Truchas User's Guide

R.L. Martin, T.V. Russo, B.H. Lengsfield III, P.W. Saxe, M. Page, G.J. Tawa, B.I. Schneider, M.O. Braunstein, P.J. Hay, and A.K. Rappe

April 18, 1995

Contents

List of Figures

List of Tables

Preface

Truchas[8] is a suite of programs which can be used to perform and analyze Hartree-Fock (HF) and density-functional-theory (DFT) calculations. It grew out of the MESA suite of electronic structure programs. The latter is based on Hartree-Fock, multi-configuration-self-consistent-field (MCSCF), and configuration-interaction (CI) methods. Whereas MESA is primarily useful for performing accurate CI expansions on the ground and excited states of small molecules ($2-10$ atoms), Truchas is geared towards the ground states of moderately large molecules ($10-500$ atoms). The DFT code is still under active development, and many features which have become "routine" in the Hartree-Fock world are still not implemented. An important example as of this writing are analytical second derivatives of the DFT energy with respect to nuclear motion.

Many people have contributed to the development of this program. The MESA program was designed and much of it written by Paul Saxe and Rich Martin at Los Alamos National Laboratory (LANL) from 1985-1987. Byron Lengsfield (Lawrence Livermore National Laboratory) contributed the MCSCF code and derivatives of MCSCF and CI wavefunctions. Mike Page (North Dakota State University) contributed the frequency and reaction path links. More recently, Tom Russo (LANL) added DFT links and second derivatives of effective-core-potential integrals. Greg Tawa (LANL) is responsible for the solvent reaction field capabilities. Additional contributions were made by Barry Schneider (LANL), Jeff Hay (LANL), and Tony Rappé (Colorado State University).

This manual assumes some familiarity with the concepts and methods of quantum chemistry. Most features of the code are described, but not explained in detail. I have tried to reference representative papers where appropriate, but for more general background material, consult Foresman

and Frisch,[3] Szabo and Ostlund,[12] and Parr and Yang.[10]

On conventions: throughout the manual, text which you might enter from a keyboard, such as keywords and options to the program, environment variables in your shell script, or disc file names are denoted with `typewriter font`.

Chapter 1

Getting Started

Truchas performs an electronic structure calculation by running a sequence of programs (links) in order, communicating the results of one calculation to another through disk files. The location of these disk files is passed along to the programs by means of environment variables which you must set up in your `.cshrc` file. The actual execution of the programs is performed by a shell script, whose location must also be made known to your `.cshrc` file. This chapter explains how to set up your personal files in order to run Truchas.

1.1 Environment Variables

When you submit a Truchas job, a shell script begins running which invokes the actual executables. It sends and receives messages from the executables and determines which program to run next. The script must therefore know the disk directory where the executables are to be found. It does this by reading an environment variable `(MESA_BIN)`. Once the programs begin running, Truchas accesses a file called `mesa.dat` in order to extract information concerning the sequence of programs which will be run, the values of orbital exponents and contraction coefficients for often used basis sets, etc. The shell finds this data file by accessing the environment variable `(MESA_HOME)`. In addition, certain scratch files are generated and discarded in the course of a calculation, and Truchas determines on which disk these files should reside based on the value of the environment variable `(MESA_TMP)`. In the instal-

lation here, the executables reside in the directory `/t12/mesa/hp`, the data file on `/usr2/mesa`, and the scratch directory depends on which machine you are running on. Since the two-electron integrals and other large, heavily accessed files reside on this scratch directory, it should be local to the cpu doing the calculation. Otherwise, two-electron integrals may be passed over the network, causing an ether storm which you may be sure will introduce you to others on your LAN you may not yet have met. My `.cshrc` file has a section

```
setenv MESA_BIN /t12/mesa/hp
setenv MESA_HOME /usr2/mesa
setenv HOST ‘hostname‘
if ( $HOST== "cesium") then
   setenv MESA_TMP /cesium/rlm/tmp
else if ( $HOST== "platinum") then
   setenv MESA_TMP /pt0/rlm
else if ( $HOST== "aluminum") then
   setenv MESA_TMP /usr4s/tmp
else if ( $HOST == "silicon") then
   setenv MESA_TMP /usr5s/tmp
endif
```

which interrogates the system and assigns the scratch disk depending on the host.

1.2 The Shell Script

The shell script which runs the program is called “mesa” and should be found in the directory `$MESA_BIN/mesa`. Make sure your path variable has this appended so that your shell can find the script. An easy way to do this is to place a line toward the end of your `.cshrc` file

```
set path = ($path $MESA_BIN)
```

You will generally be providing three arguments to the shell script: the name of the input file, the output file, and the “check” file. The check file is a binary file which encapsulates the most important results of the run. It can be read by subsequent jobs to provide an initial guess for the wavefunction,

optimized geometries from a previous run, molecular orbital information for visualization programs, etc. Generally, you will want to save both the output and the check files from successful runs. The syntax used by the script for file replacement is:

```
mesa  inp=myinp out=myout chk=mychk
```

The input file defaults to `./mesa.inp`, the output file to `./mesa.out`, and the check file to `./mesa.chk`.

Sometimes you may want to specify the "size" parameter, or greatest amount of memory any given program can use during the run. The `siz` parameter is given in real*8 words. It defaults to 6000000 (24MBytes). If you want your job to use more, e.g. 32MBytes, then `siz=8000000` will do the trick. If this is not enough memory, the program will abort somewhere along the road and provide information about a more appropriate size.

Other sometimes useful shell commands are the `start` and `stop` directives. These tell the script to begin or end execution at specific link numbers. The invocation

```
mesa inp=myinp out=myout stop=m302
```

will begin normal execution of the job, but stop execution before M302, the one-electron integrals link, is run. At this point you might examine your output file to see if the geometry generated is what you thought it would be. If a problem is spotted, you can correct your input file and resubmit the job as above. If all looks fine, the command

```
mesa -pid xxx inp=myinp out=myout start=m302
```

will resume execution. Note that the flag `-pid` is followed by the "process identification number" of the earlier submission. This is necessary because MESA has assigned unique names to temporary scratch files based on the pid of the earlier submission. The pid is printed along with other header information at the beginning of the output file.

When you execute the mesa script, you will notice messages coming back from the script informing you of your progress. Each link, as it executes, returns a message of the invocation line and the timing statistics. These may be redirected to a log file if you submit the job in the background:

```
(mesa  inp=myinp out=myout chk=mychk siz=xxx >>&mesa.log)&
```

1.3 An Example

The input deck below performs a density-functional-theory calculation on H_2O.

```
$route
dft scf=(exchf=slater,corrf=vwn)
guess=(huckel)

$title
h2o test case: closed shell dft; slater-vwn

$geom
o
h1 o r1
h2 o r2 h1 a1

r1=1.949
r2=1.949
a1=112.3
$end
```

Note that the input is divided into sections. Each new section is recognized by the program with a "dollar sign", $, as the first character of a line. The `$route` section contains everything between `$route` and `$title`. The sections therefore need not be separated by a blank line as in the example above. Note that the last section, `$geom`, is terminated with `$end`.

The content of each section contains directives which are read in a free-format manner. The `$route` section reads keywords which determine the sequence of links to run and the options to various links in the chain. In the example, a DFT calculation is requested using the Slater exchange functional and the Vosko-Wilks-Nusair (VWN) correlation functional. The initial guess for the self-consistent Kohn-Sham procedure is to be determined from a Huckel calculation. The `$title` section contains descriptive information about the job. The `$geom` section defines the molecular geometry. In the example above, the geometry is read in Z-Matrix format (this is the default). If you are not familiar with Z-Matrices, see Chapter 3, or Foresman and

Frisch.[3] In brief, bond lengths, bond angles and dihedral angles are used to define the molecular coordinates. In the example, the hydrogen atom h1 is a distance r1 from the oxygen atom, o. The values of any symbolic variables are separated from the Z-Matrix definition by a blank line. In the case above, h1 will be placed 1.949 Å from the oxygen.

The output from the job is reproduced below:

```
put it here
```

The output consists of several sections. After some banner information is printed which documents the program version, date, machine and the person submitting the job, a list of the file names is printed. Note that the scratch file names are appended with the pid of the job. The job title is reproduced and the route taken by the links printed. After the route information, the options string is echoed. This is simply a copy of the `$route` section of the input (all nulls and redundant blanks have been removed).

The next output is from link M101. M101 processes the `$geom` input section, determining which coordinates are symbolic variables and which are constants. M101 also checks the charge and spin-multiplicity of the molecule, testing against what it thinks is acceptable given the elements entered in the Z-Matrix. Unless specified, the molecular charge and spin-multiplicity default to `q=0` and `2S+1=1`. M102 then reads or produces basis set information. In this example, no basis set was declared and so it provided the default, the "double-zeta", dz, basis set for each atom. The default bases are the $[4s2p]$ contractions by Dunning of the $(9s5p)$ primitive Gaussian basis of Huzinaga.

This information is followed by the distance matrix and interatomic angles. These are produced by M202, which converts the symbolic variables into specific Cartesian coordinates. M202 also determines the point group of the molecule from the coordinates, rotates the molecule into a standard geometry depending on its conformation, and computes rotational constants and assigns isotopic masses to the atoms.

Truchas is now set to do some work. Link M302 computes the one-electron (overlap, kinetic energy, and nuclear attraction) integrals. M312 computes the two-electron integrals, and M330 sorts them into an order which is convenient for the DFT code.

Link M401 generates an initial guess for the DFT step. In this case a Huckel calculation is performed, and the guess vectors projected onto pure

symmetry-adapted-linear-combinations (SALC's) using the symmetry information produced by M202.

The Kohn-Sham equations are solved in link M511. Output from M511 consists of the symmetries of occupied and virtual orbitals, the specific exchange-correlation (XC) operator being used, information concerning the grids utilized to integrate the XC operator, cutoffs on the density and density-matrix used in the numerical integration, and the nuclear repulsion energy. At convergence, the total energy, the number of iterations needed to converge the solution, and a measure of the convergence quality are printed.

M1951 analyzes the Kohn-Sham wavefunction and prints the bond-order matrix and Mulliken charges.

The final link, M2001, summarizes times spent in each link during the calculation and is responsible for producing the checkpoint file.

1.4 How to Read This Manual

The remaining chapters describe the major input sections of Truchas. You should read Chapter 2, which describes how keywords and their options are communicated to Truchas. It also contains a list of commonly used keywords which can serve as a quick reference guide. The remaining Chapters can be read as needed. Appendix E contains a number of sample input decks which may be helpful if the primary documentation isn't clear.

Chapter 2

Routes and Keywords

The sequence of programs (links) run by Truchas to solve a particular problem is determined by a construct known as the route. Several standard routes which execute commonly performed tasks are built into the `mesa.dat` file and are loaded into memory depending on the presence of certain keywords in the `$route` section of the input file. This chapter explains the route in a little more detail, and specifies some of the major keywords and their options.

2.1 Standard Routes

The first link in a Truchas calculation is M0. It does very little, its primary purpose being to open the output file and to print the banner information recording the date, the user, the machine, etc. It then calls the next link in the sequence, which is always M1. M1 generates the route, the list of programs that will be executed in sequence. M1 first scans the input deck for a section descriptor `$nonstd`. If found, this signals that the user has entered a special route explicitly. Nonstandard routes are discussed in detail in Appendix D. If there is no `$nonstd` section, M1 reads the `$route` section of the input deck searching for keywords which determine what type of calculation is desired. The keywords may be in any order, separated by either a blank character or a comma, and the search is case-insensitive. M1 then extracts the appropriate route from the `mesa.dat` file. A route has the appearance:

```
1//1,2;
2//2;
```

```
3//2,12,30;
4//1;
5//11;
19//51;
20//01;
```

which is the route generated for a single-point DFT calculation. The first number in each line specifies the "overlay" number, and the entries after the double slash specify programs within an overlay. For example, the route above will execute M0 and M1 – all routes do – and then M101, M102, M202, M302, M312, M330, M401, M511, M1951, and finally M2001. All programs which perform a similar function or accomplish different aspects of a common task are generally grouped together in the same "overlay", although they are not overlaid in the sense familiar to computer programmers. For example, M302 calculates one-electron integrals, M312 calculates two-electron integrals and writes them to disk, while M330 sorts the two-electron integrals into a form convenient for processing in the DFT code M511. The last function performed by each program is to call a subroutine which parses the route information, decides which program to run next, and writes the name of the next program to the standard error unit. The shell script overseeing the execution intercepts this message and invokes the next program.

2.2 Keywords

The `$route` section contains keywords which are used to define the standard route to be taken as well as to convey information to individual links about what options to invoke during the calculation. The general syntax of keyword specification is

```
key=(value1,value2,...)
```

For example, in order to specify that the geometry to be read in the `$geom` section is in Cartesian coordinates (instead of the default Z-Matrix style) and that the coordinates are in atomic units (instead of the default Å units, the directive

```
geom=(coord,inau)
```

is placed in the `$route` section. Keywords may be nested, as in:

```
scf=(cycles=64,print=(guess,one-electron),dencut=1.0d-06)
```

which tells Truchas that the maximum number of cycles the SCF code can use to converge the wavefunction is 64, to print the guess vector and the one-electron integrals, and that the density cutoff value is 10^{-6}. As the example above shows, the keyword value may be a character-string, an integer, or a floating-point number.

Some of the more often used keywords in the `$route` section are reproduced in Table 2.1:

Keyword	Values	Default	Comments
hf		.true.	request HF calculation
dft		.false.	request DFT calculation
q	integer	0	molecular charge
2s+1	integer	1	spin-multiplicity
geom		Z-matrix,Å degrees	geometry format
	coord		Cartesian coordinates
	inau		distances in bohr
	inrad		angles in radians
basis	character	dz	basis set name
grid	character	sg1	atomic grid name
sym		on	use of symmetry
	on		use symmetry
	off		turn symmetry off
ints		recompute	integral computation
	reuse1		reuse $1e^-$ integrals
	reuse2		reuse $2e^-$ integrals
	reuse		reuse $1e^-, 2e^-$ integrals
guess		core	initial guess
	core		core Hamiltonian
	huckel		Huckel Hamiltonian
	chk		read from checkpoint
	rdchk		read chk and project
	alter		alter the guess
scf			scf control
	cycles	30	maximum iterations
	convergence	8	diis error $\leq 10^{-n}$
			energy change $\leq 10^{-n-1}$
properties		mulliken analysis	properties
	e0		monople moment
	e1		dipole moment
	e2		quadrupole moment
	e3		octupole moment
	e4		hexadecapole moment
	v0		potential at nuclei
	v1		electric field at nuclei
	v2		electric field gradient at nuclei
opt		murtaugh-sargeant	geometry optimization
	fletcher-powell		fletcher-powell algorithm
	murtaugh-sargeant		murtaugh-sargeant algorithm
	berny		berny algorithm
	ts		transition state
	cycles	min(20,nvar+10)	maximum iterations
force-constants		analytic	force constants

Chapter 3

Geometry Specification

The positions of the atomic nuclei may be conveyed to Truchas by either Z-Matrix or Cartesian coordinate input. In addition, ghost centers (centers with a basis set and zero nuclear charge), point charges (centers with no basis set but a finite nuclear charge), and dummy centers (centers with neither a basis set nor a nuclear charge) may be specified. This chapter discusses the input required to specify the molecular geometry.

3.1 The $Geom Input Section

The molecular geometry is specified in the `$geom` section of the input. It is read by M101, and this section is always required, except in the case of a restart, a `geom=rdchk` or a `geom=chk` directive. The input is free-field; the items on a card may be separated by one or more blanks.

There are two means used to specify a geometry, the Z-Matrix format and the Cartesian coordinates format. These are discussed in turn below.

3.1.1 Z-Matrix Format

The Z-Matrix format provides a means of specifying the molecular geometry in terms of internal coordinates: bond lengths, bond angles, and dihedral angles. The Z-Matrix description *must* be used in the case of a geometry optimization. It is an old construct, and is the default in Truchas. Two sets of cards may be read; the first set defines the nuclear centers and the second

set is present only if symbolic parameters are used in the definition of the nuclear centers.

The first series of cards, one for each center, is read in the format:

```
Element [(basis= ,grid= ,z= ,r=])] N1 Length N2 Alpha N3 Beta J
```

Element specifies the chemical nature of the nucleus. It may consist of simply the chemical symbol, such as "H" or "Pt", or it may begin with the chemical symbol follwed by a numeric identifier: "Pt4" to designate the fourth platinum atom. Ghost centers (centers with a basis but zero charge) may be denoted by the special symbol "Zq". Dummy nuclei (centers with neither a basis set or a nuclear charge) are sometimes useful to define angles and are signalled by the elemental symbol "X" or "-". A symbol is required for every center. The remainder of the element field is optional and specifies the basis set, nuclear charge, and the atomic radius for solvent calculations.

basis= specifies the basis-set name associated with this center. If the basis field is unspecified, the basis specified in the `$route` section is supplied. If you wish to specify a basis for this nucleus different from the default, enter its' name (16 character maximum) in parenthesis. The basis-name may be either one of the basis sets stored in the `mesa.dat` file, or one specified explicitly in the input deck (see Chapter 4.

grid= specifies the integration grid to be associated with this atom. If the grid field is unspecified, the grid specified in the `$route` section is supplied.

z= can be used to modify the nuclear charge associated with the site.

r= assigns the atomic radius to be used in defining the solvent cavity. It acquires the same units as the rest of the entries.

N1 specifies the (previously defined) nucleus for which the internuclear distance to center N will be given. This item may be either an integer (N1 < N) giving the number of the card defining the previous site or the alphanumeric label of the previous center.

length is the internuclear distance R(N1,N). It may be either a positive floating point number in Å (unless modified by the `geom=inau` directive

in `$route`, or a symbolic alphanumeric string of up to 16 characters. In the latter case, the distance is represented by a parameter, which will be given a value in the second section of `$geom`. If a length is to be varied during an optimization run, it must be specified symbolically. Note that the symbolic parameter may be used to specify two distinct lengths, say the two OH distances in the H_2O molecule, in which case they are constrained to be equal. The items N1 and Length are required for all nuclei after the first. For the second center, only Element, N1, and Length are required. The nucleus is placed on the z-axis.

N2 gives the center for which the angle $\alpha(N, N1, N2)$ is given. Again, it may be either an integer or an alphanumeric string matching a previous Element entry. Note that N1 and N2 must refer to different nuclei.

Alpha is the internuclear angle, $\alpha(N, N1, N2)$. This may be a floating point number giving the angle in degrees (unless modified by the `geom=inrad` directive in `$route`) or an alphanumeric string representing a symbolic parameter. The numerical value of Alpha must lie in the range $0 < \alpha < 180$ degrees. N2 and Alpha are required for all centers after the second. For the third center, only Element, N1, Length, N2, and Alpha are required. This center is placed in the xz-plane.

N3 differs in meaning, as does Beta, depending on the value of the last item, J. If J=0 or is omitted, N3 specifies the center for which the internuclear dihedral angle $\beta(N, N1, N2, N3)$ will be given. As with N1 and N2, this may be an integer or a previously defined alphanumeric string. This center must be distinct from N1 and N2.

Beta is the internuclear dihedral angle (unless J=1,see below). This may be a floating point number giving the angle in degrees (unless modifed in `$route`), or a (signed) alphanumeric string representing a parameter. The dihedral angle is defined as the angle between the planes (N,N1,N2) and (N1,N2,N3). Beta must lie in the range $0 < \beta < 180$ degrees. The sign is positive if the movement of the directed vector N1 $\rightarrow$ N2 towards the directed vector N2 $\rightarrow$ N3 involves a right-handed screw motion.

J allows you some freedom over the specification of the dihedral angle. Although it is always possible to specify the center by a bond length, a

bond angle and a dihedral angle (which is what you must do if J=0 or is absent), it is sometimes more convenient to use a bond length and two bond angles. This is called for by using J=1. In this case, the internuclear angle $\beta(N, N1, N3)$ is specified. As before, this may be either a floating point value or an alphanumeric string representing a parameter. In the event that you specify two bond angles, there will be two possible positions for center N. This is fixed by the sign of J. Thus J=+1 if the triple vector product (N1-N) X [(N1-N2) X (N1-N3)] is positive and J=-1 if the product is negative.

Parameter Definition

If you have not specified any parameters, this completes the `$geom` section. If you have, then you must supply one last set of information, the initial values of the parameters. This set of data must be separated from the Z-Matrix by a blank card, after which one card for each parameter is read in the form:

```
parameter= [d2e= ,type]
```

parameter is the alphanumeric string defined by the preceding section. It is given a floating point value representing its' initial magnitude.

d2e= specifies how Truchas obtains initial guesses for the force constants to be used in an optimization. If omitted, internally stored guesses are provided. The d2e value may be a floating point number, which then specifies the initial diagonal second derivative associated with this parameter (in Hartrees/(bohr^2), or Hartrees/(rad^2)). If omitted, internally stored guesses are provided. The d2e specification is only used in Opt=Berny runs (see Chapter ref:optimizations.

type denotes the status of the parameter. It can be either the character string `constant`, in which case the parameter is not varied in an optimization, or `variable`, the default type. This field is included for convenience only. A `constant` could have been just as easily entered directly in the Z-Matrix as a floating point number.

3.1.2 Cartesian Coordinate Format

This case is specified by the `geom=coord` directive in the `$route` section. The input consists of a number of cards, one for each center, which specify the element name, basis set, nuclear charge, solvent radius, and Cartesian coordinates of the site:

```
Element(basis=,grid=,z=,r=)  X Y Z
```

Element is either a string (as above), or an integer specifying the atomic number of the center. The basis set and charge on the site may be defined here as in the Z-Matrix format.

X Y Z are floating point numbers giving the x, y, and z coordinates of the center in Å (unless modified by the `geom=inau` directive in `$route`.

The Cartesian coordinate format may not currently be used for geometry optimizations.

3.2 The Geom Keyword

The geom keyword in the `$route` section may be used to modify the default actions taken by M101 in processing the `$geom` section. Most of these options have been discussed above, but for completeness they are reiterated here.

Truchas expects to find a `$geom` section with a Z-Matrix whose distances are in Angstroms and angles are in degrees. This may be overridden with the directives:

```
geom=([coord,inau,inrad,chk,rdchk])
```

coord specifies Cartesian coordinate input.

inau assigns distances and coordinates in atomic units(bohrs).

inrad specifies angles in radians.

chk,rdchk signals that the geometry information is to be read from the checkpoint file. In this case, no `$geom` input section is required. For instance, frequency calculations need to be referred to the minimum

energy geometry. If the checkpoint file from the geometry optimization has been saved, it contains the parameters specifying the minimum energy geometry, and this flag is convenient.

3.3 Examples

3.3.1 Example 1

```
$geom
O
H1 O 0.96
H2 O 0.96 H1 104.5
$end
```

This specifies a geometry for the water molecule. The OH bond distance is 0.96Å, and the HOH bond angle 104.5°. The basis sets are defaulted to the values defined in $route.

3.3.2 Example 2

```
$geom
O(basis=6-31g)
H1(basis=sto-3g) 1 ROH
H2(basis=sto-3g) 1 ROH H1 Theta

ROH=0.96 constant
Theta=104.5
$end
```

This specifies the same geometry for the water molecule. The O atom will be described by the 6-31G basis set, while the two hydrogen atoms will use the STO-3G basis. The bond distance and bond angle are entered as symbolic parameters. The bond distances are constrained to be equal. If an optimization is being run, the OH bond lengths will be held constant, and only the angle will be optimized.

3.3.3 Example 3

```
$geom
O(basis=6-31g)
H1(basis=sto-3g,z=0.0) 1 ROH
H2(basis=sto-3g,z=0.0) 1 ROH H1 Theta

ROH=0.96 constant
Theta=104.5
$end
```

This specifies the same geometry for H_2O, and might be used in a counterpoise calculation of the basis-set-superposition-error for Example 2. The O atom is still described by the 6-31G basis set, and the two "hydrogen" sites with the STO-3G basis. The latter will have a nuclear charge of 0.0, however, and so this calculation will determine the energy of an oxygen atom using a rather unusual multi-center basis set.

3.3.4 Example 4

```
$geom
O           0.0   0.0   0.0
H(r=1.375)  0.0   0.0  -0.96
H2(r=1.375) 0.0   0.0  +0.96
$end
```

This specifies a geometry for H_2O in Cartesian coordinates format. Note that the molecule is linear, the two hydrogens lying 0.96Å from the central oxygen. If a solvent calculation is being run, the hydrogen radius used in assigning the cavity will be 1.375.

3.3.5 Example 5

```
$geom
O                          0.0   0.0   0.0
H1                         0.0   0.0  -0.96
H2                         0.0   0.0  +0.96
H3(basis=nobasis,z=20.0)   0.0   1.5   0.0
```

```
$end
```

This specifies the geometry for H_2O as above, but a point charge has been placed 1.5Å from the origin along the y-axis.

3.3.6 Example 6

```
$geom
N
C N RCN
X C 1.0 N Half
O C RCO X Half  N 180.0
H O ROH C COH   X 0.0

RCN=1.20
RCO=1.3
ROH=1.0
COH=105.0
$end
```

This example illustrates the use of a dummy atom, X. The parameter Half represents half of the NCO angle which is expected to be nearly linear. Note that a value of Half less than 90° corresponds to the cis arrangement.

3.3.7 Example 7

```
$geom
 ti (basis=wachters-hay)
 f1 ti r1
 f2 ti r1 f1 109.4718
 f3 ti r1 f1 109.4718 f2 120.
 f4 ti r1 f1 109.4718 f2 -120.

 r1=1.75
$end
```

This example specifies a T_d structure for the TiF_4 molecule. The bond angle of 109.4718 will be recognized by M101 as close enough to the perfect

tetrahedral angle that it will be replaced by the T_d angle with full machine precision.

3.3.8 Example 8

```
$geom
 sc (basis=wachters-hay)
 f1 sc r1
 f2 sc r1 f1  120.0
 f3 sc r1 f1  120.0 f2 180.0

 r1=1.84
$end
```

This example specifies a D_{3h} structure for the ScF_3 molecule.

Chapter 4

Basis Sets

Truchas expands the molecular wavefunction in a basis set of Cartesian Gaussian functions centered at the nuclei. A number of standard basis sets are available in Truchas, and new ones can be entered straightforwardly. It is also possible to utilize effective core potentials (ECP) in Truchas. These potentials can be used to eliminate the chemically inert core electrons from the problem, thereby economizing the number of electrons which must be treated explicitly. Perhaps more importantly, ECPs are available for heavy elements which incorporate the most important relativistic effects on the valence electrons. This chapter discusses the specification of basis sets and effective core potentials in detail.

4.1 Cartesian Gaussians

An unnormalized primitive Cartesian Gaussian on center A has the form:

$$\phi(n, l, m, \alpha_A, A) = {x_A}^n {y_A}^l {z_A}^m e^{-\alpha_A {r_A}^2} \tag{4.1}$$

where (x_A, y_A, z_A) are the components of the vector $\mathbf{r_A} = \mathbf{r} - \mathbf{A}$ with norm r_A. The normalization constant for this function is

$$N_{nlm}(\alpha_A) = N_n(\alpha_A) N_l(\alpha_A) N_m(\alpha_A) \tag{4.2}$$

where

$$N_k(\alpha) = (2\alpha/\pi)^{1/4} (4\alpha)^{k/2} [(2k-1)!!]^{-1/2} \tag{4.3}$$

The primitive Gaussians are usually combined in specific linear combinations called contractions. For example, one might contract three primitive Gaussians in order to fit a $1s$ Slater-type orbital (STO);

$$\chi(\zeta, \mathbf{r_A}) = (\zeta^{\mathbf{3}}/\pi)^{\mathbf{1/2}} \mathbf{e}^{-\zeta \mathbf{r_A}} = \sum_{\mathbf{i=1}}^{\mathbf{3}} \mathbf{c_i} \phi_{\mathbf{i}}(\mathbf{0}, \mathbf{0}, \mathbf{0}, \alpha_{\mathbf{i}}, \mathbf{A}) \tag{4.4}$$

a basis generally referred to as the STO-3G basis set.[4] Note that the expansion for an arbritrary STO exponent, ζ, can be obtained from that for an STO with $\zeta = 1$ by a uniform scaling of the distance:

$$\chi(\zeta, \mathbf{r_A}) = \zeta^{\mathbf{3/2}} \chi(\mathbf{1}, \zeta \mathbf{r_A}) = \zeta^{\mathbf{3/2}} \sum_{\mathbf{i=1}}^{\mathbf{3}} \mathbf{c_i} \phi_{\mathbf{i}}(\mathbf{0}, \mathbf{0}, \mathbf{0}, \zeta^{\mathbf{2}} \alpha_{\mathbf{i}}, \mathbf{A}) \tag{4.5}$$

Aside from an overall normalization, the contraction coefficients $\{c_i\}$ for the Gaussian expansion are seen to be independent of ζ, and one can therefore simply rescale the Gaussian exponents in order to generate additional STO's.

The complete specification of the basis set therefore requires a center, an angular momentum type, a set of orbital exponents and contraction coefficients, and (optionally) a scalefactor.

In the default mode, the parameters defining the basis sets are not printed. The directive

```
print=basis
```

in the `$route` section will print the basis sets used in the calculation.

4.1.1 An Aside

Two types of contracted basis sets are generally used.[11,2] In the segmented scheme, with minor exceptions, primitive functions occur in only one contracted function. The Dunning-Hay 3s2p "double-zeta" contractions[2] of the Huzinaga (9s5p) primitive bases[5] for the first-row atoms are an example.

```
$3s2p c
 type=s
 4233.0      0.001220    0.0         0.0
  634.9      0.009342    0.0         0.0
```

```
      146.1      0.045452    0.0         0.0
       42.50     0.154657    0.0         0.0
       14.19     0.358866    0.0         0.0
        5.148    0.438632   -0.168367    0.0
        1.967    0.145918    0.0         0.0
        0.4962   0.0         1.060091    0.0
        0.1533   0.0         0.0         1.0
     type=p
       18.16     0.018539    0.0
        3.986    0.115436    0.0
        1.143    0.386188    0.0
        0.3594   0.640114    0.0
        0.1146   0.0         1.0
$end
```

Note that the more diffuse primitives – those with smaller exponents – are generally left free as those are the primitives most important for describing modifications of the atomic density when the molecule forms. Efficient segmented contraction schemes become more difficult to generate as the number of core orbitals in the atom increases. This is because orbitals of higher principal quantum number must be orthogonal to those of lower principal quantum number; the $3s$ atomic orbital must have components which use primitives important in the $2s$ and $1s$ atomic orbitals.

General contraction schemes[11] differ in that each primitive may be used in several contractions. In this way, a contracted basis set can be generated from a set of primitives by using the atomic Hartree-Fock coefficients from a calculation in the primitive space. This makes it easy to define a contracted function which is the $1s$ function, another which is the $2s$ function, etc. Note that with this approach the contracted basis must give a total atomic energy which is identical to the primitive basis set result. Once again, the most diffuse functions are usually left free to vary in the SCF calculation. The (6s5p2d) contraction of the [14s9p5d] primitive basis of Wachters'[14] provides an example of a general contraction.

```
$wachters sc
 /14s9p6d/[6s5p2d]
 type=s
 188961.0      0.00025   -0.00007    0.00003    0.0    0.0    0.0
```

```
 28491.4      0.00193   -0.00056    0.00020    0.0    0.0    0.0
 6572.43      0.00980   -0.00292    0.00102    0.0    0.0    0.0
 1881.03      0.03940   -0.01172    0.00416    0.0    0.0    0.0
  617.979     0.12532   -0.04013    0.01415    0.0    0.0    0.0
  225.127     0.29577   -0.10358    0.03765    0.0    0.0    0.0
   88.5664    0.41900   -0.20845    0.07634    0.0    0.0    0.0
   36.5819    0.24139   -0.14211    0.05949    0.0    0.0    0.0
   11.0358    0.02141    0.49986   -0.26411    0.0    0.0    0.0
    4.49063  -0.00418    0.61679   -0.48635    0.0    0.0    0.0
    1.12935   0.00137    0.05654    0.64988    0.0    0.0    0.0
    0.454613  0.0        0.0        0.0        1.0    0.0    0.0
    0.077533  0.0        0.0        0.0        0.0    1.0    0.0
    0.030790  0.0        0.0        0.0        0.0    0.0    1.0
type=p
 1113.82      0.00223   -0.00075    0.0        0.0    0.0
  266.244     0.01764   -0.00596    0.0        0.0    0.0
   86.5763    0.08123   -0.02825    0.0        0.0    0.0
   32.5934    0.24040   -0.08635    0.0        0.0    0.0
   13.2190    0.42517   -0.16445    0.0        0.0    0.0
    5.58357   0.35972   -0.12493    0.0        0.0    0.0
    2.18594   0.0        0.0        1.0        0.0    0.0
    0.895011  0.0        0.0        0.0        1.0    0.0
    0.351975  0.0        0.0        0.0        0.0    1.0
type=d
   22.4283    0.02140    0.0
    6.03479   0.11304    0.0
    1.97905   0.31393    0.0
    0.666750  0.45818    0.0
    0.218231  0.09       1.0
$end
```

The integral routines in Truchas vectorize over primitives in a shell block. A shell block is defined as the set of all primitives on a given center with a common angular momentum. Thus all the s primitives on a center constitute a shell block. The integral loops run over shell blocks and are vectorized over the primitives in the shell blocks. When the group of primitive integrals corresponding to the current shell blocks are completed, they are transformed

to the contracted basis. General contraction schemes are therefore treated in the same manner as segmented contractions. In particular, unlike many quantum chemistry programs, they are no more inefficient to compute than segmented contractions. This is a real plus for transition metal systems, where general contractions provide a much more straightforward way to define the basis set.

4.2 Basis Set Specification

The basis to be used on a specific center is specified by its basis-name. The basis-name can be specified in the `$route` section through the "basis" directive,

```
$route
   dft,basis=dzp
$end
```

which will associate the dzp, or double-zeta polarization, basis-name with all centers. This specification can be overridden in the `$geom` section:

```
$geom
   O
   H1(basis=3sp) O 0.96
   H2(basis=3sp) O 0.96 H1 104.5
$end
```

which will assign a basis-name "3sp" with the hydrogen. It is possible to define basis-names in which embedded blanks are significant by enclosing the name in double-quotes, as in `basis=''huzinaga-dunning (9s/3s)''`.

The actual basis set parameters associated with the basis-names are read by M102. M102 first searches the input file for a line of the form:

```
$basis-name atomic-symbol
```

where the "atomic-symbol" matches the element symbol used in the `$geom` section. The "atomic symbol" does not include the atom number which may have been used to distinguish like atoms. Ghost centers may be specified by using "zq" as the atomic symbol. Thus in the example above, the code would search the input file for the string

```
$dzp O
```

to obtain the oxygen basis, and the string

```
$3sp H
```

to find the hydrogen basis set. If these strings are not found in the input deck, then the data file, `mesa.dat`, is searched.

Once this string is supplied, a free-format input statement is used to specify the orbital type, scalefactor, exponents and contraction coefficients.

```
$3sp H
 /arbritrary hydrogen basis
 type=s scalefactor=1.0
  1000.0  0.05  -0.10
   400.0  0.86  -0.33
   120.0  0.68   0.85
 type=s
     0.5  1.0
 type=p
     0.75 1.0
$end
```

The slash character,'/', is used for comment lines. The type field specifies the angular momentum associated with the block. The scalefactor field scales r, i. e. the Gaussians are given by $exp^{[}-\alpha(scalefactor*r)^2]$. Note that several contractions of the primitive exponents may be specified in a single block. In the example, two contracted s-functions are constructed from three primitive functions. There is also an s-function left completely uncontracted with exponent 0.5, and an uncontracted p-function. The contraction coefficients need not be normalized, M102 will renormalize them. Truchas uses a convenient but inconsistent definition of the normalization of primitives. Spell it out. Note that the basis might have been specified equally as well in the format:

```
$3sp H
 /arbritrary hydrogen basis
 type=s scalefactor=1.0
  1000.0  0.05  -0.10  0.0
```

```
   400.0  0.86  -0.33  0.0
   120.0  0.68   0.85  0.0
     0.5  0.00   0.00  1.0
 type=p
    0.75 1.0
$end
```

It is also possible to generate new basis sets by modifying standard basis sets with the include directive. For example, the basis

```
$whf Cr
  /Wachters-Hay basis with f polarization
  $include wachters-hay
  type=f
  0.5  1.0
 $end
```

modifies the standard Wachters-Hay basis found in `mesa.dat` by adding an f type polarization function.

The standard basis sets included in `mesa.dat` are listed in Table 4.1.

4.3 Effective Core Potentials

Effective core potentials are available. Discuss them, comments about ecp1 vs. ecp2.

Basis-name	Availability
"Huzinaga-Dunning 9s/4s"	H,B-F
"Huzinaga-Dunning 9s/5s"	H,B-F
"Huzinaga-Dunning 5p/2p"	B-F
"Huzinaga-Dunning 5p/3p"	B-F
"Huzinaga-Dunning 11s/6s"	Al-Cl
"Huzinaga-Dunning 7p/4p"	Al-Cl
dz	H,B-F,Al-Cl
tz	H,B-F,Al-Cl
"Dunning polarization"	H,B-F
"Standard polarization"	Al-Cl
dzp	H,B-F,Al-Cl
tzp	H,B-F,Al-Cl
"Dunning Rydberg 3s"	B-F
"Dunning Rydberg 3p"	B-F
"Dunning Rydberg 3d"	B-F,Al-Cl
"Dunning Rydberg 4s"	B-F,Al-Cl
"Dunning Rydberg 4p"	B-F,Al-Cl
"Dunning Rydberg 4d"	B-F,Al-Cl
"Dunning negative ion 2p"	B-F,Al-Cl
3s2p	C,N,O
3s2p1d	C,N,O
ccvdz	H,B-O
ccvtz	H,B-O
sto-3g	H-Ar
3-21g	H,C,N,O
3-21g	H,C,N,O
4-31g	H
6-31g	H-Ar
6-31g*	H,Li-Ar
6-31g**	H,Li-Ar

Table 4.1: All electron basis sets known to Truchas.

Effective Core Potential	Basis-name	Availability
ecp1	mbs-ecp1 dz-ecp1	Na-Ba,Tl-Bi
ecp2	mbs-ecp2 dz-ecp2	K-Cu,Rb-Ag,Cs-La,Hf-Au

Table 4.2: Effective core potentials and corresponding basis sets known to Truchas.

Chapter 5

Symmetry

Given the molecular geometry, M202 determines the point group and generates associated information such as the symmetry-adapted-linear-combinations of basis functions, etc. In general, only D_{2h} and its subgroups are supported, although a few representations of higher order such as O_h are supported. If the molecule has a point group symmetry higher than D_{2h}, M202 produces information for a subgroup contained in D_{2h}. Input to this section of the code is governed by the "sym" keyword.

Truchas makes limited use of this information. For instance, symmetry is not used to reduce the number of integrals which must be calculated, but it is possible to enforce symmetry constraints in the wavefunction determination for both HF and DFT calculations. Options affecting the SCF symmetry utilization are invoked through the "scf" keyword.

5.1 Symmetry Determination

The symmetry routines in M202 process the geometry to determine the point group of the molecule. This code originated with the early Gaussian packages and was written by Doug DeFrees. Once the point group is determined, a number of symmetry related operators are formed using routines written by Paul Saxe, Rich Martin, and Tom Russo. Options to the symmetry package in M202 include:

```
sym=([off,norotate,force_group=,force_axis=,patoms=,ghosts=,dump])
```

off Unconditionally turn symmetry off in the calculation. There are times when the guess link has problems projecting pure symmetry vectors. In this case, this option can be useful.

norotate By default, Truchas rotates the molecule to a standard orientation. This flag shuts it off.

force_group Force the calculation to be performed in a specified subgroup of the full Hamiltonian.

force_axis Force the principal axis of the representation to be along a specified direction. For example, by default, the principal axis in C_{2V} symmetry is placed along "z". The directive `sym=(force_axis=y) will instruct the code to assume the` C_2 `axis lies in the y-direction. This is usually used in conjunction with the ‘‘norotate’’ option (make sure that the ‘‘y’’ axis is indeed the` C_2 `axis in the input orientation).`

patoms `This flag instructs the program to use only the first n atoms, sym=(patoms=n), in determining the point group and SALC information. This can be useful when a calculation is being performed in a large point charge field, and only the first n atoms have basis sets. Be careful, though, that the atoms omitted have the full symmetry of the ones included or the results will be incorrect.`

ghosts `By default, the program executes the symmetry package even if there are ghosts and/or dummies in the geometry input. The directive ‘‘sym=(ghosts=(off)) disables use of symmetry if ghosts are present.`

dump `Dumps intermediate information useful for debugging.`

5.2 Symmetry Constrained SCF

Symmetry constraints are enforced by determining the symmetries of all the occupied molecular orbitals in the guess wavefunction and demanding that this number is preserved in each iteration of the SCF step. The symmetries

of the guess are obtained in M401 through a projection on the SALC's. In the SCF links, the Fock matrix is block diagonalized over each irreducible representation. Note that if the guess has an incorrect symmetry, this will be propagated throughout the SCF process.

The only symmetry input to the SCF links is the list of occupied orbital symmetries. This is usually unecessary, since they will have been determined in the initial guess, but they can be overridden here.

```
scf=([occsym=])
```

The argument "occsym" is an array of dimension (nirrep,nshell) where nirrep is the number of irreducible representation in the point group and nshell is the total number of shells in the SCF calculation. Recall that a closed shell SCF has two shells: closed and virtual; a high-spin open-shell calculation has three: closed, open, and virtual. Occsym gives the number of orbitals of a given symmetry in each shell. For example,

```
scf=(occsym=(4,0,1,1, 6,2,2,4))
```

would be appropriate for a closed shell calculation (nshell=2) in C_{2V} symmetry where there are four irreducible representations.

Chapter 6

Integrals

Truchas employs a number of analytic Gaussian integral programs based on the Rys polynomial method. In addition to the standard one- and two-electron integrals, effective-core-potential(ECP) and property integrals are supported. First and second derivatives of the integrals with respect to nuclear coordinates are also determined by these codes. In addition, numerical integration of the exchange-correlation functionals which appear in density-functional theory is supported. These are based on the molecular partitioning concepts of Becke[1] and the radial and angular quadratures of Murray, Handy and Laming,[9] Treutler and Alrichs,[13] and Lebedev.[6,7]

6.1 Analytic Integration

These integral codes require no input other than the specification of the basis set described elsewhere.

The one-electron integrals needed to determine a nonrelativistic energy are generated by M302 and stored on the read-write file(RWF). The two-electron integrals are generated by M312 and stored on a temporary raw-integral file(RINT). They are sorted into a "triangle of triangles" order [ij|kl], i≥j, k ≥ l, in M330, and stored on the integral file(INT). The raw-integral file is destroyed in M330 after the first half-transformation in order to conserve disk space.

There are several (too many) versions of the derivative integral programs. In some versions the primitive derivative integrals are traced with density ma-

trix elements(M702, M711, M712), while in others(M303,M323) the derivative integrals over contracted basis functions are constructed and written to disk. In general, the latter approach is taken in analytic second derivative calculations where coupled-perturbed HF equations must be solved, while the former is all that is needed for analytic derivatives of most wavefunctions.

6.1.1 Options

Integral Accuracy

There are two options available in these codes which affect the accuracy of the ultimate results. They are `preex` and `cutoff` which are read from the `$route` input. The parameters default to `preex=15` and `cutoff=15` in M312. The integers determine the criteria for computing integrals based on a preexponential factor, 10^{-preex} and the criterion for retaining an integral after it has been computed, $10^{-cutoff}$. They may be modified via the `$route` directive

```
int=(preex=xx,cutoff=yy)
```

These parameters are also used in the computation of derivative integrals in M323. The input format and defaults are

```
m323=(preex=13,cutoff=9).
```

Print options

The integrals can be printed in the various integral links through a `$route` directive. For one-electron and one-electron gradient integrals, the syntax is

```
print=( [int,gradient] = ([s,t,v,ecp]) )
```

The first option is available in M302, the one-electron integral link, while the second is effective in the derivative integral links M303, and M702. Second derivative integrals are also computed in M702, and

```
print=(gradient=([s,t]) )
```

e. g. , will print both first and second derivatives of the overlap and kinetic energy operators if both are being computed.

The two-electron integrals may be printed, if you desire, after the integral sort in M330, and in the derivative two-electron links, M323 and M712.

6.2 Numerical Integration

6.3 Reusing Integrals

It is possible to avoid the recomputation of the integrals through a combination of a `$route` directive and the shell script file replacement syntax.

To avoid recomputing the one- and two-electron integrals the directive

```
int=reuse
```

or, alternatively,

```
int=noints
```

can be specified in the `$route` section. The code generally destroys the integral file and all intermediate files such as the RWF at the end of a calculation. In order to avoid this, the shell must be told to keep the integrals around. This is accomplished by specifying the directive

```
mesa inp=x out=y chk=z saveint=filename
```

in the execution line. In subsequent runs, the directive

```
mesa inp=x out=y chk=z int=filename saveint=newfilename
```

in conjunction with the `$route` command will cause Truchas to read the one- and two-electron integrals from filename and save them at the end of the calculation in the file "newfilename".

The `$route` directive

```
int=reuse2
```

will avoid computation of the two-electron integrals, but recompute the one-electron integrals. This is sometimes useful. For example, if only the point charge field is changing from one calculation to the next, the two-electron integrals need not be recomputed.

A directive

```
int=reuse1
```

is also supported, and does what you think it would.

Chapter 7

Initial Guesses

The HF and Kohn-Sham codes must have an initial guess for the wavefunction in order to initiate the self-consistent-field iterations. A good starting set of molecular orbitals can significantly speed convergence, a poor set may lead to a diverging energy in the iterative solution. The guess is generated by M401, and it provides a number of ways to skin this particular cat. Some, in our experience, are better than others, and none is fool-proof. In this chapter we present the possibilities.

7.1 Guesses Available

7.1.1 Core Guesses

The default in Truchas is to generate an initial guess by diagonalizing the core Hamiltonian. This is usually a poor guess, but is at least possible for any calculation with any basis set.

The core Hamiltonian is simply the one-electron Hamiltonian, $h = T + V$, where T and V are the kinetic energy and nuclear attraction one-electron operators. The major problem with this approach is that it completely ignores the electron-electron repulsion in the Hamiltonian. For simple molecules, where the basic character of the occupied orbitals is sometimes known to the user, a core guess used in conjunction with the `alter` directive(discussed below) can be acceptable. Generally, the Huckel guess discussed next is preferable.

7.1.2 Huckel Guesses

A Huckel guess is requested with the

```
guess=(huckel)
```

directive in the `$route` section. The Huckel calculation is also based on a one-electron Hamiltonian, but the matrix elements of the Huckel operator are empirical, and have been chosen so as to incorporate the effects of electron-electron repulsion in as sensible a way as possible. Huckel parameters are available for all the elements, although some are more reliable than others. For example, parameters for the *f* elements, the Lanthanides and Actinides, have not been as thoroughly studied as those for the earlier portions of the Periodic Table.

The Huckel calculation is an example of a guess which must be projected onto the basis set being used in the calculation. It is a minimum basis set technique; the Huckel basis consists of a single orbital for each occupied atomic orbital of the atom. In addition, the specific form of the orbital, its' exponent, is not well defined in the sense that there is no set of basis orbitals which will give the Hamiltonian matrix that is generated by the Huckel approach. The resulting wavefunction must therefore be projected onto the ab initio basis. This is done in Truchas through the technique of corresponding orbitals.

7.1.3 Checkpoint Guesses

A guess can be read from a previously converged calculation with the directive:

```
guess=(chk)
```

In such a case, the vectors are simply read from the chk file, Schmidt orthogonalized, and passed onto the SCF link.

A more general directive:

```
guess=(rdchk)
```

can be used to project a set of vectors stored on the chk file onto a different basis. This can be used effectively in cases where convergence is difficult, by

first performing the calculation in a smaller space such as a minimum basis set. The converged wavefunction can then be projected against a larger basis, and is usually a reasonable starting set of vectors.

7.1.4 RWF Guesses

In this case, the orbitals are read from the read-write-file(RWF). This option is generally only used internally in the codes in the case of geometry optimizations. The RWF always has a copy of the current orbitals produced by the last pass through the SCF link, and so during an optimization M401 picks up these orbitals, Schmidt orthogonalizes with the current overlap matrix, and passed them onto the SCF link.

7.1.5 Input File Guesses

When all else has failed and you believe you can generate a better guess "by hand", it is possible to read the starting vectors from the input file. This is accomplished by the directive:

```
guess=(rdinp)
```

If this directive is specified in $route, M401 looks for an input section `$vectors`. The input vectors follow, two cards per vector. The first card of the set is there just for the convenience of the user and may be used for a title, orbital number, etc. . It is ignored by the reading routine, while the second card gives the MO coefficients for orbital i, $(c(i,j), j = 1, nbf)$, where nbf is the dimension of the basis set.

The default for a `rdinp` guess is to read only the occupied orbitals. It is possible to direct M401 to read an enlarged set (occupied and virtuals, for example) via:

```
guess=(rdinp,nocc=xx)
```

where nocc is the number to be read.

7.2 Altering the Guess

The MO vectors in Truchas are ordered by eigenvalue. The SCF routines expect the doubly occupied orbitals to come first, followed by the open-shell

orbitals, followed by the virtuals. It is sometimes necessary to reorder the guess vectors:

```
guess=(alter)
```

which must be accompanied by an input section

```
$alter
   orbital1, orbital2
   ...
   ...
```

For example,

```
$alter
   2,6
   5,7
```

will switch orbitals 2 and 6, and 5 and 7.

Chapter 8

SCF Calculations

Self-consistent-field wavefunctions can be obtained within either the Hartree-Fock or Kohn-Sham approximation. The HF code, link M503, is included in the route by specifying

In the HF mode, the Fock operator is defined by a set of occupation numbers and coupling coefficients as described below. Default values are store internally for closed-shell and high-spin open-shell wavefunctions, and so for these cases the only input required by the SCF process is the spin-multiplicity. At present M503 must use the two-electron integrals generated by M312 and sorted by M330, and therefore is limited to systems containing $\sim$ 150 basis functions, depending on the disk capacity and transfer rate available.

The DFT code may be used to determine closed-shell and high-spin open-shell wavefunctions using either local density approximations for the exchange and correlation functionals, or from gradient-corrected functionals. A "direct' algorithm is available with DFT where the Coulomb integrals are recomputed each iteration as needed, and so there is no inherent limitation to the size of problem which may be treated other than the speed of the CPU and the patience of the user.

Since both codes are doing similar things, several options used to alter the convergence behavior, the method of solution, etc. are shared by both. These common options are described in the next section, after which follows a discussion of those unique to HF or DFT.

8.1 The SCF Keyword

The scf keyword in the `$route` section is used to modify default actions taken by the SCF codes. Options common to both HF and DFT include:

```
scf=(pulay,page,sym=,occsym=)
```

pulay Use Pulay's one-hamiltonian method to solve the SCF equations. (default=.true.)

page Use Page-Mciver pseudo-second-order approach. There is a bug in this routine somewhere. Avoid it. (default=.false.)

sym=off Do not enforce symmetry in the solution of the Kohn-Sham equations. Symmetry in enforced by projecting the atomic orbital Fock matrix onto the basis set SALC's. This prediagonalizes the matrix into blocks corresponding to each irreducible representation of the point group. Each block is then diagonalized and the occupied orbitals chosen by an aufbau procedure using an array which specifes the number of occupied orbitals in each block. (default=on)

occsym= This is an array which specifes the number of occupied orbitals in each irreducible representation. It is generated in the guess routine, M401, but may be overridden here if desired. See Chapter 5 for more information.

Convergence control options include:

```
scf=([cycles=,convergence=,diis,startdiis=,maxdiis=,maxolap])
scf=([level-shift=,noabort])
```

cycles= The maximum number of cycles allowed in the iterative solution before the program aborts. (default=30)

convergence=n The SCF convergence criterion. Either the DIIS error becomes less than 10^{-n}, or the energy changes by less than 10^{-n-1}. (default=8).

diis Use DIIS procedure to hasten convergence. (default=.true.)

startdiis= The maximum DIIS error matrix element allowed before the DIIS procedure is initiated. Initializes DIIS extrapolation when this is $\leq 10^{-n}$. (default=-4). Note the default effectively turns DIIS on immediately.

maxdiis= The maximum number of prior DIIS error matrices to use in the extrapolation process. (default=5).

maxolap Force the new vectors at each iteration to be determined according to the maximum overlap with the previous iterations' vectors as opposed to the lowest eigenvalues. (default=.false.)

level-shift= Apply a level-shift to the diagonal elements of the Kohn-Sham matrix corresponding to virtual orbitals. (default=0.0d0). A level-shift of the order of 0.3 can sometimes be very effective in the early courses of the solution.

noabort Write the final SCF vectors to the chk file even though the calculation may not have converged. Sometimes useful for generating initial guesses for subsequent runs. (default=.false.)

8.2 HF

Closed-shell and high-spin open-shell SCF calculations require little input. If a general SCF calculation is desired, several additional parameters which specify the operator must be read. The form the general Fock operator takes is:

$$F_i = f_i h + \sum_{j}^{nshell} \alpha_{ij} J_j + \beta_{ij} K_j \tag{8.1}$$

where h is the sum of the one-electron kinetic and potential energy operators and J_j and K_j are the Coulomb and echange operators, respectively. The general electronic SCF energy is then:

$$E = \sum_{i}^{nshell} \sum_{j}^{orbitals(i)} < \phi_i \mid f_i h + F_i \mid \phi_j > \tag{8.2}$$

where ϕ_j is an orbital in shell i. The vector coupling coefficients, f_i, α_{ij} and β_{ij}, can be input in two ways. They may defined directly or by specifying

a numerator and denominator for each coefficient. We must also specify the number of shells in the calculation and the number of orbitals in each shell. This input is controlled through options to the `scf` keyword:

```
scf=([gscf,nshell=,nfock=,ncoul=,nexch=,ndmat=)
```

gscf Requests a general scf calculation.

nshell= The number of occupied shells in the Hamiltonian.

nfock= The number of Fock operators. default=nshell.

ncoul= The number of Couomb operators. default=nshell.

nexch= The number of exchange operators. default=nshell.

ndmat The number of density matrices. default=nshell.

```
scf=([shlnbf=,f=,fn=,fd=,alpha=,an=,ad=,beta=,bn=,bd=)
```

shlnbf The number of orbitals in each of the nshells.

f The nshell occupation numbers.

fn The nshell occupation number numerators.

fd The nshell occupation number denominators.

alpha The α_{ij} coupling coefficients. The (nshell,nshell) matrix is read as a lower triangle.

an The α_{ij} coupling coefficient numerators. (nshell,nhshell) lower triangle.

ad The α_{ij} coupling coefficient denominators. (nshell,nshell) lower triangle.

beta The β_{ij} coupling coefficients. (nshell,nshell) lower triangle.

bn The β_{ij} coupling coefficient numerators. (nshell,nshell) lower triangle.

bd The β_{ij} coupling coefficient denominators. (nshell,nshell) lower triangle.

As an example, consider an open shell singlet with 7 doubly occupied orbitals:

```
scf=(gscf,nshells=3,shlnbf=(7,1,1),f=(1,0.5,0.5),
     alpha=(2.0,1.0,0.0,1.0,0.5,0.0),
     beta=(-1.0,-0.5,0.0,-0.5,0.5,0.0) )
```

Alternatively, one could specify:

```
scf=(gscf,nshells=3,shlnbf=(7,1,1),
     fn=(1,1,1),fd=(1,2,2),
     an=(2,1,0,1,1,0),ad=(1,1,1,1,2,1)
     bn=(-1,-1,0,-1,1,0),bd=(1,2,1,2,2,0) )
```

Some special convergence techniques are available for general SCF wavefunctions.

```
scf=([extrapolate=,diagonals=,level-shift1=,damp=])
```

If you ever find yourself in need of special convergence tricks for a general scf wavefunction – good luck – and look at M503 for more details on what these do.

8.2.1 Print options

Various print options are available. Look to M503 for more information.

```
if (logkey(ops,'scf=print=diis-error',.false.,' ')) then
if (prnt.and.logkey(ops,'print=scf=energy',.false.,' ')) then
if (logkey(ops,'scf=print=ao-diis-error',.false.,' ')) then
if (logkey(ops,'scf=print=mo-diis-error',.false.,' ')) then
if (logkey(ops,'scf=print=diis-matrix',.false.,' ')) then
if (logkey(ops,'print=scf=lagrangian=mo',.false.,' ')) then
if (logkey(ops,'print=scf=lagrangian=ao',.false.,' ')) then
if (logkey(ops,'scf=print=pseudolagrangian',.false.,' ')) then
printj=logkey(ops,'scf=print=j',.false.,' ')
printk=logkey(ops,'scf=print=k',.false.,' ')
if (logkey(ops,'scf=print=kinetic',.false.,' ')) then
if (logkey(ops,'scf=print=potential',.false.,' ')) then
if (logkey(ops,'scf=print=one-electron',.false.,' ')) then
if (logkey(ops,'scf=print=guess',.false.,' ')) then
if (logkey(ops,'scf=print=density',.false.,' ')) then
```

```
if (logkey(ops,'scf=print=q-matrix',.false.,' ')) then
if (logkey(ops,'scf=print=mo_fock_matrix',.false.,' ')) then
if (logkey(ops,'scf=print=diagonal_fock',.false.,' ')) then
if (logkey(ops,'scf=print=rotation-matrix',.false.,' ')) then
if (logkey(ops,'scf=print=viters',.false.,' ')) then
if (logkey(ops,'scf=print=coefficients',.false.,' ')) then
if (prnt.and..not.logkey(ops,'print=scf=energy',.false.,' ')) then
if (logkey(ops,'print=m503=vector',.false.,' ')) then
if(logkey(ops,'scf=punch',.false.,' '))then
if (logkey(ops,'print=scf=core-lagrangian',.false.,' ')) then
```

8.2.2 GVB

It is also possible to perform simple GVB/PP calculations. These are limited to a single pair, and invoked with the `scf=tcscf` directive.

8.3 DFT

The major options to the DFT code specify the exchange and/or correlation functionals to be employed. There are two exchange and three correlation options available. Either the Slater "local" exchange functional or the Becke "gradient-corrected" functional may be specified. Correlation options include the VWN, or Vosko-Wilks-Nusair fit of the electron gas correlation functional, the LYP, or Lee-Yang-Parr gradient-corrected functional, and the NULL option, or no functional at all. The combination of Slater exchange with VWN correlation, (S-VWN), is what is usually referred to as the "local-density-approximation", or LDA. The gradient-corrected approximation, Becke-LYP, produces significantly better energies, but is roughly 3 times more expensive. By default, the program chooses the LDA, but can be overridden by:

```
scf=( [exchf=[slater,becke], corrf=[vwn,lyp,null]] )
```

exchf=slater Use the Slater exchange functional.

exchf=becke Use the gradient-corrected exchange functional due to Becke.

corrf=vwn Use the correlation functional from the Vosko-Wilks-Nusair fit of the electron gas correlation energy.

corrf=lyp Use the Lee-Yang-Parr gradient-corrected correlation functional.

corrf=null Omit a correlation functional.

8.3.1 Integration grids

The exchange and correlation functionals are not simple mathematical expressions and the integrals which must be performed to generate the Kohn-Sham operator and evaluate the DFT energy must be done numerically. This introduces an additional source of error as a balance between economy and accuracy is struck. An estimate of the error in the energy due to the grid can be obtained from the total integrated charge printed at the end of the DFT calculation. A rule of thumb is that the energy is good to about the number of digits past the decimal as the charge. The default atomic grid is defined by the `grid=gridname` directive in `$route`. There are standard grids available for the atoms H-Kr.

The first set, from Gill et al. is denoted "standard grid1", and invoked with the command `grid=sg1` in the `$route` section. They are available for H-Ar, and generally give energies accurate to $\sim 10\mu - Hartrees$.

The second set originate from Treutler and Ahlrichs, and are labeled "tag1","tag2",and "tag3". They are available for H-Kr. The tag1 grids are very economical, but yield energies which are uncertain at about the level of 1 milli-Hartree. They may be of use for initial exploratory calculations. The tag3 grid is roughly the size of the sg1 grid of Gill, et al. and of comparable accuracy ($\sim 10\mu - Hartree$). The "tag3" grid should be used for geometry optimizations, as the increased accuracy is necessary to ensure a reliable gradient and geometry.

A third, more general, type of grid is available. This is called for with the directive `grid=general` in the `$route` section. It requires the specification of the number of radial and angular quadrature points associated with each atom.

The grid to be used on each atom may be specified in the `$geom` section of the input. This allows one to mix and match atomic grids. More information can be found in Chapter 3.

```
scf=([mxgrid=,radgrid=,lebord=,dmcut=,dencut=,])
```

mxgrid= The maximum atomic grid size allowed. (default=8000). This is used for core allocation purposes and should not have to be modified unless a general grid exceeds this limit, in which case the job will abort and you will be notified to increase mxgrid.

radgrid The number of points to be used in the radial quadrature for 'general' grids. Note that the Euler-McLaurin scheme generates a point at the origin with zero weight, so that radgrid=51 will result in a 50 point radial grid. (default=51).

lebord The order of the Lebedev angular quadrature for 'general' grids. lebord=23 will integrate spherical harmonics through l=23. (default=23). Lebedev grids are available for l=(3,5,7,9,11,13,15,17,19,23,29).

denmat-cutoff= Threshold value for declaring elements of the density matrix to be identically zero. default=10^{-16}.

density-cutoff Threshold value for declaring the numerical value of the density on the grid to be identically zero. default=10^{-50}.

8.3.2 Direct DFT

The DFT routine contains an option for "direct" Kohn-Sham calculations. Options which control aspects of the direct computation of the Coulomb matrix include:

```
scf=[directj,int=(preexponential=),rhotest]
```

directj Use the direct algorithm to compute the Coulomb integrals and Coulomb matrix. (default=.false.)

int=preexponential= The threshold prefactor value below which the computation of a two-electron integral is skipped. default=10^{-12}.

rhotest Whether to do a preliminary scan of which integrals to compute based on the prefactor and an estimate of the appropriate density matrix element. (default=.true.). The threshold is given by the preexponential factor multiplied by the the maximum element of the density matrix in the shell block being computed.

8.3.3 Print options

There are several print options which pertain to the DFT code. More information may be obtained by looking at M511. A list of the available ones are:

```
if (logkey(ops,'scf=print=ao-diis-error',.false.,' ')) then
if (logkey(ops,'scf=print=diis-matrix',.false.,' ')) then
if (logkey(ops,'scf=print=diis-error',.false.,' ')) then
printj=logkey(ops,'scf=print=j',.false.,' ')
printk=logkey(ops,'scf=print=k',.false.,' ')
prteexch=logkey(ops,'scf=prteexch',.false.,' ')
if(logkey(ops,'print=properties=v0',.false.,' ')) then
if(logkey(ops,'scf=print=kinetic',.false.,' ')) then
if(logkey(ops,'scf=print=potential',.false.,' ')) then
if(logkey(ops,'scf=print=one-electron',.false.,' ')) then
if(logkey(ops,'scf=print=guess',.false.,' ')) then
if(logkey(ops,'scf=pseudo',.false.,' ')
if (logkey(ops,'scf=print=q-matrix',.false.,' ')) then
if (logkey(ops,'scf=print=salc-fock-mtx',.false.,' '))then
if (logkey(ops,'scf=maxolap=debug',.false.,' ')) then
if (logkey(ops,'scf=maxolap=debug',.false.,' ')) then
if (logkey(ops,'scf=print=viters',.false.,' ')) then
if (ops,'print=scf=energy',.false.,' ')) then
if (logkey(ops,'print=m511=vector',.false.,' ')) then
if (chrkey(ops,'print=m511=vector',' ',' ').eq.'all') then
if(logkey(ops,'print=scf=charges',.false.,' ')) then
if(logkey(ops,'scf=punch',.false.,' '))then
if (logkey(ops,'print=scf=lagrangian=mo',.false.,' ')) then
if (logkey(ops,'print=scf=lagrangian=ao',.false.,' ')) then
if (logkey(ops,'scf=print=pseudolagrangian',.false.,' ')) then
```

8.4 Convergence Aids

8.5 Examples

Chapter 9

IVO Calculations

The virtual orbitals which are determined by an HF calculation are dependent on the operators used to perform the SCF. In the case of a closed shell molecule, the virtual orbitals are determined by the closed shell Fock operator. Since the closed shell operator contains Coulomb and exchange terms for all N electrons of the molecule, the virtual orbitals are determined in a field which sees N electrons in the core. A better set, in the sense that they resemble the excited orbitals of the molecule, can be obtained through the IVO (improved-virtual-orbital) approximation. This technique works by defining a new Fock operator for the virtuals in which one of the core-electrons (the one being excited) is missing.

9.1 The IVO Operators

Two Fock operators are constructed by M502 during an IVO calculation. A closed shell Fock operator defined as

$$F_c = h + 2J_c - K_c \tag{9.1}$$

where J_c and K_c are the coulomb and exchange matrices constructed from the doubly occupied electron density matrix defined by the orbitals in shell "1". h is the matrix of one-electron integrals. The IVO Fock operator is diagonalized to determine the virtual orbitals and is defined as:

$$F_{IVO} = \sum_{i}^{nshell-1} f_i h + \alpha_i J_i + \beta_i K_i \tag{9.2}$$

The energy of excited state m in the IVO approximation is

$$E_m = E_{closed} + E_{open} + \epsilon_m + E_{nuc} \tag{9.3}$$

where E_{closed} is the closed shell energy.

$$E_{closed} = \sum_{ij}^{nbf} d^c_{ij}(h_{ij} + F^c_{ij}) \tag{9.4}$$

$$E_{open} = \sum_{ij} d^o_{ij} F^c_{ij} \tag{9.5}$$

The density matrix constructed from the closed shell orbitals, shell "1", is d_c and that constructed from the open shell orbitals, shell "2", d^o. The quantity ϵ_m is eigenvalue m of the IVO operator, and E_{nuc} is the nuclear repulsion energy.

9.2 Options

The coupling coefficients, the α_i and β_i which define the IVO operator for a single electron outside a doublet core are stored in M502. In that case, if no input is provided to M502, it will generate the operator for a hole in the highest-occupied-molecular-orbital(HOMO), coupled triplet to an electron in the virtuals. The options below allow one to use singlet coupling of the open-shells, to redefine the orbital with the hole, and to provide a comletely general operator for the virtuals which you can tailor to your own needs.

```
ivo=([symmetry,chk,homo=,singlet,neutral,
      nshell=,fn=,an=,bn=,fd=,ad=,bd=,f=,alpha=,beta=,noshell=)
```

symmetry Enforce symmetry in the calculation of the IVO's.

chk Pick up the orbitals from the chk file. By default, Truchas looks for them on the RWF.

homo Redefines the open-shell core orbital. Defaults to the highest-occupied-molecular-orbital (HOMO) from the SCF calculation.

singlet Singlet couple the open-shell orbitals.

neutral Couple open-shell orbitals to an average of the triplet and singlet states.

nshell A general set of coupling coefficients can be read to define any operator for the virtual block you desire. The format is identical to the general SCF input described in Chapter 8.

fn

an

bn

fd

ad

bd

f

alpha

beta

noshell

9.3 Examples

Chapter 10

Properties

Various atomic charge measures, population analyses, and properties are are available for both HF and DFT wavefunctions. The property integrals are computed in M1902 and traced with the appropriate one-electron denstiy matrix in M1951. M1951 also performs the population analyses.

10.1 Population Analyses

A simple Mulliken charge and bond order population analysis is performed at the conclusion of each Truchas calculation by default. Normally, only the charge and bond-order information contracted down to atomic centers is printed. If you wish to turn off this summary, use the directive:

```
property=(populations=[nocharge,nobond])
```

It is also possible to get even more information. The full population matrix over all basis functions, or the bond order contracted to atomic orbitals, can be obtained with the directive:

```
property=(populations=[full,orbital])
```

10.2 Voronoi Charges

Mulliken charges do not always give a meaningful description of the electronic charge distribution in the molecule. In those instances where diffuse

functions are present in the basis set, it is possible for a diffuse function on one center to be used extensively by a neighboring center to expand the wavefunction. In this case, the Mulliken approach will assign electrons to the center on which the basis function resides, even though it is really being used by electrons on neighboring sites. The *reductio ad adsurbum* is to consider performing a calculation on a diatomic molecule using a basis set centered on only one of the atoms. If the basis is large enough (complete), the exact HF wavefunctions could be obtained. The Mulliken analysis for this wavefunction would suggest that all the electrons are on the site with the basis and that the other atom has no electrons at all.

An alternative approach is to assign regions of space to the atoms in the molecule, and numerically integrate the electron density in each atomic region to obtain the charge. This is exactly what is done in the DFT code in order to integrate the exchange-correlation functional over all space. Individual atomic regions, whose union covers all of three-dimensional space, can be defined via a technique due to Voronoi. We refer to the charge obtained this way as the "Voronoi charge". Voronoi chaarges can be obtained for either HF or DFT wavefunctions by submitting a nonstandard route, executing M611 in the sequence after the wavefunction is obtained. For example, for the DFT calculation of Example 1,

```
$nonstd
  1//1,2;
  2//2;
  3//2,12,30;
  4//1;
  5//11;
  6//11;
  19//51;
  20//01;
$end
```

There are options available in M611 associated with the details of the numerical quadrature which should normally not have to be worried about. The reader is referred to an appendix for more details.

10.3 Electrostatic Potential Derived Charges

Another way to get an estimate of the electronic charge distribution in the molecule that is generally more reliable than the Mulliken definition is usually referred to as the "electrostatic potential derived charge", or ESP. In this approach a surface surrounding the molecule is constructed by considering the union of spheres centered at each nucleus. The radii associated with the spheres is usually of the order of the van der Waals radius. Given the enclosing surface, the electrostatic potential at a number of points on the surface can be computed from the *ab initio* wavefunction, and the results used to determine a set of atomic charges which reproduce the potential at these points in the best "least-squares" sense.

The ESP charges can be obtained from the sequence of links: M618,M619, and M620. M618 defines the enclosing surface, M619 computes the electrostatic potential at the surface, and M620 fits these values to atom-centered multipoles.

The ESP charges are signified by the `$route` directive:

```
cox-williams=(charges,dipoles,quadrupoles,print-potential])
```

Generally, one need only specify Cox-Williams and the default options will suffice. The first three refer to the level that the atom-centered multipole expansion takes. By default, atomic charges, dipoles, and quadrupoles are used in the fit. If just a charge fit is desired, appending "no" in front of the option will turn it off;

```
cox-williams=(nodipoles,noquadrupoles)
```

10.4 Molecular Properties

By default, Truchas does not compute molecular properties. In order to evaluate properties, a directive

```
property=([e0,e1,e2,e3,e4,v0,v1,v2])
```

is required. The strings e0,e1,e2,e3,e4 refer to electric moment properties; the monopole, dipole, quadrupole, octupole, and hexadecapole moments,

respectively. The strings v0, v1, v2 refer to electrostatic potential derivatives; the potential, electric field, and electric field gradient, respectively.

There are three other property integrals supported. These are the momentum, the mass-velocity relativistic operator, and the Fermi contact interaction. The Fermi integrals also make it possible to compute the one-electron relativistic Darwin term in M1951.

```
property=([del,mv,fermi])
```

Chapter 11

Solvent Calculations

At the present time, the incorporation of solvent effects in molecular electronic structure calculations generally is accomplished with polarizable continuum models. There are two main classes of approximations: analytic cavity models, such as the Onsager reaction field, and numerical cavity models. The former have the desirable advantage that integrals describing the solute-solvent interaction can be represented analytically; a drawback is that one is limited to spherical or ellipsoidal shaped cavities—a severe problem when determining the properties of irregularly shaped molecules in solution. The second class of models utilize arbritrarily shaped cavities, e.g. a union of van der Waals spheres each centered on an atom of the solute. In this way a cavity can be chosen that is appropriate for the molecule in question, but the solute-solvent interaction term is very complicated and must be computed numerically. In most of these approaches the perturbation of the solute Hamiltonian is given by a reaction field represented by a set of virtual charges residing on the cavity surface.

The approach used in Truchas is the numerical one. The determination of the reaction-field (RF) is begun with an electronic structure calculation in the gas phase. A cavity is constructed around the solute molecule; it is composed of a union of spheres centered at each atom of the solute. Each sphere has a radius roughly equivalent to the van der Waals radius of the atom upon which it is centered. It is assumed that the solvent outside this cavity is a polarizable dielectric continuum characterized by a dielectric constant ϵ. The region inside the cavity is assigned an internal dielectric constant of 1.0. The charge density determined from the electronic structure calculation is used

to compute the response of the dielectric by solving the appropriate Poisson equation. The reaction field acting back on the molecule is represented as a set of virtual charges on the cavity surface. The solvation energy is then given by an integral of the molecular density interacting with the surface charges:

$$E_{solvation} = -1/2 \sum_{s} \int \rho q_s / r_{is} \tag{11.1}$$

If the virtual charges which represent the polarization of the solvent are used to augment the one-electron Hamiltonian of the molecule:

$$H = H_0 - 1/2 \sum_{i,s} q_s / r_{is} \tag{11.2}$$

and the procedure repeated to self-consistency, the approach is then referred to as a self-consistent-reaction-field (SCRF).

11.1 Cavity Definition

The solvation energy is most sensitive to the shape and size of the cavity, and hence the sphere radii centered at the atoms. We recommend the sphere radii used in AMBER and reproduced in Table 11.1. These were determined from molecular dynamics simulations; a distinguishing characteristic of these radii is their dependence on chemical environment. The results are especially sensitive to the value of the hydrogen radius. Note that the hydrogen radius in a hydrocarbon differs from that in an alcohol.

Note that each atom may be given a unique radius through the input in the `$geom` section, Chapter 3. There are also some default values available, but they are still under development and should be treated with caution.

The cavity is generated in link M618.

11.2 The Representation of the Potential

The electric field at the cavity surface due to the molecule's charge density is needed in the solution of the Poisson equation governing the repsonse of the dielectric. An accurate solution for a complicated surface requires that it be computed at a large number of points (npoints). The electric field is a rather

time-consuming integral to compute from the *ab initio* density, and so it is approximated by the field produced by a set of multipoles (charges,dipoles, quadrupoles) centered on the atoms of the molecule. This atom centered multipole expansion can be approximated by the Mulliken charges, a very poor approximation, or by a set of multipoles chosen to give the best fit to the "exact" electrostatic potential computed at a smaller set (nfocus) of representative points on the surface. This approximation works well.

Three links are involved in computing the multipole expansion. The first, M618, computes the cavity surface from the atomic radii. The second, M619, computes the electrostatic potential from the density at the small (nfocus) set of representative points. The "best fit" multipole expansion is determined by link M620. This code is a modified version of one written by D.E. Williams obtained from QCPE.

11.3 Reaction Fields

The representation of the reaction field in terms of the virtual surface charges is performed by M621. As discussed above, this code uses the atom-centered multipoles to generate the electric field at the cavity surface at a large number of sampling points (npoints). The Poisson equation is solved using boundary sampling integration techniques and the solution represented on the surface by a smaller number of virtual surface charges (nfocus). It is important that this number be kept as small as possible, because we must compute "nuclear attraction" integrals between all the basis functions and these surface points in order to generate the interaction potential.

11.4 Options

The options recognized by links M618,M619,M620, and M621 are:

```
solvent=([epsilon=,convergence=,npoints=,nfocus=,mulliken,
          print=,nsources=,ncenters=,nobs=])
```

epsilon The dielectric constant to be used outside the cavity.

convergence The convergence criterion on the change in energy between one cycle of the SCRF and the next. Defaults to 10^{-6}.

npoints The total number of points used to represent the molecular surface. Defaults to 40000.

nfocus The number of focus points on the surface. Defaults to 512.

mulliken Use Mulliken charges to generate the potential at the cavity surface. The default is to use the Cox-Williams multipoles.

print See the appendix.

nsources Obsolete. Defaults to the number of atoms.

ncenters Obsolete. Defaults to the number of atoms.

nobs Obsolete. Defaults to the number of atoms.

In addition, M620 recognizes a number of options relating to the determination of the multipoles. These are transmitted with the keyword "cox-williams". They are explained in Chapter 10.

11.5 Examples

Element	Radius($\mathring{A}$)	Environment
H	1.0	OH or sp^2 NH
	1.375	CH or sp^3 NH
C	1.850	carbonyls
	1.800	all others
N	1.850	sp^3 N
	1.750	sp^2 N, N $=$ C
O	1.600	OC
	1.650	OH,ON

Table 11.1: Recommended atomic radii for solvent calculations.

Chapter 12

Geometry Optimization

Truchas is able to determine optimum geometries and transition states using analytic derivatives of the energy for HF wavefunctions. At present, analytic derivatives for DFT wavefunctions are limited to closed shell molecules. Open shell analytic DFT gradients will be available soon. Numerical gradients are available in this instance although they can be expensive.

A geometry optimization in Truchas requires Z-matrix (internal coordinate) geometry input. There are three general algorithms available for finding the stationary point. The most general algorithm, the Fletcher-Powell, does not require analytical gradients and is available for all wavefunctions. The Murtaugh-Sargeant and Berny algorithms utilize the information from analytical gradients and differ in the way they estimate the Heesian matrix from a series of gradient determinations.

Transition state searches can be performed with the Murtaugh-Sargeant or Berny algorithms. The initial Hessian in obtained either by running an analytic or finite-difference force constant calculation or by specifying a negative diagonal element of the Hessian.

An optimization is invoked by the keyword "Opt". Generally, the only directive you must provide is the method to be used. A transition state calculation using the Berny algorithm is requested with:

```
opt=(ts,berny)
```

12.1 Fletcher-Powell

The Fletcher-Powell algorithm proceeds by performing finite-difference calculations of the gradient along each of the internal coordinate directions. This requires two determinations of the energy for each coordinate. At the end of this first "cycle", the gradient information is used to determine a new geometry, and the process repeated again until the energy is minimized. This can be quite expensive in the general case, and many times this procedure will be used only for a few coordinates with the others held constant.

The Fletcher-Powell algorithm is invoked by:

```
opt=(fletcher-powell,[cycles=,convf=,tstcrv,restart,chkpnt,debug,dump])
```

where the optional arguments are:

cycles The maximum number of cycles allowed for convergence. Defaults to 5.

convf The convergence criterion on the gradient. Defaults to $3.0x10^{-4}$.

tstcrv Whether to test the second derivative matrix (Hessian) for the correct number of negative eigenvalues. Since Fletcher-Powell can only find minima, where all eigenvalues of the Hessian should be positive, this will shut the calculation down if there is a negative eigenvalue indicating the calculation is heading for a transition state. Defaults to .true.

restart Requests that the calculations be restarted using the information provided on a chk file.

debug Turns on informative print statement useful for debugging.

dump Turns on printing sometimes useful for correcting a calculation gone astray.

12.2 Murtaugh-Sargeant

The Murtaugh-Sargeant algorithm utilizes analytic gradients and can find either the minimum or the transition state. It is the default method, and the default Hessian is the unit matrix. It recognizes the options:

```
opt=([convf=,tstcrv,restart,chkpnt,debug,dump,
      cycles=,negeig=,dxmaxt=,fmaxt=,eigmax=,eigmin=,
      expert,clnd2e,srcd2e])
```

where those available on the first line have the same meaning as in the Fletcher-Powell case. The additional arguments control:

cycles The maximum number of cycles allowed for convergence. Defaults to min(20,nvar+10), where nvar is the number of variables to be optimized.

negeig The number of negative eigenvalues characterizing the desired stationary point. This is normally not needed, as it is set to 0 for a minimum and 1 for a transition state, but note that it may be greater than 1.

dxmaxt The maximum stepsize allowed in the search. Defaults to $2.0x10^{-2}$.

fmaxt The maximum allowable gradient element. Defaults to 1.0.

eigmax The maximum Hessian eigenvalue allowed before aborting run. Defaults to $2.5x10^{1}$.

eigmin The minimum Hessian eigenvalue allowed before aborting run. Defaults ro $1.0x10^{-4}$.

expert Expert optimization. Adjusts dxmaxt,fmaxt,eigmax, and eigmin so as to avoid shutting the calculation down when it thinks you are in a hopeless region of configuration space.

srcd2e Source of the initial Hessian. The default is to read the diagonal elements from the set defined in the `$geom` section. Other options include

analytic An analytic second derivative calculation is done to obtain the initial Hessian.

chk,rdchk Force constants from a previous calculation are retrieved from the checkpoint file.

rdinp The initial cartesian force constants are read from an input section entitled `$cfc`. The lower triangle is read in free-format. Note that this option presupposes that M732 has been run to transform these into internal coordinates before M201 is initially executed. This will require a nonstandard route.

clnd2e Obsolete.

12.3 Berny

The Berny algorithm is probably the most robust in Truchas. It recognizes all the options above, and two additional ones:

srcht The search type. There are four possibilities. "srcht=lq", the default, includes both linear and quadratic terms in the search. "srcht=q" includes only quadratic. "srcht=lsd",does linear and steepest descent, and "srcht=lsd?" does linear and steepest descent when allowed.

fswitch The minimum force for which the linear search is activated. Defaults to zero.

12.4 Examples

12.4.1 Example 1

In the following example we are searching for the transtion state in the cis-trans isomerization of HONO.

```
$route
  opt=(berny,ts)
$geom
  o1
  n  o1 r1
  o2 n  r2 o1 ang1
  h  o1 r3 n  ang2 o1 dih

  r1=1.3      constant
```

```
r2=1.4
r3=0.9
ang1=110.0
ang2=120.0
dih=80.0    d2e=-1.0
$end
```

Note that the dihedral angle, dih, associated with the rotation of the hydrogen atom out of the plane formed by the ONO atoms is expected to be the internal coordinate which dominates the description of the reaction path. This is the diagonal element of the initial force constant matrix which is given a negative value. One also has the ability to freeze internal coordinates during an optimization procedure by declaring them to be constants. This option was employed in the HONO input as the O1-N and distance was not allowed to vary.

Chapter 13

Force Constants and Frequencies

The second derivative of the energy with respect to nuclear motion gives the force constants of the molecule. When appropriately weighted with the atomic masses, the eigenvalues of this matrix give vibrational frequencies.

Truchas can compute analytic second derivatives of the energy for HF wavefunctions. At present, force constants for DFT wavefunctions are limited to finite-differencing first derivatives. This chapter describes the finite-difference and analytic methods used in Truchas to dermine force constant matrices and harmonic vibrational frequencies.

Note that force constants and frequencies are normally computed at the equilibrium geometry of the molecule. A convenient way to use the equlibrium geometry is to employ the `geom=rdchk` option in conjunction with a previous optimization run.

13.1 Force Constants

13.1.1 Analytic

The computation of force-constants is controlled by the `force-constants` keyword:

```
$route
  force-constants
```

```
$end
```

The default is to compute the force constant matrix analytically. In this case, one must also specify the directive `nderiv=2` in the $route section to tell the derivative links to compute through the second derivative.

13.1.2 Numerical

Numerical force-constants are invoked in `$route` by:

```
force-constants=(numerical=[cartesian,two-point,stepsize=])
```

The options are:

cartesian Use Cartesian displacements for the finite difference. The default is to use internal coordinates. Note that this can still be used even when the Z-matrix is used to specify the geometry, and in fact, it should be used if the Z-matrix contains dummy atoms or if there are not 3N-6 unique internal coordinates explicitly defined in the Z-matrix.

two-point Use a two-point difference formula to compute the second derivatives. The default is single point.

stepsize The step size to use for the finite difference. The default is 0.01 in radians and atomic units.

13.2 Frequencies

Frequencies are requested in the `$route` section by adding the keyword

```
force-constants,frequencies
```

This keyword takes no options, but note that force-constants must also be present in the route to activate this option.

13.3 Examples

Chapter 14

Visualization

Truchas interacts with the tool ChemistryViewer available from Molecular Simulations. This allows the chk file to be interrogated interactively to view the geometry, plot molecular orbitals, examine electrostatic potential surfaces, etc. ChemViewer can generate color postscript files so it is possible to get hard copies.

ChemViewer utilizes AVS graphics, so you must also purchase a copy of AVS. For additional documentation, consult the AVS and ChemViewer manuals.

A B/W figure would be nice here.

Appendix A

IOSys

Truchas performs word addressable i/o through a package of routines called IOSys. This package was written by Paul Saxe and is an attempt (and a successful one we believe) at a flexible, user-friendly i/o package which hides the machine dependent aspects of i/o. All i/o in Truchas other than formatted input and output is performed with IOSys.

A.1 Directives

The form of a call to IOSys is:

```
call iosys(directive,arg1,arg2,arg3,arg4)
```

where "directive" is a character string giving the action to be taken, and teh remaining four arguments depend on what is being done. In the following description of the directives, ¡file¿ is an IOSys internal unit name; parentheses, (), indicate a choice of words, one of which must be used; and brackets, [], indicate an optional keyword or phrase. Also, all non-essential words such as "no", "on", and "of" may be omitted wherever they appear; however, their inclusion is suggested as an aid to readability.

The possible directives are:

```
‘‘length of <file> on <unit>’’
‘‘get length of <file> on <unit>’’
‘‘get maximum length of <file> on <unit>’’
‘‘get read pointer of <file> on <unit>’’
```

```
‘‘get write pointer of <file> on <unit>’’
‘‘does <file> exist on <unit>’’
‘‘write [character integer real] <file> (to,on) <unit>
       [without rewinding] [asynchronously]’’
‘‘read [character integer real] <file> from <unit>
      [without rewinding] [asynchronously]’’
‘‘end <file> on <unit>’’
‘‘endfile <file> on <unit>’’
‘‘create [character integer real] <file> on <unit>’’
‘‘open <unit> [as (new,old,unknown,scratch)] [on ssd]’’
‘‘dump’’
‘‘destroy [<unit>]’’
‘‘close (all,<unit>)’’
‘‘wait for (all,<unit>)’’
‘‘rewind (all,<file>) on (all,<unit>)
        [(read,write,read-and-write)]’’
‘‘set alignment (for,of) <unit>’’
‘‘reset alignment (for,of) <unit>’’
```

IOSys manages collection as “files” on “units”, which are actually physical disc files. IOSys uses a directory to keep track of its logical “files” on each of its logical “units”. The first step in IOSys is to open one or more logical units using the “open” directive. This is the only stage where the user must be cognizant of the physical file name associated with the logical units. After a unit is opened, it is always referred to by its logical name embedded in a directive string.

There are two ways to create a logical file on a unit: the first is simply by writing data to a new file name using the “write” directive, the second is with an explicit “create” directive. The first method will create a file exactly the length of the data being written; whereas, a “create” directive allows the user to specify the length of the file. As a user, you will only need to explicitly create files only if all the data cannot be written to the file in one write. There is also a variant of the explicit create invoked by giving a length of -1. In this case, an “endless” file is created, on which an unspecified amount of data may be written. An “endless” file is active until an “end” or “endfile” directive is issued, at which point the “endless” file is fixed to the current length and may not be further extended. There are restrictions on the use

of "endless" files: first, only one endless file can be active on a given unit at one time, and second, no new files may be created implicitly or explicitly on a unit with an active endless file.

Logical file and unit names may have up to 16 characters; any beyond 16 are truncated. If the name contains embedded blanks, it must be enclosed in double quotes, e.g. "myfile". The quotes are treated as part of the name, so a name surrounded with quotes is not the same as the name without quotes. IOSys can currently handle 20 units open simultaneously with a total of 1000 files. These limits are set in the parameter statement and are readily modified.

The length of a file or of a transfer (read or write) are in terms of the type of data on the file. Integer and real files are in terms of integer of real words, respectively, while character data is by character. Also note that for character data, the length need not be given as it can be assumed to be the length of the character string passed in.

The "alignment" directive has to do with the physical storage of data. If alignment is set, new files created will be positioned starting at physical disc sectors. This will speed up i/o operations, but will also make the physical file larger since gaps will be left between logical files. Another ramification of alignment is that the "maximum length" of files can be larger than their "length" since IOSys will allow a file to expand into the gap left by alignment. Alignment is turned off with a "reset alignment" directive. By default, units are not aligned. Finally, the alignment of a unit can be changed as often as desired since it only affects new files being created.

A.2 Arguments

This section will describe briefly those directives associated with manipulating units. A "u" will denote that the argument is not referenced when using a directive, and the user should use a "0" or any variable in the calling sequence.

A.2.1 Manipulating Units

Open

A file is opened with the directive:

```
call iosys(‘‘open <unit> [as (new,old,unknown,scratch)] [on ssd]’’,
            size,u,u,name)
```

name The character name or a unit number of the physical disc file in use. It must have 7 characters or less.

size Obsolete. It referred to the size of family members in the CTSS operating system.

unknown The default for an open directive. If the file exists, it will be opened; if not, it will be created.

old Specifies that the file must exist; otherwise, there is an error.

new Guarantees a new file; if it already exists, it will be deleted and then recreated.

scratch Can be used to create a unique temporary file. In this case, no physical file name is needed as IOSys will create one, and when the unit is closed, the physical file will be deleted.

on ssd Obsolete? Specified that the file should be put on a solid-state disc if possible. If there is not room, or no SSD, the file is created on a normal disc. The SSD was only used for heavily used, temporary files that only exist as scratch in one program link. This may be useful once again in some machine environments.

Close

A file is closed with the directive:

```
call iosys(‘‘close (all,<unit>)’’,u,u,u,u)
```

This directive closes either a specific unit or all open units. When a unit is closed, the directory information is saved on the unit so that a subsequent open will restore the unit to the identical state as when it was closed — including the read and write pointers for files.

dump

A file is dumped with the directive:

```
call iosys(‘‘dump’’,u,u,u,u)
```

This will dump a listing of the file names, pointers, etc. for all units to the output file.

destroy

A file is destroyed with the directive:

```
call iosys(‘‘destroy [<unit>]’’,name,u,u,u)
```

This directive works in two fashions: if an IOSys name is given in the directive, that IOSys unit is closed and the physical file is destroyed. In this case, the argument “name” is not referenced. Otherwise, the physical file “name” is destroyed, regardless of whether it is an IOSys file or not.

set and reset alignment

File alignment is handled with the directives:

```
call iosys(‘‘set alignment on <unit>’’,u,u,u,u)
call iosys(‘‘reset alignment on <unit>’’,u,u,u,u)
```

The first turns on and the second turns off the alignment of new files to physical sector boundaries.

A.2.2 Manipulating Files

create

A file is created with the directive:

```
call iosys(‘‘create (character integer real) <file> on [<unit>]’’,
            length,u,u,u)
```

This call create a file of length “length” data elements on an IOSys unit. If “length” is -1, an “endless” file is created.

write

A file is written with the directive:

```
call iosys(‘‘write (character integer real) <file> (to,on) <unit>
             [without rewinding] [asynchronously]’’,nwords,array,offset,c)
```

nwords The number of data elements to transfer. If -1, IOSys wil use the current size of the file and transfer that number of data elements, which is a nice shortcut for rewriting a file.

array The array to transfer from.

offset The default is to rewind the file before writing. If this is not overridden with “without rewinding”, then “offset” specifies a location in the file to begin writing (starting at 0). Otherwise, “offset” is added to the current write pointer location before writing. Thus, repeated calls “without rewinding” and an “offset” of 0 will sequentialy write a file. Note that the read and write pointers are separate and do not affect each other.

c If writing characters, “c” is the character string to be written. In this case, the length need not be given, in which case the length of “c” will be used.

read

A file is read with the directive:

```
call iosys(‘‘read (character integer real) <file> from <unit>
             [without rewinding] [asynchronously]’’,nwords,array,offset,c)
```

Reading is exactly the inverse of writing, and all options and arguments are identical to those above.

end and endfile

Endless files are terminated with the directive:

```
call iosys(‘‘end <file> on <unit>’’,u,u,u,u)
```

The length of the file is fixed at the length at the time of this call.

wait

To wait for the completion of an asynchronous i/o request:

```
call iosys(‘‘wait for <unit>’’,u,u,u,u)
```

An array being written asynchronously cannot safely be modified until a "wait" call has been made. Conversely, data being read asynchronously cannot be used before a call to "wait".

rewind

To rewind the read and/or write pointers of one or all of the files on one or all units:

```
call iosys(‘‘rewind (all,<file>) on (all,<unit>)
           [read write read-and-write)]’’,u,u,u,u)
```

A.2.3 Inquiries About Files

length

To get the length of the data written to a file:

```
call iosys(‘‘length of <file> on <unit>’’,length,u,u,u)
call iosys(‘‘get length of <file> on <unit>’’,length,u,u,u)
```

If the file does not exist, -1 is returned.

maximum length

To get the maximum amount of data which can be stored on a file:

```
call iosys(‘‘get maximum length of <file> on <unit>’’,length,u,u,u)
```

If the file does not exist, -1 is returned.

get write pointer

To get the address in a file (counting from 0) where the next write would occur if the file is not rewound:

```
call iosys(‘‘get write pointer of <file> on <unit>’’,pointer,u,u,u)
```

If the file does not exist, -1 is returned.

get read pointer

To get the address in a file (counting from 0) where the next read would occur if the file is not rewound:

```
call iosys(‘‘get read pointer of <file> on <unit>’’,pointer,u,u,u)
```

If the file does not exist, -1 is returned.

does file exist

To see if a file exists:

```
call iosys(‘‘does <file> exist on <unit>’’,u,u,u,answer)
```

The “answer” is returned as “yes” or “no”.

Appendix B

Print Flags

A complete list of the print flags available in truchas follows.

Appendix C

Links

The responsibilities of the various links in Truchas are:

M0 Truchas initialization link. Creates new output file and prints header information.

M1 Reads the `$route` section and generates a route.

M101 Reads the `$geom` section.

M102 Reads the basis set information and computes the number of electrons.

M201 Drives the geometry optimization.

M202 Generates molecular coordinates from Z-matrix.

M204 Computes harmonic vibrational frequencies given a force constant matrix.

M205 Drives finite difference force constant calculation.

M302 Computes one-electron and ECP integrals.

M303 Computes derivative contracted one-electron and ECP integrals.

M312 Computes two-electron integrals.

M323 Computes derivative contracted two-electron integrals and writes a chained output file. All of the derivative integrals associated with a particular degree of freedom are chained together.

M330 Orders two-electron integrals.

M401 Generates an initial set of molecular orbitals for SCF calculations.

M501 HF program.

M502 IVO program.

M503 General symmetry-constrained HF program.

M511 DFT program.

M611 Performs Voronoi charges analysis.

M618 Generates molecular cavity surface.

M619 Computes electrostatic potential on cavity surface.

M620 Cox-Williams fit of the cavity surface potential.

M621 BSJ solution of the solvent Poisson equation.

M701 Transforms the contracted one-electron AO density to the primitive basis.

M702 Computes derivative primitive one-electron integrals and traces them with the density.

M711 Computes derivative primitive Coulomb integrals and traces them with the density.

M712 Computes derivative primitive two-electron integrals and traces them with the density.

M721 Computes derivative primitive exchange-correlation integrals and traces them with the density.

M731 Transforms the cartesan gradient into internal coordinates.

M732 Transforms the cartesian force constant matrix into internal coordinates.

M1010 Constructs an SCF Lagrangian with derivative integrals.

M1011 Constructs generalized SCF Lagrangian.

M1012 Constructs the CPHF Hessian.

M1013 Builds the right hand side of the CPHF equation.

M1014 Builds omega term for SCF second derivatives.

M1020 Solves CPHF equations.

M1022 Assembles final SCF force constant.

M1902 Computes property integrals.

M1951 Analyzes molecular orbitals and evaluates molecular properties.

M2001 Prints timing informaiton and copies data from the RWF file to the CHK file.

Appendix D

Nonstandard Routes

Truchas performs an electronic structure calculation by running a sequence of programs (links) in an order specified by the route. A route has the appearance:

```
1//1,2;
2//2;
3//2,12,30;
4//1;
5//11;
19//51;
20//01;
```

which is the route generated for a single-point DFT calculation. The first number in each line specifies the "overlay" number, the field between the slashes allows one to specify options, and the entries after the option field specify programs within an overlay. An overlay specification is terminated with a semi-colon.

An individual link in the chain is distinguished by concatenating the overlay number with the program number. For example, the route above will execute M0 and M1 – all routes do – and then M101, M102, M202, M302, M312, M330, M401, M511, M1951, and finally M2001. Programs which perform a similar function or accomplish different aspects of a common task are generally grouped together in the same overlay.

The individual programs are not overlaid in the sense familiar to computer programmers. The last function performed by each program is to call a

subroutine which parses the route information, decides which program to run next, and writes the name of the next program to the standard error unit. The shell script overseeing the execution intercepts this message and invokes the next program. A link number "M998" signals the script that the calculation is finished.

The route above will appear familiar to users of the Gaussian codes. This basic communication and execution design was modeled after the early Gaussian programs. The route is actually capable of greater program control than the example above indicates, however. It is possible to encode labels, loops, and conditional structures in the route. The area between the slashes may be used to modify the memory allocated to any particular link, and to change various options to individual links.

As long as you are doing conventional calculations and haven't added any links of your own to Truchas, you probably will have no need to understand the structure of the route, or how M1 generates one. For the curious, however, M1 condenses the information contained in the `$route` section into a character string of the form

```
'GEOM/WFN1/WFN2/PROP'
```

where GEOM dictates the geometry field, i.e. whether a single-point calculation or an optimization is to be performed, WFN1 specifies the calculation used to obtain the one-electron molecular orbitals, WFN2 characterizes the many-body approximation to be performed, and PROP defines the properties to be computed from the wavefunction. A single-point DFT calculation, e. g. , is characterized by the string:

```
'SP/DFT/-/-'
```

where only the default properties are to be computed from the density. A Fletcher-Powell geometry optimization with a full calculation of molecular properties at the optimum geometry is characterized by:

```
'FLETCHER-POWELL/DFT/-/FULL'
```

After M1 assembles this string from the keywords in `$route`, it searches the data file mesa.dat for this identifier and loads the full route. The mesa.dat file contains routes, standard basis sets, and information about which files to checkpoint at the end of the calculation. It is a formatted file, and so new routes or basis sets may be added to it at any time.

D.1 Looping

D.2 The Options Field

Appendix E

Examples

A number of examples of standard Truchas calculations follow.

E.1 HF

E.1.1 Closed shell HF

This input deck requests a standard single-point HF calculation. Note that since the basis set is not specified, it will default to "dz".

```
$route
 2s+1=1
 print=(basis,scf=vector)
 hf scf=(pulay)
 guess=(huckel)
 sym=off

$title
 h2o test case: closed shell scf

$geom
 o
 h1 o r1
 h2 o r2 h1 a1
```

```
 r1=1.949
 r2=1.949
 a1=112.3
$end
```

E.1.2 Open shell HF

This example performs an open-shell HF calculation on the lowest triplet state of water.

```
$route
 2s+1=3
 hf scf=(pulay,cycles=64)
 guess=(huckel)
 basis=dz,sym=off
$title
  h2o test case: open-shell scf
$geom
 o
 h1 o r1
 h2 o r2 h1 a1

 r1=1.949
 r2=1.949
 a1=112.3
$end
```

E.1.3 General SCF

An example of general SCF as well as a general basis set read from the input deck. Note that the geometry is defined by Cartesian coordinates with the distances in atomic units.

```
$route
 2s+1=2,q=0,geom=(coord,inau)
 print=(scf=(vector=all)),sym=off,guess=core
 hf,scf=(convergence=5,gscf,nshell=2,shlnbf=(2,3),
    fn=(1.,1.),     fd=(1.,6.),
```

```
      an=(2.,1.,0.),  ad=(1.,3.,1.),
      bn=(-1.,-1.,0.),bd=(1.,6.,1.))
 $title
  boron test case: general scf
 $geom
  b(basis=dz)  0.0  0.0  0.0
 $dz b
  type=s
  2788.4100  0.002122
   419.0390  0.016171
    96.4683  0.078356
    28.0694  0.263250
     9.3760  0.596729
     1.3057  0.230397
  type=s
     3.4062  1.0
  type=s
     0.3245  1.0
  type=s
     0.1022  1.0
  type=p
     2.4360  0.110339
     0.6836  0.383111
     0.2134  0.647860
  type=p
     0.0701  1.0
$end
```

E.1.4 Properties

In this example several properties are evaluated for the HF wavefunction. These are multipoles through hexadecapole, electrostatic properties through the electric field gradient at each nucleus, and the Fermi contact term. The Mulliken population analysis, orbital by orbital, will also be printed. note the nonstandard route.

```
 $route
```

```
 hf,q=1,2s+1=2,scf=(cycles=32,guess=(huckel),convergence=6,pulay),
 geom=coord,
 print=(m501=vector),sym=off,
 properties=(populations=orbital,e4,v2,fermi)
$nonstd
  1//1,2;
  2//2;
  3//2,12,30;
  4//1;
  5//1;
  6//11;
  19//2,51;
  20//01;
$title
 CuH
$geom
 cu(basis=wachters)          0.000000    0.000000    0.000000
 j (basis=6-31g)             1.889650    0.000000    0.000000
$end
```

E.2 DFT

This set of decks tests various DFT single-point calculations.

E.2.1 Closed Shell DFT

A closed-shell S-Null calculation on methylene. Note the use of a general grid.

```
$route
dft,2s+1=1,scf=(exchf=slater,pulay,lebord=23,radgrid=51),
guess=huckel,print=(m511=vector),grid=general
$title
  ch2 singlet/ slater-null
$geom
c(basis=sto-3g)
h1(basis=sto-3g) c r
```

```
h2(basis=sto-3g) c r h1 theta

r=1.130
theta=100.9
$end
```

E.2.2 Open shell DFT

An open-shell S-Null calculation on methylene. The standard grid, sg1, will be used on all centers.

```
$route
dft,2s+1=3,scf=(exchf=slater,pulay,
density-cutoff=1.0d-20)
guess=(huckel),print=(guess,m511=vector),basis=sto-3g,grid=sg1
$title
 ch2 triplet  slater
$geom
c
h1 c r
h2 c r h1 theta

r=1.092
theta=134.4
$end
```

E.2.3 Becke-LYP

A gradient-corrected DFT calculation. The final vectors are printed in M511. Note the alternative means of specifying the standard grid.

```
$route
 dft,2s+1=1,scf=(exchf=becke,corrf=lyp,pulay),
 guess=huckel,print=(m511=vector)
$title
 ch2 singlet  becke-lyp
$geom
 c(basis=6-31g,grid=sg1)
```

```
 h1(basis=6-31g,grid=sg1) c r
 h2(basis=6-31g,grid=sg1) c r h1 theta

 r=1.130
 theta=100.9
$end
```

E.2.4 Direct DFT

A DFT calculation using direct formation of the Coulomb matrix.

```
$route
 dft,2s+1=1,scf=(directj,exchf=becke,corrf=lyp,pulay,
 density-cutoff=1.0d-20)
 guess=(huckel),print=(guess,m511=vector),grid=tag3
$title
 h2o singlet  becke/lyp
$geom
 o(basis=6-31g*)
 h1(basis=6-31g) o r
 h2(basis=6-31g) o r h1 theta

 r=1.000
 theta=104.5
$end
```

E.3 Optimization and Frequencies

The third group tests scf optimization and second derivative links.

E.3.1 Closed shell HF, Fletcher-Powell

This deck requests a geometry optimization using the Fletcher-Powell finite difference algorithm.

```
$route
 2s+1=1
```

```
  hf scf=(pulay)
  guess=(huckel)
  opt=(fletcher-powell)
  basis=dz,sym=off
$end
$title
 h2o test case: closed shell scf optimization(fletcher-powell)
$geom
 o
 h1 o r
 h2 o r h1 a

 r=0.949
 a=112.3
$end
```

E.3.2 Closed-shell HF, Murtaugh-Sargeant

An optimization using the default Murtaugh-Sargeant optimization algorithm.

```
$route
 2s+1=1
 hf scf=(pulay)
 guess=(huckel)
 opt
 basis=dz,sym=off
$title
 h2o test case: closed shell scf optimization(murtaugh-sargent)
$geom
 o
 h1 o r1
 h2 o r2 h1 a1

 r1=0.949
 r2=0.949
 a1=112.3
```

```
$end
```

E.3.3 Closed shell HF, Berny

A Berny optimization run.

```
$route
 2s+1=1
 hf scf=(pulay)
 guess=(huckel)
 opt=(berny)
 basis=dz,sym=off
$end
$title
 h2o test case: closed shell scf optimization
$geom
 o
 h1 o r1
 h2 o r2 h1 a1

 r1=0.949
 r2=0.949
 a1=112.3
$end
```

E.3.4 DFT optimization, Berny

Berny optimization, DFT

```
$route
 2s+1=1
 dft=(corrf=lyp,exchf=becke) scf=(pulay)
 guess=(huckel)
 opt=(berny)
 basis=6-31g*,sym=off,grid=sg1
$end

$title
```

```
  h2o test case: closed shell dft optimization
$geom
 o
 h1 o r1
 h2 o r2 h1 a1

 r1=0.949
 r2=0.949
 a1=112.3
$end
```

E.3.5 DFT optimization, Fletcher-Powell

Fletcher-Powell optimization, DFT. Note that this is currently the only available means of optimizing open-shell DFT cases.

```
$route
 2s+1=1
 dft=(corrf=lyp,exchf=becke) scf=(pulay)
 guess=(huckel)
 opt=(fletcher-powell)
 basis=6-31g*,sym=off,grid=sg1
$end

$title
 h2o test case: closed shell dft optimization
$geom
 o
 h1 o r1
 h2 o r2 h1 a1

 r1=0.99
 r2=0.99
 a1=102.3
$end
```

E.3.6 Open shell HF optimization, Berny

An open-shell optimization.

```
$route
 2s+1=3
 hf scf=(pulay)
 guess=(huckel)
 opt=(berny)
 basis=dz,sym=off
$end
$title
 h2o test case: open-shell scf optimization
$geom
 o
 h1 o r1
 h2 o r2 h1 a1

 r1=0.949
 r2=0.949
 a1=112.3
$end
```

E.3.7 Closed shell transition state, Berny

An example of a transition-state optimization.

```
$route
 2s+1=1
 hf scf=(pulay)
 guess=(huckel)
 opt=(ts,berny)
 basis=dz,sym=off
$end
$title
  h2o test case: closed shell scf transition state optimization
$geom
 x
```

```
 o x 1.0
 h1 o r1 x a1
 h2 o r2 x a1 h1 180.0

 r1=0.949
 r2=0.949
 a1=90.0
$end
```

E.3.8 Analytic HF frequencies

An example of analytic frequencies with the HF wavefunction. Note that this input deck assumes that the molecule has been optimized in a previous step; it picks up the geometry and gruee wavefunction from the chk file.

```
$route
 2s+1=1
 hf scf=(pulay)
 frequencies force-constants
 guess=(chk) geom=rdchk
 basis=dz,sym=off
$end
$title
 h2o test case: scf/analytic frequencies
$end
```

E.3.9 Numerical frequencies, two-point internal

An example of numerical finite-difference frequencies.

```
$route
 2s+1=1
 print=(scf=vector)
 hf scf=(pulay)
 guess=chk geom=rdchk
 frequencies force-constants=(numerical=(two-point))
 basis=dz,sym=off
$end
```

```
$title
  h2o test case: closed shell scf forceconstantsinternal
                              /two-point difference formula
$end
```

E.3.10 Numerical frequencies, Cartesian two-point

An example of numerical force constants in cartesian coordinates. A two-point finite difference method is used.

```
$route
 2s+1=1
 print=(gradient=(total))
 hf scf=(pulay), guess=chk, geom=rdchk
 frequencies
 force-constants=(numerical=(cartesian,two-point))
 basis=dz,sym=off
$title
 h2o test case: closed shell finite-difference force constants
           / cartesian two-point
$end
```

E.3.11 Numerical frequencies, internal one-point

An example of numerical force constants in which the default internal coordinates and one-point difference formula are used.

```
$route
 2s+1=1
 print=(scf=vector)
 hf scf=(pulay)
 guess=chk geom=rdchk
 frequencies force-constants=numerical
 basis=dz,sym=off
$end
$title
  h2o test case: closed shell scf forceconstantsinternal
                              /single-point difference formula
```

```
$end
```

E.3.12 Closed shell optimization using analytical second derivatives

This optimization uses analytic sceond derivatives to find the optimum geometry.

```
$route
 2s+1=1
 hf scf=(pulay)
 guess=(huckel)
 basis=dz,sym=off,force-constants
 opt=(berny,srcd2e=analytic)
$nonstd
  1//1,2;
  :opt;
   2//1(:endopt);
   2//2; 3//2,12,30; 4//1; 5//1; 10//11; 7//1,2,12; 8//11; 3//3,23;
   8//24; 10//12,13,14,20,22; 7//31,32(:opt);
  :endopt
  19//51; 20//01;
$end
$title
   h2o test case: scf/analytic second derivative optimization
$end
$geom
 o
 h1 o r1
 h2 o r2 h1 a1

 r1=1.1
 r2=1.1
 a1=104.0
$end
```

E.4 Solvation

An example of a solvation calculation follows. The denstiy is fit with the Cox-Williams approach. Note that the solvent radii are read in the geometry section.

```
 $route
  hf,
  q=0,
  cox-williams=(print=potential),
  print=(scf=energy,basis,m501=vector),
  scf=(pulay,convergence=6),
  solvent=(epsilon=80.0,npoints=20000,nfocus=128,convergence=1.0d-04),
  geom=(coord),
  guess=huckel,
  sym=off
 $title
  methanol, solvation free energy, cox-williams charges
  used sto-3g basis used
 $geom
  c1(basis=6-31g**,r=1.8)      .04884      .67018      .00000
  o1(basis=6-31g**,r=1.65)     .04884     -.75982      .00000
  h1(basis=6-31g**,r=1.0)     -.86453    -1.07432      .00000
  h2(basis=6-31g**,r=1.375)   1.10612      .97734      .00000
  h3(basis=6-31g**,r=1.375)   -.46265     1.07727      .88592
  h4(basis=6-31g**,r=1.375)   -.46265     1.07727     -.88592
$end
```

References

[1]A.D. Becke. A multicenter numerical integration scheme for polyatomic molecules. *J. Chem. Phys.*, 98:2547, 1988.

[2]T. H. Dunning, Jr. and P. Jeffrey Hay. *Gaussian Basis Sets For Molecular Calculations*, pages 1–27. Plenum Press, 1977.

[3]James B. Foresman and A. Frisch. *Exploring Chemistry with Electronic Structure Methods: A Guide to Using Gaussian.* Gaussian Inc., 4415 Fifth Avenue, Pittsburgh, PA., 15213, 1993.

[4]W.J. Hehre, R.F. Stewart, and J.A. Pople. Self-consistent molecular-orbital methods. i. use of gaussian expansions of slater-type atomic orbitals. *J. Chem. Phys.*, 51:2657–2664, 1969.

[5]S. Huzinaga. Gaussian-type functions for polyatomic systems. i. *J. Chem. Phys.*, 49:1293–1302, 1964.

[6]V.I. Lebedev. Density functional calculations on first-row transistion metals. *Zh. Vyschisl. Mat. Mat. Fiz.*, 15:48–54, 1975.

[7]V.I. Lebedev. Density functional calculations on first-row transistion metals. *Zh. Vyschisl. Mat. Mat. Fiz.*, 16:293–306, 1976.

[8]R.L. Martin, T.V. Russo, B.H. Lengsfield, P.W. Saxe, G.J. Tawa, M. Page, P.J. Hay, and A.K. Rappé. *Truchas*, 1994.

[9]C.W. Murray, N.C. Handy, and G.J. Laming. Quadrature schemes for integrals of density functional theory. *Mol. Phys.*, 78:997–1014, 1993.

[10]R. G. Parr and W. Yang. *Density-Functional Theory of Atoms and Molecules.* Oxford University Press, New York, 1989.

[11]R.C. Raffenetti. General contraction of gaussian atomic orbitals: Core,valence, polarization and diffuse basis sets; molecular integral evaluation. *J. Chem. Phys.*, 58:4452–4458, 1973.

[12]A. Szabo and N.S. Ostlund. *Modern Quantum Chemistry. Introduction to Advanced Electronic Structure Theory.* McGraw-Hill, New York, 1982.

[13]O. Treutler and R. Alrichs. Efficient molecular numerical integration schemes. *J. Chem. Phys.*, 102:346–354, 1995.

[14]A.J.H. Wachters. Gaussian basis set for molecular wavefunctions containing third-row atoms. *J. Chem. Phys.*, 52:1003–1036, 1970.