numerical approximations must be used. In molecules, the integrand $F(\mathbf{r})$ is dominated by cusps at atomic nuclei. A popular solution to this multicenter integration problem is the nuclear weight function scheme proposed by A. D. Becke. Two algorithms based on Becke's approach were developed by P. M. W. Gill et al. and by Becke. The latter was slightly modified by O. Treutler and R. Ahlrichs. These two different algorithms were written in Fortran 90 and incorporated in MUNgauss.This work investigates in detail Becke's scheme and the application of the previous two algorithms to integrate the charge density of a set of test molecules.

Acknowledgements

There are several individuals I would like to thank for their guidance and support while this work was being completed. First, I would like to thank my supervisor Raymond Poirier for his advice and assistance which was invaluable in the completion of this work. Also, I would like to thank the Departments of Chemistry and Physics for the use of their facilities and resources and my colleagues Tammy Gosse, and Darryl Reid for their continued support and encouragement. Thanks to the School of Graduate Studies, the Computational Science committee, the National Sciences and Engineering Research Council of Canada, and Memorial University of Newfoundland for financial support and giving me the opportunity to complete my Masters programme here.

I would like to extend a special thanks to my family and friends, especially my wonderful mother. Their continued support and encouragement throughout the completion of this project will always be greatly appreciated.

Contents

List of Tables

List of Figures

Glossary of terms and symbols

K: number of basis functions.

m: degree of the quadrature formulae.

M: number of nuclei.

n: number of dimensions of the space under consideration.

N: number of nodes in the integration interval.

N_a: number of atoms in a molecule.

N_e: number of electrons.

ρ: charge density.

R : atomic radius.

Ω: domain of integration.

P: density matrix.

ϕ: basis function, or a general function in Numerical Integration.

w_i: weight factor.

r_i: radial point.

N^r: number of radial points.

N^Ω: number of angular points.

TA grid: O. Treutler and R. Ahlrich numerical integration grid.

SG-1 grid: standard grid for numerical integration, P. M . W. Gill et al.

Chapter 1

Introduction

1.1 The need for numerical integration

Many problems of a scientific nature, in particular those arising in applied mathematics, theoretical physics, theoretical chemistry or astronomy may be formulated in terms of derivatives, the solutions of which may involve the evaluation of integrals. In practice, for most integrals, it is impossible to derive analytical solutions, and in order to evaluate them, one must apply techniques of numerical integration. This is the case in density functional theory, DFT, which is used to calculate molecular properties. Integrating the charge density obtained in ab initio calculations is another area of interest. The rest of this chapter represents the role of numerical integration in these areas. A brief historical background is introduced as well.

1.2 Density Functional Theory

For a molecular system of N_e-electrons and M-nuclei, the Schrödinger equation is

$$\hat{H}\Psi = E\Psi \tag{1.1}$$

E is the total energy of the system, $\hat{H}$ is the Hamiltonian operator, and Ψ is the wave function describing the system. The electronic Hamiltonian operator for molecular systems may be written, in atomic units, within the Born–Oppenheimer approximation, as

$$\hat{H} = \hat{T} + \hat{V}_{ne} + \hat{V}_{ee} \tag{1.2}$$

$$= -\frac{1}{2}\sum_{i=1}^{N_e} {\nabla_i}^2 - \sum_{i=1}^{N_e}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N_e}\sum_{j>i}\frac{1}{r_{ij}} \tag{1.3}$$

The first term represents the kinetic energy of the electrons, while the second term is the potential energy due to the electrostatic attraction between the electrons and the nuclei bearing nuclear charges Z_A. The last term represents the potential energy due to electron-electron repulsion.

Hohenberg – Kohn [1] showed that there exists a unique functional of the electron density ρ that yields the exact ground state energy of the system. Kohn and Sham [2] connected the exact functional with a reference state consisting of a set of non-interacting one-particle orbitals. The set of one-particle Schrödinger equations which determine the reference orbitals are called the Kohn-Sham (KS) equations [3]

$$(-\frac{1}{2}\nabla_i^2 + \hat{V}_{ks})\,\psi_i = \varepsilon_i\,\psi_i \tag{1.4}$$

where $\hat{V}_{ks}$ is a local one-body potential defined such that the total density ρ of the non-interacting system is the same as the real system, where

$$\rho(\mathbf{r}) = \sum_{i=1}^{N_e} \mid \psi_i \mid^2 \tag{1.5}$$

The KS potential $\hat{V}_{ks}$ has three components:

$$\hat{V}_{ks} = \hat{V}_{ne} + \hat{v}_{el} + \hat{v}_{xc} \tag{1.6}$$

where $\hat{v}_{el}$ is the electron-electron repulsion Coulomb potential of the electrons, $\hat{V}_{ne}$ is as before, and $\hat{v}_{xc}$ is the exchange-correlation (XC) potential, which contains everything else. The exact energy of the fully interacting system is expressed as

$$E = E_k + \int \rho(\mathbf{r})\, \hat{V}_{ne}\, d\mathbf{r} + \frac{1}{2} \int \int \frac{\rho(\mathbf{r}_1)\, \rho(\mathbf{r}_2)}{r_{12}}\, d\mathbf{r}_1\, d\mathbf{r}_2 + E_{xc} \tag{1.7}$$

where E_k is the kinetic energy of the non-interacting reference system, the second term represents the attraction energy between the electrons and the nuclei, the third term is the Coulomb repulsion energy between electrons, and the remaining term is the exchange–correlation energy related to the exchange–correlation potential by

$$v_{xc} = \frac{\delta E_{xc}}{\delta \rho} \tag{1.8}$$

The exact exchange–correlation functional is unknown and E_{xc} maybe approximated by

$$E_{xc} = \int F(\rho, \nabla\rho, \ldots)\, d\mathbf{r} \tag{1.9}$$

where F is a function of the electron density $\rho(\mathbf{r})$. The function F is too complicated to be integrated analytically. Recently, a new theorem by P. G. Mezey[4], stronger than the Hohenberg-Kohn theorem, has proven that all molecular information is also present in any nonzero volume part of the electron density. This theorem is known as the Holographic Electron Density Theorem. In chapter 3 some of the techniques to evaluate integrals like the ones in Eqs. 1.7 and 1.9 will be presented. In particular, Becke's scheme [5] for calculating numerical integration of functions $F(\mathbf{r}), \mathbf{r} \in R^3$, associated with molecular properties such as the charge density.

1.3 Molecular orbital charge density

The charge density for a closed shell molecule is given by [6]

$$\rho(\mathbf{r}) = 2 \sum_{a}^{N_e/2} \psi_a(\mathbf{r})\, \psi_a^*(\mathbf{r}) \tag{1.10}$$

where ψ_a is the a^{th} molecular orbital and N_e is the total number of electrons. ψ_a can be expanded in terms of the basis functions $\phi_i(\mathbf{r})$, $i = 1, \ldots, K$ in the form

$$\psi_a = \sum_{i=1}^{K} C_{ia}\, \phi_i. \tag{1.11}$$

Substituting Eq. 1.11 in Eq. 1.10 we get

$$\rho(\mathbf{r}) = 2 \sum_{a}^{N_e/2} \sum_{i} C_{ia}\, \phi_i(\mathbf{r}) \sum_{j} C_{ja}^*\, \phi_j^*(\mathbf{r}) \tag{1.12}$$

$$= \sum_{ij} P_{ij}\, \phi_i(\mathbf{r})\, \phi_j^*(\mathbf{r}) \tag{1.13}$$

where,

$$P_{ij} = 2 \sum_{a}^{N_e/2} C_{ia} C_{ja}^{*} \tag{1.14}$$

is the density matrix. The charge density $\rho(\mathbf{r})$ can be integrated to give the total number of electrons belonging to each atom in a molecule. However, an integration of the form

$$\int \rho(\mathbf{r})\, d\mathbf{r} = \sum_{ij} P_{ij} \int \phi_i(\mathbf{r})\, \phi_j^{*}(\mathbf{r})\, d\mathbf{r} = N_e \tag{1.15}$$

can not be, generally, carried out analytically.

1.4 Electron density for large molecules

P. Duane Walker and Paul G. Mezey [7,8] developed a new technique, Molecular Electron Density Lego Assembler (MELDA), to construct the electron density distribution of large molecules by combining distributions of small fragments present in the molecule. The method assumes that the contribution to the total molecular electron density of a molecular fragment on different molecules or different locations of the same molecule is quite similar provided that the molecular environments are similar. First, a database of ab initio electron densities of various molecules is generated where the fragments are obtained from smaller molecules for which ab initio computations are economic. The large molecule whose density distribution is being constructed is partitioned into fragments which appear in the database. From Eq. 1.13, the electron density is given by

$$\rho(\mathbf{r}) = \sum_{ij} P_{ij}\, \phi_i(\mathbf{r})\, \phi_j^{*}(\mathbf{r}). \tag{1.16}$$

The density matrix for a molecule can be partitioned into NxN fragment density matrices (P^l) where,

$$\begin{aligned} P^l_{ij} &= P_{ij} \text{ if both } \phi_i(\mathbf{r}) \text{ and } \phi_j(\mathbf{r}) \text{ are atomic basis functions centered} \\ &\quad \text{on nuclei of the fragment} \\ &= \frac{1}{2}P_{ij} \text{ if } \phi_i(\mathbf{r}) \text{ or } \phi_j(\mathbf{r}) \text{ is centered on a nucleus of the fragment} \\ &= 0 \text{ otherwise} \end{aligned}$$

For every partitioning of a molecule into M_{frag} molecular fragments there will be M_{frag} fragment density matrices $P^l, l = 1, \cdots, M_{frag}$. The total density matrix P is the sum of the M_{frag} fragment density matrices,

$$P_{ij} = \sum_{l=1}^{M_{frag}} P^l_{ij} \tag{1.17}$$

The electron density $\rho^l(\mathbf{r})$ of a given molecular fragment is

$$\rho^l(\mathbf{r}) = \sum_{i=1}^{K}\sum_{j=1}^{K} P^l_{ij}\,\phi_i(\mathbf{r})\,\phi^*_j(\mathbf{r}) \tag{1.18}$$

and the total electron density of a molecule is the sum of the densities for the fragments,

$$\rho(\mathbf{r}) = \sum_{l=1}^{M_{frag}} \rho^l(\mathbf{r}) \tag{1.19}$$

In this way Walker–Mezey were able to calculate the charge density for large molecules with more than 1000 atoms. Yet, this is another area where numerical integration is a valuable tool to integrate the charge density and obtain the number of electrons belonging to each atom, and for energy calculations using DFT, as well. The fact

that numerical integration algorithms are very suitable for parallelizing makes it even more valuable in ab initio calculations for very large molecules, such as proteins.

1.5 Historical background

According to the Oxford English Dictionary, cubature is the determination of the cubic contents of a solid, that is, the computation of a volume. Thus cubature formulae are formulae which estimate volumes [9].

Ancient Egyptians and Babylonians were the first to try to derive cubature formulae. They had precise and accurate rules for finding the areas of triangles, trapezoids, circle, and the volumes of parallelpipeds, pyramids, and cylinders (for the Babylonians π equalled 3 for the Egyptians 235/81) [10].

Around 450 B.C., one of Pythagoras' pupils, Bryson, is thought to have been a major contributor to the method of exhaustion which effectively covers the area between a curve and an inscribed polygon by successively increasing the number of sides of the polygon. Around 430 B.C., Antiphon considered the area of a circle using the same approach. About 60 years later, Eudoxus used the method to find the volumes of the cone and the cylinder. By 225 B.C. Archimedes had used the method of exhaustion for finding the area of a segment of a parabola and the volume of a sphere.

The start of the modern study of volume computation is usually linked to Kepler (1615). Kepler computed the volume of the cask by discretizing the volume into a series of shallow cylinders. The first cubature formula was constructed by Maxwell (1877). The MonteCarlo method, developed in 1945, is another way to integrate a function. Other quasi-MontCarlo methods have evolved since then.

1.6 Outline

Chapter 2 presents a background for numerical integration in one- and multi-dimensions. In chapter 3 an overview of numerical integration schemes designed especially for polyatomic systems will be presented. In particular Becke's method will be reviewed in some detail. Chapter 4 introduces the standard grid SG-1 proposed by Peter M.W.Gill et al. along with a description of the three-dimensional grids developed by O. Treutler and R. Ahlrichs. Chapter 5 is an overview of the new features added to FORTRAN 77, namely Fortran 90. Chapter 6 investigates the application of SG-1 on a variety of molecules. Chapter 7 is the conclusion of this work along with some ideas for future work.

Chapter 2

Numerical integration

2.1 One dimensional integration

Given a function $Y = \phi(x)$ defined on the interval $[a, b]$, the general procedure to integrate this function numerically is to consider the integral as the area under the curve $Y = \phi(x)$, Figure 2.1. If we divide the range of integration $[a, b]$ into $N-1$ equal intervals (x_i, x_{i+1}) of length h and height $\phi(x_i)$, $i = 1, 2, .., N$ with $x_1 = a$ and $x_N = b$, then the integral of $\phi(x)$ as $N \to \infty$, can be written as

$$I = \int_a^b \phi(x)\, dx = \lim_{N\to\infty} \sum_{i=1}^{N} h\, \phi(x_i) \tag{2.1}$$

this summation is known as a quadrature. In general the intervals do not need to be equal. A more general approximation can be given by:

$$I_N = \int_a^b \phi(x)\, dx = \sum_{i=1}^{N} w_i\, \phi(x_i). \tag{2.2}$$

The weight factors, w_i, are proportional to the interval widths between the nodes x_i.

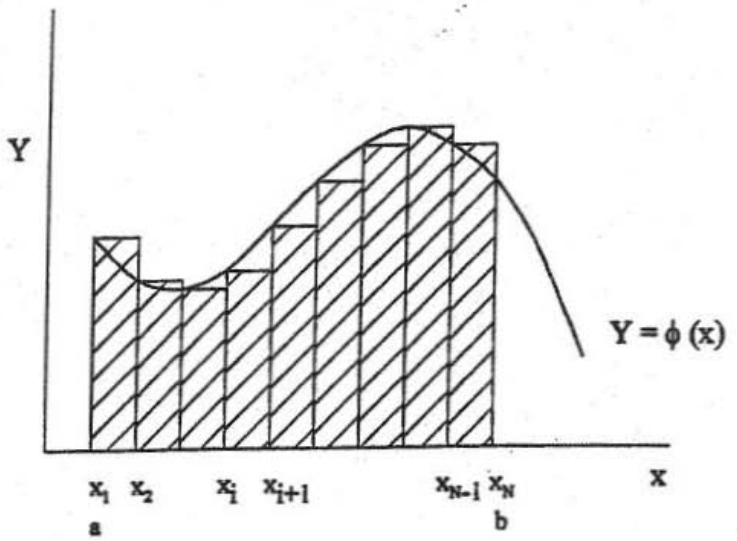

Figure 2.1: Numerical integration of a simple function $Y = \phi(x)$

The points x_i are also called abscissae. Numerical integration [10,11,12,13,14,15] methods differ in the way they select the abscissae. Some methods choose the base points for convenience, e.g., to correspond to discrete data points. Other methods are developed to give a 'best' approximation to the integral and the abscissae are positioned to achieve the highest accuracy. Both methods approximate the function $\phi(x)$ by interpolating polynomial $P_m(x)$ of degree m. A polynomial is chosen such that it intersects $\phi(x)$ at the abscissae i.e.,

$$P_m(x_i) = \phi(x_i) \qquad i = 1, 2, ..., N \tag{2.3}$$

Two simple polynomials of degree one and two are shown in Figure 2.2.

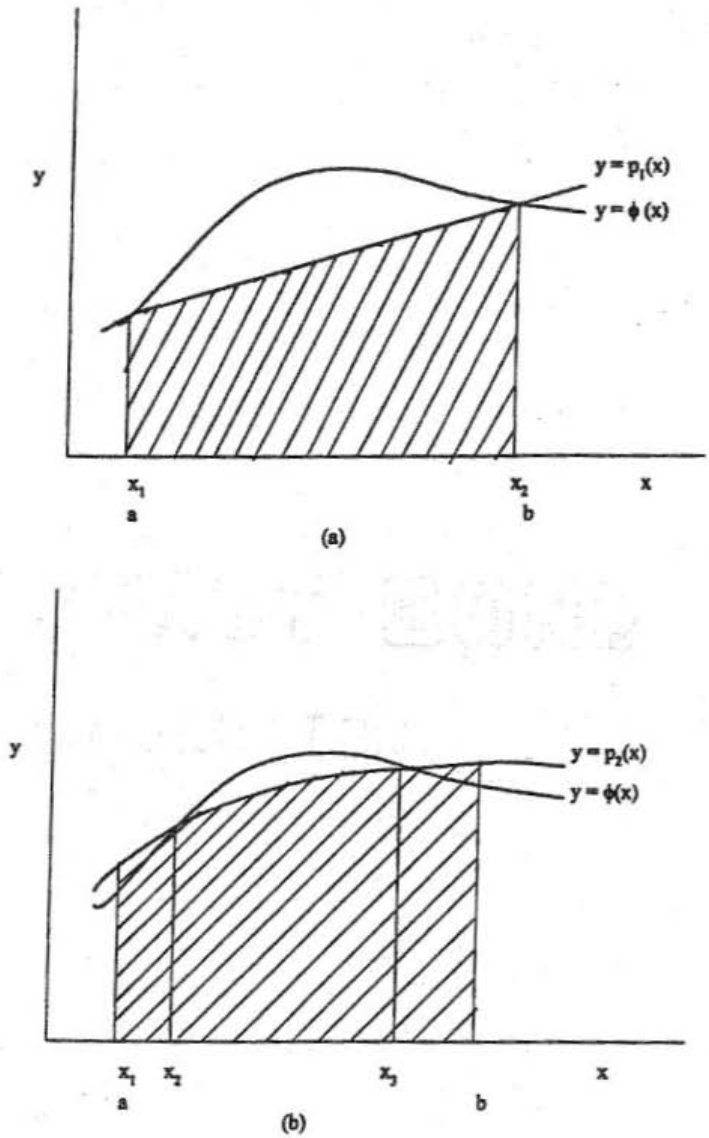

Figure 2.2: Numerical integration of a function by approximation with polynomials of degrees: (a) $m = 1$, (b) $m = 2$

Eq. 2.2 can also be written as

$$I_N = \int_a^b w(x)\, f(x)\, dx = \sum_{i=1}^{N} c_i\, f(x_i) \tag{2.4}$$

where

$$\phi(x) = w(x)\, f(x)$$

and c_i are the weight factors.

The function $w(x)$ is called the weight function. It is a part of the integrand that could arise so frequently that it is usual to write it as a separate entity within the integrand [13]. It could be an awkward part like a singularity. The weight function does not explicitly appear in the resulting quadrature rule. This implies that the weights and the abscissae of any resulting quadrature rule will depend on $w(x)$. For example

$$w(x) = \frac{1}{\sqrt{1-x^2}} \tag{2.5}$$

in the case of Gauss–Chebyshev formula of the first kind (see Eq. 2.13 below). When $w(x) = 1$ and the abscissae are equally spaced, the quadrature formulae are known as Newton–Cotes type quadrature formulae. Examples of these quadratures are the Trapezoidal rule and Simpson's rule. The Trapezoidal rule is obtained when the integrand $f(x)$ is fitted by the straight line through the end points $(a, f(a))$ and $(b, f(b))$ [10]. That is

$$\int_a^b f(x)dx \simeq \int_a^b (ux + v)dx = \frac{(b-a)}{2}(f(a) + f(b)) \tag{2.6}$$

where

$$u = \frac{f(b) - f(a)}{b - a} \tag{2.7}$$

and

$$v = \frac{bf(a) - af(b)}{b-a}. \tag{2.8}$$

Simpson's rule is generated by fitting a quadratic to $f(x)$ at the three points $(a, f(a)), (b, f(b))$ and $(c, f(c))$ with $c = (a+b)/2$. This process yields:

$$\int_a^b f(x)dx \simeq \int_a^b (ux^2 + vx + w)dx = \frac{h}{3}(f(a) + 4f(c) + f(b)) \tag{2.9}$$

where

$$u = \frac{(c-b)f(a) + (a-c)f(b) + (b-a)f(c)}{(c-a)(b-c)(a-b)} \tag{2.10}$$

$$v = \frac{(b^2-c^2)f(a) + (c^2-a^2)f(b) + (a^2-b^2)f(c)}{(c-a)(b-c)(a-b)} \tag{2.11}$$

$$w = \frac{bc\,(c-b)f(a) + ac\,(a-c)f(b) + ab\,(b-a)f(c)}{(c-a)(b-c)(a-b)} \tag{2.12}$$

and h is the distance between two nodes. Trapezoidal formula has degree one, Simpson's rule has degree three although it was designed to integrate a quadratic.

When the abscissae are not equally spaced, the quadrature formulae are said to be of Gaussian type. Examples of Gaussian quadrature rules are given in Table 2.1.

The Gauss–Chebyshev rules stand out from the other rules in that their weights and abscissae can be expressed in closed analytic form, rather than as tables of high precision real numbers. The Gauss-Chebyshev formula of the first kind is given by [13],

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)\,dx \simeq \frac{\pi}{N} \sum_{i=1}^{N} f(x_i). \tag{2.13}$$

Table 2.1: Examples of Gaussian quadrature rules

$w(x)$	interval of integration	quadrature rule
1	$[1,-1]$	Gauss-Legendre
$(1-x^2)^{-1/2}$	$[1,-1]$	Gauss-Chebyshev
$\exp(-x^2)$	$(-\infty,\infty)$	Gauss-Hermit
$\exp(-x)$	$[0,\infty]$	Gauss-Laguerre

The weights are all equal, π/N, for all i and the abscissae x_i the zeros of the Chebyshev polynomial

$$T_N(x) = \cos(N \arccos(x)) \tag{2.14}$$

are given by

$$x_i = \cos\frac{(2i-1)\pi}{2N}, \quad i = 1, \ldots, N. \tag{2.15}$$

The Gauss-Chebyshev formula of the second kind is

$$\int_{-1}^{1} \sqrt{1-x^2}\, f(x)\, dx \simeq \sum_{i=1}^{N} c_i\, f(x_i) \tag{2.16}$$

where

$$c_i = \frac{\pi}{N+1} \sin^2(\frac{i\pi}{N+1}) \tag{2.17}$$

and

$$x_i = \cos(\frac{i\pi}{N+1}). \tag{2.18}$$

2.2 From one-dimensional to multi-dimensional integration

The problem to be addressed with multi-dimensional integrals is to evaluate [10,12]

$$I = \int \cdots \int_{\Omega} \Phi(x_1, x_2, \ldots, x_n)\, dx_1\, dx_2 \ldots dx_n \tag{2.19}$$

where Ω is a region in the n-dimensional space, $dx_1 dx_2 \cdots dx_n$ is the n-dimensional volume element. The usual approach is to treat Eq. 2.19 as a recursive set of one-dimensional integrals to yield a product rule of the form

$$I \simeq \sum_{i_1=1}^{N_1} \cdots \sum_{i_n=1}^{N_n} w_{i_1}\, w_{i_2} \ldots w_{i_n}\, \phi(x_{i_1}, x_{i_2}, \ldots, x_{i_n}) \tag{2.20}$$

where the weights w_{ij} and the abscissae x_{ij} are chosen to be appropriate for the specific dimension to which they are applied, $N_1, N_2, \ldots, N_n$ are the number of abscissae in the n-dimensions. In passing from one-dimension to several dimensions, a number of new difficulties arise by comparison with the one-dimensional case. In one-dimension we can restrict ourselves to three different types of integration: the finite interval, the singly infinite interval, and the doubly infinite interval. Whereas in several dimensions, there are potentially an infinite number of different types of regions. Another difficulty is that the behavior of functions of several variables can be considerably more complicated than that of functions of one variable. It is also apparent that each dimension will require some minimum number of integration points for suitable accuracy, therefore, even for quite small n, a large number of function evaluations can be expected. Non-product rules may exist which have an equivalent degree m to a given product rule but with fewer points. In two dimensions, some

formulae have been published for the regular polygons, the square, the triangle, the circle, and the surface of a unit sphere [11]. In three and more dimensions, a few formulae are known for special regions like the n-cube and the n-sphere (n-dimensional space). In practice, the more expensive product rule yields far greater accuracy than the non-product alternative.

Not all regions of integration are feasible for the use of a product rule, but more complicated regions can often be handled by breaking them up into subregions and applying a product rule to each subregion.

2.3 n-dimensional formulation of the numerical integration problem

Assume that $\phi(\mathbf{x})$ is a function continous in some domain $\overline{\Omega}$ in R^n (n-dimensional space) and integrable over a bounded subdomain Ω with $\Omega \subset \overline{\Omega}$. The main problem of numerical integration consists in approximating the integral [15]

$$I(\phi) = \int_{\Omega} \phi(\mathbf{x})\, d\mathbf{x} \tag{2.21}$$

where $\mathbf{x}$ is an $n-$dimensional coordinate vector. The integral over Ω of $\phi(\mathbf{x})$ can be replaced by a linear combination of the values of $\phi(\mathbf{x})$ at the N points of the set

$$K = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\} \subset \overline{\Omega} \tag{2.22}$$

namely

$$I_N(\phi) = \sum_{i=1}^{N} c_i\, \phi(\mathbf{x}^{(i)}), \qquad \mathbf{x}^{(i)} \in K \tag{2.23}$$

This gives the approximate equality

$$I(\phi) \approx I_N(\phi). \tag{2.24}$$

As in the case of one-dimension, the numbers $c_i \in R$ are called the weights, and the points $\mathbf{x}^{(i)} \in R^n$ are called the nodes. The nodes of K that lie outside Ω are not excluded. However, it is preferable that the nodes to be inside the region Ω because it is not always known in advance whether it is possible to evaluate the integrand in a point outside Ω. The weights c_i are positive. If $n = 1$ then I_N is called a quadrature formula. If $n \geq 2$ then I_N is called a cubature formula. The choice of the nodes $\mathbf{x}^{(i)}$ and the weights c_i is independent of the function ϕ. They are chosen so that the formula gives a good approximation for some class of functions. The error of the cubature formula is given by

$$e = I(\phi) - I_N(\phi) \tag{2.25}$$

The cubature formula for $\phi(\mathbf{x})$ is exact when the error is zero. It is natural to consider sequences of cubature formulae with errors $e^{(N)}$ as the number of nodes N increases. In this case we speak of a cubature process. A cubature process converges if for every function in $\overline{\Omega}$ and integrable over Ω , the quantity $I_N(\phi)$ converges to $I(\phi)$ as $N \to \infty$. Convergence of the error to zero may be strong or weak. The cubature formula 2.23 is exact for a given set of functions

$$\phi_1, \phi_2, \ldots, \phi_M \tag{2.26}$$

i.e., the error is zero, if it is exact for each member ϕ_i. The problem of determining the weights c_i of an exact formula is dual to the interpolation problem of finding a

linear combination of the functions given in Eq. 2.26, e.g.,

$$\Phi = a_1\phi_1 + a_2\phi_2 + \ldots + a_M\phi_M\,, \tag{2.27}$$

taking preassigned values at the points given by Eq. 2.22. The case in which the functions 2.26 consist of polynomials is the most common. If m is the highest degree of these polynomials, the interpolation problem is to find a polynomial $p(\mathbf{x})$ of degree at most m agreeing with $\phi(\mathbf{x})$ at the given points,

$$p(\mathbf{x}^{(k)}) = \phi(\mathbf{x}^{(k)})\,, \quad k = 1, 2, \ldots, N. \tag{2.28}$$

Thus the problem of approximate integration reduces to construction of cubature formulae exact for polynomials of degree m. The cubature formulae arising in this way are often called formulae of interpolatory type.

Let P^n be the vector space of all polynomials in n variables, and P_m^n be a subspace of P^n with all polynomials of degree at most m. A cubature formula I_N for an integral I has degree m if [14]

$$I_N(p) = I(p)\,, \quad \forall p \in P_m^n \tag{2.29}$$

and

$$\exists g \in P_{m+1}^n, \text{ such that } I_N(g) \neq I(g) \tag{2.30}$$

i.e. the integer m is the order of a cubature formula 2.23 if it is exact for a polynomial of degree m and not exact for a polynomial of degree $m+1$. Because P_m^n is a vector space , the conditions 2.29 and 2.30 are equivalent to

$$I_N(p_j) = I(p_j)\,, \qquad j = 1, 2, \ldots, \text{ dim } P_m^n \tag{2.31}$$

where p_j form a basis for P_m^n . If the basis and the number of nodes N are fixed, then conditions 2.31 form a system of nonlinear equations

$$\sum_{i=1}^{N} c_i\, p_j(\mathbf{x}^{(i)}) = I(p_j)\,, \quad j = 1, 2, \ldots, \text{ dim } P_m^n. \tag{2.32}$$

Each equation in 2.32 is a polynomial equation. Each node introduces $n+1$ unknowns: the weight and the n coordinates of the node. A cubature formula of degree m with N nodes is determined by a system of dim P_m^n nonlinear equations in $N(n+1)$ unknowns.

2.4 Rotation invariant cubature formulae

If the integration domain admits a sufficiently simple finite group of transformations to itself, the calculation of the nodes and weights of a cubature formula invariant under this transformation group maybe very economical [15].

Definition 2.1 *Let G be a rotation group in R^n consisting of the elements $g_1, g_2, \ldots, g_q$ where q is the order of G. All points of the shape $g_i\mathbf{x}^{(k)}$, $\mathbf{x}^{(k)} \in R^n$, make a $G-$orbit of the point $\mathbf{x}^{(k)}$.*

Definition 2.2 *A set $\Omega \in R^n$ is said to be invariant with respect to a group G if $g(\Omega) = \Omega$ for all $g \in G$. An invariant polynomial of G is a polynomial p which is left unchanged by every transformation in G. The vector space of all invariant polynomials of G is denoted by $P_n(G)$ and the subspace of $P_n(G)$ with only the polynomials of degree $\leq m$ is denoted by $P_m^n(G)$.*

Definition 2.3 *A cubature formula is said to be invariant with respect to a group G, or simply $G-$invariant, if the integration domain Ω is invariant under G and the set*

of nodes $\mathbf{x}^{(k)}$ *is the union of* $G-$*orbits, and the nodes of the same orbit have the same weight.*

Sobolev's theorem

Let the cubature formula I_N be G-invariant, then I_N has degree m if

$$I(p) = I_N(p) \quad , \forall p \in P_m^n(G)$$

and

$$\exists\, g \in P_{m+1}^n \text{ such that } I(g) \neq I_N(g).$$

Sobolev's theorem suggests that we look for invariant cubature formulae, that is, solutions of the equations

$$I_N(p_j) = I(p_j), \quad j = 1, 2, \ldots, \text{ dim } P_m^n(G) \tag{2.33}$$

where p_j form a basis for $P_m^n(G)$. Thus the number of linearly independent polynomials for which the cubature formula must be exact equals dim $P_m^n(G)$ which is less than dim P_m^n. In this way the problem of determining the nodes and weights of the formula becomes much simpler.

It is convenient to view a rotation group as a group of transformations of the unit sphere S_n of R^n into itself [15]. For $n \geq 3$ there are finite rotation groups of regular polyhedra. The group that corresponds to an n-dimensional N-hedron is denoted by G_n^N. For $n = 3$ we have the following nonequivalent finite rotation groups of the sphere S_3 which keep invariant some regular polyhedron: the tetrahedral group G_3^4,

the octahedral group G_3^8 and the icosahedra group G_3^{20}. From now, the subscript 3 will be dropped. The group G^* can be generated from the group G by adding the reflection transformations to the group G. The reflection transformation assigns to a point of the unit sphere with coordinates θ, ϕ the point θ_1, ϕ_1 with

$$\theta_1 = \pi - \theta \quad \text{and} \quad \phi_1 = \pi + \phi$$

Lemma

Each polynomial $p(x)$ invariant under the octahedral group with reflection $(G_n^{2^n})^*$ is uniquely representable as a polynomial in $\sigma_1, \sigma_2, \ldots, \sigma_n$ where $\sigma_1, \sigma_2, \ldots, \sigma_n$ are symmetric functions given by [15]

$$\sigma_1 = \sum_{i=1}^{n} {x_i}^2, \qquad \sigma_2 = \sum_{i<j} {x_i}^2 {x_j}^2, \cdots, \tag{2.34}$$

$${\sigma_n}^2 = {x_1}^2 {x_2}^2 \ldots {x_n}^2 \tag{2.35}$$

2.5 Rotation invariant cubature formula on the sphere in R^3

This section describes, briefly, Lebedev's scheme [16,17,18] for the integration over the surface of a sphere. Let S be a unit sphere of R^3 , $S = \{(x, y, z) : x^2+y^2+z^2 = 1\}$. We can construct a cubature formula

$$I(f) = \int_S f(x, y, z)\, dS \simeq I_N(f) \tag{2.36}$$

so that it is G^{8^*}-invariant. Let $\mathcal{H}$ be the octahedron inscribed in the sphere and whose rotations generate G^{8^*}. Let its vertices be on the coordinate axes and given by:

$$\theta = 0; \quad \theta = \frac{\pi}{2}, \quad \phi = 0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}; \quad \theta = \pi \tag{2.37}$$

in the spherical coordinates (θ, ϕ). A sought cubature formula takes the form

$$\begin{aligned} I_N(f) &= A_1 \sum_{i=1}^{6} f(a_i^{(1)}) + A_2 \sum_{i=1}^{12} f(a_i^{(2)}) + A_3 \sum_{i=1}^{8} f(a_i^{(3)}) \\ &\quad + \sum_{k=1}^{N_1} B_k \sum_{i=1}^{24} f(b_i^{(k)}) + \sum_{k=1}^{N_2} C_k \sum_{i=1}^{24} f(c_i^{(k)}) + \sum_{k=1}^{N_3} D_k \sum_{i=1}^{48} f(d_i^{(k)}) \quad (2.38) \end{aligned}$$

which exactly integrates all linear combinations of spherical harmonics up to and including order m. The number of points N_i, $i = 1, 2, 3$ is chosen so that the total number of unknowns in the system obtained is equal to the number of equations. The projections of the points $a_i^{(1)}$, $a_i^{(2)}$, $a_i^{(3)}$, $b_i^{(k)}$ and $c_i^{(k)}$ onto $\mathcal{H}$ are at the vertices, the mid-points of the edges, the centers of the faces, and on the bisectors and edges of the faces of $\mathcal{H}$, respectively. The points $d_i^{(k)}$ are nodes of general position. A face of $\mathcal{H}$ is illustrated in Fig. 2.3

The nodes $a_i^{(k)}$, $b_i^{(k)}$, $c_i^{(k)}$, $d_i^{(k)} \in S$ have the following coordinates:

$$a_i^{(1)} : (0, 0, \pm 1), (0, \pm 1, 0), (\pm 1, 0, 0)$$

$$a_i^{(2)} : (\pm 2^{-1/2}, \pm 2^{-1/2}, 0), (\pm 2^{-1/2}, 0, \pm 2^{-1/2}), (0, \pm 2^{-1/2}, \pm 2^{-1/2})$$

$$a_i^{(3)} : (\pm 3^{-1/2}, \pm 3^{-1/2}, \pm 3^{-1/2})$$

$$b_i^{(k)} : (\pm l_k, \pm l_k, \pm m_k), (\pm l_k, \pm m_k, \pm l_k), (\pm m_k, \pm l_k, \pm l_k)$$

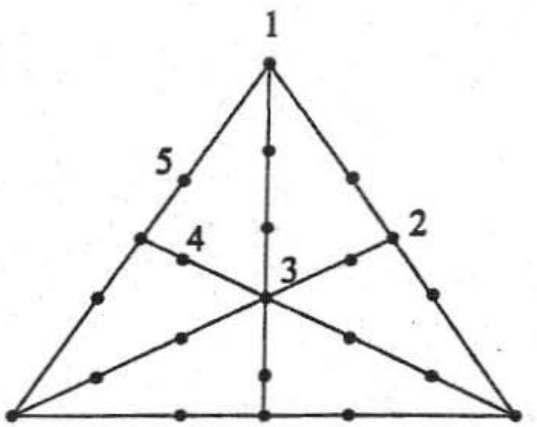

Figure 2.3: One of the eight faces of the octahedron

$$c_i^{(k)} \; : \; (\pm p_k, \pm q_k, 0), (\pm p_k, 0, \pm q_k), (0, \pm p_k, \pm q_k), (\pm q_k, 0, \pm p_k)$$

$$d_i^{(k)} \; : \; (\pm r_k, \pm u_k, \pm w_k), (\pm r_k, \pm w_k, \pm u_k), (\pm u_k, \pm r_k, \pm w_k),$$
$$(\pm u_k, \pm w_k, \pm r_k), (\pm w_k, \pm u_k, \pm r_k), (\pm w_k, \pm r_k, \pm u_k)$$

where

$$2{l_k}^2 + {m_k}^2 = 1 \tag{2.39}$$

and

$${p_k}^2 + {q_k}^2 = 1. \tag{2.40}$$

According to Sobolev's theorem it is sufficient to require the quadrature be exact only for polynomials invariant with respect to G^{8*}, and using the fact that all polynomials invariant with respect to G^{8*} may be expressed on S as a polynomial in

$$\sigma_2 = x^2y^2 + x^2z^2 + y^2z^2 \qquad \text{and} \qquad \sigma_3 = x^2y^2z^2 \tag{2.41}$$

where

$$\sigma_1 = x^2 + y^2 + z^2 = 1. \tag{2.42}$$

Lebedev was able to find cubature formulae up to the order 53 [18]. Figure 2.4 shows the nodes of the quadrature of order 53.

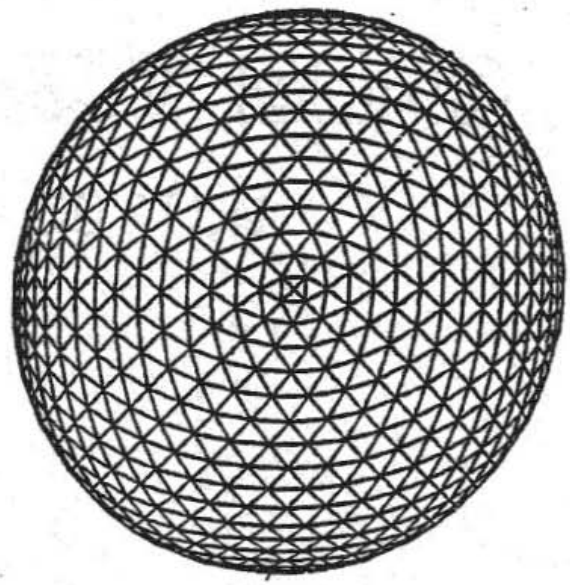

Figure 2.4: Nodes of quadrature of order 53, from Ref. 18

The quality of the quadrature is given by,

$$\eta = \frac{(m+1)^2}{3N} \tag{2.43}$$

where $(m+1)^2$ is the total number of integrable spherical harmonics up to and including order m, and N is the number of nodes in the quadrature with non-zero weights. For quadratures with a minimum number of nodes, $\eta \to 1$ as $m \to \infty$.

Chapter 3

Molecular numerical integration

Three-dimensional integrals of the type

$$I = \int F(\mathbf{r})\, \mathbf{dr} \tag{3.1}$$

where the integrand $F(\mathbf{r})$ extends over the molecular space, occur frequently in the calculations of the electronic structure of molecules. The presence of a cusp singularity at each nucleus makes the problem of numerical integration quite difficult. Two different approaches were developed to deal with this problem. The first one [19,20,21] divides the molecular space into discrete Voronoi polyhedra, while the second [5,22,23] uses nuclear weight functions. In both cases the integral is approximated by a weighted sum of the values of the integrand at a set of N_{Grid} abscissae $\mathbf{r}$. The set of the N_{Grid} abscissae with their associated integration weights constitute a three-dimensional grid. Another innovative way to deal with this problem is the work of Bader and co-workers [24,25,26]. Bader has defined atomic fragments in molecular systems on the basis of the topology of the total electron density. An ingenious numerical technique has been devised for integration within these fragments [25]. Section 3.1 follows closely the work of F. König and co-workers. Section 3.2 presents

a brief review of the basic ideas of the numerical integration over discrete Voronoi polyhedra. Section 3.3 introduces a review of the popular Becke method which adopts the weight function method.

3.1 Atoms in molecules

Bader proposed a theory of molecular structure called quantum topology [24]. According to this theorem the definition of an atom is based on a partitioning of real space where an open region of the three-dimensional space can be assigned to a subsystem. A subsystem is defined in terms of a property of the charge distribution ρ. For a given nuclear configuration $(\mathbf{R_1}, \mathbf{R_2}, \ldots, \mathbf{R_M})$ where $(\mathbf{R_1}, \mathbf{R_2}, \ldots, \mathbf{R_M})$ are the coordinates of the M nuclei, the topological properties of ρ are mapped onto the associated gradient field $\nabla\rho$. This vector field is exhibited via its trajectories in real space, which are called the gradient paths. A gradient path $g(\mathbf{r}_m)$ through a given point $\mathbf{r_m} \in \mathbf{R^3}$ is thus defined by

$$g(\mathbf{r}_m) = \{\mathbf{r}(s) \in R^3 : \frac{d\mathbf{r}(s)}{ds} = \nabla\rho(\mathbf{r}(s), X),\ s \in R,\ \mathbf{r}(0) = \mathbf{r}_m\} \tag{3.2}$$

and is orthogonal to a contour of $\rho(\mathbf{r}, X)$ at any of its points. $\rho(\mathbf{r}, X)$ describes the probability density of finding any electron at the point $\mathbf{r}$ and the nuclei in the configuration X. An attractor of a vector field $\mathbf{V}$ over R^3 is, by definition, a closed subset $G \in R^3$, which:

(1) is invariant with respect to the flow of $\mathbf{V}$; i.e., any trajectory of $\mathbf{V}$ that contains a point of G is wholly contained in G;

(2) contains all trajectories originating in G;

(3) is contained in some open subset $B \in R^3$, such that any trajectory originating in

B has its terminus in G. The maximal neighborhood B_{max} of G satisfying requirement(3) is called the basin of the attractor. The only closed subsets of R^3 exhibiting the previous properties with respect to $\nabla\rho$ are the singletons determined by the local maxima in the charge distribution.Thus the nuclei act as the attractors of the gradient vector field derived from the charge distribution. A result of this identification is that the space of the molecular charge distribution is partitioned into disjoint regions, the basins, each of which contains one point attractor or nucleus. An atom, free or bound, is defined as the union of an attractor and its associated basin.

Another way of defining an atom is to define it in terms of its boundary. The regions encompassing the atoms are delimited by surfaces S that satisfy the zero-flux condition:

$$\nabla\rho(\mathbf{r}).\mathbf{n}(\mathbf{r}) = 0\,, \quad \forall\mathbf{r} \in S \tag{3.3}$$

where $\mathbf{n}$ is the unit vector normal to the surface at $\mathbf{r}$. According to Bader [24], any atomic property F is the average over the atomic basin of an effective single–particle density $f(\mathbf{r})$. Thus the value of the property F for an atom i is

$$F(i) = \int_{\Omega_i} f(\mathbf{r})\, d\mathbf{r} \tag{3.4}$$

where Ω_i is the volume of the atom i. The integration in the last equation is extremely complicated due to the fact that the atomic surface which bounds the region Ω does not in general have a local definition. The integration method developed by F. König et al. [25] avoids the direct determination of the atomic surface by implementing the definition of an atom as the union of an attractor and its basin. By integrating along the trajectories of $\nabla\rho(\mathbf{r})$ which terminate at a given nucleus, the basin of an atom is necessarily covered and because of the zero-flux surface condition it is impossible to

cross an interatomic surface into the basin of a neighbouring atom. As an illustration of his method, König calculated the electron population of an atom i by integrating the density over the volume Ω_i

$$N_e(i) = \int_{\Omega_i} \rho(\mathbf{r})\, d\mathbf{r} \tag{3.5}$$

for the molecules LiF, CO, BeH, BeH_2, Table 3.1. The primary applications of Bader's formalism are theoretical. However, the numerical integration technique can be adopted for general–purpose integration.

Table 3.1: Electron population, using König algorithm (Ref. 25)

Molecule	Atom	$N(\Omega)$
LiF	Li	2.0630
	F	9.9370
	Total	12.0000
	exact	12.0000
CO	C	4.6544
	O	9.3459
	Total	14.0003
	exact	14.0000
BeH	H	1.8677
	Be	3.1321
	Total	4.9998
	exact	5.0000

3.2 Molecular integration using discrete polyhedra

The Voronoi polyhedron around a nucleus maybe defined as the minimum volume bounded by the planes that orthogonally bisect the line segment joining the nucleus with all other nuclei [21]. It contains all points closer to that nucleus than to any other one. Voronoi polyhedra of different atoms are non-overlapping and the conjunction of them fills all the space exactly. For a particular polyhedron, the origin of the (local) coordinate system is chosen at the atom and an atomic sphere is introduced inside the polyhedron. Thus the integration is separated into one over the sphere and one over the remaining part of the polyhedron. The latter is split into a sum of (truncated) pyramids, each having its top at the atom and its base is one of the faces of the polyhedron [19]:

$$\int_{\text{polyhedron}} F(\mathbf{r})\, d\mathbf{r} = \int_{\text{atomic sphere}} F(\mathbf{r})\, d\mathbf{r} + \sum_{\text{faces}} \int_{\text{trunc pyramid}} F(\mathbf{r})\, d\mathbf{r}. \tag{3.6}$$

The integration over the atomic sphere is carried by separating the variables and using the product form

$$\int_{\text{sphere}} F(\mathbf{r})\, d\mathbf{r} = \int_0^R g(r)\, dr \tag{3.7}$$

$$g(r) = \int_{\text{spherical surface}} F(r, \Omega)\, d\Omega. \tag{3.8}$$

The spherical surface maybe treated with a product formula, therefore reducing it to the one-dimensional case

$$\int F(\Omega)\, d\Omega = \int_{-1}^{1} d\cos\theta \int_0^{2\pi} F(\theta, \phi)\, d\phi. \tag{3.9}$$

It proves to be advantageous to use the special formulae of Lebedev for the integration over the sphere [16,17,18]. The radial integral can be evaluated by one of the Gaussian quadratures, such as the Legendre scheme. For the pyramids a threefold product formula is used. The pyramid base is a polygon and may have any number of vertices. Product formulae of quadrangles and triangles are well known. So a polygon with any number of vertices can be repeatedly split into quadrangles, until a quadrangle or a triangle remains. A final further splitting in subregions is performed, writing the pyramid as a sum of pyramids with each a quadrangular (or triangular) base. In a very sophisticated way G.te Velde and Baerends were able to perform the integration over the pyramids.

3.3 Nuclear weight function methods: Becke's method

In this class of methods, the molecular integration is broken down into atomic contributions by means of a set of N_a weight functions $w_i(\mathbf{r})$, $i = 1, 2, \ldots, N_a$, also called partition functions, where N_a is the number of atoms in the molecule. That is the partition functions w_i decompose the three-dimensional integral into a series of atomic like integrals, which are much easier to handle. There is a variety of molecular partition schemes in the literature [5,22,23]. The following is an outline of the molecular partition scheme proposed by Becke [5].

The molecular function $F(\mathbf{r})$, $\mathbf{r} \in R^3$, which has to be numerically integrated, is partitioned into atomic contributions $F_i(\mathbf{r})$,

$$F(\mathbf{r}) = \sum_{i=1}^{N_a} F_i(\mathbf{r}) \tag{3.10}$$

where, the summation is over the number of atoms, N_a. The atomic contributions $F_i(\mathbf{r})$ at each grid point $\mathbf{r}$ are defined by the normalized atomic weight functions $w_i(\mathbf{r})$

$$F_i(\mathbf{r}) = w_i(\mathbf{r})F(\mathbf{r}) \tag{3.11}$$

The molecular integral

$$I = \int F(\mathbf{r})\, d\mathbf{r} \tag{3.12}$$

can be written as

$$I = \sum_{i=1}^{N_a} \int F_i(\mathbf{r})\, d\mathbf{r} = \sum_{i=1}^{N_a} I_i \tag{3.13}$$

which is a sum over the atomic integrals I_i. The molecule is partitioned based on the definition of the atomic weight function which depends only on the atomic coordinates. The partition function $w_i(\mathbf{r})$ is required to fullfill:

- $w_i(\mathbf{r}) \geq 0$
- $\sum_{i=1}^{N_a} w_i(\mathbf{r}) = 1$ at any point $\mathbf{r}$ in the space.
- Every $w_i(\mathbf{r})$ is zero or almost zero in the vicinity of all the nuclei of the molecule, except in the vicinity of the i^{th} nucleus, where it should be almost unity.

The regions in which the partition function is close to zero should be large enough to extinguish the big values of the density at nuclear positions.

To generate w_i, Becke started by partitioning the molecular space into the conventional Voronoi polyhedra such that each nucleus is enclosed in one of these polyhedra. At every grid point, $\mathbf{r}$, confocal elliptical coordinates between all pairs of atoms, see Figure 3.1, are defined as

$$\mu_{ij} = (r_i - r_j)/R_{ij} \qquad -1 \leq \mu_{ij} \leq 1 \tag{3.14}$$

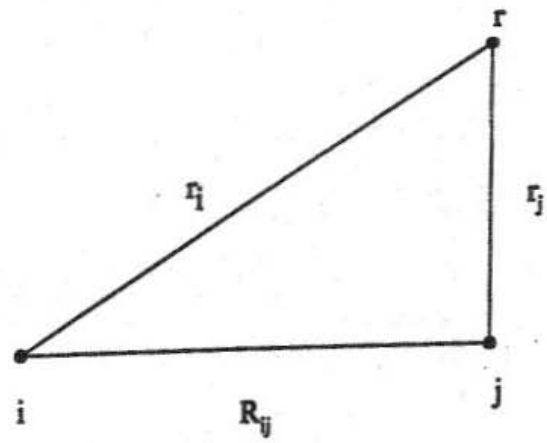

Figure 3.1: Two nuclei i, j in the molecular space, where r is a grid point, r_i and r_j are the distances to the nuclei i and j, and R_{ij} is the internuclear distance.

and Becke defined the Voronoi polyhedron on nucleus i by the product

$$w_i(\mathbf{r}) = \prod_{j \neq i} s(\mu_{ij}) \tag{3.15}$$

$w_i(\mathbf{r})$ is called a 'cell function' and has value unity if r lies inside the cell, and zero if r lies outside. $s(\mu_{ij})$ is a step function given by

$$s(\mu_{ij}) = \begin{cases} 1 & -1 \leq \mu_{ij} \leq 0 \\ \\ 0 & 0 < \mu_{ij} \leq +1 \end{cases}$$

Then Becke 'softened' the discontinuity at $\mu_{ij} = 0$, the mid-point between atoms i and j such that the $w_i(\mathbf{r})$ are now fuzzy, overlapping, and analytically continuous cells. A cubic polynomial,

$$h(\mu_{ij}) = \frac{3}{2}\,\mu_{ij} - \frac{1}{2}\,{\mu_{ij}}^3 \tag{3.16}$$

is designed with the property that

$$h(1) = 1\,, \qquad h(-1) = -1, \tag{3.17}$$

$$h'(1) = 0\,, \qquad h'(-1) = 0 \tag{3.18}$$

where h' is the first derivative of h. This polynomial varies smoothly between the end points -1 and $+1$, but it is not sharp enough. Thus, $h(\mu_{ij})$ is iterated three times,

$$g(\mu_{ij}) = h\,\{h\,[h(\mu_{ij})]\} \tag{3.19}$$

$s(\mu_{ij})$ is defined in terms of $g(\mu_{ij})$ as

$$s(\mu_{ij}) = \frac{1}{2}[1 - g(\mu_{ij})]. \tag{3.20}$$

Becke has reported that the cell function becomes too "step function like", when employing more than three iterations and leads to numerical instabilities. With one or two iterations the cell function becomes too soft and does not properly extinguish the nuclear cusp nearby the nuclei, Figure 3.2. Other authors have also recommended only three iterations.

The normalized weight functions entering the actual quadrature is given by

$$W_i(\mathbf{r}) = \frac{w_i(\mathbf{r})}{\sum_{j=1}^{N_a} w_j(\mathbf{r})} \tag{3.21}$$

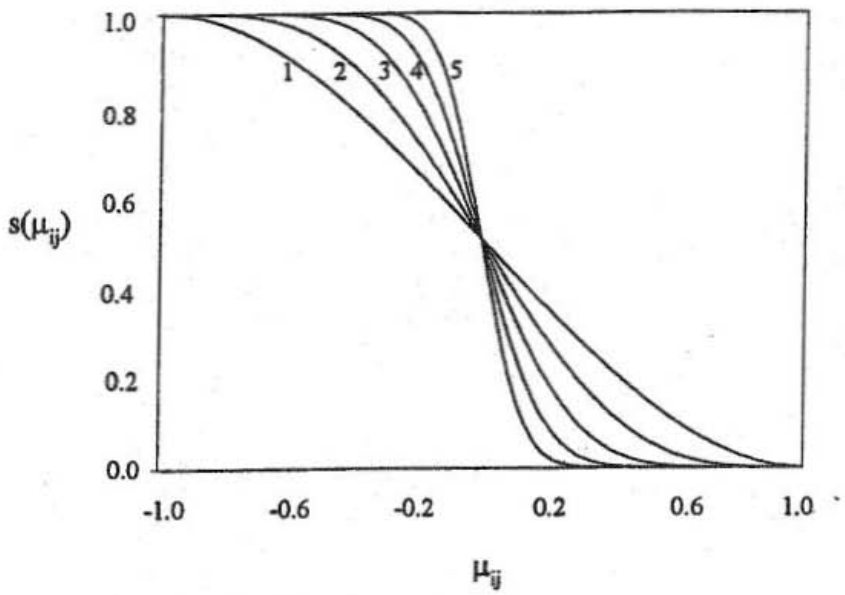

Figure 3.2: The step function for iterations 1 to 5, the 3 iteration case is given by Eq. 3.19

where the summation over j in the denominator includes all nuclei in the system.

In the scheme given so far, the space is divided equally between two atoms. Becke recognized that it is important to have regions of different sizes around each atom. Therefore, Becke introduced a change of variable

$$\nu_{ij} = \mu_{ij} + a_{ij}\left(1 - {\mu_{ij}}^2\right) \tag{3.22}$$

$$a_{ij} = \frac{u_{ij}}{{u_{ij}}^2 - 1} \tag{3.23}$$

$$u_{ij} = \frac{\chi - 1}{\chi + 1} \tag{3.24}$$

$$\chi = \frac{R_i}{R_j} \tag{3.25}$$

where R_i denotes the Bragg-Slater radius of atom number i . The partitioning of molecular integrals into atomic contributions solves the problem of near singularities at nuclear locations which are in general at arbitrary positions in space. But the price to be paid is appreciable. Since the partition function ω_i must switch rather rapidly from 1 to 0 in (or near) the middle of the bond, the atomic contributions F_i, $F_i = w_i F$ (F is the function to be integrated), show considerably more pronounced variations in these areas of space than the density itself.

3.4 Atomic integration

While the partition scheme proposed by Becke is general and is used by most of the scientific community, the single center integrals can be carried out in different ways, hoping to achieve the best efficiency possible. Becke [5] used the following algorithm. Each single-center integral I_i in equation 3.13 can be written in spherical polar coordinates as

$$\int_0^\infty \int_0^\pi \int_0^{2\pi} F_i(r,\theta,\phi) r^2 \sin\theta \, dr \, d\theta \, d\phi. \tag{3.26}$$

It is easy to rearrange the above triple integral into successive one- and two-dimensional integrals. In general, the integration of a one-dimensional function $I_i(x)$ can be numerically approximated by

$$\int_a^b I_i(x) \, dx \approx \sum_{j=1}^{N} w_j \, I_i(x_j) \tag{3.27}$$

There are many radial quadrature schemes proposed in the literature to integrate $I_i(x)$ numerically. Becke prefers the transformed Gauss–Chebyshev quadrature (GC)

of the second kind [11]. This quadrature has the advantage that the abscissae and the weights are simple closed form expressions. Following [27,28] the abscissae x_i and the weights w_j are defined by

$$x_j = \frac{N+1-2j}{N+1} + \frac{2}{\pi}\left[1+\frac{2}{3}\sin^2(\frac{j\pi}{N+1})\right] \times \cos(\frac{j\pi}{N+1})\sin(\frac{j\pi}{N+1}) \tag{3.28}$$

and

$$w_j = \frac{16}{3(N+1)}\sin^4(\frac{j\pi}{N+1}). \tag{3.29}$$

This quadrature rule generates a set of abscissae in the interval $[-1,1]$. Thus the abscissae have to be mapped into $0 \leq r_i < \infty$. Becke [5] suggested the transformation

$$r_i = R\frac{1+x_i}{1-x_i} \tag{3.30}$$

where R is a parameter corresponding to the midpoint of the integration interval at $x = 0$. R was chosen as half of the Bragg-Slater radius of the respective atom, except for hydrogen where the factor $1/2$ was not applied. For ligand atoms or atoms in linear coordination, Becke assigned 20 radial points for the hydrogen atom and an additional five points for each additional atomic shell. For central atoms in polyatomic molecules, an extra 10 radial points are added. For each radial shell an angular grid has to be established. The octahedra grids developed by Lebedev [16,17,18] are known to be more efficient than angular product grids.

Chapter 4

Two molecular numerical integration schemes

Many of the codes developed to deal with molecular integration are based on the Becke scheme [29,30,31,32,33]. The Becke scheme is easier to use than Velde's scheme[19] and much easier than the Bader theory [24,25], although Bader theory provides additional information about the atomic properties. While these codes agree on the way to divide the three dimensional molecular space, they choose different quadrature formulae to evaluate the one- and two-dimensional integrals over the atomic centers. Lebedev grids is a popular choice for the integration over the atomic surface. Less attractive is the product formula. For the one-dimensional integral, Gauss–Chebyshev and Euler Maclaurin quadratures are quite competitive. This chapter presents in some detail two popular Becke-based schemes for molecular numerical integration. The first is the standard grid, SG-1 grid, proposed by Peter M.W. Gill [29] et al. and has been incorporated into the Q-chem program. The second was introduced by Becke in his original paper [5] and later modified by O. Treutler and R. Ahlrichs [31]. These two algorithms have been incorporated into the ab-initio package MUNgauss

[34]. A variety of molecules of different sizes was used to evaluate the numerical integration scheme. The results, including the integration of the charge density and CPU time required are discussed in chapter 6

4.1 The standard grid SG-1

The integral,

$$I = \int F(\mathbf{r}) d\mathbf{r} \tag{4.1}$$

$$= \int_0^\infty \int_0^\pi \int_0^{2\pi} F(r,\theta,\phi)\, r^2\, dr\, \sin\theta\, d\theta\, d\phi \tag{4.2}$$

is evaluated by first separating the radial and angular integrations by employing product quadrature formulae (see Eq. 2.20),

$$I \approx \sum_{i=1}^{N^r} w_i^r \sum_{j=1}^{N^\Omega} \omega_j^\Omega\, F(r_i,\theta_j,\phi_j) \tag{4.3}$$

N^r is the number of the radial points and N^Ω is the number of the angular points. w_i^r and w_j^Ω are the radial and angular weights, respectively. The inner sum corresponds to quadrature on the surface of a sphere. Gill [29] used Lebedev grids of degree $m = 3, 9, 15, 23$ which have $N^\Omega = 6, 38, 86, 194$. The remaining problem is to select w_i^r and r_i values, i.e. a quadrature formula of the form

$$\int_0^\infty r^2\, G(r)\, dr \approx \sum_{i=1}^{N^r} w_i^r\, G(r_i) \tag{4.4}$$

is sought, where

$$G(r) = \int_0^\pi \int_0^{2\pi} F(r,\theta,\phi)\, \sin\theta\, d\theta\, d\phi \tag{4.5}$$

Table 4.1: Atomic radii R, in bohr, used in the SG-1 grid (Ref. 29)

atom	H	He	Li	Be	B	C
radius	1.000	0.5882	3.0769	2.0569	1.5385	1.2308
atom	N	O	F	Ne	Na	Mg
radius	1.0256	0.8791	0.7692	0.6838	4.0909	3.1579
atom	Al	Si	P	S	Cl	Ar
radius	2.5714	2.1687	1.8750	1.6514	1.4754	1.3333

and

$$G(r_i) = \sum_{j=1}^{N^\Omega} \omega_j^\Omega F(r_i, \theta_j, \phi_j). \tag{4.6}$$

G is assumed to be bounded for $r = 0$ and to decay at least exponentially for large r. Gill adopted the Euler Maclaurin scheme [32], where

$$w_i^r = 2R^3(N^r + 1)i^5(N^r + 1 - i)^{-7} \tag{4.7}$$

$$r_i = Ri^2(N^r + 1 - i)^{-2} \tag{4.8}$$

and $N^r = 50$. R is called the atomic radius and corresponds to the maximum of the radial probability function $4\pi r^2|\phi(\mathbf{r})|^2$ of the valence atomic orbital $\phi(\mathbf{r})$ given by Slater's rules[35]. The atomic radii R which follow from this definition are given in Table 4.1 for the atoms H to Ar.

Gill et al. utilized the fact that as a nucleus is approached from the valence region in a molecule, the electron density becomes more spherically symmetrical and therefore can be treated by progressively less sophisticated angular grids. In this way N^Ω is dependent on i, and different Lebedev grids are used on different concentric spheres. Few angular points are used in the core region and relatively many angular points

Table 4.2: Partitioning parameters used in the SG-1 grid (Ref. 29)

Atom	α_1	α_2	α_3	α_4
H–He	0.2500	0.5000	1.0000	4.5000
Li–Ne	0.1667	0.5000	0.9000	3.5000
Na–Ar	0.1000	0.4000	0.8000	2.5000

are used in the valence region. Given four numbers $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ and an atomic radius R, the four spheres of radii $\{\alpha_1 R, \alpha_2 R, \alpha_3 R, \alpha_4 R\}$ partition an atom into five regions where the fifth region is that between the sphere $\alpha_4 R$ and the sphere on the 50^{th} radial point, see Figure 4.1. Lebedev grids with $6, 38, 86, 194, 86$ points were employed in the five regions respectively. The α values for the atoms from H to Ar are given in Table 4.2, therefore, the specification of the SG-1 grid for the atoms H to Ar is complete. SG-1 does not adopt the 'atomic size adjustments' proposed by Becke.

4.2 Treutler–Ahlrichs grids

O. Treutler and R. Ahlrichs [31] developed new grids for three-dimensional molecular numerical integration. They used the partition functions proposed by Becke, but with slight modification in the atomic size adjustments. Treutler and Ahlrichs replaced the parameter χ in Eq. 3.25 by

$$\chi = \sqrt{\frac{R_i}{R_j}}. \tag{4.9}$$

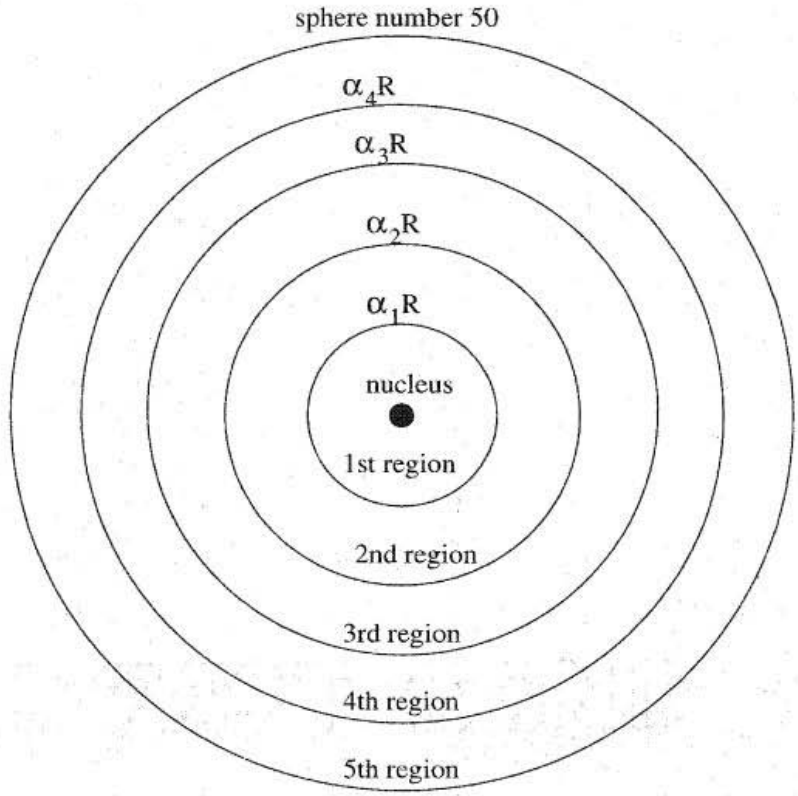

Figure 4.1: The SG-1 Partitioning of the molecular space.

They recorded that the new definition of χ has led to a better performance than Becke's adjustment. For the radial part of the integration they used Gauss–Chebyshev quadratures of the first kind (T1) and of the second kind (T2). To map the integration interval $[-1, 1]$ onto the interval $[0, \infty]$ they used four mappings. Table 4.3 gives these mappings, M, and the corresponding Jacobians.

M2 is equivalent to Becke's proposal, Eq. 3.30, where ξ replaced R. The scaling parameter ξ has first been optimized in isolated atom calculations and the resulting values have then been checked in molecular calculations. The four mappings are normalized to map the center of the x interval onto $r = \xi$. M1 shows the highest density of grid points near $r = \xi$ and low density near $r = 0$. In M2 (Becke), the grid

Table 4.3: The radial mappings and the corresponding Jacobians

M	r	dr
M1	$r = \xi \frac{1+x}{\sqrt{1-x^2}}$	$dr = \xi \frac{1+x}{(1-x^2)^{3/2}}\, dx$
M2	$r = \xi \frac{1+x}{1-x}$	$dr = \xi \frac{2}{(1-x)^2}\, dx$
M3	$r = \frac{\xi}{\ln 2} \ln \frac{2}{1-x}$	$dr = \frac{\xi/\ln 2}{1-x}\, dx$
M4	$r = \frac{\xi}{\ln 2} (a+x)^{\alpha} \ln \frac{2}{1-x}$	$dr = \frac{\xi}{\ln 2} \left(\frac{(a+x)^{\alpha}}{1-x} + \alpha (a+x)^{\alpha-1} \ln \frac{2}{1-x} \right) dx$

points are concentrated near $r = 0$ and are extended to much larger values than for M1. Thus it does not project enough points into the chemical bonding region around $r = \xi$, which means a waste of mesh points. M3 combines the advantages of both mappings M1, M2, i.e. a good representation of the regions $r = \xi$ and $r = 0$. M4 is an extension of M3, where an increase of α shifts grid points toward $r = 0$ and $r = \infty$. The best performance is that of the quadrature T2 with the mapping M4 for $\alpha = 0.6$. For the angular part of the integration Treutler-Ahlrichs used Lebedev grids of degree $m = 5, 11, 17, 23, 29, 35$ which correspond to $14, 50, 110, 194, 302, 434$ angular points respectively. Thus five three-dimensional grids with the number of radial points ranging from 20 to 45 were built and tested. Treutler and Ahlrichs 'pruned' their grids which means using small Lebedev grids in regions near the nucleus. They divided the molecular space into three regions and used

$$N^{\Omega} = 14, \quad \text{for } r_i, \quad i \leq \frac{N^r}{3} \tag{4.10}$$

$$N^{\Omega} = 50, \quad \text{for } r_i, \quad \frac{N^r}{3} < i \leq \frac{N^r}{2} \tag{4.11}$$

where N^{Ω} is the number of the angular points, N^{r} is the number of the radial points, and r_i are the radial points. For the third region they used:

- $N^{\Omega} = 50, N^{r} = 20$ for grid 1
- $N^{\Omega} = 110, N^{r} = 25$ for grid 2
- $N^{\Omega} = 194, N^{r} = 30$ for grid 3
- $N^{\Omega} = 302, N^{r} = 35$ for grid 4
- $N^{\Omega} = 434, N^{r} = 45$ for grid 5

Treutler and Ahlrichs performed numerous calculations using the five grids and reached the conclusion that grids larger than grid 3 do not lead to improvements in the results. Therefore, a grid corresponding to $N^{\Omega} = 194$ and $N^{r} = 30$ was built along with SG-1. Chapter 6 presents the results obtained on implementing these two grids.

Chapter 5

Why Fortran 90?

Fortran 90 provides superior facilities for dealing with numerical data, and it is the best language for most applications that are dominated by mathematical, engineering, or scientific analysis. Fortran 90's array-handling capabilities are outstanding. Also, Fortran 90 provides excellent mechanisms for partitioning large codes into smaller, more manageable pieces. Furthermore, it supports features that facilitate data hiding and data abstraction [36].

5.1 Drawbacks of FORTRAN 77

By today's standards FORTRAN 77 is outmoded. Many other languages have been developed which allow greater expressiveness and ease of programming. The main drawbacks have been identified as [37]:

- FORTRAN 77 awkward 'fixed form' source format. Each line can only be 72 characters long, any text in column 73 is ignored, the first 5 columns are reserved for line numbers, the 6th column can only be used to indicate a continuation line, only upper case letters are allowed anywhere in the program, variable

names can only be 6 characters, no in-line comments are allowed, comments must be on a line of their own.

- Lack of inherent parallelism. FORTRAN 77 has no in-built way of expressing parallelism.

- Lack of dynamic storage. Temporary short-lived arrays can not be created on the fly. All FORTRAN 77 programs must declare arrays 'big enough' for any future problem size which is very unattractive restriction abscent in the current popular high-level languages.

- Lack of numeric portability. Problems arise with precision when porting FORTRAN 77 code from one machine to another, this means that the code is unportable.

- Lack of user-defined structures. In FORTRAN 77 user-defined data are not available as they are in ADA, C, C++, etc...

- Reliance on unsafe storage and sequence association features. In FORTRAN 77, global data is only accessible via the open-to-abuse common blocks. The rules which applied to common blocks are very lax and can lead to terrible mistakes.

5.2 Fortran 90: New features

FORTRAN 77 is an essential part of Fortran 90, thus any program that compiles with the older standard also compiles with the new standard. Roughly speaking, FORTRAN 77 constitutes two-thirds of Fortran 90 [36]. Thus the main data types are still **integer**, **real**, **logical**, and **character**. **If** and **DO** statements remain the basis of most control structures. As indicated by Figure 5.1 , some of the new features

of Fortran 90 are simple extensions to FORTRAN 77.

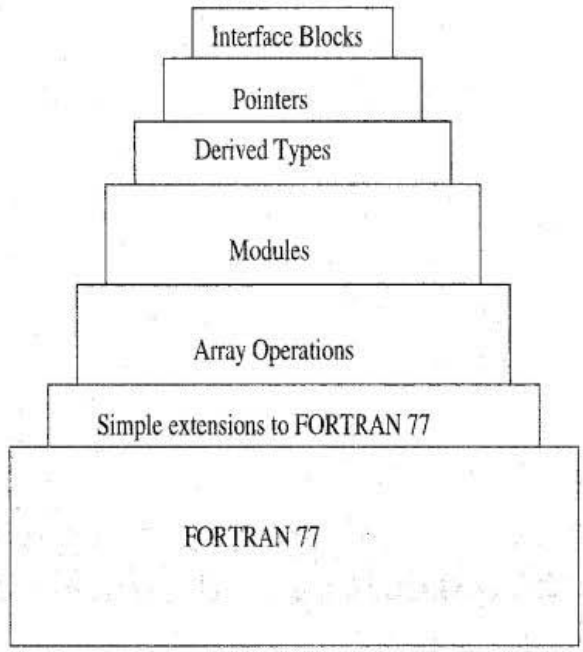

Figure 5.1: Major building blocks of Fortran 90

An example is the free source code form which permits:

- Any part of a statement to appear anywhere on a line.
- Up to 132 columns per line.
- More than one statement per line.
- In-line comments are allowed.
- Upper and lower case letters are allowed.
- longer and more descriptive object names up to 31 characters.

- Names can be punctuated by underscores making them more readable.

Also, a new form of type declarations, characterized by the presence of a double colon (::), makes it possible to specify all attributes of a variable in a single statement. Fortran 90 supports control structures that make it possible to code lengthy and complicated programs without using labels, and therefore it is never necessary to use any form of a **go to** statement. **implicit none** statement has been made standard in Fortran 90. The **end do** is a widely available extension that becomes standard in Fortran 90.

5.2.1 Array operations

In Fortran 90 new array operations were introduced which can perform simple array manipulations. Some operators, including, $+, -$, and $*$, may be used with a pair of arrays. Also, many Fortran 90 intrinsic functions, such as **sqrt**, **abs**, **sin**, etc... may be used with an array argument, in which case the function is applied independently to each element of the array and returns an array result [36]. In Fortran 90 we can deal with sections of array like A(m,n,k). This section of array runs from m to n in steps of k, where m, n, and k can take positive or negative values. Another very important feature of Fortran 90 is the introduction of dynamic storage allocation, which means storage can be allocated and freed during program execution. An allocatable array can be declared and allocated as in the following piece of code:

```
* x-coordinate of the Angular grid points.
double precision, dimension(:), allocatable :: Apointx
...
integer :: NRpoint        ! Number of radial points.
```

```
...
allocate (Apointx(NRpoint*38))
...
deallocate(Apointx)

    ...
```

First, the keyword **allocatable** appears in the attribute list of Apointx. The **colon** inside the parenthesis following the keyword **dimension** indicates that Apointx is a one-dimensional array. The number of colons represents the number of dimensions of the allocatable array. When the **allocate** statement is executed, a storage for NRpoint*38 double precision values will be allocated, and the Apointx subscript will have been declared to have a lower bound of 1 and an upper bound value of NRpoint*38. More than one allocatable array maybe allocated with a single allocate statement like

```
...
allocate(Apointx(NRpoint*38), Apointy(NRpoint*38),
Apointz(NRpoint*38))
...
```

When storage is no longer needed for an allocatable array, execution of **deallocate** statement will free this storage which is a good programming practice.

5.2.2 Derived types

Derived types and pointers support much more sophisticated data structures than have traditionally been available in Fortran. A major new feature in Fortran 90 is

its support for data aggregates made up of individual pieces of data that maybe of different data types. Derived type enables the programmer to group a collection of related pieces of data under a single name [37]. The programmer can refer either to individual components of the aggregate or the aggregate as a whole. The programmer defines the composition of such an aggregate, and in standard Fortran 90, each such definition is considered as creating a new data type. The components of any of the programmer-defined data types are of type integer, real, character, and so forth. So the programmer-defined data types are known as 'derived types' while the data types built into the language are called 'intrinsic types'. An example of a derived type is:

```
type :: grid_point
double precision ::    x      ! x-coordinate of grid_point
double precision ::    y      ! y-coordinate of grid_point
double precision ::    z      ! z-coordinate of grid_point
end type grid_point
...
type(grid_point) ::    a_point
type(grid_point), dimension(6) ::   pointA1
type(grid_point), dimension(24) ::   pointB1
...
pointA1%x = (/0.0D0,0.0D0,0.0D0,0.0D0,Radius,-Radius/)
pointA1%y = (/0.0D0,0.0D0,Radius,-Radius,0.0D0,0.0D0/)
pointA1%z = (/Radius,-Radius,0.0D0,0.0D0,0.0D0,0.0D0/)
...
```

The series of statements beginning with **type :: grid_point** and ending with the **end type** statement defines a **derived type**. The statement **grid_point :: a_point** declares a_point to be a variable of type grid_point. Thus, a_point is a composite object made up of three components, one corresponding to the x-coordinate, another corresponding to the y-coordinate, and a third to the z-coordinate. It is also possible to declare an array, each of its elements is a derived type. The statement **type(grid_point), dimension(6) :: pointA1** declares pointA1 to be a one-dimensional array, each of its elements is a derived type grid_point, the next statement defines the array pointB1 to be an array of the same type. To refer to individual components of the derived type, the variable name, pointA1, is followed by the percent sign (%), followed by the component name x, y , or z. Radius is a variable that is declared and initiated earlier in the code

5.2.3 Pointers

Conceptually, an ordinary variable contains data, while a pointer rather than contain data, points to a storage area that does [36]. In Fortran 90, a pointer usually points to an array of values or to a derived type. A limitation of an ordinary, nonpointer variable is that it has a fixed association with a specific data object. This limitation is not shared by a pointer variable. A programmer can dynamically change the association of a pointer during the execution so that at one time it is associated with one data object and later it is associated with another. The following is an example of pointer declaration:

integer :: **n**

...

```
double precision, dimension(:), pointer ::      X
double precision, dimension(n), target ::       Y
...
X => Y
...
allocate(X(Npoint))
...
```

The keyword **pointer** must always be attribute to the variable which is declared as a pointer. The presence of the keyword double precision in the declaration of the pointer X means that it is capable of pointing to any rank one array of type double precision. Fortran 90 standard requires that in order to be pointed to, a variable must be given the **target** attribute. The target must have the same type and rank as the pointer variable. The n-dimensional array Y is a target that is pointed at by the pointer X. The statement $X => Y$ associates X to Y. Later in the example, the association between X and Y is broken and the allocate statement allocates a storage for a nameless array of Npoint elements. A description of the array is placed in X and X is associated with, points to, the array.

5.2.4 Modules

Modules are new to Fortran. A module is a program unit whose functionality can be exploited by another program unit which attaches it via the **use** statement [37]. Modules eleminate the need for common blocks. A module can contain the following:

- Global object declarations. If global data is required, then objects declared in a module can be made visible wherever desired by simply attaching the module.

- Procedures declarations. Procedures can be encapsulated into a module which will make them visible to any program unit which uses the module.

- Controlled object accessibility. Variables and procedures declarations can have their visibility controlled by access statements within the module. It is possible for specified objects to be visible only inside the module.

Modules have a very wide range of applications. They permit separately compiled program units to share data in a manner that is less error-prone than that provided by the common blocks. In addition, because the module can contain subprograms as well as data specifications, it thus becomes the basis for important new ways to partition large programs. Modules can be used for global definitions, and to provide functionality whose internal details are hidden from the user (data hiding). The general form of a module program unit is given in the following example.

```
module type_basis_set
* modules needed:
USE molecule              ! Molecular information
USE constants             ! Global constants
...
implicit none
* Data in this module:
integer ::  Natoms        !The number of atoms in the molecule
...
type basis_primitives
double precision EXP      ! The gaussian exponents
double precision CONTRC   ! The contraction coefficients
```

```
end type basis_primitives
...
type basis_set
...
type(basis_primitives), dimension(:), pointer :: gaussian
...
end type basis_set
...
CONTAINS        ! Functions go here
Subroutine NORMALIZE_BASIS
...
...
end Subroutine NORMALIZE_BASIS
* other subprograms could be defined here.
...
end Module type_basis_set
```

The first statement in a module consists of the keyword **module** followed by the module name. As a module maybe used by a main program, subroutine subprogram, and function subprogram, it could be also used by another module. The first part of the module given in the example is reserved to **use** the other modules that are needed by this one. The next part declares the variables of the modules including the derived types basis_primitives and basis_set. The type_basis_set declares a pointer to the type basis_primitives.

The source code for all module subprograms is placed between a **contains** statement and the module's **end** statement. The part of the module that remains when any subprograms it contains and the **contains** statement is removed is called the specification part. By default, a module user has access to the names in the specification part and the names of the subprograms contained in the module. To block this access, the keyword **private** can be employed which minimizes the likelihood of name clashes.

5.2.5 Interface blocks

The interface block permits interfaces between an external subprogram and its caller to be more complex than in previous versions of Fortran. Fortran 90 permits an array argument to carry the number of subscripts allowed for each dimension along with the beginning address of the array, a function to return an array of values, and subprogram arguments and function results to be pointers [36,38]. These advanced features represent no particular problems for module subprograms and internal subprograms. But when these features are used with external subprograms, a new mechanism known as the "interface body" must be employed. An interface body permits the programmer explicitly to specify every detail of the interface with an external procedure. For example, whether it is involved as a subroutine or a function; the number, order and data types of its arguments; and if it is invoked as a function, the data type of the result returned. Another important use of the interface block is when there is a need to define a generic procedure. A generic procedure refers to a set of different procedures with different specific names that all have the same (generic) name [38]. This is the case when the purpose of a set of procedures is virtually identical. It is diserable to refer to each of them with the same generic name. The type of the argument is

sufficient to identify which of the specific procedures is involved in a given reference. An interface body is always placed inside an interface block which is a collection of code starting with the keyword **interface** and ending with the keyword **end interface**. An example for a module defining an interface block used in the menu part of the code is,

```
module menu_gets
implicit none
interface GET_value
module procedure GET_integer
module procedure GET_integer8
module procedure GET_real
module procedure GET_logical
module procedure GET_string
module procedure GET_title
module procedure GET_Ilist
module procedure GET_Rlist
end interface
...
contains
subroutine GET_integer
...
subroutine GET_string
...
...
```

end MODULE menu_gets

GET_value is the **generic** name for a generic procedure which returns different data types integer, real, ...etc, depending on the data type of the argument as specified by the caller. GET_integer, GET_real, ...etc, are the specific names of the procedures which do the actual work when the generic procedure is called. Their definitions are contained in the module.

5.3 Numerical Integration Code

To evaluate the numerical integration of the charge density over a molecule, see Figure 5.2, we need to:

1- **Determine the coordinates of the radial points, and consequently the positions of the spherical grids, and their radial weights.**
In the case of the SG-1 grid, Euler–Maclaurin scheme was used:

$$w_i^r = 2R^3(N^r+1)i^5(N^r+1-i)^{-7} \tag{5.1}$$

$$r_i = Ri^2(N^r+1-i)^{-2} \tag{5.2}$$

w_i^r is the weight of the point r_i, R is the atomic radius (as given in Table 4.1), and N^r is the number of the radial points. In the case of the grid developed by O. Treutler and R. Ahlrich, it will be refered as TA grid, Gauss–Chebyshev scheme was applied:

$$x_j = \frac{N^r + 1 - 2j}{N^r + 1} + \frac{2}{\pi}\left[1 + \frac{2}{3}\sin^2(\frac{j\pi}{N^r + 1})\right] \times \cos(\frac{j\pi}{N^r + 1})\sin(\frac{j\pi}{N^r + 1}) \qquad (5.3)$$

$$w_j = \frac{16}{3(N^r + 1)}\sin^4(\frac{j\pi}{N^r + 1}). \qquad (5.4)$$

The points $\{x_j\}$ defined on the interval $[-1, 1]$ were mapped onto the interval $[0, \infty]$ using, M4,

$$r_j = \frac{\xi}{\ln 2}(a + x_j)^\alpha \ln\frac{2}{1 - x_j} \qquad (5.5)$$

where ξ, and α are parameters defined in section 4.2.

2- **To calculate the coordinates of the points on the concentric spherical shells and their weights**.

These points constitute the angular grids and are given by Lebedev's grids. The integration of the charge density over the surface of a sphere of radius unity is given by Eq. 2.38

$$\begin{aligned} I(\rho) &= A_1 \sum_{i=1}^{6} \rho(a_i{}^{(1)}) + A_2 \sum_{i=1}^{12} \rho(a_i{}^{(2)}) + A_3 \sum_{i=1}^{8} \rho(a_i{}^{(3)}) \\ &+ \sum_{k=1}^{N_1} B_k \sum_{i=1}^{24} \rho(b_i{}^{(k)}) + \sum_{k=1}^{N_2} C_k \sum_{i=1}^{24} \rho(c_i{}^{(k)}) + \sum_{k=1}^{N_3} D_k \sum_{i=1}^{48} \rho(d_i{}^{(k)}). \end{aligned} \qquad (5.6)$$

The definitions of the points $a_i{}^{(k)}$, $b_i{}^{(k)}$, $c_i{}^{(k)}$ and $d_i{}^{(k)}$ are given in section 2.5. The subroutine **NIM_Gill** calls the subroutines **Angular_grid1, Angular_grid2, Angular_grid3**, and **Angular_grid4** to build 50 spherical shells around each atom.

The subroutine **NIM_TA** calls the subroutines **Angular_grid5, Angular_grid6**, and **Angular_grid4** to build 30 angular grids around each atom of the molecule.

3- **To calculate the charge density at each grid point.**

From Eq. 1.13 the charge density is given by

$$\rho(\mathbf{r}) = \sum_{ij} P_{ij}\, \phi_i(\mathbf{r})\, \phi_j^*(\mathbf{r}) \tag{5.7}$$

where P_{ij} is a density matrix element. $\phi_i(\mathbf{r})$ and $\phi_j^*(\mathbf{r})$ are basis functions. Both of **NIM_Gill** and **NIM_TA** use the subroutine **get_density** to calculate $\rho(\mathbf{r})$.

4- **To obtain the nuclear weight functions $w_i(\mathbf{r})$.**

The subroutine **Beckew** implements the algorithm given in section 3.3 to calculate these functions.

The least number of grid points for a given atom is 3300. Thus, calling the subroutines **get_density** and **Beckew** for each grid point would be expensive. To make the code more efficient, **get_density** and **Beckew** were called only once for the whole set of grid points of each atom. The following piece of code, from **NIM_Gill** demonstrates how this can be achieved:

integer :: **tot_Napoint** ! Total number of angualr points.
integer :: **NRpoint1** ! The number of radial points in the first region.
integer :: **JApoint, IApoint** ! Any angular points, I and J.
integer :: **jbegn** ! The first angular point in any of the five subregions.

integer :: Iend ! The last angular point in any of the subregions.

* The radial points of the first region.

double precision, dimension(:), allocatable :: Rad1

* The radial points of the five regions.

double precision, dimension(:), allocatable :: Rad

* Three allocatable arrays to hold the x, y, and z coordinates

* of the angular points in one of the five regions

double precision, dimension(:), allocatable :: ApointX, ApointY, ApointZ

*Angular weight of the points of one of the five regions.

double precision, dimension(:), allocatable :: Aweight

* Three allocatable arrays to hold the x, y, and z coordinates

* of all of the angular points of the whole atom.

double precision, dimension(:), allocatable :: NApointX, NApointY, NApointZ

* The weight of all of the grid points according to Becke's definition

double precision, dimension(:), allocatable :: weight

* The charge density of all of the grid points.

double precision, dimension(:), allocatable :: charge

* The angular weight of all of the angular points.

double precision, dimension(:), allocatable :: tot_Aweight

...

...

allocate (NApointX(tot_NApoint), NApointY(tot_NApoint), NApointZ(tot_NApoint), tot_Aweight(tot_NApoint))

allocate (ApointX(NRpoint1*6), ApointY(NRpoint1*6), ApointZ(NRpoint1*6))

```
allocate (Aweight(NRpoint1*6), Rad1(NRpoint1))
Rad1(1:NRpoint1) = Rad(1:NRpoint1)
call Angular_grid1(Rad1, NRpoint1, ApointX, ApointY,
ApointZ, Aweight)
JApoint = 0
jbegn = 1
Iend = NRpoint1*6
do IApoint = jbegn,Iend
JApoint = JApoint + 1
NApointx(IApoint) = ApointX(JApoint) + CARTESIANS%X(Iatom)
NApointY(IApoint) = ApointY(JApoint) + CARTESIANS%Y(Iatom)
NApointZ(IApoint) = ApointZ(JApoint) + CARTESIANS%Z(Iatom)
tot_Aweight(IApoint) = Aweight(JApoint)
end do
deallocate (ApointX, ApointY, ApointZ, Aweight, Rad1)
```

Both subroutines, **NIM_Gill** and **NIM_TA** divide the atomic space into subregions. Five subregions in the case of SG-1 grid and three subregions in the case of TA grid. In the above example, which presents the first region from **NIM_Gill**, the total number of angular points of any subregion is not known at the beginning of the compiling, and thus the total number of angular points of the whole of the atom under consideration is not known. Once the number of angular points for each subregion is calculated, the arrays ApointX, ApointY, and ApointZ are dynamically allocated to hold the x, y and z coordinates of the angular points of that region. Also, the arrays NApointX, NApointY and NApointZ are dynamically allocated to hold the x, y, and z coordinates of the total number of angular points for the whole atom. Then **NIM_Gill**

calls the subroutine **Angular_grid1** which takes as input the radial points stored in the array Rad1, and Rad1's dimension stored in NRpoint1. **Angular_grid1** returns the angular points and their angular weight in the arrays ApointX, ApointY, ApointZ, and Aweight. **Angular_grid1** assumes that the atom is at the origin of the coordinate system. To obtain the actual x, y, and z coordinates of the angular points according to the real spacial position of the atom, the statements:

NApointx(IApoint) = ApointX(JApoint) + CARTESIANS%X(Iatom)
NApointY(IApoint) = ApointY(JApoint) + CARTESIANS%Y(Iatom)
NApointZ(IApoint) = ApointZ(JApoint) + CARTESIANS%Z(Iatom)

are executed, where **CARTESIANS** is a derived type. The x, y, and z coordinates of an atom are components of this derived type. Also, these three statements store the angular points of the first subregion in the arrays NApointX, NApointY, and NApointZ. The last statement in the above example deallocates the arrays ApointX, ApointY, ApointZ, Aweight, and Rad1. The same process is repeated for each one of the subregions and later the arrays NApointX, NApointY, and NApointZ will hold all x, y, and z coordinates of all of the angular points of the atom under consideration. The statements:

allocate (charge(tot_NApoint), weight(tot_NApoint))
call get_density (NApointX, NApointY, NApointZ, tot_NApoint, charge)
call Beckew (NApointX, NApointY, NApointZ, tot_NApoint, Iatom, B_sltr, weight)

allocate the arrays charge, weight dynamically with the size of the total number of

angualr points "tot_NApoint", and call the subroutines **get_density** and **Beckew** with the whole set of angular points only once. The variable Iatom stands for the atom number I, while B_sltr is an array containing the Bragg-Slater radii of the atoms H to Ar.

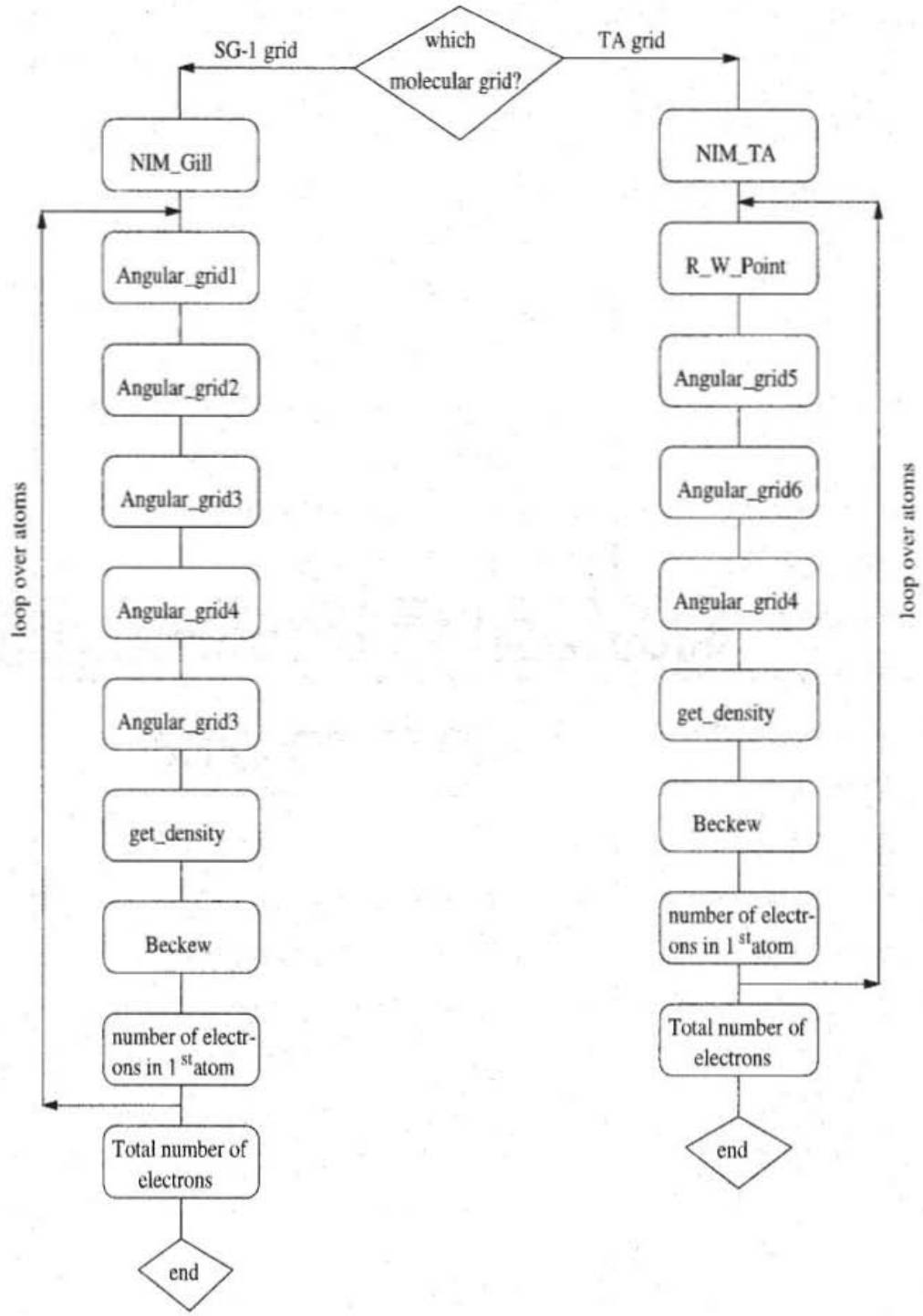

Figure 5.2: Diagram for the numerical integration of the charge density

Chapter 6

Performance of the numerical integration code: Numerical results

6.1 Introduction

A set of atoms and molecules ranging in size from He to $C_{10}H_{10}Cl_2$ was used to test the performance of the two subroutines **NIM_Gill** and **NIM_TA**. The atoms and molecules were chosen carefully so that they cover a fairly large range of electrons, from He (2 electrons) to $C_{10}H_{10}Cl_2$ (104 electrons). Some of the molecules are transition state structures, while the others are ground state structures. The input file to integrate the electron density of a molecule, say CCl_4, is as follows:

molecule **! command for defining a molecule**

title="CCl_4"

Z-matrix **! command for defining the Z-matrix**

C

CL1 C CCL

CL2 C CCL CL1 CLCCL

```
CL3    C    CCL    CL1    CLCCL    CL2    120
CL4    C    CCL    CL1    CLCCL    CL2    -120
end              ! Z-matrix
define
CCL = 1.711
CLCCL = tetra
end              ! define
end              ! molecule
basis name= 6-31G* end
OEP
method = Gill
end
output object=CHARGE_DENSITY:ATOMIC_NUMERICAL
end
stop
```

The input file starts by defining the Z-matrix of CCl_4. The command **OEP** informs the code which grid will be used for numerical integration. In the above example, the keyword **Gill** stands for SG-1 grid, the standard grid introduced by P. M. W. Gill et al. [29]. If the required integration grid is TA grid suggested by O. Treutler and R Ahlrichs [31], the keyword **TA** should be used. The result of running the above input file is an output file which contains the following:

Welcome to Mungauss - Development version (May 23, 2001)

Free format Z-Matrix for: CCl4

C

CL1 C CCL

CL2 C CCL CL1 CLCCL

CL3 C CCL CL1 CLCCL CL2 120

CL4 C CCL CL1 CLCCL CL2 -120

VARIABLES:

CCL = 1.71100000 CLCCL = 109.47122

...

...

Atomic electron densities:

atom	electron density
1	7.01893437731397
2	16.7761444015232
3	16.7517451354538
4	16.7220571437624
5	16.7244282733172

Total electron density: 73.9933093313705

CPUtime (seconds): 154.4049

In addition, the output file contains information related to the geometry of the molecule and its energy. The electron density of each atom is printed according to its order in the Z-matrix. Total electron density is the total number of electrons for the molecule, while CPUtime is the total calculation time in seconds.

The rest of this chapter discusses the performance of both SG-1 and TA integration

grids in integrating the electron density of the test atoms and molecules in terms of both the accuracy and the efficiency. Accuracy is defined in terms of the error of the calculated total number of electrons of a molecule against the exact total number of electrons of that molecule. Another measure of accuracy is the capability of the integration grid to assign equal atomic electron densities of symmetric atoms. The efficiency of the integration method is given by the CPU time.

6.2 Total number of electrons

The total number of electrons for a set of molecules containing only hydrogen and carbon atoms was calculated. These molecules are CH_4, C_2H_2, C_2H_4, C_2H_6, C_4H_{10}, C_6H_{14}, C_7H_{16}, C_9H_{20}, $C_{10}H_{22}$, $C_{11}H_{24}$, C_9H_{14}(TS), and $C_{12}H_{16}$(TS), where TS stands for transition state structure. The rest of the molecules are ground state structure. Table 6.1 gives the exact number of electrons and the calculated number of electrons using SG-1 integration grid and TA integration grid for the molecules under consideration. Table 6.1 shows that both grids overestimate the total number of electrons in ground state structures, except for TA grid in the case of CH_4, and underestimate the total number of electrons in transition state structures.

Figure 6.1 shows the relationship between the exact number of electrons and the error in the calculated number of electrons for both grids. Both integration grids perform quite similarly except for the molecules C_9H_{20}, $C_{10}H_{22}$, and $C_{11}H_{24}$, where TA grid overestimates the number of electrons more than SG-1 grid. While the error for all of the molecules is less than 0.1, the error for the molecules in their transition state is underestimated by as much as 0.5511 (C_9H_{14}(TS) (SG-1 grid)), much larger than that of the molecules in the ground state. This should not be surprising since transition state structures generally have bond lengths which are longer than in ground state

structures. Thus, the electronic charge is distributed over a larger space, and as the nuclear weight functions decay to zero away from the nuclei, a portion of the electron density is not accounted for.

Table 6.1: Total number of electrons using SG-1 and TA integration grids for molecules containing only C and H atoms

molecule	exact	SG-1 grid	TA grid
CH_4	10	10.0007	9.9961
C_2H_2	14	14.0106	14.0121
C_2H_4	16	16.0087	16.0109
C_2H_6	18	18.0062	18.0020
C_4H_{10}	34	34.0057	34.0047
C_6H_{14}	50	50.0051	50.0161
C_7H_{16}	58	58.0050	58.0316
C_9H_{14}(TS)	68	67.4489	67.4552
C_9H_{20}	74	74.0048	74.0565
$C_{10}H_{22}$	82	82.0048	82.066
$C_{12}H_{16}$(TS)	88	87.5130	87.4990
$C_{11}H_{24}$	90	90.0044	90.0670

Table 6.2 presents the exact number of electrons and the error in the calculated number of electrons for atoms and molecules containing atoms from the first and second rows of the periodic table. Figure 6.2, shows the error as a function of the number of the electrons for all molecules from He to $C_{10}H_{10}Cl_2$ including those containing C and H atoms only. It shows trends similar to those in Figure 6.1: transition state structures have larger errors than ground state structures, SG-1 and TA integration

grids have almost the same accuracy but for the molecules C_9H_{20}, $C_{10}H_{22}$, and $C_{11}H_{24}$, SG-1 grid is more accurate than TA grid.

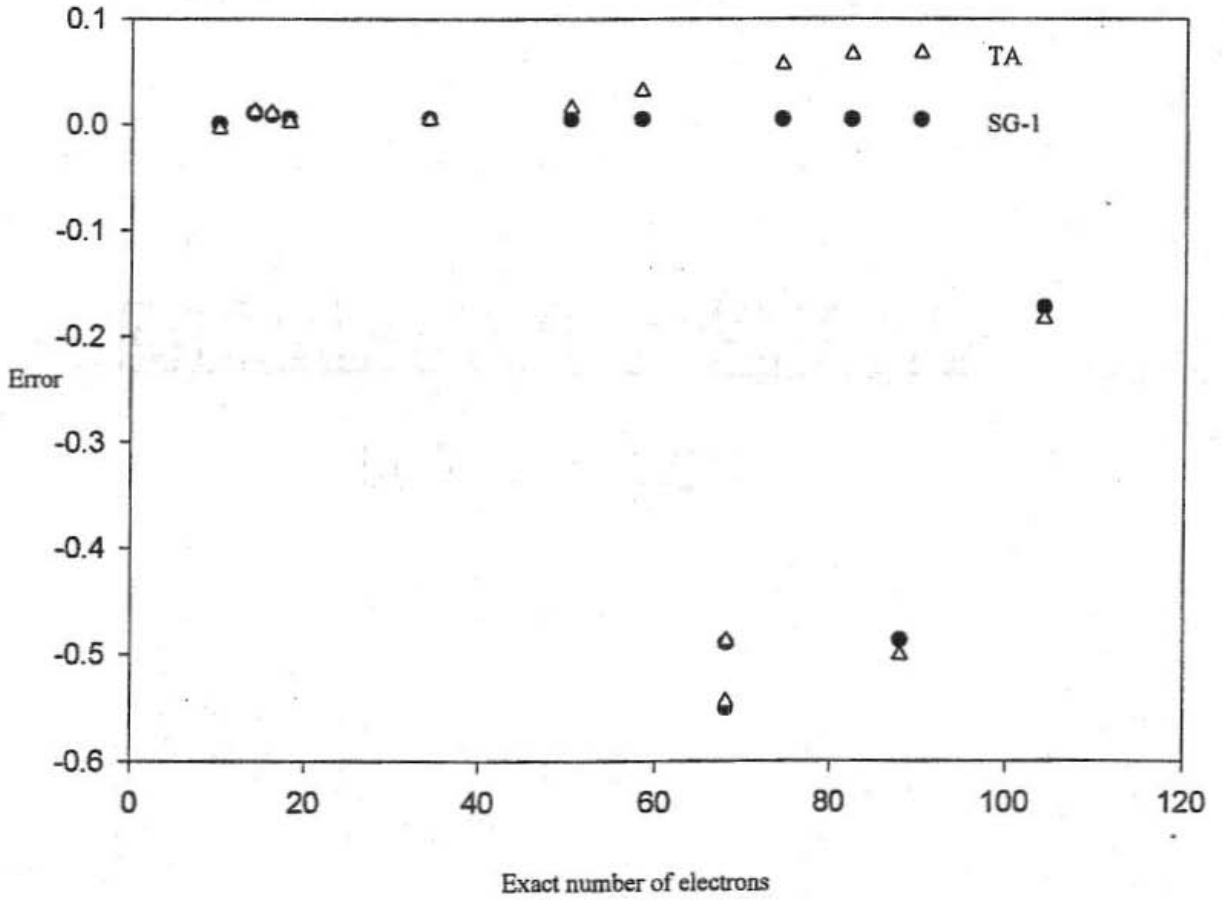

Figure 6.1: The error in the calculated total number of electrons versus the exact total number of electrons for molecules containing C and H atoms.

Table 6.2: Total number of electrons using SG-1 and TA integration grids for molecules containing atoms of the first and second rows of the periodic table

molecule	exact	SG-1 grid	TA grid
He	2	2.0000	2.0000
C	6	5.9999	6.0005
Ne	10	9.9990	10.0010
H_2O	10	10.0007	9.9995
NH_3	10	10.0011	9.9993
Mg	12	12.0000	11.9955
HCN	14	14.0114	14.0088
COH_2	16	16.0147	16.0113
CH_2NH	16	16.0129	16.0163
Ar	18	18.0000	18.0060
CH_3F	18	18.0011	18.0014
CH_3NH_2	18	18.0068	17.9994
CH_3OH	18	18.0079	18.0122
CH_3Cl	26	26.0066	26.0071
Cl_2	34	34.0070	34.0136
CHF_3	34	33.9628	33.9603
$FClO_2$	42	42.0040	42.0070

continued

molecule	exact	SG-1 grid	TA grid
ClH_2F_3	46	46.0313	46.0295
$FClO_3$	50	50.0542	50.0520
$ClHF_4$	54	54.0425	54.0317
PF_5	60	60.0174	60.0242
$SiHCl_3$	66	65.9339	65.9258
CCl_4	74	73.9933	74.0076
$C_7H_{14}Si_2$(TS)	84	83.5428	83.5060
$C_7H_7N_3O_2$(TS)	86	85.5110	85.5253
$C_{10}H_{10}F_2$	88	87.6468	87.6506
$C_{10}H_{10}Cl_2$	104	103.8275	103.8165

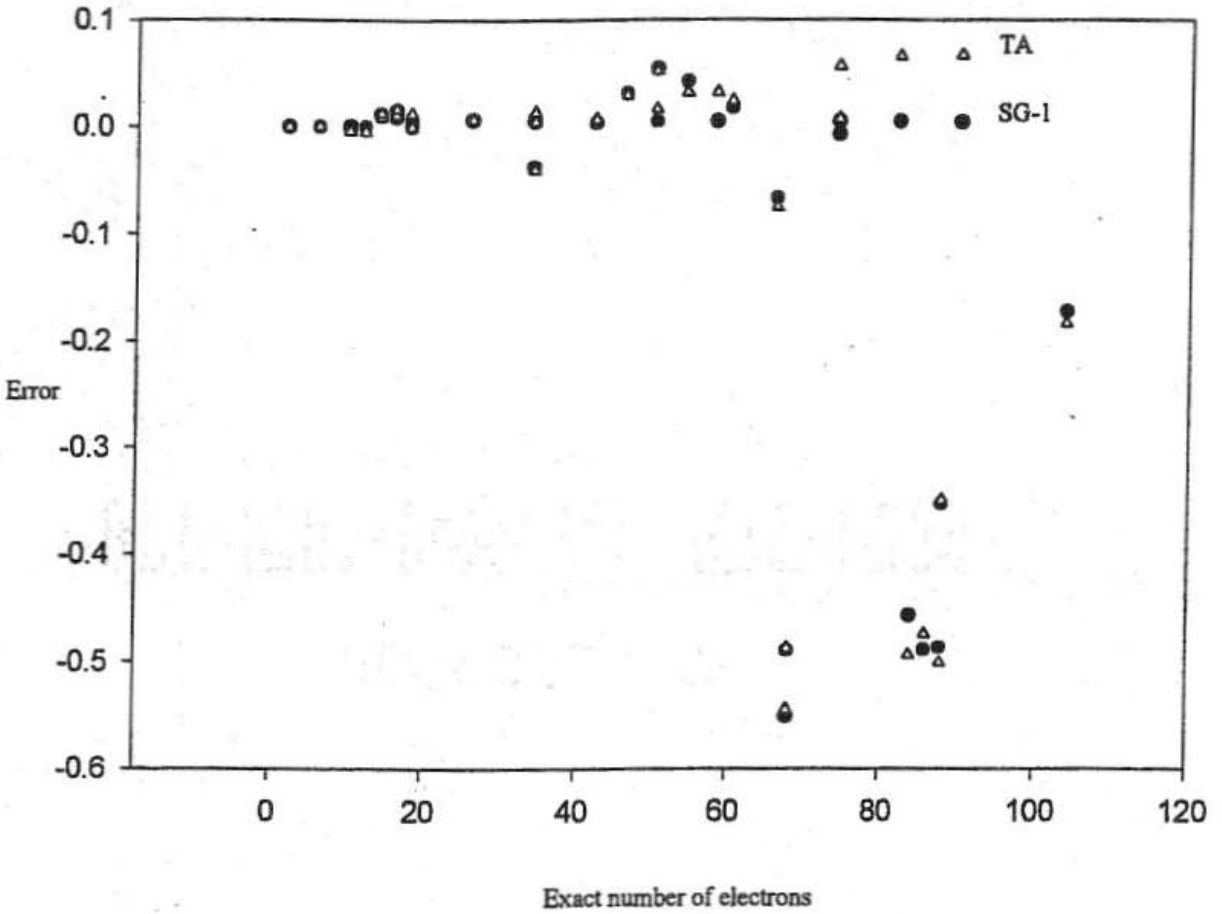

Figure 6.2: The error in the calculated total number of electrons versus the exact total number of electrons (He to $C_{10}H_{10}Cl_2$)

6.3 Atomic electron density

Although, SG-1 and TA integration grids calculate the total number of electrons to a reasonable accuracy, Table 6.3 shows that they fail to calculate reasonable electron densities belonging to each atom in a molecule. For instance, in a molecule like CHF_3, one would expect the hydrogen atom to be positively charged while the three fluorine atoms to be negatively charged. However, both SG-1 grid and TA grid predict the hydrogen atom to be negatively charged, 1.4336 in case of SG-1 grid and 1.0095 in case of TA grid. SG-1 grid predicts the fluorine atoms to be only "slightly" negatively charged, 9.1644, 9.1587, 9.1624, while TA grid predicts two of them to be positively charged, 8.9967, 8.9996. The same analysis applies to the rest of the molecules in Table 6.3. The failure of TA and SG-1 in assigning sensible electron densities to individual atoms is a result of the fact that Becke's scheme divides the molecular space into fuzzy overlapping Voronoi polyhedra cells. Therefore the boundary of each atom is not defined and consequently, the number of electrons belonging to each atom is not accurate. It is interesting that although of dividing the molecular space in this fuzzy way, the electron densities of atoms still add up to the right number of the total electrons.

Also, since CHF_3 has C_{3v} symmetry, the fluorine atoms should have the same electron density. Both of the grids violate the symmetry of the molecule, and the fluorine atoms have different electron densities. Angular grids around symmetric atoms do not preserve this symmetry. This is illustrated in Figure 6.3. The points r_1 and r_2 in the water molecule are equivalent by symmetry and thus $\rho(\mathbf{r_1}) = \rho(\mathbf{r_2})$. But, in terms of angular grids, point r_1 corresponds to the point r_3, not r_2, which may have different value for the electron density.

Table 6.3: Atomic electron densities using SG-1 and TA grids

molecule	SG-1	TA
Cl_2		
Cl1	17.0048	17.0080
Cl2	17.0022	17.0055
CH_3Cl		
C	4.8474	5.7239
Cl	16.7020	17.1599
H1	1.4857	1.0410
H2	1.4844	1.0396
H3	1.4872	1.0427
CHF_3		
C	5.0438	5.9512
H	1.4336	1.0095
F1	9.1644	9.0038
F2	9.1587	8.9967
F3	9.1624	8.9996
$ClHF_4$		
Cl	14.5714	16.4788
H1	1.5874	0.8526
F1	9.4710	9.1751
F2	9.4707	9.1747
F3	9.4714	9.1755
F4	9.4706	9.1749

molecule	SG-1	TA
$FClO_3$		
Cl	14.7507	16.3111
F	9.3905	8.9709
O1	8.6427	8.9709
O2	8.6327	8.2513
O3	8.6376	8.2551
PF_5		
P	12.6694	14.5453
F1	9.5055	9.1419
F2	9.4502	9.0700
F3	9.5111	9.1427
F4	9.4404	9.0600
F5	9.4408	9.0603
$SiHCl_3$		
Si	12.9652	13.6454
H	1.7401	1.0783
Cl1	17.0818	17.0737
Cl2	17.0688	17.0596
Cl3	17.0779	17.0687

ClH_2F_3		
Cl	14.3676	16.2919
H1	1.5295	0.8111
F1	9.6704	9.4454
H2	1.7467	0.9409
F2	9.3583	9.2699
F3	9.3589	9.2701
$C_7H_7N_3O_2$		
C1	5.1768	5.5964
C5	5.4196	5.8980
H1	1.3307	0.9247
H5	1.5216	1.0514
C2	5.3688	5.8753
C4	5.3489	5.8400
H2	1.5091	1.0148
H4	1.4574	0.9816
C3	5.0778	5.8499
H3	1.4634	1.0041
H6	1.5083	1.0058
N7	7.1763	7.1507
N8	7.1525	7.1266
C6	5.6137	5.8731
C9	5.6408	5.9010
O6	8.3381	8.2531
O9	8.3449	8.2605

N10	6.6351	7.0010
H9	1.4270	0.9174
H6	1.5083	1.0058
N7	7.1763	7.1507
N8	7.1525	7.1266
C6	5.6137	5.8731
C9	5.6408	5.9010
O6	8.3381	8.2531
O9	8.3449	8.2605
N10	6.6351	7.0010
H9	1.4270	0.9174
O9	8.3449	8.2605
N10	6.6351	7.0010
H9	1.4270	0.9174
$C_{12}H_{16}$		
C1	5.1214	5.6218
C2	5.4751	5.9359
C3	5.5893	6.0723
C4	5.4582	5.9682
C5	5.0612	5.8558
C6	4.9709	5.7653

H1	1.5178	1.0713
H2	1.4182	0.9810
H3	1.5176	1.0406
H4	1.4571	1.0033
H5A	1.4749	1.0492
H5B	1.4804	1.0501
H6A	1.4532	1.0445
H6B	1.4933	1.0611
C7	5.4555	5.9564
C8	5.4740	5.9540
C9	5.5102	5.9867
C10	5.4800	5.9862
C11	5.0607	5.8588
C12	5.0773	5.8749
H7	1.5214	1.0416
H8	1.5139	1.0256
H9	1.4973	1.0389
H10	1.5084	1.0583
H11A	1.4776	1.0573
H11B	1.4869	1.0398
H12A	1.4836	1.0567
H13B	1.4778	1.0434

$C_{10}H_{10}F_2$		
C1	5.2696	5.7062
C2	5.4599	5.9226
C3	5.5228	6.0003
C4	5.3319	5.9229
C5	5.4836	5.9435
C6	5.4891	5.9643
C7	5.3692	5.9742
C8	5.4391	5.9063
C9	5.5261	5.9958
C10	5.4896	5.9663
H1	1.4584	1.0086
H2	1.4550	1.0089
H3	1.4743	1.0216
H41	1.4714	1.0222
F42	9.2916	9.1242
H5	1.4669	1.0110
H6	1.4720	1.0024
F71	9.2853	9.1114
H72	1.4718	1.0119
H8	1.4898	1.0229
H9	1.4648	1.0061
H10	1.4645	0.9971

molecule	SG-1	TA
$C_7H_{14}Si_2$		
C1	5.1959	5.6836
C2	5.4801	5.9627
H1	1.4927	1.0593
H2	1.4007	0.9730
C3	4.9979	5.7578
C4	5.0962	5.8630
H3	1.4558	1.0252
H4	1.4969	1.0445
H5	1.5011	1.0610
H6	1.4626	1.0326
C5	5.4773	5.9390
C6	5.5148	5.9767
C7	6.5791	6.3126
H7	1.5305	1.0327
H8	1.5222	1.0254
Si1	11.7603	13.4858
Si2	11.7626	13.4703
H9	1.6338	1.1398
H10	1.6195	1.1389
H11	1.6300	1.1357
H12	1.6323	1.1208
H13	1.6439	1.1265

C_9H_{14}(TS)		
C1	5.1780	5.6313
C5	5.4611	5.9724
H1	1.3793	0.9707
H5	1.5557	1.0836
C2	4.9880	5.8066
C4	4.9954	5.8174
H2A	1.4608	1.0411
H4A	1.4594	1.0392
H2B	1.4961	1.0273
H4B	1.5001	1.0300
C6	5.4951	6.0018
C7	5.4817	5.9968
C8	5.8779	6.1081
H6	1.4801	1.0060
H7	1.5132	1.0321
C9	4.7060	5.7666
C10	4.6890	5.7646
H9B	1.4592	1.0659
H9C	1.4666	1.0601
H9D	1.4464	1.0570
H10B	1.4564	1.0569
H10C	1.4599	1.0659
H10D	1.4427	1.0529

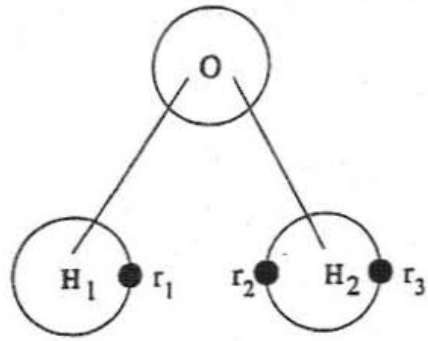

Figure 6.3: The symmetry problem illustrated in H_2O molecule

6.4 Efficiency: CPU timing

In this section CPU time is used as a criterion to compare the efficiency of both the SG-1 and TA integration grids. The following piece of code calculates the CPU time:

```
call CPU_time(tbegin)
...
...
call CPU_time(tend)
tcpu = tend - tbegin
```

The function CPU_time (Fortran 95 intrinsic function) was called at the beginning of both grids and the initial time of execution was stored in the variable **tbegin**. At the end of both grids the CPU time function was called again and the final time was stored in the variable **tend**. The variable **tcpu** contains the time of execution of the individual grid which also includes the time required to perform the SCF (self consistent field). The CPU time calculated in this way is still reliable in comparing the efficiency of both grids, since SCF time is independent of the integration grid for

the same molecule. Table 6.4 presents the CPU time for both grids for all the test molecules. From Table 6.4, it is clear that the CPU time for TA is always smaller than CPU time for SG-1 except for C_4H_{10}, and PF_5, where, the CPU time is almost the same. Thus, TA integration grid is, in general, more efficient than SG-1 integration grid.

Table 6.4: Total CPU time using SG-1 and TA (seconds)

molecule	SG-1	TA
CH3Cl	44.0	37.3
CHF3	63.1	53.6
ClH2F3	82.8	70.4
ClHF4	107.4	90.3
FClO3	113.1	94.6
C_4H_{10}	119.1	121.8
SiHCl3	128.1	108.0
PF_5	156.3	157.3
C_6H_{14}	351.3	307.2
C_7H_{16}	541.2	464.4
C_9H_{20}	1098.6	947.7
$C_7H_7N_3O_2$	1469.4	1248.9
$C_{10}H_{22}$	1525.7	1321.4
C_9H_{14}	1541.5	1336.7
C_9H_{14}	1545.4	1341.4
$C_{10}H_9F_2$	1902.9	1706.3
$C_7H_{14}Si_2$	1922.4	1669.3
$C_{11}H_{24}$	2010.9	1748.8
$C_{12}H_{16}$	3033.0	2612.8

Chapter 7

Conclusions and future work

7.1 Conclusions

The two three-dimensional integration grids SG-1 and TA have almost the same accuracy except for the molecules $C_{10}H_{22}$ and $C_{11}H_{24}$ where SG-1 grid is superior to TA grid. However, TA grid is more efficient than the SG-1 grid in terms of CPU time for almost all of the molecules. For transition state structures, the error in the integration of the electron density is, in general, much larger than in ground state structure which could lead to a serious error if these integration techniques are used in DFT energy calculations. Any numerical integration algorithm based on Becke's scheme to calculate the electron densities of individual atoms is seriously inaccurate. So far, Bader's theory is is still the only reliable method for calculating meaningful atomic electron densities.

7.2 Future Work

One of the interesting problems that could be pursued is to solve the symmetry problem. A possible solution to this problem could be to associate a coordinate system with each atom instead of a single coordinate system for the whole molecule. The atomic coordinate system would have to reflect the environment the atom is in. In this way, atoms equivalent by symmetry would have the same local coordinates and therefore the same integration grid.

Another interesting problem is to use this code to calculate the energy using DFT and investigate the effect of the quality of the numerical integration technique on the energy calculations espicially the ones in ground state structures versus transition state structures.

Bibliography

[1] P. Hohenberg and W. Kohn, *Phys.Rev.* 136, B864 (1964).

[2] W. Kohn and L. J. Sham. *Phys.Rev.*, 140, A1133 (1965).

[3] B. G. Johnson. in *Modern Density Functional Theory: A Tool for Chemistry*, edited by J. M. Seminario and P. Politzer p. 169 (ELSEVIER, 1995)

[4] P. G. Mezey. *Mol. Phys.*, 96, 169-78 (1999).

[5] A. D. Becke. *J. Chem. Phys.*, 88, 2547 (1988).

[6] *Modern Quantum Chemistry: Introduction to advanced Electronic structure Theory*, A. Szabo and N. S. Ostlund (McGraw-Hill, 1989)

[7] P. D. Walker and P. G. Mezey. *J. Am. Chem. Soc.*, Vol. 115, No. 26 (1993) p. 12423–12430

[8] P. D. Walker and P. G. Mezey. *J. Am. Chem. Soc.*, Vol. 116, No. 26 (1994) p. 12022–12032

[9] R. Cools. *Acta Numerica*, Vol. 6 1997 p. 1

[10] G. Evans. *Practical Numerical Integration* , Wiley, (1993).

[11] A. H. Stroud. *Approximate Calculation of Multiple Integrals*, Prentice-Hall, (1971).

[12] P. J. Davis and Philip Rabinowitz. *Methods of Numerical Integration*, Academic Press, (1984).

[13] H. V. Smith. *Numerical Methods of Integration*, Chartwell-Bratt, (1993).

[14] R. Cools. *Numerical Integration Recent Developments, Software and Applications*, edited by Terje O. Espelid and Alan Genz. NATO ASI Series, Vol. 357, 1-24 (1991).

[15] S. L. Sobolev and V. L. Vaskevich. *The Theory of Cubature Formulae*, Kluwer Academic Publishers, (1997).

[16] V. I. Lebedev. *Zh. Vychisl. Mat Mat. Fiz.*, 15, 1, 48-54 (1975).

[17] V. I. Lebedev. *Zh. Vychisl. Mat Mat. Fiz.*, 16, 2, 293-306 (1976).

[18] V. I. Lebedev and A. L. Skorokhodov. *Russ. Acad. Sci. Dokl. Math.*, Vol. 45, No.3, 587 (1992).

[19] G. te Velde and E. J. Baerends. *J. Comput. Phys.* , 99, 84 (1992).

[20] M. R. Pederson and Koblar A. Jackson. *Phys. Rev. B*, Vol. 41, No. 11, 7453 (1990).

[21] F. W. Averill and G. S. Painter. *Phys. Rev. B*, Vol. 39, No. 12, 8115 (1989).

[22] R. Eric Stratmann, Gustavo E. Scuseria, Michael J. Frisch, *Chem. Phys. Lett.*, 257, 213-223 (1996).

[23] B. Delley, *J. Chem. Phys.*, 92(1), 508 (1990).

[24] R. F. Bader and Nguyen-Dang, T. T., *Adv. Quantum Chem.*, 14, 63, (1981).

[25] F. W. Biegler-Koening, Nguyen-Dang, T. T., Y. Tal, R. F. Bader, and A. J. Duke, *J. Phys.* B 14, 2739 (1981).

[26] R. F. Bader, in *Encyclopedia of Computational Chemistry*, Vol. 1, 64, (1996)

[27] J. M. Pérez-Jordá, E. San-Fabián and F. Moscardó *Comput. Phys. Comm.*, 70, 271-284 (1992).

[28] J. M. Pérez-Jordá, A. D. Becke, E. San-Fabián. *J. Chem. Phys.*, 100(9), 6520 (1994).

[29] P. M. w. Gill, B. G. Johnson, J. A. Pople. *Chem. Phys. Let.*, Vol. 209, No. 5, 506 (1993).

[30] M. Krack and M. Koester. *J. Chem. Phys.*, Vol. 108, No. 8, 3226 (1998).

[31] O. Treutler and R. Ahlrichs. *J. Chem. Phys.*, Vol. 102, No. 1, 346 (1995).

[32] C. W. Murray, N. C. Handy, and G. J. Laming. *Mol. Phys.*, Vol. 78, No. 4, 997 (1993).

[33] M. E. Mura and P. J. Knowles. *J. Chem. Phys.*, Vol. 104, No. 24, 9848 (1996).

[34] R. A. Poirier, MUNgauss 1.0 (Fortran 90 version). *Chemistry Dept., Memorial University of Newfoundland, St. John's, NF, 2001.* with contributions from S. Bungay , A. El-Sherbiny, T. Gosse, D. Keefe, C. C. Pye, D. Reid, M. Shaw, Y. Wang, and J. Xidos.

[35] J. C. Slater. *Phys. Rev.*, Vol. 36, No. 57, (1930).

[36] C. Redwine. *Upgrading to Fortran 90,* Springer-Verlag (1995).

[37] A. C. Marshall. *HPF Programming Course Notes,* University of Liverpool (1997).

[38] J. C. Adams, W. S. Brainerd, J. T. Martin, B. T. Smith, and J. L. Wagener. *FORTRAN 95 HANDBOOK Complete ISO/ANSI reference,* MIT press (1997).