

PQS *Ab Initio* Program Package
version 4.0

USER'S GUIDE

*This document is available on your computer in
/usr/local/share/PQS/DOC/pqs-man-4.0.pdf*

**Parallel Quantum Solutions
2013 Green Acres, Suite A
Fayetteville
Arkansas 72703
U. S. A.**

For sales contact: sales@pqs-chem.com
For technical support: tech@pqs-chem.com

CONTENTS

GENERAL PRINCIPLES AND PROGRAM CAPABILITIES	2
INPUT FORMATS: PQS STYLE AND POPLE STYLE	4
PQS INPUT AND OUTPUT - TWO EXAMPLES	4
OVERVIEW OF PQS COMMANDS	13
FULL DESCRIPTION OF THE PQS STYLE INPUT FILE AND KEYWORDS	16
MEM	19
FILE	21
CPU	22
GEOM	23
BASIS	27
GUESS	37
INTE	41
SCF	42
FORCE	49
NUMGRAD	51
NUMHESS	53
HESS	54
NUMPOLAR/POLAR	55
FREQ	56
SQM	58
NMR/VCD	59
MP2 and DUAL-BASIS MP2	62
CORR	66
POP	69
NBO	70
PROPERTY	71
COSMO	72
SEMI	75
FORCE FIELD (FFLD)	78
OPTIMIZE	87
CLEAN	97
DYNAMICS	98
QM/MM	99
POTENTIAL SCAN	100
REACTION PATH	102
DESCRIPTION OF THE POPLE STYLE INPUT FILE	104
RUNNING JOBS	107
PROGRAM FILES	109
EXAMPLES	112
RESTARTS AND CHECKPOINTS	152
RUNNING PQS IN PARALLEL	155
SUN GRID ENGINE (SGE) PARALLEL JOB QUEUE	158
INPUT DESCRIPTION FOR SQM (Scaled Quantum Mechanical Force Field)	159
FREQUENTLY ASKED QUESTIONS	167
REFERENCES	169

INPUT DESCRIPTION FOR PQS version 4.0

INTRODUCTION

This documentation describes the general philosophy, the functionality, and the input file for the PQS *ab initio* quantum chemistry suite of programs. The PQS input file, unlike the route information in some programs, is fairly easy to generate, change and read, and after a little practice it is as easy to work with (although perhaps less compact) than the input file for any other quantum chemistry program. A graphical user interface (GUI) for input generation is available, combined with parallel job submission and post-job visualization, and is described in a separate manual. There is also a more compact, Pople-style input, familiar to users of, e.g., the popular Gaussian program package, although this is only available for the more common job types.

PROGRAM CAPABILITIES

PQS has a wide range of capabilities, and is under continuous development and improvement. Current capabilities include

- **An efficient vectorized Gaussian integral package allowing high angular momentum basis functions and general contractions**
- **Abelian point group symmetry throughout; utilizes full point group symmetry (up to I_h) for geometry optimizations, numerical gradients, numerical Hessian evaluation and for the CPHF step in analytical Hessian evaluation**
- **Closed-shell (RHF) and open-shell (UHF) SCF energies and gradients, including several initial wavefunction guess options.**
- **Closed-shell (RHF) and open-shell (UHF) density functional energies and gradients including all popular exchange-correlation functionals: VWN local correlation, Becke 88 nonlocal exchange, Handy-Cohen optimized exchange (OPTX), Lee-Yang-Parr nonlocal correlation, B3LYP etc...**
- **Fast *and* accurate pure DFT energies and gradients for large basis sets using the Fourier Transform Coulomb (FTC) method**
- **Efficient, flexible geometry optimization for all these methods including Eigenvector Following (EF) algorithm for minimization and saddle-point search, GDIIS algorithm for minimization, use of Cartesian, Z-matrix and delocalized internal coordinates. Includes new coordinates for efficient optimization of molecular clusters and adsorption/reaction on model surfaces**
- **Enforced geometry optimization (i.e., geometry optimization in the presence of an external force)**

- Full range of geometrical constraints including fixed distances, planar bends, torsions and out-of-plane bends between *any* atoms in the molecule and frozen (fixed) atoms. Atoms involved in constraints do *not* need to be formally bonded and - unlike with a Z matrix - desired constraints do *not* need to be satisfied in the starting geometry
- Analytical second derivatives for Hartree-Fock and DFT wavefunctions, including the calculation of vibrational frequencies, IR intensities and thermodynamic analysis
- Effective Core Potentials (ECPs), both relativistic and non-relativistic, including energies, gradients and analytical second derivatives.
- Nuclear magnetic shieldings for closed-shell HF and DFT wavefunctions
- Vibrational Circular Dichroism for the above
- Canonical, closed-shell RMP2 energies and analytical gradients and dual-basis RMP2 closed-shell energies
- Numerical closed-shell RMP2 second derivatives
- Unrestricted open-shell UMP2 energies
- Correlated energy module with closed-shell MP3, MP4, CID, CISD, CEPA-0, CEPA-2, QCISD, QCISD(T), CCD, CCSD and CCSD(T) wavefunctions
- Numerical gradients for all of the above
- Potential scan, including scan + optimization of all other degrees of freedom
- Reaction Path (IRC) following using either Z-matrix, Cartesian or mass-weighted Cartesian coordinates
- Population analysis, with bond orders and atomic valencies (free valencies for open-shell systems). Includes Charges from Electrostatic Potential (CHELP) and (optionally) Weinhold's Natural Bond Order (NBO) analysis, including natural population and steric analysis
- Properties module with charge, spin-density and electric field gradient at the nucleus
- Polarizabilities and hyperpolarizabilities and dipole and polarizability derivatives
- COSMO solvation model, including energies, analytical gradients and numerical second derivatives, for HF, DFT and canonical MP2 wavefunctions. Also available with NMR (closed-shell HF and DFT).

- Full Semiempirical package, both open (unrestricted) and closed-shell energies and gradients, including MINDO/3, MNDO, AM1 and PM3. For the latter, all main group elements through the fourth row (except the noble gases) as well as Zinc and Cadmium, have been parametrized
- Molecular Mechanics using the Sybyl 5.2 and Universal Force Fields
- QM/MM using the ONIOM method
- Molecular dynamics using the simple Verlet algorithm
- Pople-style input for quick input generation and compatibility with other programs

INPUT FORMATS: PQS STYLE AND POPLE STYLE

The compact input format introduced by J. A. Pople in the Gaussian 70 program and its successors is widely used by quantum chemists. To assist users of the PQS suite who are familiar with this input, we have an alternative Pople-style input reader in PQS. The program recognizes Pople-style input by a hash mark (#) as the first non-zero character on any input line. The Pople-style input is converted internally to standard PQS-style input, which is saved in the file <jobname>.pqs .

Note that the Pople-style input is compatible with the input of other programs, notably the Gaussian series, only in its general features. It is not guaranteed that an input file designed for any other program will work correctly for PQS, or that a Pople-style PQS input file will run correctly with any other program system. Because of the difference in the features of different programs, complete compatibility is impossible to achieve. Nevertheless, our Pople-style input should appear familiar to many quantum chemists.

Note also that the Pople-style input, because of its simplicity, does not recognize all of the PQS keywords, only the more routine ones. To access all features of PQS, use the PQS input, or edit the <jobname>.pqs file generated by the program from an initial Pople-style input file.

PQS INPUT AND OUTPUT - TWO EXAMPLES

Native PQS input consists of a series of commands, instructing the program to perform a program step. A command line begins with a single reserved keyword on a separate input line, and may be followed by options, separated by spaces. Options have the form of either a single keyword (e.g. **BOHR**) or **KEYWORD=value** where *value* is either a numerical value, a set of 2 or 3 (but not more than 3) numerical values, or a single character string. Commands are processed sequentially, but loops may be set up to execute the same group of commands repeatedly via the **JUMP** command. Normally only the first four characters of each keyword are significant, but more can be used to

facilitate reading.

Job output is written to standard output and is typically saved on job completion in the output file <jobname>.out. As well as the full output, there is a summary output (or short output) which is saved in the file <jobname>.log. The log file contains only output considered to be of direct interest to the user, such as the final energy, optimized geometry and any computed molecular properties. The log file can be saved as a reliable summary of a successful job, whilst the larger output file can be deleted.

Before giving an overview of the program modules and a detailed description of the PQS input file and keywords, we begin with a couple of examples, describing the input file, how to actually submit and run the job, and showing the log file that each job produces.

Example 1

Our first example is a full geometry optimization for water, followed by an NMR chemical shift calculation, using the B3LYP hybrid density functional and the 6-31G* basis set. The input file for this, `water.inp`, is:

```
TITLE Water geometry optimization + NMR chemical shifts
GEOM=pqs GEOP
O      0      0      0
H      0      0.8    0.6
H      0      -0.8   0.6
BASIS=6-31G*
GUESS=HUCKEL
OPTI          -----
SCF DFT=B3LYP          | basic optimization loop
FORCE          |
JUMP          -----
NMR
```

Anatomy of the input file

This input uses nine command keywords: **TITLE**, **GEOM**, **BASIS**, **GUESS**, **OPTimize**, **SCF**, **FORCE**, **JUMP** and **NMR**.

The **TITLE** command is trivial and simply gives a job title or heading. The title string given will be echoed in the output and log files. (In fact the entire input file will be echoed.)

The **GEOM** command controls input of the molecular geometry. **GEOM=pqs** means that we use the PQS native style to define the initial molecular geometry. PQS has a flexible geometry reader, and can read a number of different input formats, e.g., Z-matrix or Protein Database (pdb) styles, and geometries from several graphical modeling programs. The `GEOP` (Geometry Parameters) option requests the printing of *chemically relevant* bond distances, angles and torsions. The **GEOM** command is followed by the molecular geometry in PQS format (in the simplest case - as here - the atomic symbol and X, Y, Z Cartesian coordinates, default in Å, free format)

The **BASIS** command specifies the Gaussian basis set, in this case the 6-31G* basis of Pople and coworkers. PQS has a flexible basis set input, and can read basis sets from the input file or from an external file, and can augment the specified basis with additional basis functions. The **BASIS** command is required in all *ab initio* calculations.

The **GUESSs** command specifies the initial wavefunction guess for the molecule, in this case an extended Hückel wavefunction. In most cases the **GUESSs** command is optional; if it is left out, a well-defined sequence of default guesses is tried automatically by the program.

The **OPTI** command specifies a geometry optimization. This is a so-called loop command, which involves repeated execution of a sequence of commands until some condition is reached which exits the loop. The **JUMP** command denotes the end of the command sequence. Thus in this example the commands **OPTI**, **SCF** and **FORCE** will be executed repeatedly until some appropriate exit condition is satisfied (hopefully convergence to the optimized geometry). The PQS optimization module has a rich set of options, but in standard cases no additional options are required.

The **SCF** and **FORCE** commands, respectively, specify a self-consistent field calculation, followed by the evaluation of the nuclear gradient, i.e., the forces on the atoms. The gradients are required for the geometry optimization. The **SCF** command has an option, **DFT=B3LYP** which specifies that the hybrid B3LYP exchange-correlation functional is to be used.

Finally, the command **NMR** initiates the calculation of NMR chemical shifts.

The log file (the short output file) produced by running `water.inp` is:

```

=====
PQS  Ab Initio Program Package running on dirac
Date       : Tue Jan 11  9:43:37 2011
Executable : /home/pqs1/PQSV40/INTEL64/pqs.x
Type       : ELF 64-bit LSB executable, AMD x86-64, version 1 (SYSV),
            for GNU/Linux 2.4.1, statically linked, stripped
Intsize    : 1
=====
TITLE  Water geometry optimization + NMR chemical shifts
GEOM=PQS GEOP
O      0      0      0
H      0      0.8  0.6
H      0      -0.8  0.6
BASIS=6-31G*
GUESS=HUCKEL
OPTI
SCF DFT=B3LYP
FORCE
JUMP
NMR

Empirical Formula: H2O

      Cartesian Coordinates in Standard Orientation

              Coordinates (Angstroms)
      ATOM          X          Y          Z
1  o          0.000000    0.000000   -0.400000
2  h          0.000000    0.800000    0.200000
3  h          0.000000   -0.800000    0.200000
Point Group: C2v  Number of degrees of freedom: 2

```

```

Charge: 0.000000 Multiplicity: 1
Wavefunction: RDFT XC potential: b3lyp
Basis set: 6-31g-d
Number of contracted basis functions: 19

```

```

** Cycle 1 Energy -76.406968491 RMSG 0.02925 RMSD 0.07618 **
** Cycle 2 Energy -76.408906347 RMSG 0.00300 RMSD 0.01613 **
** Cycle 3 Energy -76.408955138 RMSG 0.00018 RMSD 0.00064 **
** Cycle 4 Energy -76.408955243 RMSG 0.00002 RMSD 0.00004 **

```

CONVERGED GEOMETRY
Coordinates (Angstroms)

	X	Y	Z
o	0.0000000000000000	0.0000000000000000	-0.39913730451687
h	0.0000000000000000	0.76155207034049	0.19956865225844
h	0.0000000000000000	-0.76155207034049	0.19956865225844

dipole/D = 0.000000 0.000000 2.095284 total= 2.095284

NMR SHIELDINGS

O	Atom= 1	Isotropic shielding=	316.58563	Anisotropy=	39.87370
H	Atom= 2	Isotropic shielding=	31.95379	Anisotropy=	17.39368
H	Atom= 3	Isotropic shielding=	31.95379	Anisotropy=	17.39368

```

Charge: 0.000000 Multiplicity: 1
Wavefunction: RDFT XC potential: b3lyp
Basis set: 6-31g-d
Number of contracted basis functions: 19

```

Energy is: -76.408955243 au

dipole/D = 0.000000 0.000000 2.095284 total= 2.095284

```

=====  

Total master CPU time = 0.05 Elapsed = 0.05 min  

Termination on Tue Jan 11 9:43:41 2011  

=====

```

The log file starts with a header indicating which machine you are running on (here `dirac`), the date and time the job was started, the location and type of the PQS executable, and the integer size (the number of integers in a double word, 8 bytes). This release of PQS is fully 64-bit and an `Intsize` of 1 indicates that this is a 64-bit executable (an `Intsize` of 2 would be 32-bit). The entire input file is then echoed.

Then follows output from the **GEOM** command: the empirical formula, Cartesian coordinates (possibly reoriented), the point group symmetry and the number of degrees of freedom. This is followed by the charge and multiplicity, the theoretical method (here restricted (closed-shell) DFT with the B3LYP functional) and the basis set.

There is then a summary of each optimization cycle, showing the cycle number, the energy, and the root-mean-square gradient and displacement, respectively. This

particular optimization converged in four cycles. The final converged geometry is printed, together with the final dipole moment.

The chemical shifts for each atom (from the **NMR** module) are then given, together with a final summary giving the optimized energy.

This format is fairly typical of the log file. The precise content depends, of course, on the actual job.

Example 2

Our second example is a Pople-style input file: a DFT geometry optimization on triplet dioxygen with a good quality basis set (`o2.com`). (It has the input extension `.com` to distinguish it from PQS style input which has the default extension `.inp`)

```
%MEM=3
# BPW91/6-311G(2df,2pd) OPT

Bond distance in triplet O2 by BPW91 and a good basis

O 3
O
O 1 R

R=1.2
```

The Pople input style is recognized by the hash mark as the first character of the main command line. The very first line in the input sets the maximum amount of memory that can be utilized by the job (in this example 3,000,000 MWords, i.e., 24 MB – this is a reduction from the default, which is 6,000,000 MWords; it is included here only to demonstrate this feature, as this small job does not need even this much memory). The second line is the main command line (also called the *route card*). It defines the quantum chemical method used (DFT using the BPW91 functional), the basis set, and the type of calculation (a geometry optimization). An empty line concludes this section.

Line 4 is the *title*, again terminated by an empty line. Line 6 defines the molecular *charge* (0) and *multiplicity* (3). Line 7 onwards defines the molecular geometry in Z-matrix format, using the variable R as the O-O bond length. This section is also terminated by a blank line. Line 10 onwards defines initial values for the geometry parameters (in this case the initial value for the O-O bond distance, R); again a blank line is used as a terminator.

The Pople input should be familiar to users of the popular Gaussian series of programs, and we have adhered to many of the standard input conventions used in Gaussian. Within PQS, the Pople style input is translated internally to PQS style, and a new file (with the extension `.pqs` - here `o2.pqs`) is written and used to run the job.

<p>NOTE: The Pople input style has been provided primarily as an aid to potential users who are familiar with Gaussian and similar packages that use this input style. We do not recommend its general use within PQS.</p>

The log file (the short output file) produced by running o2.com is:

```
=====
PQS  Ab Initio Program Package running on dirac
Date      : Tue Jan 11 10: 0:16 2011
Executable : /home/pqs1/PQsv40/INTEL64/pqs.x
Type       : ELF 64-bit LSB executable, AMD x86-64, version 1 (SYSV),
             for GNU/Linux 2.4.1, statically linked, not stripped
Intsize    : 1
=====
%MEM=3
# BPW91/6-311G(2df,2pd) OPT

Bond distance in triplet O2 by BPW91 and a good basis

O  3
O
O  1  R

R=1.2

Empirical Formula: O2

      Cartesian Coordinates in Standard Orientation

                Coordinates (Angstroms)
      ATOM          X          Y          Z
1  o          0.000000    0.000000   -0.600000
2  o          0.000000    0.000000    0.600000
Point Group: D*h   Number of degrees of freedom: 1

SCF Energy:  -149.467868275  iterations:  6  basis:          3-21g

Charge:  0.000000  Multiplicity:  3
Wavefunction:  UDFT          XC potential:  bpw91
Basis set:  6-311g-2df2pd
Number of contracted basis functions:  60

** Cycle  1  Energy  -150.369899116  RMSG  0.02997  RMSD  0.02857 **
** Cycle  2  Energy  -150.370423795  RMSG  0.00706  RMSD  0.00881 **
** Cycle  3  Energy  -150.370457186  RMSG  0.00048  RMSD  0.00064 **
** Cycle  4  Energy  -150.370457362  RMSG  0.00001  RMSD  0.00002 **

      CONVERGED GEOMETRY
      Cartesian Coordinates (Angstroms)
                X          Y          Z
o          0.0000000000000000    0.0000000000000000   -0.61005740412078
o          0.0000000000000000    0.0000000000000000    0.61005740412078

dipole/D =      0.000000  0.000000  0.000000  total=  0.000000
Expectation value of S**2:  2.0038500  Multiplicity:  3.0025650

Empirical Formula: O2

      Cartesian Coordinates in Standard Orientation
```

```

                Coordinates (Angstroms)
      ATOM          X          Y          Z
1   o             0.000000    0.000000   -0.610057
2   o             0.000000    0.000000    0.610057
Point Group: D*h   Number of degrees of freedom: 1

```

```

Charge: 0.000000 Multiplicity: 3
Wavefunction: UDFT                XC potential: bpw91
Basis set: 6-311g-2df2pd
Number of contracted basis functions: 60

```

```

Energy is: -150.370457362 au

```

```

dipole/D = 0.000000 0.000000 0.000000 total= 0.000000
Expectation value of S**2: 2.0038500 Multiplicity: 3.0025650

```

```

=====
Total master CPU time = 0.27 Elapsed = 0.27 min
Termination on Tue Jan 11 10: 0:32 2011
=====

```

The log file is similar to that produced for the first example, `water.inp`. The only major difference is that, as the system is not a closed-shell, the expectation value for $\langle S^2 \rangle$ and the corresponding multiplicity are printed out for an unrestricted wavefunction. As can be seen, the spin contamination is very small ($\langle S^2 \rangle$ should be exactly 2 for a triplet, compared to a calculated value of 2.00385). Note that, before the optimization is started using the requested 6-311G(2df,2pd) basis set, a preliminary SCF is done for 6 iterations only using the smaller 3-21G basis. This is a recommended procedure within PQS to provide a better initial wavefunction guess to start off the larger basis set calculation.

Running jobs

A detailed description on how to run jobs, in particular parallel jobs on Linux systems, is given later. The following lines serve only as an introduction. To run, say the input file `water.inp`, on a Linux/Unix system in the background on a single processor, go to the directory where the input file is located and type

```
pqs water &
```

at the Linux/Unix prompt. This assumes that PQS has been installed and correctly configured on your system (see below). For input extensions other than the default `.inp`, type the whole input file name, e.g., `pqs o2.com &`

On a Windows system, open a DOS window, or “Command prompt” (Start → Programs → Command Prompt), change directory first to your PQS installation directory (say, `D:\PQS`), and execute the script `setpqs.bat` (see below). This is achieved by typing at the DOS prompt (say `C:\>`)

```

C:\>D:
D:>cd PQS
D:\>setpqs

```

This needs to be done *once only* when the DOS window is opened. Now change to the directory which contains the input file (say C:\Myjobs) and run the job by typing at the DOS prompt (assumed to be still D:\> from above)

```
D:\>C:
C:\>cd myjobs
C:\>pqs water
```

Note that Linux/Unix is case sensitive while DOS/Windows is not. In both cases, the results will appear in the file `water.out`; the summary output is in `water.log`. These files can be viewed with a text editor (such as `vi` or `emacs` in Linux/Unix, or `Notepad` or `Wordpad` under Windows). The program also generates a number of internal files (e.g. `water.coord` or `water.basis`), both in the current directory and in the scratch directory. These are sometimes needed to continue a calculation. In the present case we can get rid of them by typing

```
tidy water
```

at the Linux/DOS prompt.

For a successful calculation, you should have the directory containing the `pqs` script (`pqs.bat` under Windows) in your *search path* (otherwise, you will have to explicitly specify its location, e.g., `/progs/PQS/pqs` or `C:\PQS\pqs`, not just `pqs` as above). You also need to set three *environmental variables*, and have a valid license file, `pqs_lic` in your PQS installation directory. The environmental variables tell the program the location (path) of the installation directory, the basis set library, and the scratch directory. In this order, they are `PQS_ROOT`, `PQS_BASDIR` and `PQS_SCRDIR`. The value contained in `PQS_ROOT` (`$PQS_ROOT` under Linux/Unix, `%PQS_ROOT%` under Windows/DOS) is the directory where the executable program, the scripts and the license file reside. `$PQS_BASDIR` (`%PQS_BASDIR%` under Windows) contains the basis set library. Finally, `$PQS_SCRDIR` (`%PQS_SCRDIR%`) is a scratch directory which stores temporary scratch files generated by the PQS program. On a Linux/Unix system, these variables can be set in your startup script, e.g., `.cshrc` in your home directory (if you are using the `csh` or `tcsh` program interpreters). On a Windows system, they can be set by executing the `setpqs.bat` script (see above). On both Windows and Linux/Unix, you can check if the environmental variables are defined by typing, e.g.

```
echo $PQS_BASDIR          (Linux/Unix)
echo %PQS_BASDIR%        (Windows)
```

On a Linux/Unix system, `PQS_ROOT` may be `/usr/local/share/PQS` or simply `/progs/PQS`. `PQS_BASDIR` is then `$PQS_ROOT/BASDIR`, i.e., `/progs/PQS/BASDIR` or `/usr/local/share/PQS/BASDIR`. The default scratch directory is `/scr/$USER`; e.g., if your login name is `smith`, it will be `/scr/smith`, but it can be any other directory. To set these variables under Linux/Unix using `csh/tcsh`, and add them to your path, include in your `.cshrc` file the following lines:

```
setenv PQS_ROOT /progs/PQS
setenv PQS_BASDIR $PQS_ROOT/BASDIR
setenv PQS_SCRDIR /scr/$USER
setenv PATH $PATH:$PQS_ROOT
```

If your shell is the Bourne shell or one of its derivatives, you need to make the analogous changes in your `.profile` file. The `pqs` script resides in the `$PQS_ROOT` directory (e.g., `/progs/PQS` or `/usr/local/share/PQS`), and we highly recommended that you use it every time you invoke the PQS program (as opposed to invoking the executables directly).

If you are running PQS for the first time, and your system does not have a valid license file, then the program will stop and print out a *lock code*. This is a fairly long string of characters and numbers. Send this lock code to PQS to obtain your temporary or permanent *license code*. This can be done by visiting our web site <http://www.pqs-chem.com>, and filling out an on-line form, or by mailing your lock code directly to sales@pqs-chem.com. Your license code will be emailed back to you. The license code, typically three sets of 8 to 10 digit numbers separated by one or more spaces, followed by the program name (`PQS serial` or `PQS parallel`) followed by further numbers and characters, must be put in the file `pqs_lic` in your PQS installation directory (`$PQS_ROOT`).

<p>**IMPORTANT** Once you have run PQS and generated your lock code DO NOT ATTEMPT TO RUN PQS AGAIN UNTIL YOU HAVE RECEIVED YOUR LICENSE CODE AND INSTALLED YOUR LICENSE FILE. Doing so may invalidate the lock code you have just generated and the program will not work properly.</p>

OVERVIEW OF PQS COMMANDS

The program structure is modular, each command module does a specific task, e.g., read input, set up basis set, construct initial MO guess, solve the SCF equations etc...

The modules communicate in two ways; through files or via a common storage area in memory (henceforth known as the "depository"). Some information is stored both on file and in the depository. Simple, commonly used, single item data are stored both in the memory and on the <control> file. Module specific and matrix (or vector) data are usually stored on a separate file (e.g., the molecular geometry is stored on the <coord> file, the gradient vector on the <grad> file etc...).

All data useful for either a restart or for potential use in another run with, e.g., a different basis set, are kept at the end of the job. These files can be archived into a *single* file using the script `arch`. Temporary, job-specific, files are deleted on job completion. The files generated by the various modules are described on page 82.

A summary of the program commands is given below.

MEMORY (%MEM)

Reserves virtual memory for the job (total available and incore/disk usage for integrals). If present, **this must be the very first line of the input**. If included later, it has no effect.

FILE

This command changes the scratch directory (if different from the default). It can also take files (e.g., geometry, molecular orbitals) from another calculation with a different file name.

CPU

Sets computer parameters, like cache size, integer size, accuracy of double-precision floating point numbers, and the amount of *real* (as contrasted with virtual) memory.

GEOM

Reads in the input geometry (in various formats, either as Cartesian coordinates or a Z-matrix), determines point group symmetry, orients the molecule, and calculates geometrical parameters.

BASIS

Sets up the basis set (including ECPs), either from the basis set library, an optional file, or directly from the input. It is also possible to augment an existing basis sets with extra basis functions.

GUESS

Generates the initial SCF guess. Invoked automatically in most cases and needed only to override the default or in special cases, e.g., UHF open shell singlets.

INTE

Sets up thresholds for integral evaluation. Needed only in special cases.

SCF

Solves *ab initio* SCF equations for closed-shell restricted and open-shell unrestricted wavefunctions. Does standard Hartree-Fock (HF) plus a range of DFT functionals.

FORCE

Calculates the forces on the nuclei (negative gradient or first derivative) for *ab initio* SCF and MP2 wavefunctions. Note that the MP2 gradient code is preliminary and is currently serial only.

NUMGrad

Calculates the gradient by central-difference on the energy.

NUMHess

Calculates the Hessian matrix by central-difference on analytical gradients.

HESS

Calculates the Hessian matrix analytically. Available for all SCF wavefunctions (closed and open-shell, HF and all DFT functionals).

NUMPol

Calculates polarizability (and optionally dipole and polarizability derivatives) by finite-difference on the energy/analytical gradients in an external field.

FREQ

Does vibrational and thermodynamic analysis, using the Hessian matrix produced by the HESS or NUMHESS modules.

SQM

Does SQM scaling on the Hessian matrix prior to a vibrational and thermodynamic analysis using standard or user-defined scale factors

NMR

Calculates NMR chemical shifts. Also activates VCD rotational strengths.

MP2

Canonical MP2 energy and wavefunction.

CORR

Post Hartree-Fock single-point energies (closed-shell only).

POP

Mulliken and Löwdin population analysis program. Includes CHELP and Cioslowski atomic charges.

NBO

F. Weinhold's Natural Bond Orbital analysis (NBO version 5.0).

PROP

Preliminary properties package. Computes charge and spin-density at the nucleus and the electric field gradient.

COSMO

Klamt's Conductor-like Screening solvation model (COSMO).

SEMI

Calculates energy *and* gradient for semiempirical wavefunctions. Includes MINDO/3, MNDO, AM1 and PM3.

FFLD

Calculates energy and gradient (and optionally the Hessian) for molecular mechanics force fields. Currently the Sybyl 5.2 and Universal force fields are available.

OPTimize

Geometry optimization.

CLEAn

Removes files associated with a geometry optimization.

DYNAMics

Direct Newtonian molecular dynamics.

QMMM

General QM/MM energies and gradients using Morokuma's ONIOM method.

SCAN

Potential scan, including scan + optimization.

PATH

Follows a reaction path downhill from a transition state. Path can be defined in Z-Matrix coordinates, Cartesian coordinates or mass weighted Cartesians.

JUMP

Jumps *back* to the next jump target in the input file for iterative loops (e.g., geometry optimization, reaction path, etc...). It can be used with an optional integer argument, e.g., **JUMP 5**, requesting a jump back 5 cards (lines).

DETAILED DESCRIPTION OF THE PQS INPUT FILE AND KEYWORDS

The following typographical convention is used: keywords in **BOLD FACE CAPITALS** must be typed as shown, with the proviso that they are not case sensitive, **and only their first 4 characters are significant** (although more characters can be added to facilitate reading). E.g., the following forms of the **FORCE** keyword (a command name) are equivalent: **FORC FORCES force** or **Force**. To emphasize this point, the first 4 characters of a keyword will be printed in **BOLD CAPITAL** letters throughout this document, although this is not necessary in the actual input. A line must be shorter than 300 characters. **All keywords corresponding to program steps must start in the first column on each line with at least one blank space between all keywords on the same line.** A question mark or an exclamation mark in column 1 of a line renders the whole line a comment line; it has no effect on the computation. An exclamation mark anywhere on a line makes the rest of the line, beyond the exclamation mark, a comment (Fortran style). This is convenient to add comments to the input, or to temporarily suppress some input options without removing them permanently. Unknown commands are considered comments and are printed in the output but not processed.

Text in angle brackets <...> requires the substitution of an appropriate text string or value. E.g. <command> represents any of the valid commands, <basisname> represents a valid basis set name, <integer> is an integer number, <string> is an arbitrary string etc...

Optional input is set in square brackets. E.g. **[THREs=<thr1>[,<thr2>]]** means that the whole construct is optional (because of the outer square brackets). Here <thr1> and <thr2> are user-defined floating-point values. If this option is present, it can have any of the following forms:

THRE=9.5 or **threshold=10** or **Threshold=10.4** or **THRE=9,7** or **THRE=(10.5,7)**

This last form shows that if several numerical parameters belong to a single keyword, then they can be enclosed in parentheses, separated by commas.

The general format of a command line is a command name, followed optionally by a set of options; the command name and the options are all separated by one or more blanks:

<command> [option] [option] [option]....

In a few cases (e.g. **GEOM, BASISs**), the form of the command is

<command>=<value> [option] [option]

For example **SCF ITER=15 THRE=4** instructs the program to perform an SCF iteration, with the maximum number of iterations set to 15 (instead of the default 50), and the SCF threshold set to 1.0E-4 (instead of the default which is 1.0E-5).

BASIs=6-31G* NEXT instructs the program to use the 6-31G* basis, augmented with additional basis functions. The latter are defined below the **BASIS** command, and may be extra polarization functions or a different basis set on specific atoms.

Options can have the following forms:

<keyword>

<keyword>=<integer> or **<integer>,<integer>** or **<integer>,<integer>,<integer>**

<keyword>=<real> or **<real>,<real>** or **<real>,<real>,<real>**

<keyword>=<character string>

They begin with a keyword, the first 4 characters of which are significant. The simplest (logical) options consist of the keyword only. More complex options set a numerical (integer or real) value or several (up to 3) numerical values, or a string value. There should be no blanks within an option. As mentioned above, if 2 or 3 numerical values are set in an option, they can be enclosed in parentheses.

Currently we have the following reserved words for program steps (some quite trivial). Most will be discussed separately below, except **TITLE**, **TEXT** and **STOP**.

%MEM	requests memory
TITLE=<title>	defines a title
FILE	defines archive and scratch files
CPU	defines computer parameters
TEXT=<arbitrary text>	prints text
GEOMetry	molecular geometry and symmetry
BASIs	basis set
GUESs	SCF guess
INTEgrals	parameters for integral computation
SCF	SCF iteration
FORCe	gradient evaluation
NUMGrad	numerical gradient evaluation
NUMHess	numerical Hessian calculation
HESS	analytical Hessian calculation (closed-shell only)
NUMPolar	numerical polarizabilities and polarizability derivatives
FREQuency	vibrational frequencies
SQM	SQM scaling of the Hessian matrix
NMR	NMR chemical shieldings (+ VCD rotational strengths)
MP2	canonical MP2 energies
CORR	post Hartree-Fock closed-shell single-point energies
POP	population analysis
NBO	Weinhold's natural bond order analysis
PROP	properties computed at the nucleus
COSMo	Klamt's conductor-like screening solvation model
SEMI	semiempirical energy and gradient
FFLD	molecular mechanics energy, gradient and Hessian
OPTImize	geometry optimization step
CLEAn	removes files associated with geometry optimization
DYNAmics	direct classical molecular dynamics
QMMM	general QM/MM using ONIOM method
SCAN	potential scan step
PATH	reaction path step

JUMP
STOP

go back in the program unless a condition is satisfied
instructs the program to stop

Some of the more input-intensive steps, in particular **GEOM**, **BASIs** and **OPTimize** can be followed by further input information, either in the input itself or in a separate file. E.g., the nuclear positions for **GEOM**, the basis set for **BASIs**, constraints for **OPTimize**. In general, the extra information can also be read from a file. This is usually shown by the **FILE=<filename>** option. This feature simplifies the input file and is often useful, e.g., when a customized basis set is shared by several input files, or for large molecules where the geometry data take up too much space.

In the following, the options and additional input for each program step are described:

1. %MEM command

Controls the amount of core memory (in 8-byte double words or in megawords, MW, or alternatively with a specific unit, MB or GB, given) requested for the job.

Usage: **%MEM**=<integer>{unit}
(Alternative form **MEMO**=<integer>{unit})

The default (if no %MEM card is present) is 6 MW=6,000,000 double words=48 MB.

If no unit is specified and <integer> is small (<2000) it will be assumed to be in megawords (e.g., 9 will be interpreted as 9,000,000 double words); if <integer> is large (≥2000), it is interpreted as words.

An alternative to avoid any confusion is to give a specific unit, either MB or GB. Thus **%MEM=2GB** requests 2 GB of memory. Note that there must be NO spaces between the number and the unit. We recommend that a unit be given; however the old method has been retained for backwards compatibility.

There are two options which control the amount of storage available for the in-core and disk storage of integrals (for use in semi-direct SCF)

Options: **CORE**=<integer>{unit} **DISK**=<integer>{unit}

Thus **%MEM=240MB CORE=640MB** requests 240 MB for the main program plus an *additional* 640 MB of core memory exclusively to store integrals.

Unlike physical memory, the units for disk storage if none are given are MB *not* MW. Thus **DISK=1000** requests 1000 MB (i.e., 1 GB) of disk space for integral storage. Note that at least 100 MB of disk storage *must* be requested with this option, i.e., **DISK must** be at least 100. If it is *less* than this, the command will be ignored.

For parallel jobs, the memory requested is the *request per CPU*. Thus if you have a machine with multiple cores (CPUs) per processor (virtually all of them at the time of writing) and you specify **%MEM=4GB** and are running 8 CPUs per node, then the total memory demand on each node will be 8 x 4 = 32 GB. Make sure you have sufficient resources on your computer, i.e., enough RAM and swap space, to meet the request..

****IMPORTANT**** From a practical point of view, when writing to disk data does *not* go direct to the hard drive but into an I/O buffer. Only when the buffer is full is the data physically written to the disk. PQS does not specify any buffer size when files are opened and the default under Linux is to keep expanding the buffer until no more physical memory is available. As the I/O buffer resides in physical memory, storing integrals “on disk” is essentially equivalent to storing them in “in-core” memory provided the buffer size is not exceeded. Once this happens, real disk access occurs and, because I/O speed is very much less than CPU speed, the program slows down significantly. Consequently, semi-direct SCF calculations that request *more* disk storage than the available memory are nearly always slower than the same job ran fully direct.

NOTE: When requesting memory you should clearly be aware of the configuration of your machine, i.e., how much RAM you have and how much swap space has been configured on your system. The absolute maximum of memory a program can request is the sum of RAM and swap space, which is the maximum allowed by the operating system. E.g., with 2 GB of RAM and 2.5 GB swap space per CPU (typical for the nodes on PQS machines), you have a total of 4.5 GB=562.5 MW per CPU. This means that two similar processes, both running on the same CPU, can each have 281 MW. Of course, a single program cannot use all the available memory as continuously running service programs also need some memory. In a parallel job, the slaves as well as the master need memory. This means that if the master computer runs the master process and 4 slaves, and the memory demand on the master is 10 MW then the total memory demand will be 50 MW. In addition, the slaves also allocate any in-core integral storage space specified. (In practice, the memory demand on the master process is usually *less* than on the slaves, which after all are supposed to do most of the work.)

If the master node cannot accommodate the master process as well as the slave processes it is running because of restrictions on the total memory available, consider running less slave processes on the master node. This can be done by asking for fewer processes than the number available when you submit your job (see RUNNING PQS IN PARALLEL). E.g., on an 8-CPU node run only 7 (or less) slaves on the master node.

Requesting more memory than needed does not confer any advantage, and, if extreme amounts of memory are requested, may prevent other jobs from running, but it usually does no harm if used within reasonable limits.

It is often difficult for the first-time user of PQS to know how much memory is needed for a given job; this will depend on the size of the system (number of atoms), the basis set and the methodology used. The following table gives the high water mark - which is the maximum amount of dynamically allocated memory used (in double words – to convert to MB multiply by 8 and divide by 1 000 000) for a number of different job steps and should serve as a guide as to how much memory to actually request via %MEM.

System	Job	Natoms	Nbasis	High water	
Octanol	OLYP/TZVP OPT	27	279	1096030	SCF
				1588058	FORCE
Sucrose	RHF/6-31G** FREQ	45	455	2321453	SCF
				44239840	HESS
Taxol	BVWN/3-21G E+NMR	113	660	5160937	SCF
				6004102	NMR
C ₅₄ H ₁₈	B97/6-31G* E+NMR	72	846	8390317	SCF
				9867618	NMR
Yohimbine	OLYP/PC-2 E only FTC+SEMI	52	1144	48297184	SCF
Chlorophyll	B3LYP/vdzp-ahlrichs E+G	137	1266	19594892	SCF
				19594892	FORCE
Calix[4]arene	RHF/cc-pvtz	68	1528	26882729	SCF

Some jobs, for example MP2, will allocate essentially the full memory request, so the high water mark will simply be the amount of memory available for the job.

2. FILE command

In most cases this command can be omitted as the defaults are usually appropriate. However, in some cases, notably for restarting calculations, it may be needed. **Note that the root filename for all files associated with a particular job is that of the input file.**

Options: **CHK**=<string> **SCR**=<string>

E.g. `FILE CHK=<path and file basename> SCR=<scratch directory path>`

If the **CHK**=<filename> option is defined, the program copies the files from a previous calculation (<filename>.control, <filename>.coord, <filename>.mos etc...) to the current jobname (<jobname>.control, <jobname>.coord, <jobname>.mos). The main use of this is to start a calculation with the coordinates, molecular orbitals, and other data determined in a previous run, e.g., with a smaller basis. For example, the command

```
FILE  CHK=C60-3-21G
```

in the input deck C60-6-311G.inp will cause the files (e.g., C60-3-21G.mos) of the previous small calculation to be copied to the current jobname (C60-6-311G.mos). This is useful to, e.g., get starting orbitals. Note that this example assumes that all files are in the current working directory; if this is not the case, the full file path must be given.

The **SCR** option redefines the scratch directory (in which all temporary runtime files are stored). If it is not given, the scratch directory is taken from the environmental variable **PQS_SCRDIR**. The script **pqs** sets this variable to `/scr/$USER`, i.e., to `/scr/guest` for user `guest`. **This directory must exist.** There is normally no need to include this command unless scratch file locations different from the default are desired.

3. CPU command

The CPU command allows the redefinition of certain machine parameters to fit the actual hardware being used. **Most of its options are seldom necessary, the only exception being MEMR for correlated calculations under Windows.** The program assumes certain plausible values for the physical characteristics of the computer and the operating system/compiler used, and tries to determine others. E.g., on Linux systems, the program is able to determine the actual amount of *real memory* (as opposed to *virtual memory*). For a running program, virtual and real memory appear identical. However, accessing real memory takes tens of nanoseconds, while accessing virtual memory takes milliseconds, so the distinction is important for programs which must handle large amounts of data, like traditional correlation programs.

Options: **INTS**=<integer> **ACCU**=<integer> **CACHe**=<integer> **MEMR**=<integer>
DOUB=<integer>

MEMR is the size of the *real memory* in megawords (1 million 8-byte words). If your computer has 256 MB memory, then **MEMR** should be set to 32 or slightly less. Telling the program that it has more physical memory than it actually has can lead to severe performance degradation in several job steps, e.g., MP2. This is the only **CPU** option frequently used. Note that it is not needed on Linux systems, as the program can determine the size of the physical memory.

INTS is the number of integers in a double-precision word. It is 2 for 32-bit systems and 1 for 64-bit systems.

ACCU is the accuracy of double-precision words, more precisely the value $-\log \epsilon$ so that $1.0d0+\epsilon > 1.0d0$ in Fortran. It is approximately 15, i.e., ϵ is about $1.0d-15$ for 64-bit systems.

CACHe is the cache size *in 8-byte words*.

DOUBle=size of a double-precision word in bytes. It is 8 on almost all computers in use today. Do not alter this value on the current generation of computers.

4. GEOM command

Options: **GEOM**=<string> **FILE**=<string> **BOHR SYMM**=<real> **AXES GEOP D2HS**
CHARGE=<integer> **MULT**=<integer> **NOORient NOCM PRINT**=<integer>
FIELD=<real>,<real>,<real>

GEOM: gives the style of the geometry input. **NUCL** is a synonym for **GEOM**. **GEOM** alone, with no option specified, is equivalent to **GEOM=READ**. For compatibility with earlier versions of the PQS software, the geometry style can also be defined separately as **STYLE**=<string>. The following input styles are defined:

- ◇ **READ**: reads the geometry from an existing *coord* file (see later)
- ◇ **TX90**: old TX90: (2X,A8,F10.2,3F10.6) gives name, atomic number(real!), x, y, z
e.g. N=C 6.0 0.0 1.123 -.975266
- ◇ **PQS**: standard input format: symbol, x, y, z, (atomic number(real!), atomic mass)
e.g. C 0.0 1.123 -.975266 8.0 13.003355

The last two fields are optional and a default atomic mass will be provided. This format also allows different molecules/groups of atoms to be defined by inserting “\$molecule” as a designator, e.g.

```
H 3.831 -6.435 -0.145
H 3.374 -7.746 -0.163
$molecule
H 4.444 -0.057 1.298
H 3.528 0.026 0.256
```

to designate two hydrogen molecules. Separate molecules/structures need to be defined in order to carry out cluster/surface optimizations or for QM/MM (see later)

- ◇ **PDB**: Protein DataBase format
- ◇ **MOP**: MOPAC Z-matrix format
- ◇ **CAR**: Biosym *.car file. Note that *.car files *must* be in a separate file and *cannot* be included in the input stream
e.g. GEOM=CAR file=molecule.car
- ◇ **MOL**: MDL *.mol file. Note that *.mol files *must* be in a separate file and *cannot* be included in the input stream
e.g. GEOM=MOL file=molecule.mol
- ◇ **ZMAT**: Gaussian Z-matrix
- ◇ **HIN**: Hyperchem input (several structures are possible)

Atomic Symbols

The program stores up to 8 characters for the atomic symbol. For a real atom, the first two (or just the first for a single symbol atom) must be those of a genuine atom in the periodic table. Dummy atoms (see below) should begin with the symbols ‘x’ (NOT xe, which will be taken as xenon), ‘q’ or ‘du’. Atoms can be numbered (numbers will be ignored). Additional symbols other than numbers are considered as “special symbols”

and are used to set different basis sets on different types of the *same* atom. Dummy atoms (atoms not carrying basis sets) and ghost atoms (atoms without nuclear charge carrying basis sets) are discussed further at the end of this section.

FILE=<filename>: take the molecular geometry from this file. This is used in conjunction with various **GEOM** options.

BOHR: the coordinates are given in atomic units. Note that atomic units are used internally by the program, and in its internal files, e.g., *coord*.

CHARGE=<integer>: total molecular charge

MULTiplicity=<integer>: multiplicity. (1 - singlet, 2 - doublet, 3 - triplet etc...)

The default charge and multiplicity, if these keywords are not present, is 0 and 1, respectively (corresponding to an uncharged closed-shell singlet).

SYMM=<symmetry threshold>: This is used to symmetrize a molecule whose coordinates are not exactly symmetrical. The default is 10^{-5} bohr. If, after a symmetry operation, the coordinates of corresponding nuclei coincide within this margin, it is assumed that the molecule is symmetrical but numerical errors (e.g., in a force field optimization) obscure the symmetry. Exact symmetry is subsequently enforced. This feature is also useful if the molecular symmetry is violated during a geometry optimization, due, e.g., to numerical errors in the gradient. To switch off symmetry during a calculation, specify **SYMM=0** or **SYMM=0.0**. Many modeling programs have fairly large errors in the optimized geometry, requiring a symmetry threshold as large as 0.1 (Bohr). A too large threshold will confuse the symmetrizer.

D2HS: An older symmetry algorithm for Abelian point group symmetry only. It can sometimes find symmetry which the default symmetrizer misses. It is possible to use both in succession, as in

```
GEOM=MOPAC  FILE=molecule.mop  D2HS  SYMM=0.3
GEOM  SYMM=0.1
```

AXES: causes the program to calculate the principal axes of inertia. It would transform the molecule to the principal axis system before symmetrization. This was sometimes useful but was removed when the new symmetry algorithm was introduced.

GEOP: prints out all *bonded* interatomic distances, all bond angles and all proper dihedral angles. This is much more useful than the indiscriminate printing of all bond distances and angles in some programs, as the latter grows with the square and cube of the number of atoms, and leads to much unnecessary printout. Equivalent to **PRINT=3** (see below).

NOORient: suppresses the symmetry orientation of the molecule. The latter may lead to an interchange of coordinate axes if the molecular symmetry changes, e.g., during the calculation of a numerical Hessian.

NOCM: suppresses the shifting of the center of mass to the origin.

FIELD=<real>,<real>,<real>: applies an external electric field of the value given (in atomic units) along the X, Y and Z axes, respectively.

PRINT=<integer>: controls the amount of printout (larger integer - more printout).

Dummy Atoms

Dummy atoms are used to mimic the effects of an applied field (by defining point charges) and - for dummy atoms with no charge - to calculate properties (currently only the chemical shift) at particular points in space.

Consider the following input

```
TEXT=    Water with dummy atoms
GEOM=PQS
O1      0.0      0.0     -0.405840
H2     -0.793353  0.0      0.202920
H3      0.793353  0.0      0.202920
X       0.0      0.0     10.000000   -1.00
X       0.0      0.0    -10.000000    1.00
X       0.50     0.50     0.50
```

This defines point charges along the z-axis at a distance of ± 10 Å to mimic the effects of an applied field. It also assigns a dummy centre at (0.5, 0.5, 0.5) at which point a chemical shift will be calculated.

Note the order of the atomic centres here. All real atoms come first, followed by all charged dummy atoms, followed by all uncharged dummy atoms. You can give your input in any order, but it will be reordered internally by the program to the ordering shown. This is done for ease of symmetry recognition and for geometry optimization, if requested.

Charged dummy atoms are included when determining the overall molecular symmetry (the applied field may break symmetry) but uncharged dummies are ignored. For geometry optimization and vibrational frequencies, *all* dummy atoms are ignored. You can optimize molecular geometries and compute vibrational frequencies in the presence of an applied field. Note that, for symmetry purposes, all charged dummy atoms with the same symbol (e.g. “x”) will be considered as the *same* type of “atom” and if the charges are different, they may be flagged as symmetry-breaking, and the program will stop. To avoid this, and still use symmetry, differently charged dummy atoms should be given different symbols. (See RUNNING JOBS, examples 9 and 10).

Note that dummy atoms for charges are deprecated, having now been essentially superseded by the **FIELD** option. The two methods should give very similar results for all computed properties except the energy (which includes additional interactions between the charges if point charges are used).

Ghost Atoms

Ghost atoms are “real” atoms for which the atomic charge has been set to zero. The ghost atom will be assigned its usual complement of basis functions, but with a zero charge, there will be no real atom there. In this way, basis functions can be centered at points in space, e.g., to take account of basis set superposition error.

For example

```
01      0.0      0.0     -0.405840     0.0
```

will assign a zero charge to the “oxygen” atom, keeping all its basis functions.

Ghost atoms are included when determining the molecular symmetry and will be recognized during a geometry optimization. The dummy centre will “move”, and its position will be optimized with respect to the real atoms (whatever this means?)

Dummy and ghost atoms may only be input using **TX90** or **PQS** formats, or from a *coord* file. Any other file format may be converted to Cartesians by running PQS with the **GEOM** command only. The file *coord* will contain the geometry, in standard format, *in Bohr* units. This can be augmented with dummy or ghost atoms and read in using the command `GEOM` or with `GEOM=PQS BOHR FILE=<coord-file>`. The **GEOM** command (without an option) reads the geometry from the file *coord* in Bohr units.

5. BASIS command

This command may be optionally followed by an equal sign (=) and a legal basis set name.

Options: **FILE=<string>** **NEXT** **GENERAL** **DUMMY** **PRINT**

Some basis sets are built into the program itself to facilitate testing. However, the program takes all standard basis sets from a basis set library if it finds one. The location of the basis set library is determined from the environmental variable **PQS_BASDIR** (default: `$PQS_ROOT/BASDIR` under Linux or `%PQS_ROOT%\BASDIR` under Windows). A number of basis sets are available in several formats, including the TEXAS/TX90/TX93 format used by PQS, on the PNNL Website at <http://www.emsl.pnl.gov:2080/forms/basisform.html>. Most of the basis sets in the basis set directory have been taken from this site. Note that we have corrected several minor errors and omissions, and changed the format of exponents and contraction coefficients to ensure higher accuracy. We acknowledge the public service of PNNL for supporting this basis library.

Most basis sets have a set of exponents and a set of contraction coefficients for each shell type (i.e., S, P, D, F shells etc...). Some basis sets group certain S and P shells together (i.e., have the same exponent for both the S and P functions) into what is called an L shell. Many of the Pople-type basis sets are of this form. Other basis sets group several primitive shells of the same type together into more than one contracted basis function, i.e., the exponents are the same for each individual function, but the contraction coefficients are different. These are known as generally contracted basis sets. Note that by default generally contracted basis sets are separated into their individual contracted functions inside the *PQS* program unless it is specifically requested that this not be done (see later).

The most important basis sets available in the library are:

Pople-Type Basis Sets

STO-2G	minimal	H-Ca,Sr
STO-3G	minimal	H-Sr,Te
STO-6G	minimal	H-Kr
3-21G	split-valence	H-Sr,Te
4-21G	split-valence	H,B,C,N,O,F
4-22G	split-valence	H-Ar
4-31G	split valence	H-Ne,P,S,Cl
6-31G	split-valence	H-Kr
m6-31G	improved 6-31G for transition metals	H-Kr
6-311G	valence triple-zeta	H-Ca,Ga,Ge,As,Br,Kr

These can be supplemented with polarization and diffuse functions, e.g. m6-31G*, 6-311+G**, or similarly in the (d,p) notation: m6-31G(d), 6-31+G(d,p), 6-311G(d).

Dunning Correlation-Consistent Basis Sets

cc-pVDZ	polarized valence double-zeta	H,He,B-Ne,Al-Ar
cc-pVTZ	polarized valence triple-zeta	H,He,B-Ne,Al-Ar
cc-pVQZ	polarized valence quadruple-zeta	H-Ar,Ga-Kr
cc-pV5Z	polarized valence quintuple-zeta	H-Ar,Ga-Kr
cc-pV6Z	polarized valence sextuple-zeta	H,B-Ne
cc-pCVDZ	polarized core/valence double-zeta	B-Ne
cc-pCVTZ	polarized core/valence triple-zeta	B-Ne
cc-pCVQZ	polarized core/valence quadruple-zeta	B-Ne
cc-pCV5Z	polarized core/valence quintuple-zeta	B-Ne

All the above basis sets are available with additional diffuse functions as, e.g., aug-cc-pVDZ etc... (the aug-cc-pV6Z basis is only available for H,B,C,N,O).

Other Basis Sets

svp_ahlrchs	Ahlrchs polarized split-valence	H-Kr
vdz_ahlrchs	Ahlrchs valence double-zeta	H-Kr
vdzp_ahlrchs	Ahlrchs polarized valence double-zeta	H-Kr
dz_ahlrchs	Ahlrchs double-zeta	H-Kr
dzp_ahlrchs	Ahlrchs polarized double zeta	H-Kr
tzv_ahlrchs	Ahlrchs triple zeta valence (1994)	H-Kr
dz_dunning	Dunning double-zeta	H,Li,B-Ne,Al-Cl
dzp_dunning	Dunning polarized double-zeta	H,Li,B-Ne,Al-Cl
tz_dunning	Dunning triple-zeta	H,Li-Ne
dzvp-dft*	polarized valence double-zeta	H-Xe
dzvp2-dft*	double-polarized valence double-zeta	H-F,Al-Ar,Sc-Zn
vtz_gamess	valence triple-zeta from GAMESS	H,Be-Ar
midi	Huzinaga valence double-zeta	H-Na,Al-Ar,K,Cs
midi1	ditto + polarization (NOT for C)	H,C-F,Si-Cl,Br,I
mini	Huzinaga minimal	H-Ca
mini-sc	ditto, but rescaled exponents	H-Ca
pc-n (n=0-4)	Jensen polarization-consistent	H,C-F,Si-Cl
aug-pc-n (n=0-4)	ditto, but with extra diffuse functions	H,C-F,Si-Cl

This list is not exhaustive. The two basis sets marked with an asterisk (*) were specifically optimized for DFT wavefunctions (unlike most of the others which were optimized for basic Hartree-Fock).

To use any of the basis sets listed above, simply specify `BASIS=<basis_set_name>` where `<basis_set_name>` is one of the names given above, E.g., `BASIS=cc-pVTZ`, or `BASIS=vdzp_ahlrchs`.

There are additional basis sets in the EXTRA subdirectory, and extra polarization sets in the POL subdirectory. There is a file `Table` that lists the filenames of the root basis sets, and a file `Symbols` which describes all the basis sets in more detail.

In addition to the default basis library, basis sets can be read in from files using the **FILE** option. E.g., `BASIS FILE=mybasis.bas` will read the basis from the named file.

Basis sets, both standard and via the **FILE** option, can be augmented via the **NEXT** command

```
BASIS=6-311G NEXT
FOR      C
D        0.8
FOR      O
D        0.8
F        0.9
```

This input will take the standard 6-311G basis for carbon and oxygen, and add to it a 5d polarization function on C with exponent 0.8, and 5-component *d* and 7-component *f* polarization functions on O with exponents 0.8 and 0.9, respectively.

Different basis sets for the *same* atom type can be handled by specifying a “special character” (!@#%\$%^&*+=<>?) on the atomic symbol during the **GEOM** input.

```
%MEM=1 core=2
TEXT= Water with different basis set on each H
GEOM=PQS
O1    0.0      0.0    -0.405840
H2   -0.793353 0.0     0.202920
H3$  0.793353 0.0     0.202920
BASIS=6-31G* NEXT
FOR      H$      BASIS=3-21G
GUESS
```

This input will perform a calculation on water with the 6-31G* basis set on O and one of the H atoms, and the 3-21G basis on the other H atom. When assigning the original basis set, only atoms without “special symbols” will be recognized (numbers are ignored). Thus only atoms O1 and H2 will be given a basis. The symbol “H3\$” is interpreted as “H\$” and will *not* be recognized in the standard basis. A full basis for the “special symbol” atoms must then be given, as in the example above. This feature is often very useful for NMR chemical shift calculations, using Chesnut’s attenuated basis method [1] (i.e., using smaller basis sets for other atoms than for the atom of interest).

Note that if the **NEXT** command is used both with an extra basis set *and* with extra basis functions, e.g.,

```
FOR      C$      BASIS=6-311G*
FOR      C$
F        0.8
```

the extra basis functions (F in this case) *must* be given *after* the basis set. The opposite

```
FOR      C$
F        0.8
FOR      C$      BASIS=6-311G*
```

will *not* work.

****IMPORTANT**** Unlike most other input keywords, basis set input is FORMATTED. The general rule is that any field (whether number or character string) occupies 10 columns with character strings starting on the *first* column of the field. Thus the string "FOR C" indicating the start of a basis input for carbon must have "FOR" starting in column 1 with the "C" starting in column 11. A detailed specification of the basis set format is given below.

GENERal: The default for generally contracted basis sets is to separate the different contractions involving the same primitive shells into individual contracted shells. In theory there should be efficiency gains if those shells containing the same primitives (same exponent) are treated together, but in practice this is only slight and several modules (e.g., the **CORR** module for correlated wavefunctions) cannot handle general contractions. If this keyword is specified general contractions will not be separated.

DUMMy: Atoms which pass through the **BASIS** module without being assigned any basis functions will be flagged, and the program will stop. If you genuinely desire that an atomic centre be given no basis functions (just a point charge, or dummy atom) you should add this keyword.

To add a GHOST atom (i.e., an atom with no charge but with basis functions, e.g., for an estimate of the basis set superposition error) the atomic charge should be set to zero in the **GEOM** module.

PRINT: prints full details of the basis set

Format For Basis Set Specification

This information is only necessary if non-standard basis sets are used. The basis set information is **formatted**. Each piece of data occupies a *field* 10 characters long, i.e., the first field occupies columns 1-10, the second columns 11-20, etc...

The basis set data for a particular atom is preceded by a line containing the keyword **FOR** in columns 1-3, and the atom name in columns 11-18. This is followed by the data for each primitive shell: type, exponent, and contraction coefficients (format (A3,7X,10F10.5)). A non-blank shell type signals the beginning of a new contraction, or, if it is not one of the legal basis function types, the end of the basis set data. The first field, the shell *type* should be blank for all primitive shells, except for the very first shell in the contraction. The following shell types are available:

blank signals the continuation of a contraction, with the same shell type

S s type

P p type

L sp type, i.e., a set of s and p functions sharing the same exponents but different contraction coefficients

D spherical harmonic (5 component) d functions

D6 Cartesian (6 component) d functions

F spherical harmonic (7 component) f functions

F10 Cartesian (10 component) f functions

G spherical harmonic (9 component) g functions

- G15** Cartesian (15 component) *g* functions
- H** spherical harmonic (11 component) *h* functions
- H21** Cartesian (21 component) *h* functions
- I28** Cartesian (28 component) *i* functions

The next piece of data is the exponent in atomic units (a_0^{-2}). It is followed by up to 9 contraction coefficients, starting in columns 21,31,... 101. A single contraction coefficient with a value of 1.0 can be omitted. For **L** type functions, two contraction coefficients are given: the first one for the **S** functions, and the next for the **P** functions. For the usual *segmented* contractions, only a single contraction coefficient is given, unless the basis function is of **L** type. If there is more than one non-zero contraction coefficient, and the function type is not **L**, it is assumed that general (Raffenetti) contractions are employed [2]. In the general contraction scheme, more than one contracted basis function is formed from the same set of primitive functions. Their most prominent representatives are the *correlation consistent (cc)* basis sets of Dunning and coworkers (see the references in your <PQS_BASDIR>/Symbols file). Atomic natural orbitals (ANOs) are also of this type but they are, in general, very inefficient.

Examples

FOR	C				
S	172.256	.0617669			
	25.9109	.358794			
	5.53335	.700713			
L	3.66498	-.395897	.23646		
	.770545	1.21584	.860619		
L	.195857				

This is the 3-21G basis for carbon. Its first contraction consists of 3 primitive s functions with exponents 172.256, 25.9109 and 5.53335, contracted to a single 1s core function. The next contraction consists of two *sp* (L) type shells (exponents 3.66498 and 0.770545). The contraction coefficients for the s type orbital are -0.395897 and 1.21584; the *p* contraction coefficients are 0.23646 and 0.860619. These form the inner part of the 2s and 2p orbitals. The last shell is an uncontracted *sp* function with exponent 0.195857. According to the rules of Fortran, the numbers must fit in their fields (10 character long) but they can be shifted right and left within the field. They are lined up in the above example but this is optional. Numbers in exponential format, e.g., 3.1763E5 are permitted. However, in this case the numbers must be *right justified*, i.e., the characters 'E5' must occupy columns 19 and 20.

For a general contraction:

FOR	N				
S	45840.0000	0.000092	-0.000020		
	6868.00000	0.000717	-0.000159		
	1563.00000	0.003749	-0.000824		
	442.400000	0.015532	-0.003478		
	144.300000	0.053146	-0.011966		
	52.180000	0.146787	-0.035388		
	20.340000	0.304663	-0.080077		

	8.381000	0.397684	-0.146722
	3.529000	0.217641	-0.116360
S	1.428000		
S	0.554700		
S	0.206700		

This is the s part of the cc-pVQZ (correlation-consistent polarized valence quadruple zeta) basis for nitrogen. This basis employs general contraction in the 1s-2s region.

Effective Core Potentials

Effective Core Potentials (ECPs) can be used to model the effect of core electrons for heavy atoms (typically fourth-row or higher, but can be used even for second-row elements) in order to reduce the complexity of the calculation. They are often constructed on the basis of relativistic calculations, and can thus be seen as an indirect way of introducing relativistic effects.

ECPs (aka pseudopotentials) usually fall into two categories: “large core”, which include in the definition of the core all but the outermost electrons, and “small core”, which leave the two outermost electronic shells in the valence space. The latter generally provide better results, but are more expensive to use, due to the larger number of electrons that are left to be treated explicitly.

To use pseudopotentials in a PQS run, simply use one of the ECP library basis sets provided, or explicitly add a pseudopotential specification to the basis set input using the NEXT option, as described below. ECPs are supported in almost every computational step: energy, gradient, analytical and numerical Hessian, and NMR, for Hartree-Fock and DFT wavefunctions, as well as energy, gradient, and numerical Hessian at the MP2 level.

When ECPs are to be used in the calculation, the output file will contain a summary description of the pseudopotentials involved, located immediately after the basis set specification. Information is provided on the number of electrons that are to be simulated by pseudopotentials, the number of ECPs involved, as well as the corrected value of nuclear repulsion energy, as in the following example:

```
Pseudopotentials will be used:
  84 electrons simulated by 13 pseudopotentials ( 46 radial
terms)

Nuclear repulsion energy after psp correction: 451.404073102
au
```

If the **PRINT** option is present, a detailed specification of the ECPs will be printed after the basis set output.

ECP Library Basis Sets

The simplest way to introduce pseudopotentials in a PQS calculation is to use one of the built-in ECP basis sets described below. To do this, use one of the names listed as the basis set name for the calculation. Note that many of these bases contain not only the pseudopotential specification and valence basis set for the atomic centers carrying an ECP core, but also a matching basis for light elements (i.e., elements without an ECP core).

Five types of pseudopotentials are available:

1. Stevens-Basch-Krauss-Jaisen-Cundari compact effective potentials (CEP) [3].

These large core ECPs are available with three different contractions of the valence basis (note that the different contractions apply only for elements H-Ar):

cep-4g	large core, minimal basis	H-Rn
cep-31	large core, double-zeta basis	H-Rn
cep-121	large core, triple-zeta basis	H-Rn

2. Hay-Wadt Los Alamos National Laboratory (LANL) relativistic ECP [4].

In this set there are both large core and small core pseudopotentials, available with a minimal and a double-zeta valence basis:

lanl1mb	large core, minimal basis	H-La, Hf-Bi
lanl1dz	large core, double-zeta basis	H-La, Hf-Bi
lanl2mb	small core, minimal basis	H-La, Hf-Bi
lanl2dz	small core, double-zeta basis	H-La, Hf-Bi, U-Pu
lanl2dzdp	ditto plus diffuse & polarization	H, C-F, Si-Cl, Ge-Br, Sn-I, Pb, Bi

Note that the lanl1 and lanl2 bases differ only for metals K-La, Hf-Au.

3. Christiansen-Ross-Ermler-Nash-Bursten (CRENB) shape-consistent relativistic ECP [5]

crenbs	large core, small basis	H-Ca, Sc-Kr, Y-Xe, La, Hf-Rn, Rf
crenbl	small core, large basis	H-Np

4. Stuttgart-Cologne relativistic ECP [6]

Several large and small core ECPs, coupled with a variety of valence basis are available from this set. We have organized them into two main library files, for large core and small core pseudopotentials, covering a large number of elements, plus additional sets covering only a limited number of elements but with some special ECPs or valence basis definitions. Additional pseudopotentials and basis sets can be downloaded from the Stuttgart-Cologne group web page at <http://www.theochem.uni-stuttgart.de/pseudopotentials/index.en.html>.

srlc	large core, double-zeta+ basis	H-Ca, Zn-Sr, Cd-Lu, La, Hg-Rn Ac-Lr
srscl	small core, double-zeta+ basis	K-Rn, Ac-Lr

srlic-cc-pvtz	large core + cc-pvtz basis	Ga-Kr, In-Xe
srlic-aug-cc-pvtz	ditto + aug-cc-pvtz basis	Ga-Br, In-I
srlic-cc-pvqz	ditto + cc-pvqz basis	Ga-Kr, In-Xe
srlic-aug-cc-pvqz	ditto + aug-cc-pvqz basis	Ga-Br, In-I
srsc-cc-pvdz	small core+ cc-pvdz basis	Ga-Kr, In-Xe, Tl-Rn
srsc-aug-cc-pvdz	ditto + aug-cc-pvdz basis	Ga-Kr, In-Xe, Tl-Rn
srsc-cc-pvtz	ditto + cc-pvtz basis	Ga-Kr, In-Xe, Tl-Rn
srsc-aug-cc-pvtz	ditto + aug-cc-pvtz basis	Ga-Kr, In-Xe, Tl-Rn
srsc-cc-pvqz	ditto + cc-pvqz basis	Ga-Kr, In-Xe, Tl-Rn
srsc-aug-cc-pvqz	ditto + aug-cc-pvqz basis	Ga-Kr, In-Xe, Tl-Rn
srsc-cc-pv5z	ditto + cc-pv5z basis	Ga-Kr, In-Xe, Tl-Rn
srsc-aug-cc-pv5z	ditto + aug-cc-pv5z basis	Ga-Kr, In-Xe, Tl-Rn
srsc-ano	small core, ano basis	La-Lu, Ac-Lr
srsc-ano-seg	ditto + segmented ano basis	La-Lu, Ac-Lr

5. Karlsruhe def2 basis sets [7]

Basis sets of split-valence, triple-zeta valence and quadruple-zeta valence quality for H-Rn (except lanthanides) developed by the Ahlrichs group at Karlsruhe, Germany. These basis sets use the Stuttgart-Cologne pseudopotentials (above) commencing with Rb (fourth row and greater). Two sets of polarization functions are available. The central idea behind these basis sets is a balanced description (similar errors) across the entire periodic table at each basis set level.

Method	Purpose of the Calculation			
	Explorative	Qualitative	Quantitative	Reference
DFT		def2-svp	def2-tzvp	def2-qzvp
HF		def2-svpp	def2-tzvpp	def2-qzvpp
MP2, CC	def2-svpp	def2-tzvpp	def2-qzvpp	

User-Defined Pseudopotentials

The PQS implementation of ECPs is consistent with the form of the pseudopotential operator first defined by Kahn and Goddard in 1972 [8]:

$$V_{\text{psp}} = (Z-n_c)/r + V_l + \sum (V_l - V_i) P_i$$

where Z is the atomic number, n_c is the number of core electrons, r is the radial distance, P_i are angular momentum projectors, V_l is the local term and $(V_l - V_i)$ are the semi-local terms. The summation goes from $i=0$ to $i=l-1$. Both local and semi-local terms are given as linear combinations of Gaussian components

$$V = \sum c_j r^{\eta_j - 2} \exp\{-\gamma_j r^2\}$$

where c_j are expansion coefficients, η_j are integers (usually in the range 0-2), and γ_j are the exponents of the Gaussian terms. The summation goes over $j=1$ to $j=m$. Thus, for each pseudopotential the following information needs to be specified: the number of core electrons n_c and the maximum value of angular momentum l , then for the local

term and for each of the semi-local terms in turn, the number of expansion terms m , and a series of values for c_j , n_j and γ_j .

User-defined ECPs can be added to the input file using the **NEXT** option (as for normal basis set augmentation). As for the latter case, the input starts with a formatted line containing the word FOR in columns 1-3, and an atomic symbol in columns 11-18. The rest of the pseudopotential input is in free format (i.e., fields must be separated by at least one space). The second line must contain the keyword **ECP**, followed by the number of core electrons n_c and the maximum angular momentum l . The keyword **PSP** may be used instead of **ECP**. Next comes the specification of the local potential term: one line containing the number of expansion terms m , followed by m lines each containing a triplet of values for c_j , n_j and γ_j in that order. Specification of the semi-local potential terms follows, comprising l sections, each starting with a line containing an m value, followed by m lines containing c_j , n_j and γ_j values.

Examples

```
FOR          AL
ECP 10      2
 1         ----- d potential -----
      -3.03055000  1      1.95559000
 2         ----- s-d potential -----
      6.04650000  0      7.78858000
      18.87509000  2      1.99025000
 2         ----- p-d potential -----
      3.29465000  0      2.83146000
      6.87029000  2      1.38479000
```

This is the specification of the CEP pseudopotential for aluminum. There are 10 core electrons and the maximum l value is 2. The local part (d potential) has only one expansion term with $c=-3.03055$, $n=2$, and $\gamma=1.95559$, respectively. The s-d semi-local potential is given as linear combination of two terms, with $c_1=6.0465$, $n_1=0$, $\gamma_1=7.78858$, and $c_2=18.8759$, $n_2=2$, and $\gamma_2=1.99025$, respectively. The second semi-local potential (p-d) is defined in a similar way. Note that the comments have been added only for the sake of clarity - they may be omitted from the actual input.

If the ECP specification follows the basis set input for the same atomic center, the initial line (the one containing the FOR keyword) may be omitted, as in the following example, which provides the complete specification of the lanl2mb pseudopotential and basis set for iron:

FOR	FE		
S	6.422000	-.392788	.095044
	1.826000	.771264	-.223080
	0.713500	.492023	-.242981
	0.102100	.000000	.586952
	0.036300	.000000	.542852
P	19.480000	-.047028	
	2.389000	.624884	
	0.779500	.472254	
P	0.074000	.517173	
	0.022000	.584079	
D	37.080000	.028292	
	10.100000	.153707	
	3.220000	.385911	
	0.962800	.505331	
	0.226200	.317387	
ECP 10	2		
3	----- d potential -----		
	-10.00000000	1	392.61497870
	-63.26675180	2	71.17569790
	-10.96133380	2	17.73202810
5	----- s-d potential -----		
	3.00000000	0	126.05718950
	18.17291370	1	138.12642510
	339.12311640	2	54.20988580
	317.10680120	2	9.28379660
	-207.34216490	2	8.62890820
5	----- p-d potential -----		
	5.00000000	0	83.17594900
	5.95359300	1	106.05599380
	294.26655270	2	42.82849370
	154.42446350	2	8.77018050
	-95.31642490	2	8.03978180

6. GUESS command

The GUESS command provides an initial set of MOs to start off an SCF calculation. In most cases, this command is unnecessary, as the SCF program sets the starting orbitals automatically. The **GUESS** command may optionally be followed by an equal sign and a guess type.

Options: **GUESS**=<string> **UHFS** **PRINT**=<integer> **SWAP**=<integer>,<integer>
SWAB=<integer>,<integer> **MIX**=<integer>,<integer> **ANGLE**=<real>
(**SWA1/SWA2/SWA3**=<integer>,<integer> **SWB1/SWB2/SWB3**=<integer>,<integer>)

GUESS=ATOM superposition of atomic densities available for all elements however not available for generally contracted basis sets which should be segmented before use (see the **BASIS** command)

GUESS=PM3|AM1|MNDO|MINDO semiempirical guess available for any elements for which the corresponding semiempirical method has been parametrized (see the **SEMI** command for a list of available atoms)

GUESS=HUCKEL extended Hückel guess available for H-Kr

GUESS=CORE modified core guess available for all elements

GUESS=READ use transformed SCF vectors from a previous calculation this can be either a previous *ab initio* calculation with the same or a *smaller* basis set (*fewer* basis functions), or a semiempirical calculation.

GUESS=USE use the SCF vectors on the <MOS> file “as is” without any further manipulation; this avoids the potentially expensive orbital projection (see below) which is unnecessary when both the geometry and the basis set are unchanged

The simplest guess is the modified core guess which gets the initial MOs by diagonalizing just the one-electron part of the Hamiltonian, modified to omit the potential from distant nuclei. Unfortunately this is usually the worst option. The extended Hückel guess [9,10] involves setting up and diagonalizing the so-called Hückel Hamiltonian, given by

$$H_{ii} = h_{ii} \quad H_{ij} = 0.5K (h_{ii} + h_{jj}) S_{ij}$$

where h_{ii} is taken to be the orbital energy, ϵ_i , associated with orbital i , and \mathbf{S} is the overlap matrix. K is a constant, typically taken to be 1.75. In our implementation of the extended Hückel method, we have used the scaled mini basis as an underlying minimal basis set, and we take (estimated rather crudely) different values for K depending on the atom type and the interaction (first row-first row has a different K than first row-third row, for example). This Hückel guess performs better than we expected, and prior to **GUESS=ATOM** was the only option for systems containing transition metals.

The MOs obtained from the Hückel method are given in terms of a particular minimal basis. These are then projected onto the actual basis via [11]

$$\mathbf{C}_1 = \mathbf{S}_{11}^{-1} \mathbf{S}_{12} \mathbf{C}_2 [\mathbf{C}_2^t \mathbf{S}_{12}^t \mathbf{S}_{11}^{-1} \mathbf{S}_{12} \mathbf{C}_2]^{-1/2}$$

where 1 represents the actual basis and 2 the minimal Huckel basis. This projection scheme is general, and is also used when reading in converged MOs from a previous calculation (either with the same or a different basis set).

The semiempirical guess sets up and solves the SCF equations using the requested semiempirical method. Because semiempirical theory does not explicitly consider atomic cores, the final semiempirical MOs are valence MOs only and the missing core orbitals need to be added before they can be used in the *ab initio* code. In the current implementation, these are simply taken to have a coefficient 1 for the core and zero for all other basis functions. The underlying basis functions for the semiempirical MOs are in fact Slater orbitals, and each Slater orbital is expanded into three Gaussians using standard fitting coefficients [12]. The core orbitals are also expanded, either in the STO-3G basis, or in Huzinaga's MINI-SC basis [13]. This set of MOs is then projected onto the actual basis in the same way as for the Hückel guess.

The best option is usually GUESS=READ and this is the default for a geometry optimization (i.e., to use the converged MOs from the previous geometry to start off the SCF at the current geometry). The best guess for a cold start is usually the superposition of atomic densities (GUESS=ATOM); however as this is still relatively new code, we have retained PM3 as the default which is attempted if no guess type is specified. PM3 is available for all main group elements through the fourth row, except for the rare gases, and also for zinc and cadmium. It is not available for any other transition metals, for which either the ATOM, HUCKEL (third row) or CORE guesses must be used. The HUCKEL guess is often better than PM3 for closed-shell systems containing third row main group elements.

If no guess type is specified, then the PM3, HUCKEL and CORE guesses are attempted, in that order, until one of them works. Note that the options that are available with the **SEMI** command in case of SCF convergence problems cannot be used with the **GUESS** command, which simply uses the default values. If you experience problems with any of the semiempirical guess options, you should converge the wavefunction using the full semiempirical module and use GUESS=READ to read in the converged MOs. We would encourage the use of GUESS=ATOM wherever GUESS=READ is not an option.

Note that **GUESS=READ** has a suboption **FILE=<filename>** where <filename> is the name of the file containing the initial guess MOs. Normally MOs will be read from the *current* MOs file in the working directory; however if you have a specific set of MOs available that you wish to use (e.g., from a related job) then these can be read via the **FILE** option.

****IMPORTANT**** You can specify a full directory path in <filename>; however the file extension - which *must* be **.mos** (**.mob** for beta MOs) - should *not* be given. Additionally there *must* be a basis file with the same filename (and extension **.basis**) in the same location as the MOS file for this option to work properly.

Other options are:

UHFS: requests an unrestricted singlet wavefunction.

PRINT=<integer>: controls the amount of printout (larger integer - more printout).

SWAP=<integer1>,<integer2>: This swaps the occupancies of occupied and virtual alpha/closed-shell MOs. The first integer should point to an occupied orbital; the second to a virtual (integer1 < integer2). If no orbitals are specified, then the HOMO and LUMO are swapped.

SWAB=<integer1>,<integer2>: Same as **SWAP**, only for beta orbitals.

MIX=<integer1>,<integer2>: Unrestricted wavefunctions only. Mixes alpha and beta occupied and virtual MOs. (If no parameters are given, the HOMO and LUMO are mixed.) This typically destroys any molecular symmetry (which, if present, should be turned off via **SYMM=0** on the **GEOM** card) and is usually used in an attempt to converge to a lower energy UHF singlet (as opposed to the closed-shell RHF singlet). The mixing is given by (integer1 < integer2)

$$\Psi_{\alpha}(1) = \text{Cos}(A) \Psi_{\alpha}(1) + \text{Sin}(A) \Psi_{\beta}(2)$$

$$\Psi_{\beta}(1) = \text{Cos}(A) \Psi_{\beta}(1) - \text{Sin}(A) \Psi_{\alpha}(2)$$

where A is the rotation angle. This angle can be specified by

ANGLE=<real>: where <real> is the rotation angle in degrees. The default is A = 30°.

*Note that SWAP and MIX do **NOT** work directly with GUESS=CORE or GUESS=ATOM*

SWA1/SWA2/SWA3=<integer1>,<integer2>

SWB1/SWB2/SWB3=<integer1>,<integer2>

There are occasions where the swapping of more than one pair of MOs is required and you can in fact swap up to three different pairs of MOs (both alpha and/or beta spin) at the same time. Make sure you know what you are doing when you try these options as there is only limited checking in the code.

If you are swapping just one pair of MOs (either one alpha spin pair or one beta spin pair or both) you should use only the regular single-swap options (**SWAP** and/or **SWAB**). Do not use **SWAP** or **MIX** with **SWA1/SWA2/SWA3** (or the corresponding beta spin options).

PRACTICAL ASPECTS

From a practical point of view the GUESS options perform best when the basis set is small, e.g., STO-3G or 3-21G.

For medium-sized and large molecules with the 6-31G* and larger basis sets, better performance (i.e., more rapid convergence) is usually achieved by starting with a few SCF cycles (say 6) using a smaller basis. For example

```
BASIS=3-21G
GUESS
SCF ITER=6
BASIS=6-31G*
GUESS=READ
SCF
```

As pointed out below, the two **GUESS** commands here are optional and can be omitted with no ill effects. They are included here only to show the logic of the calculation.

With very large basis sets, a staggered approach may be best, e.g.

```
BASIS=3-21G
GUESS
SCF DFTP=B3LYP ITER=6
BASIS=6-31G*
GUESS=READ
SCF DFTP=B3LYP ITER=6
BASIS=6311G(2d,p)
GUESS=READ
SCF DFTP=B3LYP
```

****IMPORTANT**** We have now included standard defaults within the **SCF** module and in most cases it is no longer necessary to include a **GUESS** card in the input deck. The **GUESS** card must be present only if you wish to force a particular (non-default) guess type or if you want to manipulate the final guess MOs (e.g., with **SWAP** or **MIX**). *Note that the restriction in previous versions of the program (3.2 and earlier) with GHOST atoms (see **GEOM**) to the CORE guess only has now been removed.*

7. INTE command

This command can be used to redefine the precision of the integrals used in the final and the preliminary SCF iterations, the blocking parameters, and the integral strategy (route). It is usually not needed.

In a typical SCF, a “sloppy” threshold is used either for the first 10 cycles or until the Brillouin criterion (a measure of SCF convergence) reaches 10^{-3} (whichever comes sooner); thereafter a tighter threshold is used to achieve convergence. The default thresholds are 10^{-7} and 10^{-10} , respectively. For larger molecules containing basis sets with diffuse functions or for medium/large UHF calculations, it is a good idea to increase the final integral threshold to 10^{-11} or 10^{-12} .

Options: **THREshold**=<real>,<real> **LIMIts**=<integer>,<integer>,<integer>
ROUTE=<integer> **ONEL** **STABLE** **ECP** **PRINT**=<integer>

THREshold=<real>[, <real>]: Final and optionally also the starting integral threshold, in “pH” form, i.e., its negative logarithm is given (thus 7 corresponds to 1.0E-7).

LIMIts=<integer>,<integer>,<integer>: Integral blocking parameters *limxmem*, *limblks* and *limpair*. The default values are 300 000, 300 and 100 for SCF. For FORCES and NMR, the default *limxmem* is 800 000. It is not recommended to change these parameters. If there is a severe memory problem, decreasing *limxmem* or the other two parameters may help. Also, slightly increasing them may result in minor performance enhancements. For high angular momentum basis functions (*g* and in particular *h* and *i* functions), it is necessary to increase *limxmem* and often decrease *limpair*. Note that if you change *one* of these parameters, values for *all three* must be given.

ROUTE=<integer>: Values are 1 or 2. Manually sets the integral route, i.e., the path through the integral code. Needed only in exceptional cases. **ROUTE**=2 is more efficient if there are many identical atoms while **ROUTE**=1 is better if all atoms are different. Usually, the program determines this information internally, but it can be forced to take a different route.

ONEL: only one-electron integrals are calculated (used only for testing).

STABLE: Must be ± 1 . -1 switches on stability checking for two-electron integrals; +1 switches it off (i.e., assumes all integrals are stable). Under certain circumstances, most notably for basis functions with very large exponents (normally upwards of a million) the standard integral evaluation algorithm is unstable. The potential for instability is detected during basis set read-in, and **STABLE** is set accordingly. *However, this is not foolproof.* Switching on stability checking will detect any instabilities and use a modified algorithm to evaluate the unstable integrals. Checking for instabilities does take time, and so if this option is invoked the calculation will take longer. The default can be overridden by specifically setting **STABLE** here.

PRINT=<integer>. It must be 1 or 2. If it is 1, the one-electron integrals are printed, if 2, the 2-electron ones. Both options, particularly the latter, can produce extremely voluminous output, so they should be used only for small molecules.

8. SCF command

Options: **DFTP** (or **DFT**)=<string> **THREshold**=<real> **ITERations**=<integer>
DIIS=<real> **LVSH**=<real> **PSEUdo**=<real> **STHReshold**=<real> **NODD**=<integer>
VIRT=<integer> **GRID** (or **FACTor**)=<real> **LRAD**=<integer> **LANG**=<integer>
LOCALize=<string> **LOCV**=<string> **GRANularity**=<integer> **ANNEal**=<real> **SEMI**
PWAVE **PRINT**=<integer>

DFTP (or **DFT**): the DFT exchange-correlation potential. Possible values include:

HFS	Slater local exchange [14]
SVWN	Slater local exchange + Vosko, Wilk & Nusair local correlation [15] using the RPA fit (this is the “wrong” functional but is the one used by Gaussian)
SVWN5	ditto, but using the “correct” Ceperley-Alder fit (this is the functional recommended in the original 1980 paper)
HFB	Slater local exchange + Becke’s 1988 nonlocal exchange [16]
BVWN	as HFB + VWN local correlation (RPA fit)
BVWN5	as HFB + VWN local correlation (CA fit)
BLYP	Slater local exchange + Becke’s 1988 nonlocal exchange + Lee, Yang & Parr’s nonlocal correlation [17]
BP86	Slater local exchange + Becke’s 1988 nonlocal exchange + Perdew’s 1986 local & nonlocal correlation [18]
BPV86	as BP86 but with Perdew’s 1986 local correlation replaced by VWN local correlation (CA fit) - used with COSMO
BPW91	Slater local exchange + Becke’s 1988 nonlocal exchange + Perdew & Wang’s 1991 nonlocal correlation [19]
OPTX	Slater local exchange + Handy & Cohen’s optimized exchange [20]
OVWN	as OPTX + VWN local correlation (RPA fit)
OVWN5	as OPTX + VWN local correlation (CA fit)
OLYP	Slater local exchange + Handy & Cohen’s optimized exchange + Lee, Yang & Parr’s nonlocal correlation
OP86	Slater local exchange + Handy & Cohen’s optimized exchange + Perdew’s 1986 nonlocal correlation
OPW91	Slater local exchange + Handy & Cohen’s optimized exchange + Perdew & Wang’s 1991 nonlocal correlation
PW91	Slater local exchange + Perdew & Wang’s 1991 nonlocal exchange + Perdew & Wang’s 1991 nonlocal correlation
B3LYP	hybrid 3-parameter HF-DFT functional comprising linear combination of Slater local exchange, Becke nonlocal exchange, VWN local correlation and LYP nonlocal correlation together with a portion (20%) of the exact Hartree-Fock exchange [21]
B3PW91	ditto, but LYP functional replaced by PW91 and VWN replaced by VWN5 (original 3-parameter hybrid recommended by Becke) [21]
O3LYP	hybrid 3-parameter HF-DFT functional comprising linear combination of Slater local exchange, Handy/Cohen nonlocal exchange, VWN5 local correlation and LYP nonlocal correlation with a portion (11.61%) of the exact Hartree-Fock exchange [22]
PBE	Perdew, Burke & Ernzerhof [23]
B97	Becke’s 1997 10-parameter hybrid [24]
B97-1	ditto as reparametrized by Hamprecht, Cohen, Tozer & Handy [25]

B97-2	ditto as reparametrized by Wilson, Bridley and Tozer [26]
HCTH	Hamprecht, Cohen, Tozer & Handy – parametrization 407 [27]
WAH	modified form of B3LYP (with only 5% exact exchange) used specifically for NMR chemical shifts [28]
USER	user defined combinations, i.e., make up your own functional

User-defined functionals are specified as follows. The line following the SCF command line (which must contain DFTP=USER) must start with the 5 character string \$user, followed by a list of coefficients `xf=<real>`, with at least one blank space between each coefficient definition. `xf` can be one of the following:

<code>ax</code>	coefficient for exact Hartree-Fock exchange
<code>xs</code>	coefficient for Slater local exchange
<code>xvwn</code>	coefficient for VWN local correlation
<code>xvwn5</code>	ditto for VWN5 local correlation
<code>xb88</code>	ditto for Becke's 1988 nonlocal exchange
<code>xoptx</code>	ditto for Handy & Cohen's optimized nonlocal exchange
<code>xp86l</code>	ditto for Perdew's 1986 local correlation
<code>xp86nl</code>	ditto for Perdew's 1986 nonlocal correlation
<code>xpw91x</code>	ditto for Perdew & Wang's 1991 nonlocal exchange
<code>xpw91l</code>	ditto for Perdew & Wang's 1991 local correlation
<code>xpw91nl</code>	ditto for Perdew & Wang's 1991 nonlocal correlation
<code>xlyp</code>	ditto for Lee, Yang & Parr's nonlocal correlation
<code>xpbex</code>	ditto for Perdew, Burke & Ernzerhof's 1996 nonlocal exchange
<code>xpbec</code>	ditto for Perdew, Burke & Ernzerhof's 1996 nonlocal correlation

Typical usage would be

```
SCF DFTP=USER
$user ax=0.17 xs=0.83 xvwn=0.27 xbec=0.73
```

The Becke Half-and-Half functional [29], for which there is not a specific keyword in PQS, can be accessed in this way via

```
$user ax=0.50 xs=0.50 xlyp=1.00
```

There has been a certain amount of confusion over the very popular B3LYP functional [30]. The original 3-parameter hybrid functional, as defined by Becke [21], was effectively B3PW91. When Gaussian released their first DFT implementation they had not coded the PW91 functional, and so they replaced it by the LYP functional – hence B3LYP. In addition to this, they erroneously used the wrong version of the VWN functional (the RPA fit as opposed to the Ceperley-Alder fit). Due to the wide-spread use of the Gaussian program, the Gaussian version of B3LYP became the de facto standard, despite the fact that at the time of its release there were very little published data attesting to its quality, if indeed it had been properly tested at all.

The Gaussian version of B3LYP, which is what you get by specifying this keyword, is equivalent to

```
$user ax=0.20 xs=0.80 xb88=0.72 xvwn5=0.19 xlyp=0.81
```

The version currently coded in, e.g., the GAMESS program, replaces the “incorrect” VWN5 functional by VWN, and can be accessed via

```
$user ax=0.20 xs=0.80 xb88=0.72 xvwn=0.19 xlyp=0.81
```

Our own work suggests that both the OLYP and O3LYP functionals are as good as, if not better, than B3LYP for general organic chemistry [31], although this does not hold for transition metals [32].

THREshold=<real>: maximum Brillouin matrix element at convergence in pH form; default is 5.0 (which is 1.0×10^{-5}). If you need a tighter convergence than the default (e.g., for numerical frequencies on a floppy molecule) then make sure the integral threshold is appropriate. A good rule of thumb is to set the final integral threshold to at least twice the Brillouin threshold (in pH form), e.g., if the SCF threshold is 5.5, the integral threshold should be 11.

ITERations=<integer>: maximum iteration count, default is 50.

DIIS=<real>: DIIS is a method for accelerating SCF convergence which is now used in virtually all SCF codes [33]. Not switched on until the maximum Brillouin element is less than this value, default 2.0.

LVSH=<real>: artificially shifts the energies of the virtual orbitals to help convergence (typical values from 0.1 to 4.0) [34]. The default is an initial level shift of unity for DFT wavefunctions and zero (i.e., *no* level shift) for HF, except for the *core* guess, when the default is again unity. After the first cycle, the actual shift is controlled during the SCF procedure by the HOMO-LUMO gap, except that the level shift on a particular SCF cycle cannot be less than 30% of its initial value.

PSEUdo=<real>: uses pseudodiagonalization [35] instead of full diagonalization of the Fock matrix. Switched on when the maximum Brillouin element is less than the requested threshold (default is 0.005). For unrestricted wavefunctions, or for restricted wavefunctions and less than 250 basis functions, pseudodiagonalization is switched off by default.

STHReshold=<real>: threshold for suppressing linear combinations with very low norm in nearly linearly dependent basis sets. If the lowest eigenvalue of the overlap matrix is smaller than **STHR** (default is 6.0 in pH form, corresponding to 10^{-6}), then the procedure is activated, and any linear combination of basis functions with eigenvalue less than **STHR** is eliminated. This reduces the dimension of all SCF matrices by one for each linear combination removed. If the basis set is too linearly dependent (e.g., has several interacting diffuse functions), the integral threshold should be tightened (see the **INTE** keyword, above). Basis function suppression was previously done using a penalty function approach which was far less efficient.

NODD=<integer>: switch off use of difference densities after <integer> SCF cycles. The default is to use difference densities to construct a difference Fock matrix; however sometimes this can hinder convergence due to numerical problems. Difference densities are no longer used by default if there is no convergence after 30 cycles. Note that the last cycle (at convergence) always uses full densities. Specifying **NODD** alone will use full densities every cycle.

VIRT=<integer>: number of virtual orbitals to print if MO printout is requested (see the **PRINT** keyword below).

GRID/FACTor=<real>: controls grid quality in DFT calculations. Larger values, more grid points. Suggested values are between 0.5 and 2.0 (default 1.0). **GRID** controls both the number of radial grid points and also the maximum number of angular points per radial shell (i.e., the degree of “grid pruning”). NOTE: This option was formerly **FACTor** which is now obsolescent.

LRAD=<integer>: sets the number of radial grid points in a DFT calculation. The maximum allowed value is 400; any value greater than this is reduced. The default is for the program to determine LRAD based on the value of GRID and the atom type.

LANG=<integer>: sets the angular quadrature order, i.e., the number of grid points, in each spherical shell. Must be in the range 1 to 7 with 7 being the maximum which corresponds to 434 angular grid points per shell. The default is for the program to determine LANG for each radial grid point based on the value of GRID and the atom type.

LOCALize=<string> localizes the occupied orbitals using the method specified. Currently the Boys [36] and the Pipek-Mezey [37] localizations are implemented (LOCA=Boys or LOCA=Pipek).

LOCV=<string>: as above but for the virtual orbitals (using zero for the Fermi level).

GRANularity=<integer>. This has an effect only for parallel jobs. The default is 20. To decrease the communication overhead, the program at first assigns GRAN integral blocks to the slaves. Toward the end of the calculation, this is reduced to single blocks. Too small value for GRAN results in excess communication cost, too high a value involves the risk that one slave finishes much later than the rest, resulting in idle time.

ANNEal=<real> **UHF only at present**. If this option is specified, then the molecular orbitals are given fractional occupancies $n_i = \{1 + \exp[(\varepsilon_i - \varepsilon_F)/kT]\}^{-1}$, i.e., according to Fermi-Dirac statistics. Here n_i is the occupation number of orbital i , ε_i is its orbital energy, ε_F is the Fermi energy, k is Boltzmann’s constant and T is the absolute temperature. The Fermi level is calculated from the condition that the total number of electrons is correct. The real number specified is the value of kT in atomic units E_h . A value around 0.2 is reasonable. The temperature is reduced by 30% in each SCF cycle, hence the name, annealing.

SEMI: Activates semi-direct mode (for both integral evaluation and storage *and* the DFT part of the calculation; see below).

PWAVE: Activates the Fourier Transform Coulomb (FTC) method, an alternative approach for evaluating a large part of the Coulomb term in closed-shell DFT calculations. Available for pure (i.e., non-hybrid) DFT functionals only. *Cannot* be used for Hartree-Fock calculations, only DFT (see below).

PRINT=<integer>: controls the amount of printout (larger integer - more printout). In particular, a value of 3 will print the MO coefficients.

Semidirect DFT

The DFT part of this was actually available in previous releases (from PQS v. 2.5) as an undocumented feature. It was first documented in the PQS v. 3.2 manual.

The normal DFT SCF algorithm is "fully direct" in that all quantities are recomputed on every SCF cycle. We have implemented a "semidirect" algorithm, which saves certain DFT quantities and reuses them on the next cycle; specifically these are the atomic integration grids, the density over the grid and the potential over the grid. This enables delta densities and delta potentials to be used when computing the exchange-correlation energy which can lead to significant savings for larger systems. These ideas have been discussed in a published article [38], from which more details can be obtained.

To activate semidirect DFT simply add the keyword `SEMI` onto the **SCF** command line. Note that several files (three for each symmetry-unique atom) are written to your scratch directory, both on the master *and* on the slave nodes for parallel jobs. These may not all be deleted, especially on a job crash, and so you should check your scratch directory periodically (on *each* node) and remove dead files when using this option.

The original meaning of "semidirect" was the storage of some integrals, typically the more computationally expensive ones, so that they could be reused on subsequent SCF cycles as opposed to being continuously recalculated. This has now been fully implemented, both for integrals stored in core memory or on disk. (Note that this is either one *or* the other, *not* both.)

The amount of in-core memory *or* disk space available to store integrals is specified on the memory card, via the options `CORE` or `DISK` (see the **%MEM** command). Integrals are computed (using the final accuracy threshold) on the first SCF cycle and either stored in core memory or written to disk until the amount of storage requested has been exhausted. These integrals are then reused on subsequent SCF iterations. All remaining integrals are recalculated as usual.

Note that in order to use semidirect integral storage, either `CORE` or `DISK` *must* be specified, along with a corresponding storage value. If no units are given, in-core storage is assumed to be in MW and disk storage in MB. (`DISK` must be at least 100 MB or it will be ignored.) Specifying `SEMI` on the **SCF** command line *without* a corresponding `DISK/CORE` specification will do semidirect on the DFT part of the calculation *only*.

Because of the huge disparity between CPU and I/O speed, semidirect calculations that request a large amount of disk storage are nearly always slower than the corresponding fully-direct calculation. (See the comments under the **%MEM** command regarding I/O buffer size.)

Fast Fourier Transform (FFT)

This is a potentially useful feature currently available for pure (i.e., non-hybrid) DFT closed-shell calculations of energies, gradients and geometry optimizations which can provide substantial savings in computational time with essentially no loss in accuracy [39]. It involves computing a large part of the Coulomb potential (and its derivative for the forces) using a plane wave expansion of (part of) the original Gaussian basis set. The FTC method can be considered as being related to the so-called “resolution of the identity” RI-DFT approach which expands the density in an auxiliary basis [40]. However, unlike the situation in RI-DFT where the auxiliary basis is typically a set of Gaussian functions slightly larger than the original basis set [41], in the FTC method it is an essentially infinite plane wave basis. This makes FTC potentially much more accurate than RI-DFT. It also scales better with respect to both system and basis size.

The current implementation is only preliminary and does not realize the full potential of the method. It is closed-shell only and does not make use of any symmetry that the system may have. For small basis sets (STO-3G, 3-21G) it is slower than the traditional, all-integral algorithm, but as the system size and (especially) basis set increase, so do the savings. With large basis sets containing multiple valence-shell and polarization functions, savings of up to an order of magnitude or more over the conventional calculation can be achieved. For small molecules, the FTC method requires a lot more memory than the traditional all-integral algorithm, but for larger systems the additional memory requirement over the traditional algorithm is small.

To access the FTC method, simply add the keyword **pwave** to the **SCF** command line, e.g., `SCF DFTP=OLYP pwave`

PRACTICAL ASPECTS

In most cases the standard SCF procedure will converge to the ground state wavefunction. Sometimes, e.g., if there are particular convergence problems or if high value level shifts have been used, convergence may be to an excited state. It is always a good idea to check the final orbital energies; if the orbital energy of the (supposed) LUMO is *lower* than that of the HOMO then it is virtually certain that you have converged to an excited state. This can normally be rectified by restarting the calculation, swapping the HOMO and LUMO. (See the SCF **GUESS** options.)

Although the wrong orbital energy ordering is normally a clear sign of convergence to an excited state, the correct energy ordering is unfortunately not a cast-iron guarantee that you have the ground state. The only way to be sure is to do a stability check on the wavefunction. This capability is currently unavailable in PQS but is under development and is planned for a future release.

SAVINGS

Significant savings can result with judicious use of the semidirect and FTC options. For integral semidirect, the maximum savings ensue when a large percentage of the total number of integrals can be stored. This occurs for small and medium-sized systems. As the system and basis set size increase, savings decrease. There is little point doing any integral storage for large systems. For DFT semidirect on the other hand, savings tend to increase with increasing system size, so it is always worth specifying `SEMI` with large DFT calculations, regardless of the functional type.

FTC is best used for medium and large systems with large basis sets. As the FTC part of the calculation does not use symmetry, maximum savings will occur with unsymmetrical molecules. With highly symmetrical systems the gain from FTC may be offset by the current inability to utilize the molecular symmetry. Because FTC calculations typically compute only a small percentage of the integrals that are needed for a traditional all-integral calculation, the integral semidirect option can be utilized with effect for larger systems when FTC is employed than otherwise.

<p>NOTE: Only RHF (Restricted Hartree-Fock) and UHF (Unrestricted Hartree-Fock) SCF has been implemented so far, no ROHF. The type of the calculation will be automatically set according to the multiplicity to RHF or UHF.</p>

9. FORCE command

The **FORCE** command calculates analytical forces (negative gradient) for closed- and open-shell HF and DFT and closed-shell MP2.

Options: **THR1**=<real> **THR2**=<real> **LIMIts**=<integer>,<integer>,<integer>
PRINt=<integer>

THR1=<real>: one-electron integral threshold in pH format, e.g. 12 means 1.0E-12. Default is 11, i.e. 1.0E-11.

THR2=<real>: two-electron integral threshold in pH format. Default is 10, i.e. the threshold is 1.0E-10.

LIMIts=<integer>,<integer>,<integer>: the integral blocking parameters *limxmem*, *limblks* and *limpair*. See the comments to the **INTE** command.

PRINt=<integer>: print level (needed only for diagnostics)

The FORCE module will automatically adapt to the algorithm used in the preceding energy calculation. For example, if FTC were used in the SCF step it will also be used when computing the gradient.

****IMPORTANT**** Do **NOT** change the two-electron integral threshold here for an MP2 force. The same threshold used in the MP2 energy step **MUST** be used in the force.

External Forces

It is possible to modify the force experienced by particular atoms in a molecule via the addition of an applied external force. This is known experimentally as “mechanochemistry” and its most common manifestation is probably atomic force microscopy (AFM), in which a molecule tethered to a surface interacts with a probe tip [42].

The current implementation follows that outlined in a recent paper in Molecular Physics [43]. The external force is defined between pairs of atoms in the molecule, acting along the (hypothetical) line joining the two atoms, either to force them together (PUSH) or to pull them apart (PULL). Applying the external force in this way ensures that there is no overall translation or rotation of the system. As many external forces can be applied between as many pairs of atoms as desired, you are not limited to just one external force between one pair of atoms, although this would be the most direct simulation of an AFM experiment.

External forces are specified between *\$force* and *\$endforce* cards immediately following the **FORCE** command line as follows:

```

$force
atom1  atom2  force (au)  PUSH/PULL
atom3  atom4  force (au)  PUSH/PULL
"      "      "      "
$endforce

```

Here `atom1` and `atom2` are the two atoms (integers) defining the force direction (in the same order as in the geometry input), `force (au)` is the magnitude of the applied force in atomic units (1 au = 82,387.2206 pN) and `PUSH` or `PULL` indicate how the force is to be applied, as discussed above.

Note that during a geometry optimization with an applied external force, the relative direction of the applied force will change at each optimization cycle depending on how the two atoms that define its direction move; however the magnitude will remain constant as will the force type. The entire procedure has been termed enforced geometry optimization (EGO) and has proven to be a very powerful tool for discovering new chemical structures [44,45].

Note also that this is perhaps the simplest approach to simulate an applied force and is clearly missing some of the essential physics of the “real” situation as the force is applied after the SCF procedure and has no influence on the wavefunction. Furthermore we do not include any additional energy term involving the applied force (see [43]) so the energy at any given geometry is the same as it would be without the force. As a result of this the magnitude of the external force required to induce a reaction (for example, a change from a *cis*- to a *trans*-isomer) theoretically is typically two-four times larger than observed in an actual AFM experiment.

For a geometry optimization with an applied external force see example 47 in the RUNNING JOBS section.

11. NUMGrad command

Calculates the gradient (negative force) numerically using central differences on the energy. Available for all wavefunctions for which an energy can be computed. Analytical gradients are available for closed- and open-shell HF and DFT and for closed-shell MP2 (see the **FORCE** command), so there should normally be no need to compute the gradient numerically for these wavefunctions, other than as a check. (Note however that there may be advantages in computing the closed-shell MP2 gradient numerically as the analytical MP2 gradient is still serial only.) This module was added primarily to allow the option of numerical gradients (and hence geometry optimization) for the high-level post-HF wavefunctions now available via the **CORR** command.

Numerical gradients *are* expensive. If the system under investigation has no symmetry at all, then you are facing $6N$ (where N is the number of atoms) energy calculations for each numerical gradient. Compare this to the time required to calculate a typical HF or DFT analytical gradient of around one-third of the time taken to compute the energy. For post-HF wavefunctions, however, calculation of the gradient is normally far more computationally demanding than the energy alone and usually takes much longer - in PQS the time required to compute the analytical MP2 gradient is around three times that of the energy – so this is somewhat less of an issue.

All of the numerical modules in PQS make good, although not optimal, use of symmetry. Only symmetry-unique atoms are displaced and full point-group symmetry can be employed (up to I_h). Furthermore, individual atomic displacements that are equivalent by symmetry can often be avoided. Numerical gradients are most effective for small systems (where even though they are inefficient the computational time required to calculate them is not large) or for medium-sized systems with high symmetry (where the number of finite-difference displacements, and hence energies, required is relatively small).

This is a loop command, requiring a terminating **JUMP** card. Typical usage would be:

```
NUMG
GEOM NOORIENT PRINT=1
SCF THRE=6.0
CORR=<method>
JUMP
```

The **NOORIENT** keyword on the **GEOM** command line is compulsory.

Options: **FDSTep**=<real> **PRINT**=<integer>

FDSTep=<real>: controls the finite-difference step (values in au). The recommended step size for medium-sized molecules is 0.005 for all post-HF based wavefunctions. If the molecule is floppy or if you see convergence difficulties during the energy calculations, this step size should be increased.

PRINT=<integer>: controls the amount of printout (larger integer - more printout).

If the gradient is being calculated numerically for a floppy molecule both the integral and SCF thresholds should be tightened (see the **INTE** and **SCF** keywords).

Geometry Optimization with Numerical Gradients

This requires a nested loop structure, with the optimization as the outer and the numerical gradient as the inner loop. Because the finite-difference steps in the numerical gradient will overwrite the energy, the normal ordering in the optimization loop of energy followed by force should be reversed. Thus

```
OPTIm
NUMG
GEOM NOORIENT PRINT=1
SCF THRE=6.0
CORR=<method>
JUMP
GEOM
SCF THRE=6.0
CORR=<method>
JUMP
```

Note the `GEOM` command after the first `JUMP` command (signifying the end of the numerical gradient) to restore the initial molecular geometry and symmetry, and the use of `NOORIENT` with the `GEOM` command inside the numerical gradient loop. These are common features in many numerical computations (see the **NUMHess** command).

11. NUMHess command

Calculates the Hessian (second derivative) matrix numerically using central differences on the (analytical) gradient. Analytical Hessians are now available for all Hartree-Fock and DFT wavefunctions, both closed- and open-shell (unrestricted), and so the numerical code should only be needed for: (1) Semiempirical wavefunctions; (2) MP2 wavefunctions; or (3) all wavefunctions when the COSMO solvation model is switched on. It might still be advantageous to use the numerical code over the analytical for highly symmetrical systems or if there are difficulties with the analytical code. It may also be advantageous to compute the Hessian numerically for small systems with large basis sets where the FTC method has been used. This extends to larger systems with high symmetry. (Although FTC cannot utilize symmetry, the finite-difference steps typically produce geometries where much, if not all, of the symmetry is lost, allowing FTC to be used with advantage.)

This is a loop command, requiring a terminating `JUMP` card. Typical usage would be:

```
NUMH
GEOM NOORIENT PRINT=1
SCF LOCA=PIPEK
MP2
FORCE
JUMP
```

The `NOORIENT` keyword on the `GEOM` command line is compulsory.

Options: **FDSTep**=<real> **PRINT**=<integer> **FILE**=<string>

FDSTep=<real>: controls the finite-difference step (values in au). Recommended step sizes for medium-sized molecules are 0.005 for HF and semiempirical wavefunctions and 0.02 for DFT. The default if this keyword is not present is 0.02 au.

PRINT=<integer>: controls the amount of printout (larger integer - more printout).

If the Hessian is being calculated numerically for a vibrational frequency analysis on a floppy molecule both the integral and SCF thresholds should be tightened (see the `INTE` and `SCF` keywords).

FILE=<string>: reads in a list of atoms for which a *partial* Hessian will be computed, e.g., for a TS search, calculating Hessian rows and columns for atoms in the “active site”. The first line in the file should be a text-only title (it is ignored); thereafter atoms are listed by number (in the same order as in the geometry input), maximum 10 per line (free format).

<p>**IMPORTANT** In a complex input, involving the calculation of several properties at the final geometry, the <code>NUMHess</code> loop (typically followed by <code>FREQ</code>) should either be done LAST or an additional <code>GEOM</code> card should be inserted to restore the geometry.</p>
--

12. HESS command

Calculates the Hessian (second derivative) matrix analytically. Available for Hartree-Fock and DFT (all supported functionals) wavefunctions, both closed- and open-shell (unrestricted). The current version is somewhat memory intensive, although more efficient than in some other programs.

Options: **THR1**=<real> **THR2**=<real>,<real>,<real> **THREshold**=<real>
ITERations=<integer> **RESEt**=<integer> **GRID**=<real> **NODD** **PRINt**=<integer>

THR1=<real>: one-electron integral threshold in pH format. Default is 10, i.e., 1.0E-10.

THR2=<real>,<real>,<real>: two-electron integral thresholds in pH format. The first is the threshold for the direct contribution of the two-electron second derivatives; the second is for both the contribution from two-electron first derivatives *and* the final integral threshold for the coupled perturbed Hartree-Fock (CPHF); the third is the loose integral threshold for the CPHF. The defaults are 9, 10 and 8, respectively. If these are increased (to say 12 or higher), set `STAB=-1` via the **INTE** command.

THREshold=<real>: convergence threshold for the CPHF in pH format. Default is 5, i.e., 1.0E-5.

ITERations=<integer>: maximum number of iterations during the CPHF. Default is 30.

RESEt=<integer>: resets CPHF if convergence not achieved after `reset` iterations. Default is 20.

Note that the CPHF procedure normally converges fairly rapidly (usually in 10 or less cycles). There is rarely any need to change either `ITER` or `RESEt` from their default values.

GRID=<real>: controls grid quality for DFT wavefunctions. (See the same option under the **SCF** keyword.)

NODD: switch off use of difference densities in CPHF.

PRINt=<integer>: print level (needed only for diagnostics).

Memory and disk requirements. Disk requirements are fairly modest, but memory demands can be high. The greatest memory demand occurs during derivative Fock matrix construction and, especially, the CPHF step. Each symmetry-unique atom requires storage for three derivative Fock matrices (for X, Y and Z perturbations); additionally there are several other matrices of similar dimension in the CPHF step. Total storage during the CPHF is about $3 \times 3 \times \text{NATOMS} \times (N \times (N+1))/2$, where N is the number of basis functions. With, say, $N=1000$ and 100 atoms, this comes to well over 3 GB. The memory demand can be reduced by doing multiple passes over the atoms, i.e., constructing derivative Fock matrices or solving the CPHF equations for only as many atoms as can be comfortably handled in the available memory. Perhaps surprisingly, multiple passes in the CPHF step usually have only a small effect on the job time.

For parallel jobs, each slave needs as much memory as the master, so for very large jobs you may need to reduce the number of CPUs per node to ensure that the available memory per CPU is not exceeded. The parallel efficiency in the CPHF, although improved recently, is not great, and can fall off noticeably beyond 8 processors.

13. NUMPolar/POLAr command

Calculates the polarizability tensor and (optionally) the dipole and polarizability derivatives. The latter can be used to calculate Raman intensities during a standard vibrational analysis (via the `FREQ` command).

The **POLAr** command computes the polarizability and hyperpolarizability analytically, by solving the coupled-perturbed Hartree-Fock equations. It is currently available for closed-shell Hartree-Fock and DFT wavefunctions.

NUMPolar calculates its quantities numerically and is available for all supported wavefunctions. The polarizability is determined by finite-difference on the energy in the presence of an external field; the dipole and polarizability derivatives are determined by finite-difference on the gradient in the field. Note that the field is automatically invoked by the **NUMP** command - there is no need to include the field explicitly on the `GEOM` command line.

NUMP requires a numerical loop, including an energy (and a gradient if derivative quantities are required). In this, it is similar to the `NUMH` command for the numerical Hessian. However, unlike for the Hessian, the number of finite-difference calculations is fixed regardless of the system size. For the polarizability and dipole derivatives, six calculations are needed, with external fields applied along the X, Y and Z axes, respectively. For polarizability derivatives, twelve calculations are needed, with additional external fields along the X & Y, X & Z and Y & Z axes simultaneously. Polarizability derivatives are second derivative quantities and their numerical calculation is consequently less accurate than for the dipole derivatives. Tightening the various thresholds (on the integrals and SCF convergence) is often required, especially for larger systems, to get reliable results.

Typical usage would be:

(i) for the polarizability alone

```
NUMP
GEOM NOORIENT PRINT=1
SCF THRE=6.0
JUMP
```

(ii) for dipole/polarizability derivatives

```
NUMP POLD
GEOM NOORIENT PRINT=1
SCF THRE=6.0
FORCE
JUMP
```

The `NOORIENT` keyword on the `GEOM` command line is compulsory. We also recommend an SCF convergence threshold of at least 10^{-6} (as shown).

Options: **DIPD** **POLD** **FIELD**=<real> **PRINT**=<integer>

DIPD: calculate dipole derivatives

POLD: calculate polarizability derivatives

FIELD=<real>: applied field (default is 0.0005 au)

PRINT=<integer>: controls the amount of printout (larger integer - more printout).

If no additional options are specified, just the polarizability is calculated. At the end of the calculation any derivative quantities computed will be found (as Cartesians) in the <deriv> file. The polarizability derivatives require the gradient at the initial geometry, and so a `FORCE` command should be run *before* starting the `NUMP` loop.

See example 17 in the RUNNING JOBS section.

****IMPORTANT**** In a complex input, involving the calculation of several properties at the final geometry, the `NUMP` loop (typically followed by `FREQ`) should either be done ***LAST*** or an additional `GEOM` card should be inserted after it to restore the geometry.

14. FREQUency command

Calculates vibrational frequencies from an existing Hessian matrix (taken from the <hess> file), infrared (and possibly Raman) intensities using dipole moment and polarizability derivatives (taken from the <deriv> file), VCD rotational strengths, as well as thermodynamic parameters in the rigid rotor/harmonic oscillator approximation.

Options: **TEMP**=<real>,<real>,<real> **PRES**=<real>,<real>,<real> **LOWF**=<real>
MASS=<string> **PRINT**=<integer>

TEMP=<real>,<real>,<real>: Range of temperatures for which a thermodynamic analysis should be carried out. The first value is the starting temperature, the second the ending temperature and the third the step size. All values are in degrees K.

All temperatures given must be positive, and the second must be greater than the first. If only one temperature is given, the analysis will be carried out at that temperature; if two values are given without a step size, then the analysis will be performed at both temperatures. If nothing is given the default temperature is 298.18 K (25°C)

PRES=<real>,<real>,<real>: Range of pressures for which a thermodynamic analysis should be carried out. The first value is the starting pressure, the second the ending pressure and the third the step size. All values are in atm.

All pressures given must be positive, and the second must be greater than the first. If only one pressure is given, the analysis will be carried out at that pressure; if two values are given without a step size, then the analysis will be performed at both pressures. If nothing is given the default pressure is 1 atm.

LOWF=<real>: Smallest magnitude frequency that will be printed; all frequencies with magnitude less than this will be removed as “zero” and not printed. (Default 10 cm⁻¹.)

MASS=<string>: options are **ABUN** to use the mass of the most abundant isotope for each element or **AVER** (the default) to use the isotope-averaged atomic mass. Alternatively atomic masses can be provided directly during geometry input

PRINT=<integer>: print level (needed only for diagnostics).

****IMPORTANT**** FREQ will carry out a frequency analysis on *any* Hessian matrix that is in the <hess> file, *including the approximate Hessian left over from a geometry optimization*. For a reliable analysis, make sure that an exact Hessian is available (via HESS or NUMHess) **before** the FREQ command line.

15. SQM command

Scales a Hessian matrix using a set of standard SQM scale factors. (See the description of the stand-alone SQM program at the back of this manual.) Additional scale factors can be read in, either directly embedded in the input or from an external file. There must be an existing (Cartesian) Hessian matrix on file which is read in, converted to primitive internals, scaled, converted back to Cartesian coordinates and written back to the Hessian file. The result should be an SQM Hessian that can be used in the frequency module to give a full SQM vibrational analysis.

The direct SQM implementation in PQS does NOT have the capability to optimize the scale factors. It simply scales a Hessian using existing (known) scale factors. Scale factor optimization requires the stand-alone SQM program.

Options: **NODE** **FILE**=<string> **PRINT**=<integer>

NODE: Denotes NO default scaling. The default is to automatically scale the Hessian using the scale factors derived for molecules containing the elements H, C, N, O and Cl at the B3LYP/6-31G* level in the original primitive SQM paper [113]. Thus SQM alone with no other options will scale any Hessian matrix found on the <hess> file using the following scale factors:

stretch	X-X	0.9207	bend	X-X-X	1.0144
stretch	C-Cl	1.0438	bend	X-X-H	0.9431
stretch	C-H	0.9164	bend	H-C-H	0.9016
stretch	N-H	0.9242	bend	H-N-H	0.8753
stretch	O-H	0.9527	torsion	all	0.9523
			linear bends	all	0.8847

where X denotes a non-hydrogen atom (C, N or O).

If **NODE** is specified then scale factors must be provided, either in a separate file via the **FILE** command (see below) or embedded in the input deck.

FILE=<string>: where <string> is the name of a file (maximum 24 characters) containing additional scale factors. The expected format is similar to that in SQM proper

```
$scale
stre X X 1 0.9207
bend X X X 2 1.0108
tors X X X X 3 0.9523
$endscale
```

The scaling information must be given inside the `$scale` and `$endscale` delimiters either in a separate file defined by <string> or embedded in the input deck following the SQM command. If **NODE** is specified then these will be the only scale factors used; if not they will be used in addition to the default scale factors (see above).

PRINT=<integer>: print level (needed only for diagnostics).

It is strongly recommended to read the input description for SQM version 2.0 at the back of this manual.

16. NMR command

This command calculates nuclear magnetic shieldings by the GIAO (Gauge-Including Atomic Orbital) method [46,47]. Available for HF and all supported DFT [48,49] wavefunctions. Closed-Shell ONLY. The PQS NMR module is very efficient, and the convergence of the coupled-perturbed Hartree-Fock step has recently been improved.

Options: **THR1**=<real> **THR2**=<real>,<real> **THREshold**=<real>
ITERations=<integer> **GAUge**=<integer> **LIMits**=<integer>,<integer>,<integer>
LVShift=<real> **NOCP**hf **MAL**Kin=<string> **VCD** **PRIN**t=<integer>
NICS=<integer>,<integer>,<integer> **DISP**=<real> **GRID**=<integer> **STEP**=<real>

In most applications the defaults will work well and only **NMR** is needed.

THR1=<real>: This is the neglect threshold for the one-electron GIAO integrals, in pH form, i.e., the <real>=-log(threshold1). The default is 10 (threshold1=1.0E-10).

THR2=<real>,<real>: These are integral thresholds, thre2 and thre3, in the coupled-perturbed Hartree-Fock program, in pH form (see above). thre3 is used in the preliminary CPHF steps. The default is currently 10 (i.e., 1.0E-10) for both.

THREshold=<real>: required accuracy for the first-order density matrix in the Coupled-Perturbed Hartree-Fock equations, in pH form. The default is 5, i.e., 1.0E-5.

ITER=<integer>: sets the maximum number of CPHF cycles. The default is 30 which is normally adequate. If a calculation fails to converge in 30 cycles, it probably never will.

GAUge=<integer>: The gauge origin will be the n-th atom if n is the integer.

LIMits=<integer>,<integer>,<integer>: See **INTE** and **FORCe** for this option. It is usually not required but may be necessary for high angular momentum functions (*g,h,...*) and large basis sets.

LVShift=<real>: *DFT wavefunctions only*. This option specifies a level shift for the virtual orbitals. Using an appropriate level shift can dramatically improve the accuracy of calculated NMR shieldings [50]. The recommended level shift [50] is 0.025 E_h for the BLYP, B3LYP, OLYP and O3LYP functionals. *Note - this option should not be confused with the use of a level shift to improve SCF convergence.*

NOCPhf: This suppresses the solution of the coupled-perturbed Hartree-Fock (or Kohn-Sham) equations. It is automatically invoked for the Wilson-Amos-Handy (WAH) functional, see [28].

MALKin=<string>: This option directs the program to perform the Malkin correction on the orbital energies [51], in effect a variable level shift. Malkin *et al.* [51] calculate the level shift using the simple Slater (HFS) functional [14] even if the DFT functional used is different. Our program allows the specification of the functional to be used to calculate the level shifts. **MAL**Kin=HFS recovers the original Malkin correction, while **MAL**Kin=B3LYP (in a B3LYP calculation) is more consistent, in that the same functional is used to determine the exchange-correlation energy *and* the level shift (in this case B3LYP). See ref. [50].

VCD: Does the necessary calculations (of the atomic axial tensors) for the computation of VCD rotational strengths. Note that the actual calculation is in two parts, with the first part done in the NMR module, and the second in the Hessian module. The final

rotational strengths are evaluated and printed in the Frequency module.

PRINT=<integer>: controls the amount of printout (larger integer - more printout).

NICS=<integer>,<integer>,<integer>: NICS (Nuclear-Independent Chemical Shifts) is a now well-established technique [52,53] often used as a magnetic criterion for aromaticity. This option defines a plane, P, passing through three atoms as defined by the three integers, with the chemical shift evaluated at various points in the plane. Take care that the three integers correspond to three real atoms in the system under study. (If only two integers are given, then "grid" points are generated along a line) Other options include

DISP=<real>: The plane P (above) is displaced perpendicularly by the value given (in bohr).

GRID=<integer>: defines the number of points on the plane P. If we call the integer value n_g , the grid points are generated on an $n_g \times n_g$ square, with one additional point in the center of the grid (the center of the three atoms) if n_g is even. The default if this parameter is not specified is a single grid point.

STEP=<real>: distance between the grid points along the sides of the grid square. The default is 1.0 bohr.

The following options are obsolescent, but still available

FOR=<integer>: This option, if present, instructs the program to calculate NMR shieldings only for the n -th nucleus where n is the integer. n must be between 1 and the total number of nuclei, otherwise no shielding is calculated. The default is to calculate shieldings for all symmetry-unique nuclei. If several nuclear shieldings need to be calculated, the NMR card must be repeated. This will *not* repeat the whole calculation. Note that the savings obtained by calculating only a few nuclei are not great because most of the work goes into calculating the first-order wavefunction, and this has to be done whether one or a 100 shieldings are calculated. An alternative way to specify the atoms for which magnetic shieldings must be calculated is to list the name of the atom on the NMR card (see options **HYDRO**gen to **CHLO**rine and **DUMM**y below).

HYDROgen, **BORO**n, **CARBO**n, **NITRO**gen **OXYGE**n **FLUO**rine **SODI**um **MAGNE**sium **ALUMI**nium **SILI**con **PHOS**phorus **SULFU**r, **CHLO**rine **DUMM**y: Calculate and print magnetic shieldings only for the atom types listed on the NMR card. Specifying the atoms this way is frequently simpler than using the **FOR** card. Note that the atoms listed above are the only ones included in the program. To calculate, e.g. thallium magnetic shieldings, either use the **FOR** card, or simply calculate all shieldings (no option cards).

NOTE: To convert the calculated NMR shieldings into (relative) chemical shifts requires subtraction of the calculated shieldings from those of a standard. A larger shielding results in a lower shift; if the shielding of the atom is larger than that of the standard, the shift will be negative. For consistency, nuclear magnetic shieldings for the standard should be computed at the same level of theory as the calculated shieldings.

16a. Vibrational Circular Dichroism (VCD)

Allows the computation of VCD rotational strengths in chiral molecules. The implementation of this in PQS is similar to that of Stephens and coworkers [54].

Strictly speaking, this is not a separate command, but an option to the **NMR** command. However, it requires both an NMR *and* a Hessian calculation, and they must be done in the correct order, with the NMR part first. Typical usage would be

```
NMR VCD  
HESS  
FREQ
```

The actual rotational strengths are printed out as a part of the frequency analysis.

See example 37 in the RUNNING JOBS section.

17. MP2 command

Calculates both closed- and unrestricted open-shell canonical second-order Møller-Plesset (MP2) correlation energies (also called second-order Many-Body Perturbation Theory, MPPT(2)). On PQS systems (the QuantumCube™) the MP2 code can handle systems containing up to 2000 basis functions on a single node and more if the job is run in parallel over multiple nodes. Although canonical (not local) MP2 energies are calculated, there are efficiency/accuracy advantages in localizing orbitals, and **LOCA=PIPEK is now compulsory in the preceding SCF step.**

Options: **MAXDisk**=<real> **NOFRozen** **THREshold**=<integer> **CORE**=<real>
ORBS=<integer>,<integer> **SCS**=<real>,<real> **SOS** **NOIO** **DUAL**=<integer> **GRAD**
REStart **PMIJ**=<integer> **PRINT**=<integer>

MAXDisk=<real>: maximum amount of scratch disk storage allowed (in GB); in parallel jobs the maximum disk storage *per node/process*. The default value is 50 GB. The bulk of the disk storage is required for the half-transformed integrals; if there is insufficient to store all transformed integrals, the job will crash. The second half-transformation step can be done in multiple passes and requires only a couple of GB (maximum), although it will use more if this is available. If a calculation crashes in the sort phase, the half-transformed integral file is not removed and the calculation can be restarted.

Disk storage required for the half-transformed integrals is given by $S = 5N(N+1)(n^2+2)/2$ where S is the storage in bytes, N is the number of basis functions and n is the number of correlated orbitals. This formula holds for closed-shell non-symmetrical systems, for (Abelian) symmetry S should be divided approximately by the number of symmetry operations, with the proviso that only symmetry operations that transform atoms into different atoms count, e.g., for naphthalene (D_{2h}) the molecular plane leaves all atoms stationary, and thus the number of effective symmetry operations is 4 *not* 8. In a parallel job, disk storage is evenly distributed across the nodes, so the storage calculated via the above formula should be divided by the number of slaves.

Disk storage actually *requested* should account for the second half-transformation as well as the first. Maximally, this can be twice as much as that required for the half-transformed integrals, so - if available - you should ask for twice the calculated value of S. **Do not ask for more disk storage than is available on your system.** If you are limited by disk capacity, the second-half transformation can be carried out (with multiple passes) in as little as 2-3 GB.

For example, consider C_{72} , no symmetry, 6-311G* basis set. Here $N=1332$ and $n=148$ for a frozen-core calculation. The disk storage needed for the half-transformed integrals for a closed-shell RMP2 energy is 97.2 GB. The recommended value for **MAXDisk** for a single-processor job is 200 and should be at least 100. Run in parallel on, say, 4 processors, these values can comfortably be reduced to 50 (recommended) and 27 (minimum).

Because there are both α - and β -spin MOs and hence three different integral types, a UMP2 energy calculation requires approximately four times the disk storage of the corresponding RMP2 energy.

NOFRozen: core orbitals will also be correlated; the default is to correlate only the valence orbitals.

THREshold=<real>: Set the integral threshold to $10^{-\text{real}}$. E.g., `THREsh=9` sets the integral threshold to 10^{-9} . The default threshold is set to the lower of 10^{-10} or the square of the lowest eigenvalue of the overlap matrix (which is calculated in the SCF step). The latter is a good measure of basis set stability - too low a value and the basis has severe linear-dependency problems. The threshold has a double effect. First, a lower threshold lowers the computational cost but also the accuracy. The threshold for a single-point MP2 energy can normally be lower than for the SCF (but should not be reduced if an MP2 gradient is subsequently calculated). Regardless of the magnitude of the lowest eigenvalue, the threshold will not be set lower than 10^{-12} unless forced (not recommended). Half-transformed integrals are stored in integer form, occupying 5 bytes of storage - if the threshold is set too low, large integrals may overflow this storage. Formerly the calculation simply stopped, but we have now introduced an automatic threshold reduction which reduces the threshold for any integrals which would otherwise overflow (albeit with some potential for loss of accuracy in the final computed MP2 energy). Integral overflow is unlikely to happen with the default threshold and the usual basis sets but very large basis sets, especially if they contain diffuse functions, may cause this problem.

CORE=<real>: e.g., `CORE=-2.7`. Orbitals with energies *lower* than `CORE` are considered to be core orbitals and are *not* correlated. The default is `-3.0 au`.

ORBS=<istart>,<iend> or **ORBS**=(istart,iend): Both integers. Correlate only orbitals between `istart` and `iend` (inclusive).

SCS=<real>,<real>: Spin-component scaled MP2. This was originally introduced by Grimme [55], who considered separate scaling of the MP2 energy contributions from antiparallel-spin ($\alpha\beta$ “singlet”) and parallel-spin ($\alpha\alpha$, $\beta\beta$ “triplet”) electron pairs. Grimme’s final least-squares scaling factors were 6/5 for the “singlet” scaling and 1/3 for the “triplet”. A later paper from the Head-Gordon group [56] - scaled opposite-spin (SOS) MP2 - eliminated the “triplet” contribution altogether, and scaled the “singlet” contribution by 1.3.

The two optional parameters to the `SCS` option are read-in scaling factors for the “singlet” and “triplet” contributions, respectively. Specifying `SCS` alone, without any scaling factors, is equivalent to Grimme’s original scaling. The Head-Gordon scaling can be accessed via `SCS=1.3,0.0`. (See also the `SOS` keyword.)

SOS: requests Head-Gordon’s scaled opposite-spin MP2. There are no time savings requesting this option for closed-shell systems, but there can be significant savings in UMP2 calculations as the same-spin $\alpha\alpha$ and $\beta\beta$ components do not need to be calculated.

Both `SCS` and `SOS` scaling are also available in the MP2 gradient allowing for closed-shell scaled MP2 geometry optimizations.

NOIO: requests the in-core algorithm for the second half-transformation for UMP2 energy calculations. (Not available for RMP2.) This is slower than the default bin sort algorithm, but requires no additional disk storage. Will be used by default if disk storage is at a premium for serial calculations.

DUAL=<integer>: Dual basis set calculation. Available for closed-shell RMP2 energies only. The idea here is that high angular momentum functions may be needed to adequately describe electron correlation, but have little effect in the SCF. This allows one to use a smaller basis for the SCF, and a larger one during the MP2 step.

The integer option must be either 1 or 2. The former gives the original approach as developed by Wolinski and Pulay [57]; the latter gives the modified approach of Head-Gordon and coworkers [58]. Specifying **DUAL** alone is equivalent to **DUAL=1**.

To carry out a dual basis MP2 calculation first complete a small basis SCF calculation, then include in the input file

```
BASIS=<larger basis>
GUESS=READ
MP2 DUAL <other possible MP2 options>
```

The larger basis set **must** be an extension of the smaller. Note also that **the GUESS card is mandatory and there is no SCF between the GUESS and the MP2 cards**.

GRAD: Forces the program to compute and store quantities needed for a subsequent MP2 gradient calculation even if the **FORCE** keyword is not included in the input file. Normally for timing or diagnostic purposes only.

REStart: If this option is present, the program assumes that the half-transformation is finished, and the half-transformed integrals are present on the file <scratchdir>/<jobname>.htr (in the Windows version: <scratchdir>\<jobname>.htr), and only the sorting and second half-transformation are carried out.

PMIJ=<integer>: Print the exchange matrix for pair (integer). The orbital pairs are numbered as 1,1 2,1 2,2 3,1 3,2 3,3 4,1, etc... Diagnostic only.

PRINT: print level. Much output can be produced by using **PRINT=3** or higher.

Degeneracy. The current implementation cannot handle, in general, symmetries with degenerate representations. Most likely, an error message, "Orbitals do not conform to symmetry", will appear. Two remedies are available: (1) distort the system slightly (10^{-4} au) to break the symmetry, or switch off symmetry (with **SYMM=0.0** on the **GEOM** card).

Memory requirements. The greatest memory demand arises in the calculation of integrals. The program must be able to hold all integrals ($M\nu|L\sigma$) where M and L are shells and ν, σ are contracted basis functions. This adds up to S^2N^2 where S is the largest shell size (5 for d's, 7 for f's etc...), and N is the number of basis functions. For large shells (e.g. G15 or H21) this is a considerable amount of memory. However, it scales only quadratically with the number of basis functions. This memory ought to be *real* memory, and the program must be told this, or else severe paging problems may occur. For calculations employing f functions ($S^2=49$), at least 400 MB fast memory is needed for $N=1000$, and at least 256 MB for $N=800$. This is not especially restrictive but for larger shells (G15 etc...) the problem becomes more severe. We have recently modified the MP2 integral routines to allieviate this memory bottleneck, but it is still fairly demanding.

Practical differences between RMP2 and UMP2 energy calculations. The main difference, already noted, is that a UMP2 energy requires approximately four times the disk storage of a corresponding RMP2 energy. If disk storage is a likely issue it is recommended that UMP2 energies in particular be run in parallel over multiple nodes which allows the aggregate disk storage over all the nodes to be utilized. It is also recommended not to commit all CPUs on a given node to an MP2 energy calculation as this will fully utilize all available compute resources and adversely affect disk I/O, slowing down both the bin sort and integral unpacking.

Typically a UMP2 energy will take between 1.5 and 3.0 times as long as a similar RMP2 energy depending on whether the first or the second half-transformation is dominant. Most commonly, for large basis sets, it will be the former.

For further details on the MP2 algorithm, both serial and parallel, see the published literature [59,60,61].

MP2 Gradient

Analytical gradients are now available for closed-shell, canonical MP2 wavefunctions. The code is still somewhat preliminary (for example there is *no symmetry* and it is *serial only*) but we have included it because, despite its inefficiencies, it is still faster and can handle larger systems than in many other programs.

Full details of the algorithm have been published [62], but from a practical point of view the memory requirements are somewhat greater and disk requirements two-three times greater than for the corresponding RMP2 energy calculation. The time (elapsed) required to compute an RMP2 gradient is typically around three times longer than for the corresponding RMP2 energy (summing times for both the SCF *and* MP2 steps).

As far as PQS input is concerned, there is no specific MP2 gradient keyword, and the MP2 gradient is computed as part of the `FORCE` keyword. A typical RMP2 geometry optimization loop would be

```
OPTIM
SCF LOCA=PIPEK
MP2
FORCE
JUMP
```

with the program itself determining which wavefunction to compute the gradient for (HF or MP2) by parsing the input file.

See example 31 in the RUNNING JOBS section.

18. CORR command

This command requests the calculation of post-Hartree-Fock single-point energies. This is a new module which supports a number of different electron correlation methods. CLOSED-SHELL ONLY.

Options: **CORR**=<string> **TRIPles** **NOFRozen** **THREshold**=<real> **CORE**=<real>
ORBS=<integer>,<integer> **LVSH**=<real> **DIIS**=<integer>,<integer>
CONV=<real>,<real> **MEMOry**=<integer> **PRINT**=<integer> **ITER**=<integer> **CHKP**
DUMP **REStart**

CORR: the electron-correlation method. Possible wavefunction types, all derived from a closed-shell Hartree-Fock single determinant, are:

MP2	second-order Møller-Plesset perturbation theory
MP3	third-order Møller-Plesset perturbation theory [63]
MP4	fourth-order Møller-Plesset perturbation theory [64]
CID	configuration-interaction, double excitations [65]
CISD	configuration-interaction, single and double excitations
CEPA-0	coupled-electron pair approximation, level 0 [ref]
CEPA-2	coupled-electron pair approximation, level 2 [66]
QCISD	quadratic configuration-interaction, single and double excitations [67]
CCD	coupled-cluster, double excitations [68]
CCSD	coupled-cluster, single and double excitations

Within the limitations of the basis set, the best possible wavefunction (which gives the exact energy) is what is known as a full CI. This starts from the Hartree-Fock SCF wavefunction and considers all possible excitations of all electrons from the occupied HF orbitals into all of the virtuals. It can be written as

$$\Psi^{\text{CI}} = C_0 \Psi^{\text{HF}} + \sum C_i^a \Psi_i^a + \sum C_{ij}^{ab} \Psi_{ij}^{ab} + \sum C_{ijk}^{abc} \Psi_{ijk}^{abc} + \dots$$

where the coefficients are variational parameters and the wavefunction contains all possible single excitations (from occupied MO i into virtual MO a), all possible double excitations (from i,j into a,b), all possible triple excitations (from i,j,k into a,b,c) and so on. A full CI is prohibitively expensive for all but the smallest systems and all of the correlated methods listed above can be considered as approximations to it.

Perhaps the intuitively most obvious way to approximate the full CI wavefunction is to truncate it at a certain excitation level, for example ignore anything beyond double excitations. This gives rise to the CID and CISD methods. Unfortunately any method based on this type of truncation has the disadvantage that it is not *size-consistent*. An example should make this clear. Consider the (hypothetical) He_2 molecule. This is not bound at all at the Hartree-Fock level and minimally bound with the inclusion of electron correlation. If you considered the dissociation of He_2 at the CISD level (ignoring the effects of basis set superposition error), then as far as the individual He atoms are concerned you have the best answer possible, because for He, which only has two electrons, CISD is equivalent to a full CI. However, for He_2 , CISD has clearly missed out the contribution from triple and quadruple excitations, hence its energy is not as “good” as for the dissociated He atoms. This is clearly a general phenomenon, and at any given level of truncation the energy of a system is less accurate than that of its

dissociated components. All of the other correlated methods mentioned can be considered as approximations to a full CI which maintain either approximate or exact size-consistency, i.e., the energy of a smaller fragment of the whole is as “good” as that of the whole.

The Møller-Plesset wavefunctions (**MP2**, **MP3**, **MP4**) are exactly size consistent order-by-order. The MP series is related to the corresponding CI series (full MP4 for example includes all excitations up to quadruples) except that the coefficients are fixed by the perturbation expansion and are not variational. They are consequently a lot less expensive to compute than the corresponding CI wavefunction. Higher order MP calculations will automatically compute all lower order MP energies (so requesting an MP4 energy will also give both the MP2 and MP3 energies). Note that the MP2 energy is computed as a matter of course with all other correlated methods. It should not be requested on its own in this module (`CORR=MP2`) as the stand-alone MP2 energy (via the **MP2** command) is much faster if this is all you want.

The triples contribution to the MP4 energy is dominant in terms of computational time and requesting `CORR=MP4` will give only MP4SDQ (i.e., will *not* compute the triples contribution). If you want this then add the **TRIPles** command (see below).

Coupled cluster wavefunctions are size-consistent. The coupled electron pair approximation (CEPA), of which there are many variants, is approximately size-consistent. QCISD is an exactly size-consistent CISD (or, depending on your point of view, a simplification of CCSD which retains size consistency).

Note that, aside from the Møller-Plesset wavefunctions, all the other methods are approximately (within a factor of about two) the same cost. The best balance between cost and accuracy is probably QCISD.

TRIPles: include triples contributions (calculated perturbatively). Applicable to MP4, CCSD and QCISD wavefunctions. In the case of the latter two methods, results in the CCSD(T) and QCISD(T) energies. Note that, as mentioned earlier, `CORR=MP4` *without* `TRIP` will give only the MP4SDQ energy. Note: The triples contribution is expensive and is normally the dominant component in any wavefunction in which it is included.

NOFRozen: core orbitals will also be correlated; the default is to correlate only the valence orbitals.

THREshold=<real>: set the integral threshold to $10^{-\text{real}}$. E.g., `THREsh=9` sets the integral threshold to 10^{-9} . The default threshold is set to the lower of 10^{-10} or the square of the lowest eigenvalue of the overlap matrix (which is calculated in the SCF step). The latter is a good measure of basis set stability - too low a value and the basis has severe linear-dependency problems. The threshold has a double effect. First, a lower threshold lowers the computational cost but also the accuracy. Regardless of the magnitude of the lowest eigenvalue, the threshold will not be set lower than 10^{-12} unless forced (not recommended). Unlike in the stand-alone MP2 algorithm, partially-transformed integrals are stored in full real*8 precision.

CORE=<real>: e.g., `CORE=-2.7`. Orbitals with energies *lower* than `CORE` are considered to be core orbitals and are *not* correlated. The default is -3.0 au.

ORBS=<integer>,<integer>: the first integer is the *lowest* occupied orbital to be correlated; the second is the *highest* virtual to be included in the excitation space.

LVSH=<real>: artificially shifts the energies of the virtual orbitals to help convergence (typical values from 0.1 to 4.0). Available for CI- and CC-based wavefunctions. Default value is 0.0.

DIIS=<integer>,<integer>: convergence acceleration. The first integer is the size of the DIIS subspace (default 6) and the second is the iteration on which DIIS is turned on (default 1). To switch DIIS off, specify `DIIS=0`.

CONV=<real>,<real>: threshold for convergence on the energy and the squared residuals, respectively, in pH format (i.e., threshold is $10^{-\text{real}}$). Default is 10^{-6} in both cases.

MEMOry=<integer>{unit}: specifies how much dynamic memory may be used in CORR calculations. Unfortunately, the memory handling is separate to, and independent of, the memory handling in the rest of the program and must be specified explicitly. Other than this, usage is similar to that for the more general **%MEM** command. In particular if no unit (MB or GB) is given and <integer> is small (<2000) it will be assumed to be in megawords (e.g., 7 will be interpreted as 7,000,000 double words); if <integer> is large (≥ 2000), it is interpreted as words. The default if this keyword is absent is 10 (10,000,000 double words = 80 MB) which is nowhere near enough for larger systems.

PRINT=<integer>: print level. Much output can be produced by using `PRINT=3` or higher (default is 1).

ITER=<integer>: maximum number of iterations requested (default is 20).

CHKP: save the amplitudes to disk at the end of each iteration. Should be requested for large systems to provide the option for a restart

DUMP: save the amplitudes to disk at the end of the job (either at convergence or if the maximum number of iterations is exceeded).

REStart: if requested alone will attempt to restart a failed job from the amplitudes currently stored on disk (see the **CHKP** option); if requested together with **TRIP** will assume that the stored amplitudes are converged and compute the triples contribution.

All correlated energies can be computed in parallel. The parallel efficiency is not as high as, for example, in the SCF code, because to calculate any correlated energy is highly computationally intensive, both in terms of memory and I/O. There are typically several partial integral transformations/rearrangements (compared to just one for the stand-alone MP2 energy algorithm) and multiple bin sorts. To aid the development of the correlated code, a generic I/O routine called array files was utilized [69]. This is a generalized parallel I/O routine which determines, and keeps track internally, on which node each record is written, removing the burden of doing this from the writer of the code. The side effect is that the increased generalization reduces the efficiency.

Note that in the parallel MPI version of this module (although not in the PVM version), one CPU on each node is reserved by the program as an I/O processor. So if the user requests n slaves on a particular node for his/her MPI job, only $(n-1)$ of these will be available as compute nodes. In the PVM version an extra process is spawned to handle the I/O, so no existing processes are "lost".

For further details of the algorithm, see [70,71]; for some applications see [72,73].

19. POP command

This command carries out population analysis on the wavefunction, computing and printing out atomic charges, bond orders, atomic valencies, free valencies (for unrestricted wavefunctions) and (optionally) gross orbital occupancies.

Options: **POP**=<string> **PThrsh**=<real>

POP=<string>: type of analysis requested. Should be one of

- ◇ **MULLIKEN**: Mulliken analysis
- ◇ **LOWDIN**: Löwdin analysis
- ◇ **CHELP**: Charges from electrostatic potential
- ◇ **FULL**: All three analyses, including gross orbital occupancies

POP alone, without any defining string, does both Mulliken and Löwdin analyses *without* printing the gross orbital occupancies.

PTHRsh=<real>: threshold for printing the bond orders

Only bond orders of magnitude greater than PTHRsh will be printed (default 0.01).

Notes

Population analysis is an attempt to interpret the wavefunction in terms of classical chemical concepts such as bond order (single, double, triple bonds etc...) and atomic valency. In open-shell systems, atoms with a high free valency are presumerably more “reactive” than atoms where the free valency is lower.

The essential quantity in a Mulliken analysis [74] is **PS** (density x overlap), whereas in a Löwdin analysis [75] it is the symmetrized equivalent $\mathbf{S}^{1/2}\mathbf{PS}^{1/2}$. Use of these two quantities influences the analysis in different ways. For example, gross orbital occupancies (g_i) in a Löwdin analysis satisfy $0 \leq g_i \leq 2$ for closed-shell systems, which is just what we would expect, whereas for a Mulliken analysis this does not necessarily hold and one can get unphysical *negative* occupancies. On the other hand, Löwdin bond orders are always positive and hence do not allow for unfavorable (negative) interactions between atoms in a molecule. Overall, the Löwdin analysis is more stable; the Mulliken analysis can be thrown way off if there are diffuse functions in the basis set [76].

CHELP is now a fairly common procedure which involves computing the electrostatic potential on a grid of points surrounding the molecule and deriving charges on the various atoms that best reproduce it. There are various implementations, including some that also attempt to reproduce the dipole moment; the version in PQS follows that of Singh and Kollman [77], as amended by Breneman and Wiberg [78], and produces best-fit atomic charges.

Note that if dipole derivatives are available on the <deriv> file (from a numerical or analytical Hessian calculation), then the population analysis of Cioslowski [79] is carried out, which derives atomic charges from the trace of the dipole derivatives tensor. These charges can often appear to be chemically more “sensible” than those derived from the other analyses.

20. NBO command

This is the NBO program of Prof. Frank Weinhold (University of Wisconsin). PQS (optionally) includes NBO version 5.0. The official NBO program manual is provided with the PQS documentation and should be consulted for a full list of options, keywords and references. NBO 5.0 has a limitation of 200 atoms and 2000 basis functions.

The NBO program performs an analysis of a many-electron molecular wavefunction in terms of localized electron-pair bonding units. It can determine natural atomic orbitals (NAOs), natural hybrid orbitals (NHOs), natural bond orbitals (NBOs), and natural localized molecular orbitals (NLMOs), and use these to carry out a natural population analysis (NPA), an NBO energy analysis, and other localized analyses of wavefunction properties, including natural resonance theory (NRT) analysis.

There are two methods for accessing NBO within a PQS run. For the basic NBO analysis, simply add the keyword `NBO` after the `SCF` keyword. For additional NBO options, specify `NBO` as above, then on the next line type `$NBO` followed by the desired NBO keyword(s), followed by `$END`. Then on the next line type `ENDNBO`.

For example, to request a molecular dipole moment analysis in addition to the basic NBO analysis (using the additional NBO keyword `DIPOLE`).

```
...
SCF
NBO
$NBO  DIPOLE  $END
ENDNBO
...
...
```

Note that the NBO keyword alone gives an NPA analysis by default. See examples 8 and 10 in the RUNNING JOBS section.

21. PROPErty command

Computes selected properties at the nucleus for each atom in the system. Currently the charge and spin (for open shell) densities and the electric field gradient are available for HF and DFT wavefunctions.

All properties are calculated numerically over atom-centered grids (similar to those used for DFT wavefunctions). There are two methods coded for computing the charge/spin density, the standard delta function approach (which involves expectation values evaluated at the nucleus) and a Gaussian-weighted operator originally developed by Rassolov and Chipman [80] for MCSCF wavefunctions and extended for use with DFT wavefunctions [81]. The latter has a short effective range about the nucleus and samples more of the wavefunction than does the delta function method; in this way it is hoped to reduce the basis set dependence of the delta function method and compensate somewhat for the fact that Gaussian basis functions do not have the correct asymptotic behavior near the nucleus.

Options: **SPIN** **EFG** **RADF=<real>** **LMAX=<integer>** **GRID=<real>**
PRINT=<integer>

SPIN: calculates the charge and spin densities (for open shell). Both delta function (Fermi contact) and Rassolov/Chipman densities will be computed.

EFG: calculates the electric field gradient

RADF=<real>: sets the radial factor for the Rassolov-Chipman operator. Typical values are 0.1 to 0.5 (default 0.35).

LMAX=<integer>: orbitals are expanded in spherical harmonics and LMAX sets the maximum angular momentum value used in the expansion. The default is LMAX=4 which is normally perfectly adequate. LMAX should be a positive integer between 0 and 17. Do NOT set LMAX greater than 17 as the default angular grid used (with 110 angular points) cannot integrate beyond this value.

GRID=<real>: controls radial grid quality. Larger values, more radial grid points. Suggested values are between 0.5 and 2.0 (default 1.0). See the SCF command.

PRINT=<integer>: controls the amount of printout (larger integer - more printout).

22. COSMO command

Requests a calculation using the Conductor-like Screening Model (COSMO) [82] to model the effect of a solvent on the system under study. In this model the solute forms a cavity in a dielectric continuum representing the solvent. The size of the cavity is defined by the solvent accessible surface (SAS), which is constructed on the basis of the molecular geometry.

COSMO is available for energies and gradients using Hartree-Fock, MP2 and DFT wavefunctions. (Second-derivatives are accessible via the `NUMHess` command.) It is switched on by the presence of the `COSMO` keyword (and any associated options) in the input deck (normally after the geometry and basis set have been defined). All subsequent `SCF` and `FORCE` commands will be affected. It can be switched off by adding the line `COSMO OFF` later in the input file. Note that COSMO performs best for polar solvents (large dielectric permittivity values) and results for weak dielectrics are less reliable.

The default settings of the COSMO module have been tailored for application of the COSMO-RS (COSMO for real solvents) theory [83]. In particular, the COSMO module will produce a file (`<jobname>.cosmo`) that can be used as input to the COSMOtherm suite of programs for the calculation of solvation mixture thermodynamics. The COSMOtherm package is distributed by *COSMOlogic* GmbH & Co.KG (web site: <http://www.cosmologic.de>). The COSMO-RS parameters have been optimized for DFT calculations using the BVP86 functional and the `svp_ahlrachs` and `tzvp_ahlrachs` basis sets, thus the recommended settings for running a COSMO-RS calculation are `DFTP=BVP86` on the `SCF` command line, and `BASIS=svp_ahlrachs` or `BASIS=tzvp_ahlrachs` as basis set choice.

NOTE: The heart of the COSMO code was kindly provided by *COSMOlogic*. Due to technical limitations, gradients with COSMO switched on are not as numerically reliable as normal non-COSMO gradients. Consequently, during geometry optimizations you may see the energy rise slightly at the end of the optimization, i.e., the gradient "zero" is not the true energy minimum. Such discrepancies are usually chemically insignificant.

Options: **EPSI**=<real> **RSOL**=<real> **SOLV**=<string> **RADI**=<string> **ROUT**=<real>
DISE=<real> **LCAV**=<integer> **NPPA**=<integer> **NSPA**=<integer> **AMPR**=<real>
PHSR=<real> **OFF**

EPSI=<real>: electrical permittivity of the dielectric continuum. The default is infinity if no solvent is specified, otherwise it is the value for the chosen solvent.

RSOL=<real>: additional (atomic) radius (Å) for solvent accessible surface (SAS) construction. The default is 1.3 Å if no solvent is specified, otherwise it is the value for the chosen solvent.

SOLV=<string>: name of the solvent to be used. The default corresponds to the settings for a COSMO-RS calculation (for which the **SOLV** keyword must not be present). Specifying the solvent will set the values of **EPSI** and **RSOL** (see above) to predefined values for the chosen solvent, as follows:

Solvent	alias(es)	rsol (Å)	epsi
water	h2o	1.39	78.39
methanol	ch3oh	1.86	32.63
ethanol	c2h5oh, ch3ch2oh	2.18	24.55
acetone	ch3coch3	2.38	20.70
ether	ch3ch2och2ch3	2.79	4.34
acetonitrile	ch3cn	2.16	36.64
nitromethane	ch3no2	2.16	38.20
carbontetrachloride	ccl4	2.69	2.23
chloroform	chcl3	2.48	4.90
dichloromethane	ch2cl2	2.27	8.93
dichloroethane	ch2clch2cl	2.51	10.36
dimethylsulphoxide	dmsol	2.46	46.70
aniline	c6h5nh2	2.80	6.89
benzene	c6h6	2.63	2.25
toluene	c6h5ch3	2.82	2.38
chlorobenzene	c6h5cl	2.81	5.62
tetrahydrofuran	thf	2.56	7.58
cyclohexane	c6h12	2.82	2.02
n-heptane	heptane, c7h16	3.13	1.92

Solvents that are not listed can be simulated by explicitly entering appropriate values for **RSOL** and **EPSI**.

RADI=<string>: Type of atomic radii to be used for SAS construction. Should be one of

- ◇ **BONDI**: Use van der Waals radii from A. Bondi, *J. Chem. Phys.* **68** (1964) 441
- ◇ **COSMO**: Use the COSMO optimized radius if available, otherwise use 1.17 times the Bondi radius. This is the default.

Note that the internally defined Bondi atomic radii do not cover the entire periodic table (although all of the commonly used elements are defined). If the program encounters an atom for which an atomic radius is not defined, it will stop with an error message. To proceed in such a case, the corresponding radius must be entered as a user-defined value via one of the following methods

- ◇ **USER**: User-defined radii. These are read from one or more additional lines in the input file immediately following the **COSMO** command line. These additional lines must begin with the character string **\$RADI** followed by one or more occurrences of <symbol>=<radius>, where <symbol> is a string identifying an atomic center (the use of numbers or special symbols to match the symbols used in the geometry section is permitted) and <radius> is the corresponding atomic radius (Å). Note that user-defined radii do not need to cover *all* the atomic centers in the system under study; any atoms with undefined radii after parsing the user-defined input will be assigned the default COSMO value.

◇ **USERB**: Same as USER above, except that every atom not defined will be given the Bondi radius.

◇ **<filename>**: If the string following the `RADI` keyword is not one of `BONDI`, `COSMO`, `USER` or `USERB`, it will be assumed to indicate the name of a file containing the user-defined atomic radii. The format of this user supplied file is the same as for `USER` (above), except that the initial `$RADI` string may be omitted.

Some examples of input for user-defined atomic radii are given below:

```
GEOM=PQS
O      0.000000      0.000000      -0.405840
H     -0.793353      0.000000      0.202920
H      0.793353      0.000000      0.202920
COSMO  RADI=USER
$radi  h=1.5  o=1.8
```

will assign a radius of 1.5 Å to hydrogen and 1.8 Å to oxygen

```
GEOM=PQS
O      0.000000      0.000000      -0.405840
H     -0.793353      0.000000      0.202920
H      0.793353      0.000000      0.202920
COSMO  RADI=USER
$radi  h=1.5
```

will assign a radius of 1.5 Å to hydrogen leaving oxygen with the default (1.72 Å)

```
GEOM=PQS
O      0.000000      0.000000      -0.405840
H     -0.793353      0.000000      0.202920
H$     0.793353      0.000000      0.202920
COSMO  RADI=USER
$radi  h=1.5  o=1.8
$radi  h$=1.3
```

will assign a radius of 1.5 Å to the first hydrogen, 1.8 Å to oxygen and 1.3 Å to the second hydrogen.

OFF: Turns off COSMO for all subsequent steps. This is useful if you wish to do several calculations, with and without COSMO, in the same input file. (See example 33 in the RUNNING JOBS section.)

The following options are for fine tuning and are meant for advanced users

ROUT=<real>: Factor for outer sphere construction (default 0.85). The outer sphere is used for the outlying charge correction.

DISE=<real>: Cutoff for use of basis grid points during SAS construction (default 10.0)

LCAV=<integer>: Type of cavity 0=open; 1=closed (default)

NPPA=<integer>: Total number of basis grid points per atom (default 1082)

NSPA=<integer>: Number of segments for non-hydrogen atoms (default 92)

AMPR=<real>: Amplitude factor for coordinate randomization during SAS construction. Should be 0.00001 (default) or smaller.

PHSR=<real>: Phase offset for coordinate randomization (default 0.0)

23. SEMI command

Semiempirical theories of various types have been around since the early days of quantum chemistry. One of the first (and simplest) is Hückel theory [84] which, at the time, was very successful in predicting the relative energy levels of the π orbitals in aromatic hydrocarbons. Modern semiempirical methods share most of the concepts underlying the more rigorous *ab initio* theories, such as atomic orbitals and their linear combination to form molecular orbitals, but they usually consider only orbitals in the valence shell (typically assigning one s and one p valence AO to each atom, except hydrogen which gets only an s orbital). Only the most important integrals are computed, with the effects of the “missing” integrals and the atomic “core” being parametrized using upwards of a dozen adjustable parameters along with a number of atomic constants for each element. These parameters are fit by attempting to reproduce well-defined experimental (and sometimes *ab initio*) data.

A common family of semiempirical methods are the various NDO approximations introduced by Pople and coworkers [85]. The first of these was CNDO (“complete neglect of differential overlap”) [86], followed by INDO (“intermediate neglect of differential overlap”) [87] and then MINDO/1 (“modified intermediate neglect of differential overlap, version 1”) [88]. There is also Zerner’s ZINDO [89], which was parametrized to reproduce excitation energies from UV spectroscopy.

The man most associated with the semiempirical methods in common use today is Michael Dewar. His scientific autobiography “A Semiempirical Life” is available as an ACS monograph [90]. The PQS semiempirical module has four semiempirical methods available; Dewar was directly involved in developing three of them, MINDO/3 [91], MNDO [92] and AM1 [93], and indirectly involved in the fourth, PM3 [94]. The first two are continuations of the NDO approximation; AM1 is a more refined theory, with more adjustable parameters and somewhat fewer approximations. PM3 is a continuation of AM1, with a modified scheme for parameter fitting and a larger set of experimental values involved in the fit. For more details see the original literature.

The best of the four methods overall is probably PM3, which has also been parametrized for many more elements than the other three methods. Because of the limitation of just an s and a p orbital in the valence shell, semiempirical methods can in general only be used for main group elements. Recently, Thiel has included d-orbitals and has parametrized the first-row transition metals within a MNDO approximation [95] (this is currently unavailable in PQS).

Atoms that have been parametrized for the various semiempirical wavefunctions are:

at. no.	atom	PM3	AM1	MNDO	MINDO
1	H	Y	Y	Y	Y
3	Li	Y	-	Y	-
4	Be	Y	Y	Y	-
5	B	Y	Y	Y	Y
6	C	Y	Y	Y	Y
7	N	Y	Y	Y	Y
8	O	Y	Y	Y	Y
9	F	Y	Y	Y	Y
11	Na	Y	-	-	-
12	Mg	Y	-	-	-
13	Al	Y	Y	Y	-
14	Si	Y	Y	Y	Y
15	P	Y	Y	Y	Y
16	S	Y	Y	Y	Y
17	Cl	Y	Y	Y	Y
19	K	Y	-	-	-
20	Ca	Y	-	-	-
30	Zn	Y	Y	Y	-
31	Ga	Y	-	-	-
32	Ge	Y	Y	Y	-
33	As	Y	-	-	-
34	Se	Y	-	-	-
35	Br	Y	Y	Y	-
37	Rb	Y	-	-	-
38	Sr	Y	-	-	-
48	Cd	Y	-	-	-
49	In	Y	-	-	-
50	Sn	Y	Y	Y	-
51	Sb	Y	-	-	-
52	Te	Y	-	-	-
53	I	Y	Y	Y	-

As can be seen, PM3 is available for all main group elements through the fourth row, except for the rare gases. Also included are the formal transition metals, Zinc and Cadmium. (These metals have completely filled D-shells, with a doubly occupied 4S shell; chemically they have much in common with the alkaline earths.) AM1 is parametrized for the same set of elements as MNDO, except for Lithium which is unavailable. MINDO/3 is only available for selected first and second row elements.

The semiempirical module in PQS calculates both energies *and* gradients for both closed and open-shell (unrestricted) wavefunctions.

Options: **ETHR**=<real> **DTHR**=<real> **LVSH**=<real> **DIIS**=<real> **ITER**=<integer>
NOGUess **PRINT**=<integer>

SEMI may be optionally followed by an equal sign and a semiempirical method. (One of **PM3**, **AM1**, **MNDO** or **MINDO**, see above). The default if no method is given is **PM3**.

Most of the other options control the SCF step and are similar to those in the main *ab initio* SCF module, although in several instances they have been implemented differently.

ETHR=<real>: convergence criterion on the energy change (default is 10^{-9}). Note that internally energies in the semiempirical module are heats of formation in Kcal/mol; they are converted into atomic units on the <control> file.

DTHR=<real>: convergence criterion on the *maximum* difference in the density matrix (compared element by element). The default is 10^{-5} , but typical values obtained in practice are much less than this due to the tight energy criterion.

LVSH=<real>: artificially shifts the energies of the virtual orbitals to help convergence (see the SCF command). The default value is zero, i.e., *no* level shift. If you experience convergence problems, try increasing the value to, say, 4.0. Larger level shifts slow down the SCF convergence rate but should, at least in theory, guarantee convergence.

DIIS=<real>: criterion for switching on DIIS (see the SCF command) which will not be utilized until the *maximum* difference in the density matrix between SCF cycles is less than this value (default 0.1).

ITER=<integer>: maximum number of SCF cycles (default is 200).

NOGUess: The default during a semiempirical geometry optimization is to use the converged orbitals from the previous geometry as starting orbitals for the next SCF; if this option is specified, the SCF will start with a new guess every time. Should rarely be used except for QM/MM calculations. (See RUNNING JOBS, example 26.)

PRINT=<integer>: controls the amount of printout (larger integer - more printout).

Because **SEMI** automatically includes calculation of the gradient, there is no need to include a **FORCE** command in an optimization loop, which typically would be

```
OPTIM
SEMI
JUMP
```

Prior semiempirical calculations are often useful as an aid to *ab initio* calculations, for example to preoptimize a molecular geometry prior to starting an *ab initio* optimization or for calculating a cheap starting Hessian. (See RUNNING JOBS, example 5).

24. FFLD command

Calculates energies, gradients and (optionally) the Hessian matrix using a molecular mechanics force field.

Options: **CUTOff**=<real> **FILE**=<string> **HESS** **PREOpt** **PRINT**=<integer>

FFLD may be optionally followed by an equal sign and a force field type. Currently two forcefields are available: Sybyl_5.2 [96] and UFF (universal force field) [97].

CUTOff=<real>: ignores the van der Waals term in the force field for all interatomic distances greater than the cutoff value given (in Å). If no value is given, a default of 10 Å is used.

FILE=<string>: where <string> is the name of a file containing user-defined atomic connectivities and Sybyl/UFF bonding types (see later)

HESS: calculate the Hessian matrix at the current geometry by finite-difference on analytical gradients

PREOpt: starting from the current geometry, take a few steepest descent steps to lower the energy and hopefully get a more reasonable starting structure prior to a full *ab initio* optimization. Note that this preoptimization is done entirely *within the force field module* and *not* as a part of the optimization step.

PRINT=<integer>: controls the amount of printout (large integer - more output)

This is a preliminary module for a number of different mechanics force fields that we plan to introduce. The existing module is included mainly as an aid to *ab initio* calculations on larger organic systems, for example to preoptimize a molecular geometry prior to starting an *ab initio* optimization or for calculating a cheap starting Hessian. Note that mechanics force fields *cannot*, in general, be used as an aid to locating transition states.

The Sybyl_5.2 Force Field

This is a fairly basic force field containing the following terms:

1. Bond Stretching

$$E_s = \frac{1}{2} C_s (R - R_0)^2$$

where R_0 is an "equilibrium" bond length and C_s is a scaling parameter for each defined bond type; if no parameters are known R_0 is taken as the initial bond length, R , and $C_s = 600$.

2. Non-Bonded van der Waals interaction (1,3 and H-bonds excluded)

$$E_v = C_v [1/(R/R_v)^{12} - 2/(R/R_v)^6]$$

where R_v is the sum of the van der Waals radii for the two atoms and C_v is scaling parameter taken as the square root of the product of the hardness parameters for each atom; if no hardness parameters are known $C_v = 1$

3. Bending

$$E_b = \frac{1}{2} C_b (\theta - \theta_0)^2$$

where θ_0 is an “equilibrium” bond angle and C_b is a scaling parameter for each defined bond angle; if no parameters are known θ_0 is taken as the initial bond angle, θ , and $C_b = 0.02$.

4. Torsion

$$E_t = \frac{1}{2} C_t [1 + s / \{ |s| \cos(|s| \phi) \}]$$

where C_t is a scaling parameter for each torsion ϕ and s is a small integer depending on the bond type; if no parameters are known for the two central atoms of the torsion then $s = 3$ and $C_t = 0.2$.

5. Out-of-Plane Bend

$$E_p = \frac{1}{2} C_p d^2$$

where d is the distance from the central atom to the plane defined by its three attached atoms and C_p is a scaling parameter. Only well-defined atom types have an out-of-plane bend contribution.

The Sybyl_5.2 force field was coded from an old Sybyl theory manual and directly from the original reference [96], both from 1989. As with many other mechanics force fields, carbon, nitrogen and a few other atoms that are in different bonding environments within a molecule are considered as being *different* atom types as far as the Sybyl force field is concerned, and each atom type has its own specific parameters.

Force field parameters are defined only for the atom types listed (see the following page), although van der Waals radii are defined for all atoms up to and including Xenon. There are defaults for most parameters, so the force field can be used even for molecules for which it was not originally defined. In these cases, “equilibrium” bond lengths and angles will be assumed to have values *as calculated from the original input geometry*. Side effects of this assumption are: (1) slightly different starting structures for the same system will optimize to different final geometries; and (2) if optimized structures are reoptimized, they will again change.

Standard Sybyl atom types and their numerical values are listed below:

1	C.3	carbon sp3	
2	C.2	carbon sp2	
3	C.ar	carbon aromatic	
4	C.1	carbon sp	
5	N.3	nitrogen sp3	
6	N.2	nitrogen sp2	
7	N.1	nitrogen sp	
8	O.3	oxygen sp3	
9	O.2	oxygen sp2	
10	S.3	sulphur sp3	
11	N.ar	nitrogen aromatic	
12	P.3	phosphorus sp3	
13	H	hydrogen	
14	Br	bromine	
15	Cl	chlorine	
16	F	fluorine	
17	I	iodine	
18	S.2	sulphur sp2	
19	N.pl3	nitrogen trigonal planar	
20	LP	lone pair	REDUNDANT
21	Na	sodium	
22	K	potassium	
23	Ca	calcium	
24	Li	lithium	
25	Al	aluminium	
26	Du	dummy atom	REDUNDANT
27	Si	silicon	
28	N.am	nitrogen amide	
29	S.O	sulphoxide sulphur	
30	S.O2	sulphone sulphur	
31	N.4	nitrogen sp3 positive charge	
32		unknown atom type	

Standard Sybyl bond types are:

1	single bond
2	double bond
3	triple bond
4	amide bond
5	aromatic bond

The UFF Force Field

This is a general force field covering the entire periodic table. Unlike many other forcefields (e.g., the Sybyl force field as discussed above), UFF parameters are estimated using general rules based on the *element only*. It contains the following terms:

1. Bond Stretching

$$E_s = \frac{1}{2} K_{ij} (R - R_{ij})^2$$

where R is the current interatomic distance in angstroms, R_{ij} is the sum of standard radii for atoms i and j, plus a bond-order correction plus an electronegativity correction, and K_{ij} is a stretching force constant.

$$R_{ij} = R_i + R_j - R_{bo} - R_{en}$$

where

$$R_{bo} = 0.1332 (R_i + R_j) \ln(n) \quad n = \text{bond order}$$

(C-N amide bond order is 1.41; bond order in aromatic rings is 1.5)

and

$$R_{en} = R_i \times R_j \{ \sqrt{X_i} - \sqrt{X_j} \}^2 / (X_i \times R_i + X_j \times R_j)$$

(X_i is the GMP electronegativity of atom i)

The stretching force constants are atom-based and are obtained from a generalization of Badger's rules. They are given by

$$K_{ij} = 664.12 (Z_i \times Z_j) / R_{ij}^3$$

(Z_i, Z_j are effective atomic charges)

2. Non-Bonded van der Waals interaction (1,3 interactions excluded)

$$E_v = D_{ij} [(X_{ij}/X)^{12} - 2 (X_{ij}/X)^6]$$

where D_{ij} is the well depth and X_{ij} is the van der Waals bond length

$$D_{ij} = \sqrt{(D_i \times D_j)} \quad D_i, D_j \text{ are for individual atoms}$$

$$X_{ij} = \sqrt{(X_i \times X_j)} \quad X_i, X_j \text{ are for individual atoms}$$

X is the actual distance between atoms i and j

3. Bending

(a) general nonlinear bend

$$E_b = K_{ijk} [C0 + C1 + C2 \cos(2Th)]$$

Th is the current bond angle and K_{ijk} is the bending force constant between atoms i, j and k. The three expansion coefficients are given by

$$C2 = 1 / (4 \sin^2(Th0))$$

$$C1 = -4 C2 \cos(Th0)$$

$$C0 = C2 (2 \cos^2(Th0) - 1)$$

where Th0 is the idealized equilibrium angle at the central atom (j)

(b) linear, trigonal-planar, square-planar and octahedral

$$E_b = (K_{ijk}/n^2) [1 - L \cos(nTh)]$$

(linear: $n=1, L=-1$; trigonal-planar: $n=3, L=1$; square planar/octahedral: $n=4, L=1$)

The bending force constants in both cases are given by

$$K_{ijk} = (664.12/R_{ij} \times R_{jk}) (Z_i \times Z_k / R_{ik}^5) (R_{ij} \times R_{jk})$$

$$[3R_{ij} \times R_{jk} (1 - \cos^2(Th0)) - R_{ik} \cos(Th0)]$$

4. Torsion

$$E_t = \frac{1}{2} V [1 - \cos(nDih_0) \cos(nDih)]$$

V is the torsional barrier and Dih₀ is the idealized dihedral angle
Specific general cases include (j-k is the central bond of the torsion)

(a) j=sp³ hybridized center; k=sp³ hybridized center

where n=3 and Dih₀=180° (or 60°) and

$$V = \sqrt{(V_j \times V_k)} \quad \text{with } V_j, V_k \text{ being an individual main group atom value}$$

(non main-group elements are assigned V=0)

The torsional terms for pairs of sp³ hybridized group 6 central atoms are exceptions
Here V_j=2 for oxygen and V_j=6.8 for the remaining group 6 elements, with n=2 and Dih₀=90°.

(b) j=sp² hybridized center; k=sp³ hybridized center

where n=6, Dih₀=0° and V=1.0

For a single bond involving an sp³ hybridized group 6 central atom and an sp² atom of another column, V is defined as in (c), below, with n=2 and Dih₀=90°.

For a single bond where the sp² hybridized center in (b) is connected to another sp² hybridized center (e.g., propene) then n=3, Dih₀=180° and V=2.0.

(c) j=sp² hybridized center, k=sp² hybridized center

where n=2, Dih₀=180° and

$$V = 5 \sqrt{(U_j \times U_k) [1 + 4.18 \text{Ln}(\text{BO})]} \quad \text{BO is the bond order between i and j}$$

(U_j constants take values 2.0, 1.25, 0.7, 0.2 and 0.1 for the second through the sixth period/first through the fifth rows of the periodic table)

5. Inversion

$$E_p = K_{ijkl} [C_0 + C_1 \cos(Y_{ijkl}) + C_2 \cos(2Y_{ijkl})]$$

This term applies to exactly 3 atoms (j,k,l) bonded to a central atom (i)

Y_{ijkl} is the angle between the il axis and the ijk plane

Central atoms for which an inversion term is defined are: C, N, P, As, Sb, Bi

The inversion force constants (K_{ijkl}) for all other atoms are set to zero.

For sp² hybridized and aromatic carbon atoms: C₀=1.0, C₁=-1.0, C₂=0.0

If carbon is bonded to sp² hybridized oxygen K_{ijkl}=50.0 otherwise K_{ijkl}=6.0

The UFF force field was coded directly from the original reference [97]. PQS also thanks Dr. Marcus G. Martin, Sandia Laboratories, for supplying unpublished electronegativity parameters from his UFF implementation in the Towhee molecular mechanics package.

There are a number of typos in the original paper. Several of these are corrected in the web page document <http://towhee.sourceforge.net/forcefields/uff.html>. In particular, the expression for the angle bend force constants (eq. 13 in the original paper [97]) is obviously incorrect. However, there are other mistakes, including the expression for the angle bend energy for linear systems (eq. 10) which has a sign error. The implementation in PQS corrects all *known* errors.

As befits a forcefield that covers the entire periodic table (up to Lawrencium, element 103), UFF has many more atom types than Sybyl, which is principally an organic

forcefield. As with Sybyl, several atoms have more than one atom type, depending on the bonding, and in total there are 126 different atom types recognized by UFF. A complete listing is given below (taken from Table 1 in ref. 97). The five standard bond types (single, double, triple, amide and aromatic) covered in the Sybyl forcefield transfer over to UFF as well.

Standard UFF atom types and their numerical values are given below. (Where indicated the value after the “+” sign indicates the formal oxidation state)

1	H_	normal hydrogen
2	H_b	bridging hydrogen
3	He4+4	helium (square-planar)
4	Li	lithium
5	Be3+2	beryllium sp3
6	B_3	boron sp3
7	B_2	boron sp2
8	C_3	carbon sp3
9	C_R	carbon aromatic
10	C_2	carbon sp2
11	C_1	carbon sp
12	N_3	nitrogen sp3
13	N_R	nitrogen aromatic
14	N_2	nitrogen sp2
15	N_1	nitrogen sp
16	O_3	oxygen sp3
17	O_3_z	oxygen in zeolites
18	O_R	oxygen aromatic
19	O_2	oxygen sp2
20	O_1	oxygen sp
21	F_	fluorine
22	Ne4+4	neon (square-planar)
23	Na	sodium
24	Mg3+2	magnesium sp3
25	Al3	aluminium sp3
26	Si3	silicon sp3
27	P_3+3	phosphorus
28	P_3+5	phosphorus sp3
29	P_3+q	4-coordinate phosphorus in phosphines
30	S_3+2	normal divalent sulphur
31	S_3+4	sulphur in e.g. SO2
32	S_3+6	sulphur sp3
33	S_R	sulphur aromatic
34	S_2	sulphur sp2
35	Cl_	chlorine
36	Ar4+4	argon (square-planar)
37	K_	potassium
38	Ca6+2	calcium (octahedral)
39	Sc3+3	scandium (tetrahedral)
40	Ti3+4	titanium (tetrahedral)

41	Ti ⁶⁺⁴	titanium (octahedral)
42	V ₃ ⁺⁵	vanadium (tetrahedral)
43	Cr ⁶⁺³	chromium (octahedral)
44	Mn ⁶⁺²	manganese (octahedral)
45	Fe ³⁺²	iron (tetrahedral)
46	Fe ⁶⁺²	iron (octahedral)
47	Co ⁶⁺³	cobalt (octahedral)
48	Ni ⁴⁺²	nickel (square-planar)
49	Cu ³⁺¹	copper (tetrahedral)
50	Zn ³⁺²	zinc (tetrahedral)
51	Ga ³⁺³	gallium (tetrahedral)
52	Ge ³	germanium (tetrahedral)
53	As ³⁺³	arsenic
54	Se ³⁺²	selenium
55	Br	bromine
56	Kr ⁴⁺⁴	krypton (square-planar)
57	Rb	rubidium
58	Sr ⁶⁺²	strontium (octahedral)
59	Y ₃ ⁺³	yttrium (tetrahedral)
60	Zr ³⁺⁴	zirconium (tetrahedral)
61	Nb ³⁺⁵	niobium (tetrahedral)
62	Mo ⁶⁺⁶	molybdenum (octahedral)
63	Mo ³⁺⁶	molybdenum (tetrahedral)
64	Tc ⁶⁺⁵	technetium (octahedral)
65	Ru ⁶⁺²	ruthenium (octahedral)
66	Rh ⁶⁺³	rhodium (octahedral)
67	Pd ⁴⁺²	palladium (square-planar)
68	Ag ¹⁺¹	silver (linear)
69	Cd ³⁺²	cadmium (tetrahedral)
70	In ³⁺³	indium (tetrahedral)
71	Sn ³	tin (tetrahedral)
72	Sb ³⁺³	antimony
73	Te ³⁺²	tellurium
74	I ₁	iodine
75	Xe ⁴⁺⁴	xenon (square-planar)
76	Cs	cesium
77	Ba ⁶⁺²	barium (octahedral)
78	La ³⁺³	lanthanum (tetrahedral)
78	Ce ⁶⁺³	cerium (octahedral)
79	Pr ⁶⁺³	praseodymium (octahedral)
80	Nd ⁶⁺³	neodymium (octahedral)
81	Pm ⁶⁺³	promethium (octahedral)
82	Sm ⁶⁺³	samarium (octahedral)
83	Eu ⁶⁺³	europium (octahedral)
84	Gd ⁶⁺³	gadolinium (octahedral)
85	Tb ⁶⁺³	terbium (octahedral)
86	Dy ⁶⁺³	dysprosium (octahedral)
87	Ho ⁶⁺³	holmium (octahedral)
88	Er ⁶⁺³	erbium (octahedral)
89	Tm ⁶⁺³	thulium (octahedral)

90	Yb6+3	ytterbium (octahedral)
91	Lu6+3	lutetium (octahedral)
92	Hf3+4	hafnium (tetrahedral)
93	Ta3+5	tantalum (tetrahedral)
94	W_6+6	tungsten (octahedral)
95	W_3+4	tetravalent tungsten (tetrahedral)
96	W_3+6	hexavalent tungsten (tetrahedral)
97	Re6+5	rhenium (octahedral)
98	Re3+7	rhenium (tetrahedral)
99	Os6+6	osmium (octahedral)
100	Ir6+3	iridium (octahedral)
101	Pt4+2	platinum (square-planar)
102	Au4+3	gold (square-planar)
103	Hg1+3	mercury (linear)
104	Tl3+3	thallium sp ²
105	Pb3	lead sp ³
106	Bi3+3	bismuth
107	Po3+2	polonium
108	At	astatine
109	Rn4+4	radon (square-planar)
110	Fr	francium
111	Ra6+2	radium (octahedral)
112	Ac6+3	actinium (octahedral)
113	Th6+4	thorium (octahedral)
114	Pa6+4	protactinium (octahedral)
115	U_6+4	uranium (octahedral)
116	Np6+4	neptunium (octahedral)
117	Pu6+4	plutonium (octahedral)
118	Am6+4	americium (octahedral)
119	Cm6+3	curium (octahedral)
120	Bk6+3	berkelium (octahedral)
121	Cf6+3	californium (octahedral)
122	Es6+3	einsteinium (octahedral)
123	Fm6+3	fermium (octahedral)
124	Md6+3	mendelivium (octahedral)
125	No6+3	nobelium (octahedral)
126	Lr6+3	lawrencium (octahedral)

Unlike Sybyl, there is no allowance in UFF for unrecognized atom types, and so certain compounds simply cannot be treated with UFF. For example, although all elements are covered, many metals must have a well-defined coordination (usually octahedral) or they will either not be properly recognized or distort significantly if a geometry optimization is attempted. In particular, UFF simply does not recognize a trigonal bipyramidal configuration around a central atom. On a more positive note, UFF is one of the few, general force fields that can treat organometallic systems at all – just don't expect perfection.

When the force field module is first invoked it will try, based on the input geometry, to define the atomic connectivity and the Sybyl/UFF bond types. It will then, based on this data, assign the atom types. The actual force field parameters that will be used for your system can be printed out by setting the print flag to 4 or higher.

These preliminary parameters can be overridden by reading in predefined data in a given file using the **FILE** option. The first line in the file should be a title line with user comments (it is not read by the program) followed by one or more lines containing bonding data: atom I atom J bond type free format, all integers. This shows that atoms I and J are bonded and gives the (integer) bond type. There should be *no* blank lines *anywhere* in the file.

25. OPTimize command

PQS contains a powerful suite of algorithms for geometry optimization, referred to collectively as OPTIMIZE. Capabilities include optimization of minima and transition states, optimization in Cartesian, Z-matrix and delocalized internal coordinates, and optimization with a wide range of constraints including constrained interatomic distances, angles, torsions and out-of-plane bends, and frozen (fixed) atoms. Note that desired constraints do not need to be satisfied in the starting geometry.

The main factors that affect the efficiency of a geometry optimization are: (1) the initial guess geometry; (2) the optimization algorithm; (3) the quality of the initial starting Hessian; and (4) the coordinates used to describe the system. Perhaps surprisingly, most of the advances in geometry optimization in recent years have come, not from any major improvements in the algorithms used, but rather from a better choice of coordinates. A good set of coordinates should ideally be decoupled to the maximum extent possible so that changes in the value of one coordinate should have a minimal impact on the other coordinates. Nearly all modern optimization algorithms use the Hessian matrix (the second derivative of the energy with respect to coordinate displacements), or a suitable approximation to it, to help calculate the next step, and the Hessian is easier to estimate (and to improve) if the coordinates are decoupled as off-diagonal matrix elements are small and can often be ignored.

The default coordinates used by OPTIMIZE are delocalized internals [98]. These are automatically generated from input Cartesian coordinates using a simple algorithm based on the atomic connectivity. Delocalized internals combine features from both natural internal coordinates [99] and redundant internal coordinates [100], introduced earlier by Pulay and coworkers, and reduce both harmonic and anharmonic coupling between coordinates to the maximum extent possible on a purely geometric basis. Geometric constraints (including fixed atoms) can be imposed by a very powerful Schmidt orthogonalization procedure [98], and this has been extended to include constraints that are not satisfied in the starting geometry [101] using a Lagrange multiplier algorithm originally developed for Cartesian coordinates [102]. Special delocalized cluster coordinates have also been developed for the efficient optimization of molecular clusters [103] and for adsorption/reaction on model surfaces.

The standard optimization algorithm in OPTIMIZE is the Eigenvector Following (EF) algorithm [104]. This can locate both minima and transition states, and is capable of taking corrective action if the system is in the wrong region of the potential energy surface appropriate to the stationary point being sought (i.e., if the current estimate of the Hessian matrix has the wrong eigenvalue structure). Another option, which has been coded for minimization only, is GDIIS [105].

For optimizing the structures of *single* molecules delocalized internals are the coordinates of choice. For optimizing molecular clusters (containing two or more weakly interacting molecules) use `coord=cluster`, while for a molecular system being adsorbed/reacting on a model surface, you should specify `coord=surface`. In these two latter cases, the individual molecules in the cluster or the surface/molecule interface should be separated in the geometry input by "\$molecule" (see GEOMETRY section). For a surface/molecule system, the surface coordinates should be given *first*.

This is a loop command, requiring a terminating `JUMP` card. Typical usage would be:

```
OPTIM
SCF
FORCE
JUMP
```

Options: **COORD**=<string> **REGEnerate**=<string> **CUTOff**=<real> **TYPE**=<string>
MODE=<integer> **GDII**s=<integer> **DMA**X=<real> **GTOL**=<real> **DTOL**=<real>
ETOL=<real> **STOL**=<real> **HESS**=<string> **UPDAte**=<string> **PROJect**=<string>
TRAN OPTCycle=<integer> **CTOL**=<real> **LINEar**=<real> **BACK**=<string> **NOTOr**s
SCALE=<real> **HCVrt** **QMMM** **PRINT**=<integer>

COORD coordinate system used to carry out optimization
usage: coord=cart Cartesian coordinates
coord=deloc/int delocalized internal coordinates
coord=zmat z-matrix coordinates
coord=surface surface adsorption/reaction
coord=cluster cluster coordinates (includes inverse-distance)

default: delocalized internals with automatic switch to Cartesians if there are any problems (e.g., with the back-transformation)

Note that if a given coordinate system is specified there is NO automatic switch to Cartesians if problems are encountered. For a z-matrix optimization, the geometry MUST be given in z-matrix form AND “coord=zmat” MUST be specified.

REGEnerate regenerate “best” set of delocalized internal coordinates on each optimization cycle

There are two options, regenerate just the delocalized internals using the same set of underlying primitives or regenerate both the internal coordinates *and* the underlying primitives (specify regenerate=all). The latter is recommended for cluster optimizations with a specific cutoff.

default: generate delocalized internals on first cycle ONLY and use these throughout the optimization

Note that in single-molecule optimizations, use of this option does not usually gain and is NOT RECOMMENDED.

CUTOff distance cutoff for bonding in surface/cluster optimizations
usage: cutoff=R where R is a distance cutoff (in Å)

default: 5 Å for cluster optimizations; 3 Å for surfaces

TYPE type of stationary point sought
usage: type=min search for a minimum (default)
type=ts search for a transition state

MODE which Hessian mode (eigenvector) to follow (i.e., to maximize along) during a transition state search

usage: mode=N where N is the mode number

default: follow (maximize along) the lowest Hessian mode

Note that N must be greater than 0 and NOT greater than the total number of Hessian modes (degrees of freedom). Following different Hessian modes from the *same* starting point can lead to different transition states.

GDIIS use the Pulay Geometry DIIS algorithm instead of the default Eigenvector Following (EF) algorithm
usage: gdiis=N where N is the maximum allowed size of the iterative subspace

default: specifying "gdiis" alone will give a default subspace size depending on system size (typical values are around 4)

Note do not set N too large. GDIIS can ONLY be used for minimization.

DMAX maximum allowed optimization step size
usage: dmax=R where R is a small real number, e.g., 0.15
default: 0.3

GTOL convergence criterion on maximum allowed gradient component
usage: gtol=R where R is a small real number ($R \geq 10^{-6}$)
default: 0.0003 au

DTOL convergence criterion on maximum predicted displacement
usage: dtol=R where R is a small real number ($R \geq 10^{-6}$)
default: 0.0003

ETOL convergence criterion on energy change from previous cycle
usage: etol=R where R is a small real number ($R \geq 10^{-8}$)
default: 10^{-6} hartree

****IMPORTANT**** In order to converge, the criterion for **GTOL** *must* be satisfied, together with any *one* of **DTOL** or **ETOL**, not necessarily both. To ensure, e.g., that **DTOL** is satisfied, set **ETOL** so low that it will almost never be satisfied, and vice versa.

STOL RMS gradient tolerance for steepest descent step
(If RMS gradient > STOL, steepest descent step will be taken)
usage: stol=R where R is a real number ($R \geq 10^{-3}$)
default: 0.3 au

HESS Initial Hessian matrix
usage: hess=unit take unit matrix as starting Hessian
hess=default force internal initial guess
default: read Hessian from <hess> file if one exists; otherwise estimate (nominally diagonal) Hessian (note that the default starting Hessian for a Cartesian optimization is a unit matrix)

UPDAte Hessian update
usage: update=no no Hessian update (when have exact Hessian)
 update=ms Murtagh-Sargent update
 update=powell Powell update
 update=bofill Powell/Murtagh-Sargent update
 update=bfgs BFGS update
 update=bfgs-safe BFGS update with safeguards

default: standard minimization - BFGS; GDIIS minimization - BFGS with safeguards;
transition state search - Powell/Murtagh-Sargent

Note that the BFGS update tends to preserve the positive-definite nature of the Hessian. If "bfgs-safe" is specified then the update will be skipped *if* the update does not preserve positive definiteness.

PROJect project out translations and rotations from Hessian matrix during Cartesian optimization
usage: project=no do not project
 project=partial project out translations
 project=full project out translations and rotations

default: full projection

TRAN inclusion of gradient term when transforming Hessian from Cartesian to internal coordinates

default: do not include gradient term

Note that the gradient term should normally only be included when an *exact* transformation is desired. At a stationary point, the gradient should be zero and this term is zero in any case.

OPTCycle maximum number of optimization cycles
usage: optc=N where N is the number of cycles
default: 50

CTOL tolerance for satisfying imposed constraints
usage: ctol=R where R is a small real number ($10^{-6} \geq R$)
default: 10^{-6} au

LINEar tolerance for near-linear bond angle during generation of primitive internals
usage: linear=R where R is the desired angle in degrees
default: 165.0 (do not set much below this)

If a given primitive bond angle is greater than R degrees, then it will be replaced during the generation of delocalized internals by the special colinear bend.

BACK sets backtransformation algorithm for internal coordinates
usage: back=full full backtransformation using exact inverse
 back=zmat O(N) Z-matrix backtransformation
default: back=zmat

NOTOrs do not use torsions when generating delocalized internals

This can be useful to reduce the (often large) number of primitive internals in the coordinate space in situations where all the deformational degrees of freedom can be spanned using bends alone. Often useful for surface optimizations where each atom is connected to many neighbours.

default: use torsions

SCAL scaling factor for inverse-distance coordinates used in cluster optimizations (coord=cluster, see above)

usage: scal=R where R is the desired scaling factor

default: 1.00 (typical values range from 1.00 to 10.00)

For weakly interacting clusters use the default. As the intermolecular interactions get stronger, scaling should be increased. For clusters of up to 10 water molecules, a scaling factor of 5.0 has been successfully used. Too large scaling factors tend to give a smoother optimization, but slow down the rate of convergence.

HCNVrt Hessian transformation flag (delocalized internals → Cartesians)

At the end of a successful optimization an approximate Hessian matrix in Cartesian coordinates is available in the <hess> file. If the optimization was performed using delocalized internal coordinates, the Hessian is transformed from internal into Cartesian coordinates. Normally this is done once only, at the end of the optimization. If this keyword is present, the transformation will be done every cycle.

QMMM must be included for a QM/MM optimization

PRINT sets value of print flag

usage:	print=0	NO printout (except for error messages)
	print=1	summary and warning printout only
	print=2	standard printout
	print=3	slightly more printout (including gradient)
	print=4	heavier printout (including full Hessian)
	print=5	heavier still (includes iterative printout)
	print=6	very heavy (including internal generation)
	print=7	debug printout

Additional Input Options: Constraints and Connectivity

Geometrical constraints (fixed values for various internal coordinates) and additional bond connectivity (sometimes needed to ensure that a full set of internal coordinates can be generated) can be input *either* directly following the **OPTimize** command line or via a user-defined file. *Note that formerly only the latter option was available.*

Usage: file=<myfile> user-defined file name

Constraints

\$constraint defines the beginning of the constraints section and *\$endconstraint* the end. See example below:

```
$constraint
stre   I  J           value   angstroms  value > 0.0
bend   I  J  K       value   degrees    180.0 ≥ value ≥ 0.0
tors   I  J  K  L    value   degrees    180.0 ≥ value ≥ -180.0
outp   I  J  K  L    value   degrees     90.0 ≥ value ≥ -90.0
linc   I  J  K  L    value   degrees         ditto
linp   I  J  K  L    value   degrees         ditto
$endconstraint
```

stre distance constraint between any two (different) atoms
bend planar bend constraint between any three (different) atoms
 J is the middle atom of the bend
tors dihedral angle (proper torsion) constraint between any four (different) atoms
 the connectivity is I-J-K-L, with J-K being the central "bond"
 the torsion is the angle the plane I-J-K makes with the plane J-K-L
outp out-of-plane-bend constraint between any four (different) atoms
 angle made by bond I-L with the plane J-K-L (L is the central atom)
linc colinear bending constraint between any four (different) atoms
 bending of I-J-K in the plane J-K-L
linp perpendicular bending constraint between any four (different) atoms
 bending of I-J-K perpendicular to the plane J-K-L

linc/linp are special angles used when four atoms are near-linear.

I, J, K, L are integers indicating the positions of the atoms involved in the constraint in the order they appear in the Cartesian coordinate list defining the molecular geometry.

Frozen Atoms

\$fix defines the beginning of the frozen atom section and *\$endfix* the end. See example below:

```
$fix
atom  fixed          format (I4,2X,A3)
$endfix
```

atom integer indicating which atom in the Cartesian coordinate list is to be fixed;
fixed character string (**upper case**): X, Y, Z, XY, XZ, YZ, XYZ
 indicates which Cartesian coordinate or coordinates are to be fixed
 (XYZ fixes (freezes) the entire atom)

****IMPORTANT**** If *all* atomic coordinates (XYZ) are frozen then the optimization can be carried out in delocalized internals, which are much more efficient. (Previously this option was not available.) However, in order to do this all the frozen atoms *must* be formally connected; if they are not, then additional connectivity should be specified to ensure that they are (see below).

Additional Atom Connectivity

Normally delocalized internal coordinates are generated automatically from the input Cartesian coordinates. This is done by first determining the atomic connectivity list, i.e., which atoms are formally bonded, and then constructing a set of individual *primitive* internal coordinates comprising all bond stretches, all planar bends and all proper torsions that can be generated based on the atomic connectivity. The delocalized internals are in turn constructed from this set of primitives.

The atomic connectivity depends simply on distance and default bond lengths between all pairs of atoms are available in the code. In order for delocalized internals to be generated successfully, all atoms in the molecule **MUST** be formally bonded so as to form a closed system. Formerly, for molecular complexes with long, weak bonds or in certain transition states where parts of the molecule are rearranging or dissociating, the standard atomic connectivity algorithm separated the system into two or more distinct parts, and the generation of delocalized internals failed. Additional connectivity needed to be input in order to connect the disparate fragments. This should no longer be necessary as the connectivity algorithm has now been modified and should successfully generate a full set of delocalized internals for virtually all systems.

It should only be necessary to specify additional atomic connectivity in the case where there is a subset of frozen atoms, not all of which are formally bonded (see above).

\$connect defines the beginning of the additional connectivity section and *\$endconnect* the end. See example below:

```
$connect
atom  list          format (I4,2X,8I4)
$endconnect
```

atom atom for which additional connectivity is being defined
list list of up to 8 atoms considered as being bonded to the given atom

Surface Constraints

In optimizations involving model surfaces, one or more (or sometimes all) layers of surface atoms may be kept fixed. *\$surface* defines the beginning of the surface constraints section and *\$endsurface* the end. See example below:

```
$surface
fixed <list>
$endsurface
```

<list> either a list of surface atoms to be fixed format (10X,10I4)
(the 10X starts from the beginning of the line and includes the fixed string)
or the character string all to fix all surface atoms

Dummy Atoms

\$dummy defines the beginning of the dummy atom section and *\$enddummy* the end. See example below:

```
$dummy
idum type list                    format (I4,2X,I4,2X,7I4)
$enddum
```

idum centre number of dummy atom (should be one greater than total number of
 real atoms for first dummy atom; two greater for second and so on)
type which type of dummy atom (either 1, 2 or 3; see below)
list list of up to 7 atoms defining position of dummy atom

Dummy atoms are used to help define constraints during constrained optimizations in *Cartesian* coordinates. They CANNOT be used with delocalized internals.

All dummy atoms are defined with reference to a list of real atoms and dummy atom coordinates will be generated from the coordinates of the real atoms in its defining list. There are three types of dummy atom:

1. Positioned at the arithmetic mean of the up to 7 real atoms in the defining list
2. Positioned a unit distance along the normal to a plane defined by three atoms, centred on the middle atom of the three
3. Positioned a unit distance along the bisector of a given angle

Once defined, dummy atoms can then be used to define standard internal (distance, angle) constraints as per the constraints section, above.

****WARNING**** The use of dummy atoms of type 1 has never really progressed beyond the experimental stage. Avoid if possible. Will not always work.

Rules for dummy atom placement in dihedral constraints

Bond and dihedral angles *cannot* be constrained in Cartesian optimizations to exactly $\pm 0^\circ$ or $\pm 180^\circ$. This is because the corresponding constraint normals are zero vectors. Also dihedral constraints near these two limiting values (within, say, 20°) tend to oscillate and are difficult to converge.

These difficulties can be overcome by defining dummy atoms and redefining the constraints with respect to the dummy atoms. For example, a dihedral constraint of 180° can be redefined to two constraints of 90° with respect to a suitably positioned dummy atom. The same thing can be done with a 180° bond angle (long a familiar usage in Z-matrix construction).

Typical usage is as follows:

internal coordinates

```
$constraint
tors I J K L 180.0
$endconstraint
```

Cartesian coordinates

```
$dummy
M 2 I J K
$enddummy
$constraint
tors I J K M
tors M J K L
$endconstraint
```

The atom order is important to get the correct signature on the dihedral angles. For a 0° dihedral constraint, J and K should be switched in the definition of the second torsion constraint in Cartesian coordinates.

Note that in the vast majority of cases the above discussion is somewhat academic, as internal constraints are now best imposed using delocalized internal coordinates AND there is no restriction on the constraint values.

Composite Constraints

A composite constraint is a constraint on not just one primitive internal coordinate, but on a linear combination thereof. It is defined within the `$constraint` and `$endconstraint` delimiters for a normal (single primitive) constraint, using the delimiters `#composite` and `#endcomposite` as follows

```
$constraint
<normal constraints>
#composite
type      atom list      weight      value
type      atom list      weight
  "        "              "
#endcomposite
$endconstraint
```

Here `type` is the primitive internal coordinate type and is one of either *stre*, *bend*, *outp* or *tors* (composite constraints cannot be defined for planar/perpendicular linear bends), `atom list` is the list of atoms (up to four) defining the internal coordinate, `weight` is

the weight of that particular primitive in the composite constraint (its factor in the linear combination) and `value` is the overall total value desired for the linear combination. Note that you only need to give relative weights as whatever linear combinations you provide will be normalized by the program; the default weight if nothing is given is one.

If no `value` is provided then whatever value the composite constraint has in the starting geometry will be maintained. If you do provide a `value` then you should provide it on the first line defining the composite constraint only (as shown above) and you *must* also provide a corresponding `weight`, even if it is the default value of one. The `value` must be given in atomic units, i.e., stretches in au and angles in radians.

The most common type of composite constraint is likely to be a linear combination of one type of primitive internal, e.g., all stretches or all bends. However, this is not a limitation and you can combine any or all of the four supported primitives in the same composite constraint. For example

```
#composite
stre  I  J    1.0d0
stre  J  K   -1.0d0
bend  I  J  K    2.0d0
#endcomposite
```

Here the linear combination $1/\sqrt{6} (R_{IJ} - R_{JK} + 2 A_{IJK})$ will be constrained to whatever value it had in the starting geometry. (The $1/\sqrt{6}$ is a normalization factor.) If you wanted to provide a specific value for the composite constraint, it should be given after the `1.0d0` on the line defining the first stretch.

There is no limit on the number of composite constraints that can be defined. Each one should be specified between its own `#composite` and `#endcomposite` delimiters. Note, however, that there is no checking for incompatible or impossible constraint combinations, so you should be careful in your definitions.

From a practical point of view, composite constraints that are not satisfied initially are more difficult to satisfy exactly than are straightforward individual primitive constraints. So at convergence, using the default convergence criteria, a composite constraint may not be satisfied as precisely as an individual primitive constraint (which is usually six decimal places for stretches (in Å) and three for angles (in degrees)).

One place where a composite constraint may be required is for a potential scan involving symmetry (see the **SCAN** command).

26. CLEAN command

Cleans up (removes) specific files after a geometry optimization.

This command was introduced primarily to prevent a second or subsequent optimization in the same input file from using the <opt> and <optchk> files left over from a previous failed optimization.

OPTIMIZE is designed so that if a given optimization fails, any restart will automatically attempt to continue from where the previous job left off by reading intermediate data from an <optchk> file, assuming one exists. If an input deck contains multiple optimization steps, either explicit or implicit (e.g., in an optimized potential scan), it may be desirable to continue with the current optimization even if the previous one failed. Adding the **CLEAN** command at the end of an optimization loop will delete both the <opt> and <optchk> files, ensuring that the next optimization starts properly. Note that after a *successful* geometry optimization, these files are automatically deleted.

There is one option, **CLEAN=ALL** which deletes the <hess> file as well. This prevents any subsequent optimization from picking up the previous Hessian matrix, and forces a new initial Hessian estimation. This may seem counterintuitive, as it is normally a good idea for a related optimization to make use of the previous optimization's Hessian, but sometimes this is not always the case.

One such situation is during an optimized potential scan where a bond length is being stretched to a fairly large value. (See the **SCAN** command.) Optimized potential scans are effectively constrained optimizations with the scanned variable as the constraint. Constrained optimizations can break down if the Hessian matrix develops very small eigenvalues, something that will almost inevitably occur if the same Hessian is updated repeatedly as a bond length stretches. Under these circumstances it is best to start with a newly estimated Hessian at the start of each optimization following incrementation of the scanned variable. This can be accomplished via the **CLEAN** command as in the following example input deck:

```
SCAN
OPTIM
SCF
FORCE
JUMP
CLEAN=ALL
JUMP
```

27. DYNAMics command

Initiates a direct classical molecular dynamics run.

This is a loop command, requiring a terminating `JUMP` card. Typical usage would be:

```
GEOM SYMM=0          (needed if the initial geometry is symmetrical - see below)
DYNAMics STEP=40 TEMP=1000 MAXC=500
SCF
FORCe
JUMP
```

Options: **STEP**=<real> **TEMP**=<real> **MAXC**=<integer> **SEED**=<integer>

STEP=<real>: This is the timestep in atomic units. The atomic unit of time is $\hbar/E_h \approx 2.4188843E-17$ s = 0.024 188 843 fs, and thus 41 au \approx 1 fs

TEMP=<real>: Initial starting temperature in degrees Kelvin. Calculations are started with the initial geometry and with kT random kinetic energy per degree of freedom. This is followed by removal of rotations so that the angular momentum of the system is zero. If the initial geometry is close to equilibrium, as is usually the case, then, of course, about half of the kinetic energy is converted to potential energy. In a classical assembly of coupled harmonic oscillators, the amount of kinetic and potential energy is equal, and thus the equilibrium temperature will be approximately half the initial one. There is no thermostat in the current code. The temperature of the system can be established only after it has equilibrated somewhat.

MAXC=<integer>: Maximum number of dynamics cycles, i.e., the length of the run.

SEED=<integer>. The program uses a seed, which is derived from the time and thus is truly random, to initialize the random number generator which is needed to set the initial velocities. A disadvantage of this method for is that a dynamics run is impossible to reproduce. The **SEED** option allows the user to set the random seed. If **SEED** is not specified, the random seed is printed out and can be used in subsequent jobs.

Notes

(1) Symmetry must be suppressed when carrying out a dynamics run because the atoms are started with random velocities which destroys the symmetry anyway. This can be accomplished by inserting a **GEOM SYMM=0.0** card in front of the **DYNAm** card. The preceding optimization step (if the user chooses to perform one) can use symmetry. (See RUNNING JOBS, example 18).

(2) Chemical reactions may take a very long time to occur, even if the temperature, and thus the available energy, is more than sufficient to surmount the barrier. A common trick to circumvent this difficulty is to start the trajectory from the transition state.

(3) In the current implementation, there is no thermostat. The temperature is thus not accurate, it serves only as an orientation value. The initial kinetic energy given to the atoms is kT (not $kT/2$). In a system of harmonic oscillators, the average kinetic energy is equal to the average potential energy, and thus the average kinetic energy, if the dynamics run was started at the energy minimum, will develop toward the approximately correct value $kT/2$.

(4) The trajectory is written to the file <jobname>.trajec . This file contains the Cartesian coordinates of the atoms in each timestep.

28. QMMM command

The idea behind QM/MM methods is to define a (small) region of the system of interest which will be computed using a fairly accurate, high-level model, and a (much larger) region for which a less accurate, low-level model can be used. For example, in a large model protein or enzyme, most of the chemistry takes place around the “active site” – this can be modelled using quantum mechanics, while a mechanics force field can be used for the rest of the system; hence the name QM/MM. The term has now been generalized to refer to a high level calculation for one (or more) regions of the system and a lower level (not necessarily molecular mechanics) for the rest.

The major problem with QM/MM is how to treat the interaction between the two regions. The approach which we have adopted is the so-called ONIOM method, as popularized by Morokuma and coworkers [106]. If any formal bonds are “broken” between the QM and MM regions, then “link atoms” (usually hydrogen) are added to saturate the free valency. Three calculations are done, the full system at the MM level, and the inner region (including any extra link hydrogen atoms) at the QM level and again at the MM level. The QM/MM energy is defined as:

$$E_{\text{QM/MM}} = E_{\text{MM}}^{\text{full}} + E_{\text{QM}}^{\text{inner}} - E_{\text{MM}}^{\text{inner}}$$

With this simple definition, it is straightforward to define QM/MM gradients and Hessians (although the latter are not yet included in our code) as well as other molecular properties. A useful reference, from which our own algorithm was coded, is the work of Dapprich et al. [107].

Note that if you plan to do a “QM/QM” calculation with, e.g., a large basis set for the “inner” region and a small basis for the rest of the system, this can be done within the existing code without invoking QM/MM. See the **BASIS** command for details.

To designate the “inner” (QM) part of your molecule the “\$molecule” designator (see the `GEOM` command) should be used. Atoms *before* the “\$molecule” will be treated by the chosen QM method; all atoms *after* the “\$molecule” will be treated by the low level method.

The current QM/MM module is only preliminary and needs to be revised. There are no options to the command line other than **PRINT**.

QMMM is a rather complex command. It will almost invariably be used as part of a geometry optimization. A full example showing how to place the `QMMM` cards in the input deck for a QM/MM optimization is given in the RUNNING JOBS section (example 26).

29. SCAN command

Carries out a potential scan, i.e., changes one variable while keeping the others fixed. By combining with OPTIMIZE, it is possible to carry out a scan on one variable while optimizing all remaining degrees of freedom.

The input card for **SCAN** must be EITHER of the form

```
SCAN coord <atom list> FROM <range> <step>
```

Where `coord` is one of `stre`, `bend`, `outp` or `tors`

e.g. SCAN tors I J K L FROM -30.0 30.0 5.0

OR of the form (for a composite scan)

```
SCAN comp FROM <range> <step>
#composite
<definition of composite coordinate>
#end composite
```

e.g. SCAN comp FROM 5.0 4.0 -0.1
#composite
stre 5 11
stre 6 13
#endcomposite

OR of the form (for a Z-matrix scan)

```
SCAN ZMAT <z-matrix variable> FROM <range> <step>
```

e.g. SCAN ZMAT L1 FROM 1.0 1.6 0.05

There are no other options.

Note that a potential scan alone is best done via a Z matrix, as this is the only way to define and fix the remaining variables. (The scan can be done in delocalized internal coordinates, but the remaining variables will be ill-defined, being linear combinations of stretches, bends and torsions rather than the well-defined individual internal coordinates that make up the Z matrix.) For an optimized scan, it doesn't matter how the remaining degrees of freedom are defined (only the scan variable is important) as they will be optimized in any case, so this can be done in Z-matrix coordinates or (best) delocalized internals. For input examples, see the RUNNING JOBS section.

An optimized scan using delocalized internals is done effectively in the same way as a constrained optimization, with the scanned variable as the constraint. It is possible to do a *constrained* optimized scan by specifying additional constraints for the optimization loop; the scanned variable is simply added to the constraint list.

If your molecule has symmetry which the variable to be scanned breaks (e.g., ethane scanning a H-C-C-H torsion) then you should distort the starting geometry slightly prior to starting the scan.

Scanning a composite coordinate (a linear combination of primitive internals rather than a single primitive) is often required if the system has symmetry which you wish to maintain and scanning a *single* variable (primitive) will break that symmetry. In this

event you must pick an appropriate linear combination to scan which will preserve the symmetry. See the **OPTIm** command for more on composite coordinates

It is not possible to do a potential scan in Cartesian or surface/cluster coordinates.

SCAN is a loop command, requiring a terminating **JUMP** card. Typical usage would be:

```
SCAN  ZMAT  <scan definition>
GEOM=ZMAT PRINT=1
SCF
JUMP
```

See also the **CLEAn** command.

PRACTICAL ASPECTS

Be aware of limitations imposed by internal coordinate definitions. For example, if you want to scan a bond angle through a complete 360° rotation, this must be done in two separate scans – from, say, 0° to 180° and then from 180° to 0° the other way – because the planar bend is always defined as lying between 0° and 180°. The extreme points of the range (0° and 180°) may also cause problems and should be done separately. Similar limitations apply to the out-of-plane bend which, for a complete 360° scan, should be scanned from -90° to +90° and then from +90° to -90° the other way, the problem angle here being ±90°.

When doing a Z-matrix scan, remember to include the **GEOM=ZMAT** card after the **SCAN** command so that the geometry module knows that you are using Z-matrix coordinates. (This is similar to the requirement for a Z-matrix optimization.)

30. PATH command

Follows a reaction path downhill from the transition state to the reactant and/or product.

The concept of a reaction path, although seemingly well-defined chemically (i.e., how the atoms in the system move to get from reactants to products), is somewhat ambiguous mathematically because, using the usual definitions, it depends on the coordinate system. Stationary points on a potential energy surface are *independent* of coordinates, but the path connecting them is not, and so different coordinate systems will produce different reaction paths. There are even different definitions of what constitutes a “reaction path”; the one used in PQS is based on the intrinsic reaction coordinate (IRC), first defined in this context by Fukui [108]. This is essentially a series of steepest descent paths going downhill from the transition state.

The reaction path is most unlikely to be a straight line and so by taking a finite step length along the direction of the gradient you will leave the “true” path. A series of small steepest descent steps will zig-zag along the actual reaction path (this is known as “stitching”). Ishida et al. [109] developed a predictor-corrector algorithm, involving a second gradient calculation after the initial steepest descent step, followed by a line search along the gradient bisector to get back on the path; this was subsequently improved by Schmidt et al. [110], and is the method we have adopted. For the first step downhill from the transition state this approach cannot be used (as the gradient is zero); instead a step is taken along the Hessian mode corresponding to the imaginary frequency.

The reaction path can be defined and followed in Z-matrix coordinates, Cartesian coordinates or mass-weighted Cartesians. The latter represents the “true” IRC as defined by Fukui [108]. However, if the main reason for following the reaction path is simply to determine which minima a given transition state connects (perhaps the major use), then it doesn’t matter which coordinates are used. In order to use **PATH** you must know (and input) the transition state geometry and you must have the exact Hessian available in the <hess> file.

This is a *double* loop command, requiring *two* `PATH` cards and a terminating `JUMP` card. Typical usage would be:

```
PATH
SCF
PATH
FORCE
JUMP
```

During the line search, the program loops between the two `PATH` commands. Whenever a gradient is needed following a successful line search the second `PATH` is “by-passed” and the program loops between the first `PATH` and the `JUMP` command.

Options: **COORD**=<string> **DMAX**=<real> **DTOL**=<real> **ITER**=<integer>
SIGN=<integer> **PRINT**=<integer>

COORD coordinate system used to define and follow the reaction path

usage: coord=cart Cartesian coordinates
coord=mwgt mass-weighted Cartesian coordinates
coord=zmat z-matrix coordinates

default: mass-weighted Cartesians

DMAX approximate maximum step size between points on reaction path

default: 0.15

DTOL convergence criterion on step size

default: 0.005

Normally the maximum step size will be taken. However, towards the end of the path, i.e., near the reactants or products, the gradient will be reduced and the step size will be smaller. If the predicted step size gets below **DTOL**, then the reaction path search will stop. Note that the algorithm is not designed to follow the reaction path all the way to the minimum, and normal termination is after a specific number of steps (see **ITER**).

ITER number of points to find on reaction path

default: 20

SIGN defines downhill direction from transition state

default: +1

Must be either +1 (follow eigenmode downhill) or -1 (change sign of eigenmode). In all likelihood you won't know the direction prior to the Hessian diagonalization, but setting **SIGN**=-1 will go in the opposite direction (i.e., towards the other minimum).

PRINT sets value of print flag (higher value, more printout)

DESCRIPTION OF THE POPEL STYLE INPUT FILE

The Pople style input has the following general format:

1. Preamble: %MEM %CHK %RWF cards
2. Route information
3. Title
4. Charge and multiplicity
5. Molecular geometry, possibly in two parts: geometry and parameters
6. Optional **CONV** keyword (no run, only input conversion to PQS input is requested)

The same typographic conventions apply to the Pople style input as to PQS input. These are: Case does not matter; characters following an exclamation mark (!) are removed as comments; in general, only the first 4 characters of a word are significant

The route and title sections must be terminated by a blank card. The first route card must have a hash sign (#) as its first non-blank character. If the **CONV** option is used, the whole input preceding the **CONV** command must also be terminated by a blank.

1) Preamble

- a) %MEM= n If no unit (MB or GB) is specified then if $n < 2000$, n Megawords ($8n$ Megabytes) of memory are requested. If $n \geq 2000$, n words (8 bytes) are requested.
- b) %CHK=<old_job>. This option has two effects. First, all files <old_job>.* are copied to <jobname>.*, allowing the calculation to start with data from <old_job>. Second, the program saves its own files under the jobname <old_job>. It is thus a combination of the PQS CHK and SAVE options.
- c) %RWF=<path/base filename>. The scratch files will be put in *path*, in files <base filename>.<pqs extension>. E.g., if %RWF=/MYSCR/tempfil, then the scratch files will be /MYSCR/tempfil.11, /MYSCR/tempfil.12, etc...

2) Route

The route card has the following information:

Print flag

Method/Basis set

Calculation type (e.g. single energy point, forces, optimization, NMR)

Options for the program steps

The route card is often a single line but can be written in 2 or more lines. **It is terminated by a blank line.**

- a) The first non-blank character on the route card **MUST** be a hash sign (#). The hash sign is optionally followed by the letter P (#P), signaling that the print option is on. The Pople style input has only one print option which is on or off for all program steps. It can thus produce copious amounts of printing. If finer control of printing is needed, use the PQS input, or edit the <pqs> file generated from the Pople style input.
- b) The next field, separated from the first (# or #P) by one or more blank spaces, is the method/basis information: <method>/<basis>. Currently <method> is limited to (R)HF, (R)HFS, (R)SVWN, (R)SVWN5, (R)BVWN, (R)BVWN5, (R)BLYP and (R)B3LYP, the same eight methods with R replaced by U, and (R)MP2. The first R can be omitted in the restricted methods. For more details of these methods see the SCF section on page 32.

- c) The <basis> field can be any of the basis sets listed in the PQS description (see page 24), not just the Pople-style basis sets. Note that the method and basis must be separated by a slash (/) *without spaces*.
- d) Calculation type. This can be absent, in which case a single point energy is calculated, with a few properties. Possible calculation types are currently:
- i) **SP** = single point (default)
 - ii) **FORCE**: calculate forces = negative gradients on the nuclei
 - iii) **OPT**: Geometry optimization
 - iv) **FREQ**: Calculate force constants and vibrational frequencies
 - v) **NMR**: Calculate NMR chemical shifts
 - vi) **DYNA**: run a direct *ab initio* classical molecular dynamics trajectory
 - vii) **MP2**: run an MP2 calculation *after* other job steps (e.g. optimization)
- e) Options for the program steps
- i) **GEOM**=<geometry type>. Possible types are ZMAT (default), CART (PQS Cartesian input), and many other types described in the PQS input description.
 - ii) **UNIT**=BOHR changes the default coordinate unit from Angstrom (10^{-10} m) to bohr (0.529177×10^{-10} m)
 - iii) **NOSY**mmetry: supresses symmetry. **SYMM**=<real> (e.g. SYMM=0.05) sets a threshold for symmetrization of an approximately symmetrical molecule, as obtained, e.g., from a graphical modeling program. **NOSY**m is equivalent to **SYMM**=0.0. See the **GEOM** command on page 11 for more details.
 - iv) **GUES**s=<guess type>. Allowed types are MINDo, HUCKel, READ
 - v) **SCF**=TIGHT requests tighter SCF thresholds. This is the default for all but single point calculations
 - vi) **SCFCycle**=N number of maximum SCF cycles, default is 50
 - vii) **VSHif**t=<real>. Level shift applied to the virtual orbitals, in atomic units. Note that this definition is not the same as in some other programs where VSHIFT is expressed as an integer, 1 meaning $1.0E-4$ hartrees.
 - viii) **COOR**=<coordinate type for optimization>. Possibilities are CART, ZMAT. The default is delocalized coordinates.
 - ix) **OPTCycle**=N (integer). The maximum number of optimization cycles.
 - x) **TS** (logical flag): transition state optimization is requested
 - xi) **GDII**s=N (integer). The maximum number of steps stored in a geometry DIIS procedure
 - xii) **MAXD**=<integer or real>. The maximum amount of disk space the MP2 procedure can use, *in million* (10^6) *double words* (8-byte words).

- xiii) **STEP**=<real number>: the time step for a molecular dynamics run, in atomic units (1 au time is about 0.024 fs)
 - xiv) **MAXCycle**=<integer>: the maximum number of molecular dynamics cycles
 - xv) **TEMP**=<real number>: the approximate initial temperature of a molecular dynamics run.
 - xvi) **POP** or **POP=MULLiken**, **POP=LOWDin**, **POP=FULL**, **POP=NBO**: Population analysis options (**POP** alone requests Mulliken and Löwdin charges, valencies and free valencies but no bond orders). See the description of the population analysis module and the NBO module earlier in this manual.
- f) Charge and multiplicity (2 integers), followed by the molecular geometry. If there are any parameters in a Z matrix input, they must be preceded by an empty line.

The Pople style input stream can be optionally terminated by a blank card followed by the keyword **CONV** . This tells the input reader that it should only transform the Pople style input to PQS style input but should not run the job. E.g., let us assume that the Pople style input is in the file `nic2h2.com`. The command

```
pqs nic2h2.com
```

transforms the file `nic2h2.com` , shown left, to `nic2h2.pqs`, right:

<pre>%MEM=5 # BLYP/6-31G* FORCE VSHIFT=5 Nickel-acetylene complex 0 1 Ni X 1 1.9 C 2 0.62 1 90.0 C 2 0.62 1 90.0 3 180.0 H 3 1.06 2 180.0 1 90.0 H 4 1.06 2 180.0 1 90.0</pre>	<pre>%MEM=5 TITLE=Nickel-acetylene complex GEOM=ZMAT GEOP SYMM=0.000010 Ni X 1 1.9 C 2 0.62 1 90.0 C 2 0.62 1 90.0 3 180.0 H 3 1.06 2 180.0 1 90.0 H 4 1.06 2 180.0 1 90.0 VARIABLES BASIS=3-21G SCF ITER=6 DFTP=blyp LVSH=5.0 BASIS=6-31G* SCF DFTP=blyp LVSH=5.00 FORCE</pre>
--	---

RUNNING JOBS

The best way to submit PQS jobs is via the *pqs* job script. In the Linux/Unix version, this resides in /usr/local/share/PQS; in the Windows version, in the top installation directory (e.g. C:\PQS). Simply type (for single-processor background jobs under Unix/Linux)

```
pqs jobname &
```

or (for foreground jobs under Linux/Unix and under Windows)

```
pqs jobname
```

where *jobname* is the name of the input file *without* the *.inp* extension. If the extension of the input file is not the default *.inp*, you have to type the full input file name, e.g.

```
pqs jobname.ext &
```

where the input file extension is *.ext*. Besides the *.inp* extension, a commonly used input file extension is *.com*, used for Pople-style input files. The *pqs* script assumes that the input file has one of the following four extensions: *.inp*, *.com*, *.pqs* or *.input*. The program will function for other extensions, but it may not save any old output files correctly.

The Pople style input converter produces an intermediate input file *jobname.pqs* which can be edited by hand and run with the command

```
pqs jobname.pqs &
```

The (detailed) output of the program will be in the file *jobname.out*; a concise output can be found in *jobname.log*. The *pqs* script checks whether there is already a file *jobname.out*, and, if so, renames the old file to *jobname.old*. If *jobname.old* already exists, the output will be appended to it. In earlier versions of the *pqs* script the accumulated old output file was called *jobname.out_1*.

All files produced by running PQS using the *pqs* job script will be given the prefix *jobname*; thus the <control> file will be called *jobname.control*, the <coord> file *jobname.coord* and so on. On successful job completion, all files necessary for a restart, or to resubmit the job at a higher level of theory, will be saved in the directory from which the job was submitted. These files can be collected into a job archive using the *archive* script. On a Linux machine, simply typing "*archive jobname*" will archive and compress all necessary files into a tar file *jobname.tgz*. Files can be restored by typing "*tar -xvzf jobname.tgz*". The archive script is not yet implemented for Windows.

Unwanted files can be removed using the *tidy* script. (*tidy* jobname). This is recommended before a job is resubmitted after a failed run, as some modules, in particular the optimize module, may pick up incorrect information from the intermediate files. If a job crashes or is deleted manually, other modules (e.g., the force field, the scan) may also show this problem. If a job fails to run for no apparent reason, please try first to use *tidy*. The *tidy* script will delete all files with the base filename jobname except the input file (.inp), output file (.out), the summary log file (.log) and the old outputs (.old). It will also delete files in the SCR (scratch) directory (default /scr/<username> under Linux; <Install-dir>\SCR under Windows, e.g., C:\PQS\SCR if the installation directory was C:\PQS).

An alternative to using the *pqs* job script is submission “by hand”, e.g. (for Linux/Unix)

```
/progs/PQS/pqs.x jobname &  
or  
/progs/PQS/pqs.x jobname.ext &
```

Under Windows, this will change, according to the Windows syntax, to, e.g.

```
C:\PQS\pqs.x jobname  
or  
C:\PQS\pqs.x jobname.ext
```

Finally, the program will accept (under Linux/Unix) the syntax

```
/progs/PQS/pqs.x < input-file >& output-file &
```

In this case all files created by PQS (other than the output file which has been specifically named) will be given the prefix *pqsjob*. DO NOT submit different jobs using this syntax at the *same* time in the *same* directory - this will cause severe filename conflicts. Different jobs can be safely submitted from the same directory using the *pqs* job script; however, you should make sure that any SCR files *explicitly named* in the input file are *different* for different jobs. As scratch files are usually not explicitly defined, this is not usually a limitation.

Using *tidy* without an argument will assume the argument *pqsjob*. Using “*tidy* all” will delete **all** PQS files in both the current and scratch directories (take care before using it!)

For running PQS in parallel see later. In the simplest case, if the Parallel Virtual Machine (PVM) has already been set up, the command to run PQS in parallel, say on four processors, is

```
pqs jobname 4 &
```

PROGRAM FILES

This section describes the files the PQS modules produce (*first* write to a particular file). Only the *filename extension* is given here; the actual file is prefixed with a <jobname> determined at job submission (see RUNNING JOBS). Thus, e.g., <control> denotes the file <jobname>.control

GEOM (geometry input)

Files: <control> this module is the first to write to the <control> file
<coord> contains atomic symbols, Cartesian coordinates, atomic charges and atomic masses format (A8,2X,5F20.14)
<zmat> z-matrix and parameter list (if z-matrix input)
<sym> contains symmetry information: point group symbol, number of atoms, number of symmetry operations, number of degrees of freedom, rotation matrix (relating initial to final orientation), number and list of symmetry-unique atoms, symmetry operations (as 3x3 matrices), equivalent atoms array

BASIS (basis set definition)

Files: <basis> contains basis set information: number of contracted basis functions, number of shells, number of primitive shells, definition of basis set (function type, exponents, contraction coefficients).
<basis2> copy of <basis> (for use with SCF guess using different basis)

GUESS (SCF starting orbitals)

Files: <mos> binary file containing initial guess alpha/closed-shell MOs
<mob> binary file containing initial guess beta spin MOs (for UHF)

SCF (SCF and DFT iterations)

Files: produces several temporary binary files
updates <mos>/<mob> files with latest SCF orbital coefficients
may produce any of the following files, depending on the specific job type
<los> binary file containing localized alpha/closed-shell MOs
<lob> binary file containing localized beta spin MOs (for UHF)
<nos> binary file containing naturalized orbitals
<potS> FTC potential file (deleted in a subsequent gradient step)
<potM> ditto

FORCE (Gradient of the energy)

Files: <grad> contains atomic symbols and Cartesian forces
format (A8,2X,3F20.14)

NUMGrad (Gradient from numerical differentiation of energies)

Files: <grad> contains atomic symbols and Cartesian forces
format (A8,2X,3F20.14)
<gradchk> binary file containing data pertinent to next finite-difference step

MP2 (Traditional canonical MP2 energies)

Files: produces several (potentially very large) temporary binary files
by default these are stored in the user's scratch directory
<htr> half-transformed integrals
<bins> bin-sort integral files
By default, these files are deleted upon job completion; however the
<htr> files will be kept if the `KEEP` option is specified

NUMHess (Hessian - force constant - matrix from numerical differentiation of forces)

Files: <hess> Hessian matrix in Cartesian coordinates
format keyword indicating Hessian type
Hessian dimension
lower triangle, one row at a time, free format
<deriv> dipole moment derivatives
format (10X,3F20.14)
<hesschk> binary file containing data pertinent to next finite-difference step
can be used for restarts

HESS (Analytical Hessian matrix calculation)

Files: <hess> Hessian matrix in Cartesian coordinates
format keyword indicating Hessian type
Hessian dimension
lower triangle, one row at a time, free format
<deriv> dipole moment derivatives
format (10X,3F20.14)
<aat> atomic axial tensors (in a VCD job)
format (10X,3F20.14)
produces several (potentially large) temporary binary files

NUMPol (Dipole moment and polarizability derivatives via numerical differentiation)

Files: <deriv> dipole and polarizability derivatives
format (10X,3F20.14)
<polchk> binary file containing data pertinent to next finite-difference step
can be used for restarts

NMR (NMR chemical shifts)

Files: may produce temporary binary files

POP (Population analysis)

Files: <chelp> ascii file containing various results of the CHELP analysis

NBO (Weinhold's Natural Bond Orbital analysis)

Files: opens some internal scratch files which are deleted on exit

SEMI (Semiempirical SCF calculations)

Files: <grad> and several temporary files which are deleted on exit.

FFLD (Force field module)

Files: <grad> and (optionally) <hess> files (see above)
<ffchk> binary file containing details of force field parameters

COSMO (Solvation model)

Files: <cosmo> ascii file containing data for the application of COSMO-RS
(COSMO for Real Solvents) theory
produces several (potentially large) temporary binary files

OPTimize (Geometry optimization)

Files: <opt> internal file containing optimization options
<optchk> binary file containing data pertinent to next optimization cycle
can be used for restarts
<hess> approximate (updated) Hessian matrix in Cartesian coordinates
format keyword indicating Hessian type
Hessian dimension
lower triangle, one row at a time, free format
<hprim> primitive Hessian in internal coordinates
(only if new delocalized internals are generated every cycle)
format - same as for <hess>

DYNAMics (Direct Newtonian molecular dynamics)

Files: <trajec> trajectory file

QMMM (Combined quantum mechanics – molecular mechanics)

Files: <qmmm> binary file containing data pertinent to next QM/MM step
can be used for restarts

SCAN (Potential scan, including scan + optimization)

Files: <scan> binary file containing data pertinent to next step in potential scan
can be used for restarts

PATH (Reaction path)

Files: <path> binary file containing data pertinent to next step in path search
can be used for restarts

EXAMPLES

Test	Molecule	Calculation	Page
1.	H ₂ O	6-31G* SCF geometry optimization	114
2.	NH ₃	SVWN DFT Z-matrix geometry optimization	115
3.	CH ₄	BLYP/6-31G* geometry optimization followed by analytical frequencies	116
4.	H ₂ O	3-21G constrained geometry optimization	117
5.	HCN → HNC	B3LYP 6-311G** transition state search and frequency analysis at the saddle point	118
6a.	dichloropropane	RHF/STO-3G constrained optimization using delocalized internal coordinates	118
6b.	dichloropropane	Same with Cartesian coordinates + dummy atoms	119
7.	H ₂ O	Different basis sets on the two H atoms. Geometry optimization followed by NMR chemical shifts	120
8.	ozone	RHF/3-21G optimization followed by UHF singlet calculation and NBO analysis	121
9.	benzene	RHF/6-31G* + NMR in an external electric field	122
10.	H ₂ O	BLYP/6-31G** geometry optimization + NBO analysis + NMR + frequencies in an electric field	123
11.	(H ₂) ₁₀	RHF/3-21G optimization of 10-molecule H ₂ cluster	124
12.	CO on Si surface	RHF/3-21G optimization of adsorbed molecule	125
13.	1,3,5-trifluorobenzene	B3LYP/3-21G Cartesian optimization with force field preoptimization and Hessian	126
14.	H ₂ O	Same as Test 1 with Pople style input	127
15.	CH ₄	Same as Test 3 with Pople style input	127
16.	H ₂ O	RHF/6-31G* optimization, followed by MP2	127
17.	H ₂ O ₂	BPW91/VDZP optimization + frequencies with RAMAN intensities via polarizability derivatives	128
18.	HF + H ₂ O	RHF/6-31G energy with HF as ghost atoms	129
19.	ethyl radical C ₂ H ₅	UBLYP/6-311G** optimization + nuclear properties	129
20.	H ₂ O ₂	B3LYP/6-31G* optimization + dynamics (30 steps)	130
21.	C ₂ H ₄	Z-Matrix potential scan along the C-C bond	131
22.	C ₂ H ₄	optimized scan along the C-C bond	131
23.	H ₂ CO → H ₂ + CO	BLYP/6-31G** Cartesian reaction path	132
24.	HCN → HNC	Z-matrix reaction path	133
25.	H ₂ O	Dual basis MP2: 6-311G** and 6-311G(3df,3pd)	133
26.	SeP(CH ₃) ₃	QM/MM optimization using STO-3G and PM3	134
27.	CH ₃ OH	RHF/DZP optimization + analytical frequencies	135
28.	H ₂ O	MP2 with the cc-pVQZ basis (spherical harmonic g functions)	135
29.	H ₂ CO	B3LYP/cc-pVTZ optimization + WAH NMR shieldings	136
30.	CO	OLYP/6-311G** optimization + NMR with level shift	136

31a.	HF	MP2/PC-2 optimization	137
31b.	HF	MP2-SCS/PC-2 optimization	137
32.	NO	PBE/6-311G** optimization + analytical frequencies	138
33.	CHFCICH ₂ F	OLYP/PC-2 energy with FTC + semidirect	138
34.	HCl	optimization + frequency in gas phase and water via COSMO	139
35.	C ₂ H ₅ OH	RHF/CEP-121 optimization + frequency + NMR	140
36.	H ₂ dimer	MP2/6-311G** use of ghost atoms with symmetry	140
37.	CHFCIBr	RHF/3-21G optimization + frequency + NMR + VCD	141
38.	cyclopropane	B97/3-21G optimization + frequency + full population analysis	141
39.	2-aminopropionic acid	PM3 test of multiple constraint types	142
40.	HOF	B3LYP/6-311G** optimization + frequency + polarizabilities	143
41.	1,2-dimethylbenzene	HF/6-31++G suppressed basis functions	144
42.	butane	RHF/3-21G NMR + NICS	145
43.	butane	RHF/3-21G optimization + frequency (IR+Raman) + QCISD	146
44.	H ₂ O	QCISD(T)/6-311G* numerical optimization	147
45.	O ₃	UMP2/6-311G* numerical optimization	148
46.	chlorobenzene	B3LYP/6-31G* optimization + normal & SQM frequencies	149
47.	HOF	B3LYP/6-311G** enforced geometry optimization	150
48.			151

This section contains 50 input files. They have been selected to allow a rapid test of the capabilities of the PQS program, and also serve as examples for input preparation. The Table above summarizes the job types.

1. Standard RHF/6-31G* geometry optimization of water

```
TEXT= Standard 6-31G* optimization of water
GEOM=zmat
O
H 1 L1
H 1 L1 2 A1
VARIABLES
L1 1.0
A1 105.0
BASIS=6-31G*
GUESS
OPTIM          -----
GUESS=READ    |
SCF           | Basic Optimization Loop
FORCE        |
JUMP         -----
```

Note that, even though the input geometry is via a Z-matrix, the optimization will be carried out using (default) delocalized internal coordinates (*not* Z-matrix coordinates).

Note further that in PQS version 2.3 and higher the GUESS commands and the integer on the JUMP card are no longer necessary. They are given for the sake of completeness and to show the proper position of the GUESS command in the job input.

2. SVWN/6-311G** Z-matrix optimization of ammonia

```
TEXT=  Ammonia      Z-matrix optimization      SVWN/6-311G**
GEOM=zmat
N
X   1   1.0
H   1   L1   X   A1
H   1   L1   X   A1   3   120.0
H   1   L1   X   A1   3  -120.0
VARIABLES
L1   1.0
A1   105.0
BASIS=6-311GDP
GUESS
OPTIM  coord=zmat      -----
GEOM=zmat              |
GUESS=READ             | Z-matrix Optimization Loop
SCF  dftp=svwn        |
FORCE                             |
JUMP  5                -----
```

Note the difference between the Z-matrix optimization loop and the standard (non-Z-matrix) optimization loop in example 1. The extra GEOM card (with option `zmat`) *must* be included, as *must* the `coord=zmat` option with the OPTIM card. Because of the extra GEOM card, the JUMP loop is now 5 (one more than previously).

3. BLYP/6-31G* optimization plus analytical frequencies for methane

```
%MEM=6
TEXT= Methane geometry optimization + analytical frequencies
GEOM=PQS
C      0.0000000  0.0000000  0.0000000
H      0.6293118  -.6293118  0.6293118
H      -.6293118  0.6293118  0.6293118
H      0.6293118  0.6293118  -.6293118
H      -.6293118  -.6293118  -.6293118
BASIS=6-31G*
OPTIM  gtol=0.00005  -----
GUESS=READ          |
SCF dftp=blyp      | Optimization Loop
FORCE              |
JUMP  4            -----
HESS
FREQ
```

As with examples 1 and 2, none of the GUESS commands or the integer on the JUMP cards are strictly necessary. They are given for the sake of completeness and to show the proper position of the GUESS command in the job input. In most subsequent examples, unnecessary GUESS cards will be omitted.

4. Constrained optimization on water

```
TEXT= Water constrained optimization H---H fixed
GEOM=zmat
O
H 1 L1
H 1 L1 2 A1
VARIABLES
L1 1.0
A1 105.0
BASIS=3-21G
INTE thresh=10,8
OPTIM gtol=0.00005 print=4
$constraint
stre 2 3 1.8
$endconstraint
SCF dftp=b3lyp
FORCE
JUMP
```

Here we have a constrained optimization with the constraint directly embedded in the input file. The stretch coordinate (distance) between atoms 2 and 3 (the two hydrogens) will be constrained to 1.8 Å. This stretch coordinate will not normally be one of the primitives generated as the two H atoms are not formally bonded; it will be added to the coordinate set in order to impose the constraint. Note that the desired constraint value is *not* satisfied in the starting geometry; this can be handled by the OPTIMIZE module.

5. HCN <---> HNC transition state search + frequency analysis

```
TEXT=  HCN <---> HNC  TS search + frequencies  B3LYP/6-31G**
GEOM=zmat
C
N  1  L1
H  1  L2  2  A1
VARIABLES
L1  1.126
L2  1.2
A1  90.0
NUMHESS  fdstep=0.005  -----
GEOM  noorient  print=1  | Preliminary Numerical Hessian Loop
SEMI=PM3  | using semiempirical PM3
JUMP  -----
GEOM  print=1
BASIS=6-31G**
OPTIM  type=ts  print=4  -----
SCF  dftp=b3lyp  | Optimization
FORCE  | Loop
JUMP  -----
HESS  ----- Final Analytical Hessian for ab initio
FREQ
```

Note that transition state searches are rarely successful without a good starting Hessian. In this example an initial Hessian is calculated numerically *before* starting the main optimization loop; this is done using PM3. (The recommended finite-difference step size for semiempirical wavefunctions is 0.005 a.u.) PM3-generated starting Hessians are not always suitable for transition state searches, but should certainly be tried if nothing better is available. There are no problems at all in this case.

The main level of theory is B3LYP/6-31G**. The optimization will be done in delocalized internal coordinates and the optimization options include `type=ts` for a TS search. (Actually in this example the same coordinates are generated as are present in the Z-matrix, and a Z-matrix optimization should give an identical performance.) The final DFT Hessian is calculated analytically.

6. Constrained optimization on 1,2-dichloropropane showing use of dummy atoms

```
TEXT=  Constrained optimization test  1,2-dichloropropane
GEOM=car file=dichloroprop.car
BASIS=STO-3G
OPTIM  coord=deloc/cart  print=3  file=dichloroprop.opt#
SCF
FORCE
JUMP
```

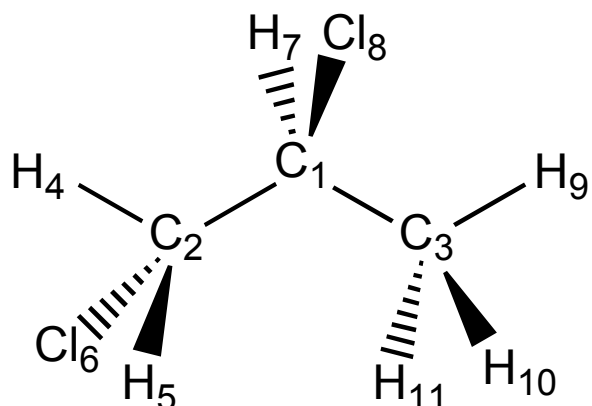
contents of user file dichloroprop.opt#

1. *delocalized internals*

```
$constraint
stre  2  3          2.5      constraints defined with respect
bend  1  3  9      110.0     to real atoms in the molecule
tors  6  2  1  8    180.0
$endconstraint
```

2. *Cartesian coordinates*

```
$dummy
12    2    6  2  1          dummy atom defined first
$enddummy
$constraint
stre  2  3          2.5
bend  1  3  9      110.0
tors  6  2  1 12    90.0     torsion constraint defined with respect
tors 12  2  1  8    90.0     to dummy and real atoms
$endconstraint
```



The two optimizations impose exactly the same set of constraints. However, because of the impossibility of *directly* imposing 0° or 180° torsion constraints in Cartesian coordinates, in the latter optimization the torsion is redefined as two 90.0° constraints with respect to a dummy atom perpendicular to the constraint planes. Carrying out the optimization using delocalized internals is far more efficient, as well as being more straightforward to set up. Note that in both examples, constraints are added via an external file; alternatively the file contents could have been embedded directly into the input file, immediately following the OPTIM command line (see example 4).

7. B3LYP/6-31G* Water with different basis set on each H

```
%MEM =3000000   core=2000000
TEXT=  Water with different basis set on each H
GEOM=pqs
O1      0.0      0.0      -0.405840
H2     -0.793353  0.0      0.202920
H3$     0.793353  0.0      0.202920
BASIS=6-31G*   NEXT
FOR      H$
S        5.447178  0.156285
         0.824547  0.904691
S        0.183192
INTE  thresh=10,8
OPTIM  coord=internal  gtol=0.00005  print=4
SCF  dftp=b3lyp
FORCE
JUMP
GEOM  geop
NMR
```

Note the use of the “special symbol” (in this case a “\$”) to get a different basis from the standard for one of the H atoms. Despite the fact that the initial coordinates show C_{2v} symmetry, only C_s symmetry will be used during the optimization and the final geometry will break C_{2v} symmetry (due to the different basis functions on the H atoms). The NMR shieldings will also be different for each H atom. The `geop` option on the final GEOM card will cause the converged O-H bond lengths to be printed out, making explicit the extent of the symmetry breaking.

This small job will be run in semidirect mode, with 2 million words of memory reserved for integral storage (sufficient in this case to store all integrals).

8. RHF/3-21G Z-matrix optimization of ozone + UHF singlet + NBO analysis

```
TEXT= Ozone Z-matrix optimization + UHF singlet + NBO
GEOM=ZMAT
O
O 1 L1
O 1 L1 2 A1
VARIABLES
L1 1.4
A1 120.0
BASIS=3-21G
OPTIM coord=zmat dmax=0.15 print=4
GEOM=zmat print=1
SCF
FORCE
JUMP
NBO --- NBO analysis on RHF wavefunction
GEOM=read symm=0.0 --- no symmetry for UHF
GUESS=READ UHFS MIX angle=45.0 --- mixing alpha & beta MOs
INTE thresh=10,8 --- restoring original integral threshold
SCF
NBO --- NBO analysis on UHF wavefunction
```

Note the GUESS card which is required in order to access the UHFS and MIX options.

9. RHF/6-31G* NMR shielding for benzene in the presence of an applied electric field

```

TEXT= Benzene      NMR Shielding along principal axis
GEOM=PQS
C      -1.207353   -0.697066    0.0
C      -1.207353    0.697066    0.0
C       0.0        1.394131    0.0
C      1.207353    0.697066    0.0
C      1.207353   -0.697066    0.0
C       0.0       -1.394131    0.0
H     -2.142871   -1.237187    0.0
H     -2.142871    1.237187    0.0
H       0.0        2.474375    0.0
H      2.142871    1.237187    0.0
H      2.142871   -1.237187    0.0
H       0.0       -2.474375    0.0
Q       5.0         0.0        50.0        9.0
Q      -5.0         0.0        50.0        9.0
Q       0.0         5.0        50.0        9.0
Q       0.0        -5.0        50.0        9.0
X       5.0         0.0       -50.0       -9.0
X      -5.0         0.0       -50.0       -9.0
X       0.0         5.0       -50.0       -9.0
X       0.0        -5.0       -50.0       -9.0
X       0.0         0.0         0.0
X       0.0         0.0         0.5
X       0.0         0.0        -0.5
X       0.0         0.0         1.0
X       0.0         0.0        -1.0
BASIS=6-31G* DUMMY
SCF
NMR

```

Here dummy atoms are used to mimic the effects of a uniform field. Note the location of the dummies and their charges. Single dummy atoms at (0,0,±50) could be also have been used, but the field would have been less uniform. Dummy atoms with a positive charge have been given the symbol “Q”, whereas those with a negative charge are denoted by “X”. This helps with symmetry recognition (C_{2v} in this case).

The nuclear magnetic shielding will be computed at all carbon atoms, all hydrogens and at the five positions given by the last five dummies (all along the principal axis). These dummies are *not* included in the symmetry determination.

10. BLYP/6-31G** Optimization + numerical frequencies + NMR + NBO analysis for water in an applied electric field + dummy atom

```

TEXT= Water various showing use of dummy atoms
GEOM=PQS
O      0.00      0.0      -0.405840
X      0.00      0.00     -50.000000    9.0
X      0.00      0.00     50.000000   -9.0
H     -0.793353  0.00      0.202920
X      0.50      0.50      0.50
H      0.793353  0.0      0.202920
BASIS=6-31G** DUMMY
OPTIM coord=internal print=3
SCF DFTP=BLYP
FORCE
JUMP
NMR
NBO
NUMHESS fdstep=0.02 -----
GEOM NOORIENT PRINT=1 | Numerical Hessian
SCF DFTP=BLYP | Loop
FORCE |
JUMP -----
GEOM NOORIENT PRINT=1
FREQ

```

Here dummy atoms are used to mimic the effects of a uniform field. The uncharged dummy atom at (0.5,0.5,0.5) will be excluded from the symmetry determination (hence the system will be recognized as C_{2v}), but will be included in both the NMR and NBO analyses. The charged dummy atoms along the principal axis will be excluded from all molecular property determinations.

IMPORTANT: The numerical hessian plus frequency should be the last properties computed as the numerical hessian loop will destroy the converged MO and binary density files. The GEOM card in the numerical hessian loop *must* include `noorient` and should also include `print=1`. The finite-difference (central differences are used) step size of 0.02 a.u. is the current recommended value for DFT wavefunctions.

Note the use of `noorient` on the final GEOM card before the frequency analysis (the geometry should not be reoriented in the presence of an applied field – compare this with example 17).

11. RHF/3-21G optimization of 10-molecule H₂ cluster

```
TEXT= Randomly generated 10-molecule H2 Cluster
GEOM=PQS
h1      9.312880896      0.068981920      1.583083577
h2      9.184311798      0.569133067      2.084799449
$molecule
h3      9.703799891      2.750525348      2.238102939
h4      9.594408699      3.447999309      2.096810187
$molecule
h5      6.990854245      0.262543127      2.119289305
h6      6.942340403      0.970578861      1.997914649
$molecule
h7      9.934168615      4.165684282      0.419657171
h8      9.609286136      4.718387243      0.747332028
$molecule
h9      7.572245704      0.701916772      3.784739353
h10     6.995634965      1.078738605      3.994323134
$molecule
h11     5.249253820      0.135389887      1.060880149
h12     5.043662023      0.803819832      1.232150174
$molecule
h13     7.628550909      4.250882101      0.974189438
h14     7.346010396      3.805632099      1.464417237
$molecule
h15     5.316176578      2.694413190      2.119907053
h16     5.951167335      2.806792553      2.440153235
$molecule
h17     7.067429324      3.464539696      4.384161347
h18     6.943045015      4.111988363      4.673537861
$molecule
h19     10.367557821     5.385222847      2.469758640
h20     10.040050347     5.746285829      2.999638227
BASIS=3-21G
OPTIM coord=cluster regen=all gtol=0.0005 etol=0.0000001 print=3
SCF
FORCE
JUMP
```

Note the various optimization options: `coord=cluster` for cluster optimization (with the individual hydrogen molecules separated by `$molecule` in the geometry input; `regen(erate)=all` for regenerating the coordinates at every optimization cycle; and reduced values of `gtol` and `etol` for tight convergence. The default distance cutoff of 5Å is used for determining the intermolecular bonding.

12. RHF/3-21G optimization of CO adsorbed on model Si surface

```
TEXT=  RHF/3-21G   CO adsorbed on model Si surface
GEOM=PQS
Si      -1.223460   0.000000  -0.337432
Si       0.611730   1.059547  -0.337432
Si       0.611730  -1.059547  -0.337432
Si       0.000000   0.000000  -2.020958
$molecule
C        0.000000   0.000000   1.915467
O        0.000000   0.000000   3.117787
BASIS=3-21G
OPTIM  coord=surface cutoff=3.0 print=4 file=surface.opt0
SCF
FORCE
JUMP
```

contents of user file surface.opt0

```
$surface
fixed  all
$endsurface
```

In this example, CO is being adsorbed onto a model silicon surface consisting of a top layer of 3 Si atoms forming an equilateral triangle, and a bottom “layer” of a single Si atom below the mid-point of the triangle. The whole system has C_{3v} symmetry with all Si atoms of the surface frozen. As with constraints (example 4) the contents of the file surface.opt0 could have been embedded directly into the input file immediately following the OPTIM command.

13. B3LYP/3-21G Cartesian optimization of 1,3,5-trifluorobenzene with force field preoptimization and Hessian

```
%MEM=2000000
TEXT=1,3,5-trifluorobenzene Force Field preoptimization + Hessian
GEOM=TX92
C    0.695062606    1.203883748    0.000000000
C    1.389792019    0.000000000    0.000000000
C    0.695062606   -1.203883748    0.000000000
C   -0.695062606   -1.203883748    0.000000000
C   -1.389792019    0.000000000    0.000000000
C   -0.695062606    1.203883748    0.000000000
F    1.359947507   -2.355498178    0.000000000
F   -2.719895015    0.000000000    0.000000000
F    1.359947507    2.355498178    0.000000000
H    2.469685167    0.000000000    0.000000000
H   -1.234842583   -2.138810094    0.000000000
H   -1.234842583    2.138810094    0.000000000
OPTIM
FFLD print=4
JUMP
FFLD print=1 HESS
BASIS=3-21G
OPTIM print=3 coord=cart
SCF DFTP=B3LYP
FORCE
JUMP
```

Here we have a force field preoptimization on 1,3,5-trifluorobenzene followed by a full force field Hessian calculated at the optimized force field geometry before starting an ab initio optimization which will use Cartesian coordinates. The print flag is set during the force field optimization so that it will print out all the force field parameters and all energy terms (stretching energy, bending energy etc...) at each optimization cycle.

14. Standard RHF/6-31G* geometry optimization of water using Pople-type input

```
# RHF/6-31G*  OPT          ---- keyword line (denoted by #)

Standard 6-31G* optimization of water  ---- text

0  1          ---- charge/multiplicity
O
H  1  L1
H  1  L1  2  A1

L1  1.0
A1  105.0
```

15. Methane optimization + numerical frequencies using Pople style input

```
#P BLYP/6-31G*  OPT  FREQ  GEOM=CART

Methane  geometry optimization + numerical frequencies

0  1
C      0.0000000    0.0000000    0.0000000
H      0.6293118   -0.6239118    0.6239118
H     -0.6293118    0.6239118    0.6239118
H      0.6293118    0.6239118   -0.6239118
H     -0.6293118   -0.6239118   -0.6239118
```

16. Water 6-31G* RHF optimization, followed by MP2

```
# RHF/6-31G*  OPT  MP2

Standard 6-31G* optimization of water + MP2

0  1
O
H  1  L1
H  1  L1  2  A1

L1  1.0
A1  105.0
```

This shortened input style will be familiar to users of programs such as GAUSSIAN.

17. BPW91/VDZP optimization of H₂O₂ with numerical frequencies including polarizability derivatives for Raman intensities

```

TEXT= H2O2 optimization + frequency including polarizability
derivatives
GEOM=ZMAT
O
O 1 L1
H 1 L2 2 A1
H 2 L2 1 A1 3 D1
VARIABLES
L1 1.4
L2 1.0
A1 105.0
D1 120.0
BASIS=vdzp_ahlrichs
OPTIM
SCF DFTP=BPW91 THRE=6.0
FORCE
JUMP
NUMHESS fdstep=0.02
GEOM NOORIENT PRINT=1
SCF DFTP=BPW91 THRE=6.0
FORCE
JUMP
GEOM print=1
SCF DFTP=BPW91 THRE=6.0
FORCE
NUMPolar dipd pold
GEOM noorient print=1
SCF DFTP=BPW91 THRE=6.0
FORCE
JUMP
GEOM print=1
FREQ

```

| Numerical Hessian
| Loop
|

--- additional geometry to restore symmetry
--- restore energy at optimized geometry
--- ditto restore gradient

| Numerical polarizability
| derivatives Loop
|

--- additional geometry to restore symmetry
--- will include both IR & Raman intensities

18. RHF/6-31G* HF + H₂O interaction energy HF as ghost atoms

```
TITLE= HF + H2O interaction energy: HF as ghost atoms
GEOM=PQS
H      0.000000  0.000000  0.000000  0.0
F      0.920300  0.000000  0.000000  0.0
O     -1.808600  0.000000  0.000000
H     -2.366997 -0.756170  0.000000
H     -2.366997  0.756170  0.000000
BASIS=6-31G*
SCF
```

This job is part of a calculation to determine the basis set superposition error (BSSE) for the HF + H₂O interaction. In this particular input, the energy of H₂O is being calculated in the presence of the basis set for HF.

19. Ethyl radical optimization + charge/spin density and electric field gradient

```
TEXT = Ethyl Radical  UBLYP/6-311G** OPT + nuclear properties
GEOM=PQS  MULT=2
C     -0.1341   0.7105   0.0000
C     -0.0900  -0.8090   0.0000
H     -0.5832   1.1308  -0.8976
H     -0.5832   1.1308   0.8976
H     -1.1161  -1.1991   0.0000
H      0.4255  -1.1809  -0.8891
H      0.4255  -1.1809   0.8891
BASIS=3-21G
SCF  DFTP=BLYP  ITER=6
BASIS=6-311G**
OPTIM
SCF  DFTP=BLYP
FORCE
JUMP
PROP  SPIN  EFG
```

20. Hydrogen peroxide optimization + 30 steps molecular dynamics

```
TEXT= B3LYP/6-31G* Optimization + Dynamics for Hydrogen Peroxide
GEOM=ZMAT
O
O 1 L1
H 1 L2 2 A1
H 2 L2 1 A1 3 D1
VARIABLES
L1 1.4
L2 1.0
A1 105.0
D1 120.0
BASIS=6-31G*
OPTIM
SCF DFTP=B3LYP
FORCE
JUMP
GEOM SYMM=0.0
DYNA STEP=50 TEMP=800 MAXC=30 -----
SCF DFTP=B3LYP | dynamics
FORCE | loop
JUMP -----
```

21. Ethylene Z-Matrix potential scan along C-C bond

```
TEXT= C2H4 HF/3-21G Z-matrix potential scan
GEOM=ZMAT GEOP
C
C 1 L1
H 1 1.0 2 120.0
H 1 1.0 2 120.0 3 180.0
H 2 1.0 1 120.0 3 180.0
H 2 1.0 1 120.0 5 180.0
VARIABLES
L1 1.2
BASIS=3-21G
SCAN ZMAT L1 FROM 1.2 1.5 0.05 -----
GEOM=ZMAT print=1 | potential scan
SCF | loop
JUMP -----
```

22. Ethylene optimized potential scan along C-C bond

```
TEXT= C2H4 HF/3-21G optimized potential scan
GEOM=ZMAT GEOP
C
C 1 L1
H 1 1.0 2 120.0
H 1 1.0 2 120.0 3 180.0
H 2 1.0 1 120.0 3 180.0
H 2 1.0 1 120.0 5 180.0
VARIABLES
L1 1.2
BASIS=3-21G
SCAN stre 1 2 FROM 1.2 1.5 0.05 -----
OPTIM ----- |
SCF | optimization | potential scan
FORCE | loop | loop
JUMP ----- |
JUMP -----
```

Note the order of the commands; the optimization loop is *inside* the potential scan loop. This job will scan the C-C distance and optimize the molecular geometry at each scanned distance. The optimization will be carried out in delocalized internals.

23. H₂CO <---> H₂ + CO Cartesian reaction path

```
TEXT  H2CO <---> H2 + CO  reaction path search
GEOM=PQS  GEOP
c    -0.48011661324879    -0.05270447746945    0.0000000000000000
o    -0.68486418742763    -1.21833101973701    0.0000000000000000
h     1.18428634708622     0.33221463198601    0.0000000000000000
h    -0.01930554640980     0.93882086522045    0.0000000000000000
BASIS=6-31G**
PATH coord=cart sign=+1 print=3 ITER=10  -----
SCF  DFTP=BLYP                          | reaction path
PATH                                     | loop
FORCE                                    |
JUMP                                     -----
```

A reaction path search downhill from the transition state. The input geometry *must* be that of the transition state at the level of theory being used; additionally the exact transition state Hessian matrix *must* be available on the <hess> file.

Here the path will be defined in Cartesian coordinates with the initial step being along the negative Hessian eigenmode (the mode corresponding to the negative eigenvalue) in a *positive* sense (i.e., exactly as output from the Hessian diagonalization - to search downhill in the opposite direction repeat this job with `sign=-1`). Ten points on the reaction path will be found (in addition to the initial transition state geometry).

Note that there are *two* PATH commands. Both must be present. During the line search along the gradient bisector the FORCE command will not be accessed (in effect the job will loop between the two PATH commands only). Once a new point on the reaction path has been found, the full loop will be implemented to compute the steepest descent and the gradient bisector.

24. HCN <--> HNC Z-matrix reaction path

```
TEXT HCN <--> HNC reaction path search
GEOM=ZMAT
C
N 1 L1
H 1 L2 2 A1
VARIABLES
L1 1.191655
L2 1.188583
A1 71.6011
BASIS=6-31G**
PATH coord=zmat print=3 sign=-1 ITER=10
GEOM=ZMAT print=1
SCF DFTP=B3LYP
PATH
FORCE
JUMP
```

25. Water dual basis MP2

```
%MEM=6
TITLE= H2O dual basis MP2
GEOM=PQS
O 0.0 0.0 0.0
H 0.0 0.8 0.6
H 0.0 -0.8 0.6
BASIS=3-21G
SCF ITER=5
BASIS=6-311G(d,p)
SCF LOCA=PIPEK
MP2 --- conventional MP2 with 6-311G** basis
BASIS=6-311G(3df,3dp) --- larger basis must be an extension of smaller
GUESS=READ
MP2 DUAL MAXDisk=50 --- dual basis MP2; 6-311G** for SCF
SCF LOCA=PIPEK 6-311G(3df,3dp) for MP2
MP2 --- conventional MP2 with 6-311G(3df,3dp)
```

26. QM/MM geometry optimization on SeP(CH₃)₃

```
TEXT= Test of new QM/MM module
GEOM=PQS SYMM=0.005 BOHR
se      .0000000000000000      .0000000000000000      .0000000000000000
p      3.98883557319641      .0000000000000000      .0000000000000000
$molecule
c      5.30083513259888      2.72307753562927      -1.50575280189514
c      5.32141399383545      .0000000000000000      3.09815073013306
c      5.30085372924805      -2.72307753562927      -1.50573432445526
h      7.38374853134155      2.66850209236145      -1.37349081039429
h      4.63818359375000      4.45043897628784      -.55669420957565
h      4.76076984405518      2.78649663925171      -3.48898339271545
h      7.41268062591553      .0000000000000000      2.96388578414917
h      4.73644876480103      -1.68607091903687      4.13668775558472
h      4.73643016815186      1.68607091903687      4.13668775558472
h      7.38376760482788      -2.66850209236145      -1.37347209453583
h      4.63822126388550      -4.45043897628784      -.55667573213577
h      4.76078891754150      -2.78649663925171      -3.48896479606628
OPTIM QMMM print=3
SEMI NOGUESS print=3      ----- MM part: PM3 calculation on full system
QMMM print=3              ----- defines model system by adding link atoms
SEMI NOGUESS print=3      ----- MM part: PM3 calculation on model system
QMMM                      ----- prepares for QM part
BASIS=STO-3G
SCF LVSH=3.0              ----- QM part: SCF calculation on model system
FORCE
QMMM                      ----- restores full system
JUMP
```

In this example, the Se=P part of SeP(CH₃)₃ will be treated quantum mechanically (RHF/STO-3G) and the methyl groups are all treated semiempirically (PM3). Note the use of the `$molecule` designator to separate the QM (upper) from the MM (lower) part of the system. A higher than normal print flag is set to provide more detail in the output file.

27. CH₃OH geometry optimization + analytical frequencies

```
%MEM=9
TEXT= Methanol RHF/DZP optimization + analytical frequencies
GEOM=ZMAT
C
O   1   L1
H   1   L2   2   A1
H   1   L3   2   A2   3   D1
H   1   L3   2   A2   3   -D1
H   2   L4   1   A3   3   180.0
VARIABLES
L1  1.4
L2  1.08
L3  1.08
L4  1.0
A1  109.5
A2  109.5
A3  108.0
D1  120.0
BASIS=DZP_dunning
OPTIM
SCF
FORCE
JUMP
HESS
FREQ
```

28. MP2 energy for water with cc-pVQZ basis (includes g-functions)

```
%MEM=10
TITLE= MP2/cc-pvqz for water (includes spherical g functions)
GEOM=pqs
O   0   0   0.118234
H   0   0.758161  -.472936
H   0   -.758161  -.472936
BASIS=3-21G
SCF ITER=5
BASIS=cc-pvqz
SCF THRE=5.5
MP2 THRE=10.5
```

29. B3LYP/cc-pVTZ optimization of formaldehyde + NMR with WAH functional

```
TITLE= Formaldehyde B3LYP/cc-pvtz optimization + WAH NMR
GEOM=ZMAT
C
O C L1
H C L2 O A1
H C L2 O A1 3 180.0
VARIABLES
L1 1.2
L2 1.08
A1 120.0
BASIS=6-31G*
SCF DFTP=B3LYP ITER=5
BASIS=cc-pvtz
OPTIM
SCF DFTP=B3LYP
FORCE
JUMP
SCF DFTP=WAH
NMR
```

30. OLYP/6-311G** optimization of CO + NMR with level shift

```
TITLE= CO OLYP/6-311G** optimization + NMR with level shift
GEOM=PQS
C 0.0 0.0 0.0
O 0.0 0.0 1.128
BASIS=3-21G
SCF DFTP=OLYP ITER=5
BASIS=6-311G**
OPTIM
SCF DFTP=OLYP
FORCE
JUMP
NMR LVSH=0.025
```

31a. MP2 optimization of Hydrogen Fluoride using Jensen's PC-2 basis set

```
%MEM=10
TITLE= HF  MP2/PC-2 optimization
GEOM=PQS
H      0.0    0.0    0.0
F      0.0    0.0    1.0
BASIS=pc-0
SCF ITER=5
BASIS=pc-2
OPTIM
SCF THRE=6.0 LOCA=PIPEK
MP2
FORCE
JUMP
```

31b. MP2-SCS optimization of Hydrogen Fluoride using Jensen's PC-2 basis set

```
%MEM=10
TITLE= HF  MP2/PC-2 optimization
GEOM=PQS
H      0.0    0.0    0.0
F      0.0    0.0    1.0
BASIS=pc-0
SCF ITER=5
BASIS=pc-2
OPTIM
SCF THRE=6.0 LOCA=PIPEK
MP2 SCS
FORCE
JUMP
```

Two MP2 optimizations, the first using regular MP2 the second using spin-component scaled (SCS) MP2.

32. PBE/6-311G** optimization + analytical frequencies for NO

```
%MEM=9
TITLE= NO PBE/6-311G** optimization + analytical frequencies
GEOM=PQS MULT=2
N 0.0 0.0 0.0
O 0.0 0.0 1.128
BASIS=3-21G
SCF DFTP=PBE ITER=5
BASIS=6-311G**
OPTIM
SCF DFTP=PBE
FORCE
JUMP
HESS
FREQ
```

33. OLYP/PC-2 energy with FTC + semidirect for CHFClCH₂F

```
%MEM=30 CORE=10
TITLE= CHFClCH2F OLYP/PC-2 FTC + semidirect
GEOM=PQS
c -0.657640 -0.119772 -0.174228
cl -0.670475 1.644447 -0.173545
h 1.167983 -0.241420 -1.034052
f -1.342668 -0.619081 1.033700
h -1.173453 -0.461608 -1.035296
c 0.655721 -0.585532 -0.173024
f 1.342668 -0.089308 1.035296
h 0.663109 -1.644447 -0.170884
BASIS=3-21G
SCF DFTP=OLYP ITER=6
BASIS=pc-2
SCF DFTP=OLYP SEMI PWAVE
POP
```

Because of the nature of the basis set, the FTC method is faster than the standard all-integral algorithm for this calculation, even for such a relatively small system.

34. BVP86 optimization + frequency for HCl in gas phase + water using COSMO

```
%MEM=9
TITLE= HCl  BVP86/svp_ahlrichs opt + freq + COSMO
GEOM=PQS
H      0.0    0.0    0.0
Cl     0.0    0.0    1.4
BASIS=svp_ahlrichs
OPTIM
SCF DFTP=BVP86
FORCE
JUMP
HESS
FREQ
COSMO SOLV=WATER
OPTIM
SCF DFTP=BVP86
FORCE
JUMP
NUMHESS fdstep=0.02
GEOM NOORIENT print=1
SCF DFTP=BVP86
FORCE
JUMP
GEOM print=1
FREQ
COSMO OFF
SCF DFTP=BVP86
FORCE
```

Here we are first carrying out a standard optimization plus vibrational analysis of HCl (i.e., in the gas phase), followed by an optimization using the COSMO solvation model with water as the solvent. Analytical second derivatives are not available with COSMO, so the Hessian matrix is calculated numerically. At the end of the calculation COSMO is switched off and a single-point energy plus gradient are computed to determine the energy of the COSMO optimized geometry and the residual forces in the gas phase at the COSMO-optimized geometry.

35. RHF/CEP-121 optimization + frequency + NMR for ethanol

```
TEXT= ethanol ecp test job cep basis
GEOM=PQS GEOP
C   -1.282208929  -0.260589603   0.015650763
H   -1.338952086  -0.842127894   0.945243893
C   -0.004407982   0.567846618  -0.032066673
H   -1.331051729  -0.955863284  -0.834513160
H   -2.163200295   0.394178188  -0.037210856
H    0.022643530   1.177808866  -0.950049011
H    0.045402298   1.252220956   0.824503097
O    1.191141650  -0.242010134   0.059924695
H    1.189850477  -0.846145158  -0.703537329
BASIS=cep-121 print
OPTIM
SCF
FORCe
JUMP
HESS
FREQ
NMR
```

A standard optimization of ethanol (no symmetry) using an ECP basis set. An analytical Hessian (plus vibrational analysis) and NMR are done at the optimized geometry, both using the ECP basis.

36. MP2/6-311G** for H2 dimer with ghost atoms and symmetry

```
TITLE=  H2 dimer    ghost atoms with symmetry
GEOM=PQS
H    0.36    1.00    0.0
H   -0.36    1.00    0.0
H$   0.36   -1.00    0.0    0.0
H$  -0.36   -1.00    0.0    0.0
BASIS=6-311G** NEXT
FOR      H$    BASIS=6-311G**
SCF THRE=6.0 LOCA=PIPEK
MP2
```

This shows how to use ghost atoms and still utilize symmetry. If the special "\$" symbol were *not* used on the ghost H atoms, then the job would stop, complaining that symmetry (D_{2h}) was being broken, as the charges on all the hydrogen atoms are not the same. Giving the ghost H atoms a different symbol ($H\$$ instead of H) allows the program to recognize a different type of hydrogen atom, lowers the symmetry to C_{2v} and enables the job to run.

37. RHF/3-21G optimization + frequency + NMR + VCD for CHFCIBr

```
TEXT=Test of VCD
GEOM=PQS
c      -0.03155715494684      0.05784758579888      0.13730601371952
f      -0.03948600303304      1.39983830048394      0.12931336665829
cl     1.64766123566997      -0.53244336383942      0.17971931585716
h      -0.55725000123065      -0.32537841681946      1.00930665044019
br     -1.01936807645943      -0.59986410562394      -1.45564534667514
BASIS=3-21G
OPTIM
SCF THRE=6.0
FORCE
JUMP
NMR VCD
HESS
FREQ
```

Note the order of the key words for the VCD analysis.

38. B97/3-21G optimization + frequency + full population analysis on cyclopropane

```
%MEM=5 CORE=5
TEXT=cyclopropane Test Job
GEOM=PQS SYMM=0.005
c      0.43212770207642      0.74846713535435      0.00000000000000
h     -1.44959369213872      0.00000000000000     -0.90285062495995
h      0.72479684606936     -1.25538496255781      0.90285062495995
h      0.72479684606936      1.25538496255781     -0.90285062495995
h      0.72479684606936      1.25538496255781      0.90285062495995
c     -0.86425540415284      0.00000000000000      0.00000000000000
h      0.72479684606936     -1.25538496255781     -0.90285062495995
h     -1.44959369213872      0.00000000000000      0.90285062495995
c      0.43212770207642     -0.74846713535435      0.00000000000000
BASIS=3-21G
OPTIM
SCF DFTP=B97 SEMI
FORCE
JUMP
HESS
FREQ
POP=FULL
```

39. PM3 multiple constraints for 2-aminopropionic acid

```
TITLE= 2-aminopropionic acid Multiple constraints test
GEOM=PQS
H          -0.033086      1.169295      1.037328
N          -0.070432      0.176872      1.465463
C          -0.073029     -0.851293      0.402646
C           1.154853     -0.742968     -0.466683
O           1.948633      0.177758     -0.363854
C          -1.347330     -0.713890     -0.468542
H           0.774958      0.039270      2.126502
O           1.328319     -1.713145     -1.360929
H           2.124181     -1.552627     -1.853694
H          -2.244634     -0.778874      0.165780
H          -0.079658     -1.845221      0.882325
H          -1.351803      0.257040     -0.987249
H          -1.388230     -1.516545     -1.221647
OPTimize PRINT=3
$fix
  2  XYZ
  3  XYZ
$endfix
$constraint
bend  1  2  7  110.0
#composite
stre  3  4
stre  3  6
#endcomposite
$endconstraint
SEMI=PM3
JUMP
```

Here we have a combination of fixed atoms, a regular bond angle constraint and a composite distance constraint. The fixed atoms will be converted to a simple distance constraint and the optimization will be done in delocalized internal coordinates.

40. B3LYP/6-311G** optimization + analytical frequencies + polarizabilities for HOF

```
TITLE= HOF OPT + FREQ + polarizability + hyperpolarizability
GEOM=ZMAT
O
F O L1
H O L2 F A1
VARIABLES
L1 1.4
L2 1.0
A1 105.0
BASIS=3-21G
SCF DFTP=B3LYP ITER=6
BASIS=6-311G**
OPTIM
SCF DFTP=B3LYP THRE=6.0
FORCE
JUMP
HESS
POLAR
FREQ
```

41. HF/6-31++G Energy for 1,2-dimethylbenzene with suppressed basis functions

```
TITLE= 1,2-dimethylbenzene  suppressed basis functions
GEOM=PQS
C      0.423830      0.599746      -0.012182
C     -0.694842     -0.247221      0.000896
C     -0.508061     -1.635163     -0.027432
C      0.777710     -2.182187     -0.024606
C      1.890093     -1.339053      0.003211
C      1.710177      0.046163      0.007150
H      1.056969      2.598551     -0.654219
H     -2.175068      1.104966      0.851863
H     -1.363110     -2.303317     -0.050066
H      0.913443     -3.259262     -0.040311
H      2.891040     -1.758647      0.021294
H      2.583887      0.689412      0.025920
C      0.260275      2.126175     -0.060432
H     -0.702373      2.399463     -0.516479
H      0.304489      2.525471      0.963509
C     -2.119125      0.325134      0.078097
H     -2.390178      0.759423     -0.895456
H     -2.859156     -0.449656      0.329245
BASIS=3-21G
SCF ITER=6
BASIS=6-31++g
SCF THRE=6.0
```

This is a standard input file for a single-point energy; however the basis set contains a large number of diffuse functions and the lowest eigenvalue of the overlap matrix is 6.29×10^{-7} , which is below the default cutoff for basis function suppression of 10^{-6} . Consequently the combination of basis functions giving rise to this eigenvalue is eliminated. (The next lowest eigenvalue, which is also printed out, is 2.49×10^{-6} .)

42. RHF/3-21G optimization plus NMR with NICS for butane

```
TITLE= NICS test on butane
GEOM=PQS
C      -1.461826      1.292042      0.000000
H      -1.461826      2.371433      0.000000
H       0.508818      1.135042     -0.881326
H      -1.970644      0.932245      0.881326
H      -1.970644      0.932245     -0.881326
C       0.000000      0.775244      0.000000
H       0.508818      1.135042      0.881326
H      -0.508818     -1.135042      0.881326
C       0.000000     -0.775244      0.000000
H      -0.508818     -1.135042     -0.881326
H       1.970644     -0.932245     -0.881326
C       1.461826     -1.292042      0.000000
H       1.970644     -0.932245      0.881326
H       1.461826     -2.371433      0.000000
BASIS=3-21G
OPTIM
SCF
FORCE
JUMP
NMR NICS=1,6,9 GRID=6 STEP=1.0 DISP=1.0
```

As well as the normal NMR chemical shifts for each atom, additional nuclear-independent chemical shifts will be computed in a plane passing through atoms 1, 6 and 9 but displaced perpendicularly (along the Z-axis) by 1.0 bohr. The grid will be a 6 x 6 square, with a distance of 1.0 bohr between the grid points and with one additional point at the “centre” of the three atoms defining the plane, for a total of 37 additional chemical shifts.

43. RHF/3-21G optimization + frequency (IR + Raman + VCD) + QCISD single-point energy for butane

```
%MEM=40
TEXT= butane RHF/3-21G OPT + FREQ(IR+Raman) + QCISD
GEOM=PQB FILE=test43.pqb
!!!!!!!!!!!!!! STEP 1 !!!!!!!!!!!!!!!
BASIS=3-21g
OPTimize
SCF THRE=6.0
FORCe
JUMP
FORCe
NMR VCD
HESS
CORR=QCISD MEMO=40
NUMPolar pold
GEOM PRINT=1 NOORient
SCF THRE=6.0
FORCe
JUMP
GEOM PRINT=1
FREQ
```

Note the additional `FORCe` command *after* the geometry optimization loop. This would appear to be redundant, but the gradient file is deleted at the end of the optimization and a computed force is needed for the `NUMPolar` loop. The single-point QCISD energy is done *before* the numerical evaluation of the polarizability derivatives which are used to derive Raman intensities in the frequency step, which is done last. There is a `GEOM` card before the `FREQ` command to ensure that any molecular symmetry (which may have been lost during the numerical polarizability loop) is restored.

This input file was generated by *PQSMol*, our graphical user interface.

44. QCISD(T)/6-311G* numerical geometry optimization for water

```
%MEM=20
TITLE  H2O  QCISD(T)/6-311G*  Numerical optimization
GEOM=ZMAT
O
H O 1.0
H O 1.0 2 105.0
BASIS=6-311G*
OPTIM print=3
NUMGrad fdstep=0.005
GEOM NOORIENT print=1
SCF THRE=6.0
CORR=QCISD TRIP MEMO=30
JUMP
GEOM print=1
SCF THRE=6.0
CORR=QCISD TRIP MEMO=30
JUMP
```

Note that the numerical gradient (the `NUMGrad` loop) is evaluated *before* the energy in the optimization loop. The gradient is computed via central differences on the energy by displacing each symmetry-unique atom. There is no point calculating the energy at the initial geometry until that geometry has been restored after gradient evaluation as the corresponding MOs at the initial geometry will be overwritten after each finite-difference displacement. With the SCF energy calculated *after* the numerical gradient, the correct MOs will be available for the QCISD step. Note the `GEOM` card after the `NUMGrad` loop which restores any symmetry before the energy is calculated.

45. UMP2/6-311G* numerical geometry optimization for ozone

```
%MEM=30
TITLE  UMP2/6-311G*  Numerical optimizationa  Ozone
GEOM=ZMAT MULT=3
O
O 1 1.4
O 1 1.4 2 120.0
BASIS=6-311G*
OPTIM print=3
NUMGrad fdstep=0.005
GEOM NOORIENT print=1 MULT=3
SCF THRE=6.0 LOCA=PIPEK
MP2 NOIO
JUMP
GEOM print=1 MULT=3
SCF THRE=6.0 LOCA=PIPEK
MP2 NOIO
JUMP
```

This input file is similar to the previous one (test 44) except that it is an open-shell UMP2 geometry optimization. Note the `NOIO` option on the `MP2` command line which forces the in-core algorithm, which is faster than the default for such a small system.

46. B3LYP/6-31G* optimization + frequency + SQM frequency for chlorobenzene

```
%MEM=100MB
GEOM=PQS
C      0.000000      0.000000      1.342139
C      1.210694      0.000000      0.643308
C      1.210986      0.000000     -0.754645
C      0.000000      0.000000     -1.453392
C     -1.210986      0.000000     -0.754645
C     -1.210694      0.000000      0.643308
Cl     0.000000      0.000000      3.093905
H      2.150247      0.000000      1.186779
H      2.151379      0.000000     -1.297357
H      0.000000      0.000000     -2.538821
H     -2.151379      0.000000     -1.297357
H     -2.150247      0.000000      1.186779
BASIS=3-21G
SCF DFTP=B3LYP ITER=6
BASIS=6-31G*
OPTIM
SCF DFTP=B3LYP THRE=6.0
FORCE
JUMP
HESS
FREQ
SQM
FREQ
```

A straightforward geometry optimization + frequency analysis followed by an SQM scaling of the Hessian matrix (using the default scaling parameters) together with a second frequency analysis. Both the output and log files will contain the two sets of frequencies (the second one scaled) and thermodynamic analyses.

Note that the first frequency analysis is performed on the Hessian matrix resulting from the HESS command; the second is on the scaled Hessian resulting from the SQM command. This will overwrite the initial Hessian, so unless this is saved it will be lost.

47. B3LYP/6-311G** enforced geometry optimization for HOF

```
TITLE=  HOF  geometry optimization with external force
GEOM=ZMAT
O
F O L1
H O L2 F A1
VARIABLES
L1 1.4
L2 1.0
A1 105.0
BASIS=3-21G
SCF DFTP=B3LYP ITER=6
BASIS=6-311G**
OPTIM
SCF DFTP=B3LYP THRE=6.0
FORCE
$force
2 3 0.03 PULL
$endforce
JUMP
```

An example of an enforced geometry optimization with an additional force of 0.03 au applied to both the F and H atoms along the line joining their centres so as to pull the two atoms apart. Compare the final optimized geometry in this example with that from example 40.

48. <Example needed>

RESTARTS and CHECKPOINTS

Files saved and available on successful job completion include:

<control>	
<coord>	final geometry (Cartesian coordinates)
<sym>	symmetry data
<basis>	basis set data
<mos>	converged alpha/closed-shell SCF MOs (binary file)
<mob>	converged beta spin SCF MOs (binary file)
<hess>	Cartesian Hessian (exact if from NUMHESS job; approximate if from OPTIMIZE)
<deriv>	dipole moment derivatives (from NUMHESS)
<zmat>	final Z-matrix (if Z-matrix optimization)

This collection of files constitutes the *checkpoint* or *data archive* for that job. Much of this data can be reused for, e.g., running a similar job on the same system at a higher level of theory or with a different basis set.

A typical example is reoptimizing a molecule at a higher level of theory. Assume you have the data archive for a RHF/3-21G optimization of a particular molecule and you want to reoptimize the geometry at B3LYP/6-31G*.

Input would be as follows:

```
FILE save=molecule
TEXT= B3LYP/6-31G* optimization from RHF/3-21G archive
GEOM=read ----- get geometry from old <coord>
BASIS=6-31G*
GUESS=READ ----- use old MOs as initial guess
OPTIM ----- will automatically pick up any
GUESS=READ pre-existing <hess> file
SCF dftp=b3lyp
FORCE
JUMP
```

The following files need to be available (and in the directory in which the new job is being run)

<control>	always needed
<coord>	
<basis>, <mos>	
<hess>	

Job Crashes

Geometry optimizations can easily be restarted from the <optchk> file following job crashes (note - this is *not* necessarily the case if the job aborts with an error message).

The OPTIMIZE module is set up to *automatically* read data from the <optchk> file if one exists. To restart a job that crashes somewhere in the main optimization loop use

```
TEXT= Restart following job crash
GEOM=read                ----- must use current geometry
BASIS=6-31G*
OPTIM                    ----- will automatically pick up data
SCF dftp=b3lyp           from <optchk> file
FORCE
JUMP
```

Note that the original input geometry (if one was given in the input file) *must* be removed and either replaced with the current geometry or (better) by a direct read from the latest <coord> file (as shown above).

<p>**IMPORTANT** The geometry optimization restart is different from earlier versions of PQS (3.1 and earlier) in that there is no longer any need to add SCF and FORCE cards before the OPTIM card as was previously the case.</p>
--

Restart of Numerical Hessian runs

Frequency runs that crash in the numerical Hessian loop can be restarted from the <hesschk> file. The restart procedure is very similar to the restart of a geometry optimization from the <optchk> file.

Initial job

```
TEXT= Numerical Hessian + frequencies on previously converged geometry
GEOM=pqs file=molecule.tex
BASIS=6-31G*
GUESS
NUMHESS fdstep=0.02
GEOM noorient print=1
GUESS=READ
SCF dftp=b3lyp
FORCE
JUMP 5
GEOM print=1
FREQ
```

restart job

```
TEXT= Numerical Hessian + frequencies Restart
GEOM=read noorient ----- read in geometry on loop
BASIS=6-31G* ----- that crashed from <coord>
GUESS
SCF dftp=b3lyp ----- recalculate energy and
FORCE ----- gradient
NUMHESS fdstep=0.02 ----- will automatically pick up data
GEOM noorient print=1 ----- from <hesschk> file
GUESS=READ
SCF dftp=b3lyp
FORCE
JUMP
GEOM print=1
FREQ
```

To successfully restart you need the files <control>, <coord> and <hesschk>

RUNNING PQS IN PARALLEL

The most computationally intensive parts of the PQS *ab initio* program package are implemented for parallel execution using the message-passing paradigm. This coarse-grained parallelism and relatively limited communication provides good scaling up to a few dozen processors.

Our implementation uses the SPMD (Single Program, Multiple Data) master-slave model for wide applicability. The program will act as a master (driving the calculation) or as a slave depending on the context of the parallel execution environment. The master takes the input from the standard input; the slaves do not read the input file. Usually the master should be running on the fastest, best-equipped processor available for optimal performance.

The communication interface of the program could easily accommodate any message-passing system. Presently two architectures are available: the widely used, flexible and efficient PVM and the industry-standard, supposedly fast MPI. We have implemented both architectures but PVM generally provides better performance on small clusters, and is somewhat easier to use.

A prerequisite of **any** parallel run is that the user should be able to rlogin (remote login) to all slave hosts without using a password. This can be achieved systemwide by specifying all machines in a cluster in the */etc/hosts.equiv* file, or individually by a read-only file *.rhosts* in the home directory of the user. This can be checked by typing *rlogin hostname* where *hostname* is the host name of a slave machine.

PVM

To use the PVM environment each user has to set up the *PVM_ROOT* and *PVM_DPATH* environmental variables, preferably in the login scripts. *\$PVM_ROOT* should be the root directory of the PVM installation (this is */progs/pvm3*). It is also useful to define an environmental variable *PVM* as *\$PVM_ROOT/lib/pvm* (use the command *setenv PVM \$PVM_ROOT/lib/pvm* in *csh* or *tcsh*). *PVM_DPATH* should be set to *\$PVM_ROOT/lib/pvmd*.

These directories/paths have already been set up on your machine in all supplied user accounts.

PVM manages tasks by daemon processes on each machine used in the computation. The daemons can be started and controlled via the PVM console, *\$PVM* (or *\$PVM_ROOT/lib/pvm* if the PVM environmental variable has not been set). New machines can be added to the setup by the *add hostname* command of the console.

Other useful commands are:

conf	check configuration
delete	removes a machine from the PVM setup
ps	used as ps a it shows all PVM jobs running, including the master (-) and the pvm group server
quit	exits the PVM setup but leaves the PVM daemon running and preserves the last PVM configuration
reset	kills a parallel job
halt	kills a parallel job, stops the PVM daemon, and removes the PVM configuration

If a parallel job needs to be stopped, or if it dies and the slaves are still running, use **reset**, otherwise the next job cannot be started. It never hurts to issue a **reset** in the PVM console before starting a parallel job. If a job dies, it may leave a file in the **/tmp** directory (**/tmp/pvmd.userid** where *userid* is your Unix number). This prevents other PVM jobs from running. The preferable way of removing this file is to issue a **halt** command in the PVM console; however sometimes PVM cannot be invoked in which case you should delete the **/tmp/pvmd.userid** file by hand. **This behavior is the leading cause of complaints about parallel jobs.** Otherwise, using the latest version of PVM, it is remarkably problem free.

Specific to Your System

Your QuantumCube™ consists of dual-processor motherboards linked by gigabit ethernet. Each motherboard has a particular ID (or hostname). These are usually simply n1 though nm, where *m* is the number of nodes on your system.

The connection to the outside world (usually 10/100 Mb/s ethernet) is <yourhostname.yourdomainname>, i.e., a full internet domain name supplied by your local network administrator.

Let's say you want to set up and run a 4 processor parallel job (input file gen.inp) using the two dual-processor motherboards on n1 and n2.

Login to, say, n1. Type **\$PVM**. At the prompt type **conf**. You should see that n1 is automatically included in the PVM configuration. Type **add** n2. Type **conf** again. You should now see that both n1 and n2 are included in the PVM configuration. Type **reset** then type **quit** to exit.

The program has the ability to assign the slaves automatically; to do this, the hosts must be listed in the home directory in the (invisible) file `.txhosts`. *This has been done on your system already.* **Note that this feature can work only if PVM is not yet running. If PVM is running already, the program will use the existing configuration.** In general, we found that it is preferable to configure the parallel virtual machines by hand. The configuration survives (for a given user) logging in and out. However, PVM must be reconfigured if the machine has been shut down.

The master process can be started now:

```
/usr/local/share/PQS/pqs_pvm.x -np 5 gen.inp &
```

Note that the absolute path names must be used for the PVM executable. This is essentially equivalent to

```
/usr/local/share/PQS/pqs gen 4 or simply pqs gen 4
```

where `pqs` is a script in `/usr/local/share/PQS`. NOTE that the script needs the number of slaves (4 in this case) while the program itself uses the number of processes (5 including the master; see below). Using the `pqs` script is preferable to calling `/usr/local/share/PQS/pqs_pvm.x` directly, as the script performs other useful checks.

The slaves are started by the master, one on each node in the configuration list including the master node. A different number of processes can be given by the number following the `-np` option. The number following `-np` should be **one larger than the number of slaves**, as the master process also counts. In the current implementation, the first slave starts on the *second* node in the list, and then the other slaves follow, with the slave on the master node (assuming there is one) *last*. The cycle is repeated if there are more slaves than nodes. Of course, only on nodes with more than one CPU per host does it make sense (apart from testing) to define more processes than CPUs.

The program has to be started with an **explicit full pathname**, because PVM does not perform a full path search. (Alternatively, the program can be copied or linked into `$PVM_ROOT/bin/$PVM_ARCH`, the only place that is searched.)

The program can perform the setup of the configuration as well, instead of using the PVM console. This can be necessary in a queuing system.

If there is no virtual machine setup when the program is started, it is possible to provide a host list file on the command line. E.g.:

```
/progs/PQS/pqs_pvm.x -np 5 [/mydir/myhostlist] < gen.inp > gen.out &
```

where the brackets denote an optional argument. This method eliminates interference from stopped or hung slaves as well. The format of the hostlist file is the same as that in, e.g., the **.rhosts** file.

PARALLEL JOB QUEUE: SUN GRID ENGINE (SGE)

Provided with the QuantumCube™ is a powerful parallel job queuing utility called Sun Grid Engine (SGE). This enables multiple users to submit jobs to a job queue, allowing a more orderly and controlled use of the CPU cycles on each processor. Sun Grid Engine apparently has the same parentage as the Distributed Queuing System (DQS) which we used to use, but is much more “professional”. At one time it was only available commercially, but has been released by SUN as part of their Linux “open-source” initiative. All the basic DQS commands still function under SGE.

The basic commands are:

qsub <i>jobscript</i>	submits a job to the queue
qstat <i>-f</i>	displays the current status of the queue
qdel <i>jobnumber</i>	deletes a job from the queue
qconf	configures the queue

A simple script to submit a job to the queue might be:

```
#$ -cwd
#$ -pe pvm 1-4
#$ -S /bin/csh
#$ -masterq n1.q
#$ -q n1.q,n2.q,n1.q,n2.q
pqsh gen 4
```

Commands to the queue are prefixed by “#\$”. The first line changes to the current working directory (all files will be read from or written to this directory). The next line declares that the job will be parallel using PVM. The third line exports the current shell to all the nodes. The fourth line indicates which node will be the master for this job and the next line defines the PVM configuration (two processors on n1 and two on n2 - note there should be *no* spaces between the commas defining the PVM). The final line submits the job. The entire script is submitted via **qsub** *jobscript*.

There are manual pages for all the SGE commands (obtainable via `man qsub`). Additionally there is an interactive graphical queue monitoring system that can be invoked by typing `qmon` at the terminal prompt; this allows full monitoring and interaction with the parallel job queue graphically.

For more details see the .pdf documentation files in **/progs/SGE/doc**. (These can be seen by, e.g., typing `xpdf <filename.pdf>`.)

DQS was developed by the Supercomputer Computations research Institute at Florida State University and is supported by the United States Department of Energy through contract DE-FC05-85ER250000 and the state of Florida. Questions and comments concerning DQS should be forwarded to <dqs@scri.fsu.edu>.

INPUT DESCRIPTION FOR SQM version 2.0

INTRODUCTION

SQM is an add-on module for the PQS program which scales force constants to produce a Scaled Quantum Mechanical (SQM) Force Field. This can correct for deficiencies in the calculated (harmonic) force constants, giving a better fit to experimentally observed vibrational fundamentals and infrared (IR) intensities. The method has considerable predictive power and is often of great help in understanding and assigning experimental vibrational spectra.

The first *ab initio* calculations of harmonic force constants were carried out at the Hartree-Fock level of theory. Hartree-Fock theory overestimates harmonic force constants significantly and empirical scaling is needed to produce force constants which can reproduce experimental vibrational frequencies. The scaling compensates for basis set deficiencies, anharmonicity and mostly for the lack of electron correlation. The need for empirical correction diminishes but is not completely eliminated if the quality of the wavefunction improves by adding electron correlation and increasing the size of the basis.

The first scaling methods applied to *ab initio* force constants used several different scale factors to correct for systematic errors in different types of molecular deformations, e.g., stretches, bends and torsions. This procedure requires the transformation of the molecular force field (the Hessian matrix) to chemically meaningful internal coordinates and cannot be applied directly to the calculated frequencies. It is thus less convenient than global scaling using a single scaling factor. Global scaling can be applied directly to the frequencies, the scale factor for frequencies being near 0.9, corresponding to a scale factor of 0.81 for force constants. Because of its simplicity, global scaling became popular, but using multiple scale factors yields much better results as was convincingly demonstrated by Blom and Altona in a series of papers starting in the mid 1970s [111]. Their method forms the basis of the SQM procedure which has been in widespread use for over 20 years [112].

In the original SQM procedure, the molecular geometry was expressed in terms of a full set of nonredundant natural internal coordinates [99]. Natural internals use individual bond displacements as stretching coordinates and localized linear combinations of bond angles and torsions as deformational coordinates. (They are the precursors to the delocalized internal coordinates used in PQS, which are linear combinations of *all* stretches, bends and torsions in the molecule [98].) On the basis of chemical intuition, the natural internal coordinates of all molecules under consideration are sorted into groups sharing a common scaling factor, and factors for each group are determined by a least-squares fit to experimental vibrational frequencies. Force constants, originally calculated in Cartesian coordinates, are transformed into an internal coordinate representation, and scaling is applied to the elements of the internal force constant matrix (*not* to the individual vibrational frequencies) according to

$$F_{ij}(\text{scaled}) = (s_i s_j)^{1/2} F_{ij}$$

where s_i and s_j are scaling factors for internal coordinates i and j , respectively.

The accuracy obtained by selective scaling in this way is naturally greater than if just a single overall scaling factor were used. Additionally, scaling the force constant matrix also affects the resultant normal modes, and hence the calculated intensities (which are unaffected if only the frequencies are scaled), leading to better agreement with experimental intensities.

The SQM procedure has been widely used in the interpretation of vibrational spectra. A further important role is the development of *transferable* scale factors which can be used to modify calculated force constants and so predict the vibrational spectrum a priori.

The SQM module uses a modified scaling procedure involving the scaling of *individual* valence coordinates [113] (*not* the linear combinations present in natural internal coordinates). This has immediate advantages in terms of ease of use, as no natural internals need to be generated (a procedure which may fail for complicated molecular topologies), and it simplifies the classification and presorting of the coordinates. In addition, the extra flexibility involved in the scaling of individual primitive internals generally leads to an increase in accuracy and to more transferable scale factors.

The user is encouraged to view the references provided, especially ref. 113.

PROGRAM CAPABILITY AND INPUT

SQM capabilities include

1. Scaling a force constant matrix using a set of one or more precalculated scale factors. Eleven optimized scale factors are available for standard organics containing H, C, N, O and Cl for force constants calculated at B3LYP/6-31G*. This is one of the most cost-effective and reliable theoretical methods currently available. The recommended scale factors are [113]:

stretch	X-X	0.9207
stretch	C-Cl	1.0438
stretch	C-H	0.9164
stretch	N-H	0.9242
stretch	O-H	0.9527
bend	X-X-X	1.0144
bend	X-X-H	0.9431
bend	H-C-H	0.9016
bend	H-N-H	0.8753
torsion	all	0.9523
linear bends	all	0.8847

where X denotes a non-hydrogen atom (C, N or O).

2. Adjusting atomic masses to give normal modes, frequencies and IR and potentially Raman and VCD intensities for isotopomers.
3. Optimizing scale factors to give the best least-squares fit to a set of experimental vibrational frequencies.
4. Carrying out a total energy distribution analysis [114] to determine how much a given primitive (stretch, bend or torsion) contributes to a particular normal mode.
5. Determining invariant diagonal force constants for all the stretches, bends and torsions in a molecule
6. Carrying out a full vibrational and thermodynamic analysis based on the scaled force constant matrix. The analysis is the same as that in PQS.

INPUT FILE

A sample input file for formamide (HCONH₂) with all keywords shown is given below:

```
$molecules
formamide
$scaling
stre  X    X          1    0.9202    fixed
stre  C    H          2    0.9163    fixed
stre  N    H          3    0.9239
bend  X    X    X      4    1.0108    fixed
bend  X    X    H      5    0.9438    fixed
bend  H    N    H      6    0.8765
tors  H    X    X    X   7    0.9525    fixed
tors  H    X    X    H   7    0.9525    fixed
$print_ted  3.0
$print_level  4
$max_atoms 10
$end
```

\$molecules: file prefix names for <hess>, <deriv>, <aat> and <evib> files

SQM needs molecular geometries, force constant (Hessian) matrices, dipole (and possibly quadrupole) derivatives and atomic axial tensors as input. The latter are available following a PQS frequency run in the files `jobname.hess`, `jobname.deriv` and `jobname.aat` - in this example these files are called `formamide.hess`, `formamide.deriv` and `formamide.aat`. **These files must exist.** (SQM will still function if the <deriv> or <aat> files are missing, but no IR, Raman or VCD intensities will be available.)

Various molecules can be grouped together (for example to determine the best scale factors for the whole set). In this case all the file prefix names should be given, one per line with no blank lines, following the **\$molecules** keyword.

\$scaling: scaling parameters

<scale type> <atom types> <scale group> <value> <action>

<scale type>: one of `stre`, `bend`, `tors`, `linc`, `linp`
(corresponding to stretches, planar bends, proper torsions, and colinear and coplanar bend, respectively. Out-of-plane bends (`outp`) are currently not accessible)

<atom types>: up to four atomic symbols (X for any non-hydrogen atom)
depending on the <scale type>

<scale group>: scale factors with the same (integer) scale group number will be
grouped together during any optimization of the scale factors.
The scale group number should start at 1 and must be listed
consecutively as shown in the example input

<value>: initial scale factor value

<action>: either `fixed` (for a fixed scale factor) or `optimize`
The default if this field is blank is to optimize the scale factor

In the formamide example, there are seven scale factors, with the two different torsions in the molecule grouped together. The scale factors for the N-H stretch and the H-N-H bend will be optimized to give the best least-squares fit to a set of experimental frequencies, keeping all the other scale factors fixed at their initial input values.

****IMPORTANT**** When applying the scale factors, SQM takes each scale factor in turn *in the order they appear in the input file* and scales any Hessian element that fits. Care must be taken that scale factors involving the use of "X" - for a general non-hydrogen atom - appear *first* in the list of each scale type (`stre`, `bend`, `tors`, `linc`, `linp`) as, if these appear *subsequent* to a specific atom type (say a C-C stretch), then the specific atoms will be taken as general non-hydrogen atoms and the wrong scale factor will be applied.

Note also that "X" *cannot* be used in place of a hydrogen atom. Thus if all torsions, say, in a given molecule/set of molecules are to be scaled with the same scale factor, then any torsions involving hydrogen must be specifically provided. Thus `tors X X X X` alone will *not* scale any torsions involving hydrogen; you also need to specify both `tors H X X X` and `tors H X X H` to scale these torsions.

The other input options are:

\$print_ted=<real>: print threshold for total energy distribution analysis. This analysis gives the percentage contribution of each primitive stretch, bend and torsion in the molecule to each normal mode. The default if no value is given is 5.0, i.e., only primitives which contribute 5% or more to a given normal mode will be printed for that mode.

\$print_level=<integer>: controls the amount of printout (larger integer - more printout). In particular, a value of 4 will print the normal modes. Values higher than 4 will progressively output more and more intermediate quantities constructed during the SQM procedure and are essentially for debug printout.

\$max_atoms=<integer>: for allocating memory. Should be set to at least the number of atoms in the largest molecule under consideration (the default is 50).

\$end terminates input (**must** be present)

The <evib> File

The <evib> file contains both the molecular geometry and, if scale factors are being optimized, the experimental vibrational frequencies. The *formamide.evib* file is given below:

```
$coordinates      angstrom
C      0.4121292508   0.0816374866   0.0000000000
O      0.4653658528  -1.1331720094   0.0000000000
H      1.3139166617   0.7266945306   0.0000000000
N     -0.7368390692   0.8133281236   0.0000000000
H     -1.6267160783   0.3334309776   0.0000000000
H     -0.7250294253   1.8221287816   0.0000000000
$point_group      cs
$nsymop           2
$frequencies      J.Raman Spectros.  25 (1994) 183
0.0
608.
646.
841.    0.0d0
1090.
1309.
1391.
1602.
1692.
2882.
3190.
0.0
$end
```

The **\$coordinates** section contains the geometry in Cartesian coordinates. The format is: atomic symbol X Y Z atomic mass (A8,2X,4F20.14).

This is essentially the same format as the PQS <coord> file *except that the atomic charge is missing*. Coordinates can be given either in bohr or angstrom; the units are specified following the **\$coordinates** string as shown. If no units are specified, the default is bohr.

Atomic Masses

There are two ways of specifying isotopomers. The atomic mass can be given following the coordinates (as per the above format) or the isotope can be specified as a part of the atomic symbol, e.g., H-2 will use deuterium instead of hydrogen for that atom, C-13 will give carbon 13. If *neither* of these options is specified, then the isotopically averaged atomic mass will be used.

The SQM program has built-in isotopically averaged atomic masses for all elements up to and including Xenon (N=54). There are also up to four individual isotope masses for each of these elements (if an element has more than four isotopes, then the four with the highest percentage abundance are available). The atomic masses for any atoms not included in the above description **must** be specified in the **\$coordinates** section.

Use of Symmetry

The inclusion of full point group symmetry is the major difference between SQM version 2.0 and previous versions. Knowing the molecular point group allows appropriate symmetry labels to be given to each vibrational mode and also allows for a proper thermodynamic analysis. The data required are the point group symbol and the number of symmetry operations.

\$point_group=<string>: character string (maximum four characters) giving the point group symbol, e.g., cs, c2v, td etc....

\$nsymop=<integer>: the number of symmetry operations in the point group, e.g., 2, 4, 24 etc....

Symmetry information can either be provided in the <evib> file following the geometry (as shown above) or it can be read from a PQS <sym> file, if one exists. Note that if a <sym> file does exist then it will be read and any symmetry information given in the <evib> file will be overwritten. If no symmetry data is given, C₁ symmetry (i.e., no symmetry) will be assumed. If there *is* symmetry and it is not used, then certain quantities calculated in the thermodynamic analysis will be incorrect.

SQM version 2.0 makes use of the same symmetry routines as PQS and we have made a conscious effort to make the output from SQM as close as possible to the corresponding output in PQS. Indeed if all you want to do is an SQM scaling of the force constant matrix using known scale factors, then this can now be done within PQS itself using the newly introduced `SQM` keyword (see the main PQS section in this manual).

The **\$frequencies** section contains the experimental (or other) frequencies that will be used in the least-squares fit. The format is <frequency> (in cm^{-1}) <weight> with each frequency on a separate line.

weight=<real>: gives the weight each vibrational frequency is given in the least-squares fit. Frequencies that are not known accurately can be given a lower weight in the fitting; conversely frequencies that are regarded as being reliable or for which a good agreement is particularly desired can be given a higher weighting. The default if no weight is given is $1000 \times \text{inverse experimental frequency (in } \text{cm}^{-1}\text{)}$.

If the experimental frequency for a particular vibration is very suspect or if it is not known at all, it should be given a zero weight. The number of fundamentals a molecule has is given - for a non-linear system it is $3N-6$, where N is the number of atoms. Quite often there are *less* than $3N-6$ vibrational fundamentals that are reliably known. In this case, the **\$frequencies** section should have one or more lines containing zeroes (which correspond to an unknown frequency with a zero weight in the fit). It may be necessary to vary the position of these zero lines in the input depending on the accuracy of the fit; note also that although frequencies should generally be given in increasing order, it may be that two fairly close values with different symmetries may need to be switched. This can often be easily detected if one mode is strongly IR active whilst the other is only weakly active or IR inactive; the theoretical mode with the large IR intensity should be fit to the experimentally IR active mode.

It is not so uncommon to find experimentally assigned bands that you simply cannot fit at all because they have, in fact, been misassigned. The SQM procedure, when used correctly with a good theoretical method (such as B3LYP/6-31G*), usually gives average errors in band positions of around 8 cm^{-1} , and maximum errors of the order of $20-30 \text{ cm}^{-1}$. If you find maximum errors significantly outside this range, there is a good chance that the experimental assignment is wrong.

\$end terminates input (**must** be present)

In the formamide example, both the lowest and the highest frequency fundamentals are not known experimentally (hence the two zero lines) and the fundamental assigned experimentally at 841 cm^{-1} is considered to be unreliable and has been given zero weight in the fit. The source of the experimental data (J.Raman Spectros. **25** (1994) 183) has been given after the **\$frequencies** keyword as a reminder to the user.

Invariant (“relaxed”) Force Constants

Force constants in internal coordinates can be obtained by a suitable transformation of the Cartesian force constant matrix. Diagonal elements of the internal coordinate force constant matrix give individual stretching, bending and torsional force constants. Unfortunately, the values of these internal coordinate force constants depend on which primitive internals have been chosen to describe the molecular geometry. For example, if a given stretch is present in two sets of internal coordinates (which contain different bends and/or torsions), both sets of which can be used to describe a molecular geometry, then the stretching force constant calculated using one set of coordinates will *not* have the same value in the other set. This fact is perhaps not as widely known as it ought to be.

However, if the force constants are defined *not* as the diagonal elements of the Hessian matrix in internal coordinates, but instead as the inverse of the diagonal elements of the *inverse* Hessian, then the two stretching force constants in the example above *will* be the same. In this way force constants in internal coordinates can be defined in an *invariant* way, independent of the precise choice of coordinates. The inverse Hessian matrix is known as the compliance matrix. For a fairly recent discussion on relaxed force constants see [115].

Invariant force constants defined in this manner can be output from the SQM program by setting `$print_level` to 4 in the SQM input file.

****IMPORTANT**** Both invariant force constants and a total energy distribution of the normal modes will only be performed for *single* molecules, i.e., there must be only one molecule filename specified in the `$molecules` section.

Program Usage

```
sqm.x filename
```

where the input file should be `filename.inp` and only the file prefix (i.e., no `.inp`) should be given. Output will be in `filename.out` and files associated with the molecule(s) specified in the `$molecules` section in `filename.inp`, namely `molecule.evib`, `molecule.hess` and (optionally) `molecule.deriv` and `molecule.aat` must exist.

The SQM program produces four temporary files: `sqm-temp1`, `sqm-temp2`, `sqm-temp3` and `sqm-temp4`. As these do not have unique names associated with the input only *one* SQM job should be run in a given directory at any one time.

FREQUENTLY ASKED QUESTIONS

I am unable to run PQS in parallel. When I try to start PVM, I get a cryptic message “Cannot start pvmd” (the PVM daemon). What is the problem?

Assuming that the network is functioning, the most likely cause of this problem is that a previous parallel calculation has been killed manually or because a computer was shut down or crashed (e.g., in a power outage). PVM keeps a file, `pvmd.nnn` where `nnn` is the user's Unix number (usually around 500 for ordinary users in Linux) in the `/tmp` directory. The presence of this file signals that a PVM job is already running and prevents another PVM job starting. This file is normally removed at the end of the job, except when the job or the machine dies. Use the `pvm_reset` script if you have one to remove these files remotely, or log in on the master and each slave and manually remove the `pvmd` file *on each computer you want to use for a parallel job. You do not have to be the superuser to do this.*

My small jobs are running OK but a large job keeps crashing, seemingly with lack of memory, even though I have increased the memory allocation beyond what may be conceivably needed. What should I do?

Check if the `%MEM` card is the *very first* card of the input deck. Due to the programming model used, dynamic memory allocation must be the first statement executed.

My energy is going up in the early stages of SCF, becoming unreasonable. What is the reason?

This is possible for heavier elements, in particular transition metals. The problem is the quality of the initial guess, in particular the guess for the core orbitals. Start the calculation with a small basis set (3-21G or even STO-3G) for a limited number of cycles, *and use level shift. A value of 2-3 E_h is recommended but particularly difficult cases may require level shifts over 5 E_h . Run only the preliminary calculation with high level shift, as large shifts slow down SCF convergence in the final stages.*

A calculation which ran perfectly well last week keeps crashing with a file read error. Why?

This is probably an optimization, force field, scan, or path job, or a dual-basis MP2 job. The program stores information accumulated during the job in intermediate files, such as the `<jobname>.optchk` file. If the calculation is killed or crashes, these files remain behind and the program tries to use the information on them. This may lead to format errors. Run `tidy <jobname>` before running PQS.

I am unable to get a semiempirical guess. The semiempirical program does not converge. What should I do?

Use a level shift. The syntax is the same as for SCF. LVSH=1 is often effective.

I am trying to get an open-shell singlet wavefunction (e.g., for ozone, O₃) but the wavefunction stubbornly gives the closed-shell solution. I have tried GUESS=UHFS MIX but it does not help. Where is the error?

In this case, the UHFS wavefunction must break the spatial symmetry (C_{2v} for O₃). Keeping the symmetry forces the system into the closed-shell state. Use SYMM=0.0 on the GEOM card.

My SCF calculation has converged but I forgot to do a population analysis and compute nuclear properties. I have the converged <mos> file, but when I read it in to repeat the SCF it takes several cycles to converge. Why doesn't it converge immediately?

The default in an SCF is to start the calculation with a loose integral threshold, then tighten the threshold for the final convergence. The final orbitals which are saved on the <mos> file are typically converged with the *tight* threshold; if these are used with the *loose* threshold, they will need additional (unnecessary) cycles to converge. When reading in converged orbitals from a previous calculation, you should use the tight threshold from the start. Add the line INTE THRE=10,10 *before* the SCF card; this should result in immediate (2 cycle) convergence in the SCF step.

REFERENCES

- [1] D. B. Chestnut and C. K. Foley, *Chem. Phys. Lett.* **118** (1985) 316;
D. B. Chestnut and K. D. Moore, *J. Comput. Chem.* **10** (1989) 648
- [2] R. C. Raffanetti, *J. Chem. Phys.* **58** (1973) 4452
- [3] (a) W. J. Stevens, H. Basch and M. Krauss, *J. Chem. Phys.* **81** (1984) 6206
(b) W. J. Stevens, M. Krauss, H. Basch and P. G. Jaisan,
Can. J. Chem. **70** (1992) 612
(c) T. R. Cundari and W. J. Stevens, *J. Chem. Phys.* **98** (1993) 5555
- [4] P. J. Hay and W. R. Wadt, *J. Chem. Phys.* **82** (1985) 270, 284, 299
- [5] (a) L. F. Pacios and P. A. Christiansen, *J. Chem. Phys.* **82** (1985) 2664
(b) M. M. Hurley, L. F. Pacios, P. A. Christiansen, R. B. Ross and W. C. Ermler,
J. Chem. Phys. **84** (1986) 6840
(c) L. A. LaJohn, P. A. Christiansen, R. B. Ross, T. Atashroo and W. C. Ermler,
J. Chem. Phys. **87** (1987) 2812
(d) R. B. Ross, J. M. Powers, T. Atashroo, W. C. Ermler, L. A. LaJohn and
P. A. Christiansen, *J. Chem. Phys.* **93** (1990) 6658
(e) W. C. Ermler, R. B. Ross and P. A. Christiansen,
Int. J. Quantum Chem. **40** (1991) 829
(f) C. S. Nash, B. E. Bursten and W. C. Ermler, *J. Chem. Phys.* **106** (1997) 5133
- [6] A complete list of references for the Stuttgart-Cologne pseudopotentials can be found at <http://www.theochem.uni-stuttgart.de/pseudopotentials/index.en.html>
- [7] F. Weigend and R. Ahlrichs, *Phys. Chem. Chem. Phys.* **7** (2005) 3297
- [8] L. R. Kahn and W. A. Goddard III, *J. Chem. Phys.* **52** (1972) 2685
- [9] M. Wolfsberg and L. Helmholz, *J. Chem. Phys.* **20** (1952) 837
- [10] R. Hoffmann, *J. Chem. Phys.* **39** (1973) 1397
- [11] D. Cremer and J. Gauss, *J. Comput. Chem.* **7** (1986) 274
- [12] W. J. Hehre, R. F. Stewart and J. A. Pople, *J. Chem. Phys.* **51** (1969) 2657
- [13] J. Andzelm, M. Klobukowski, E. Radzio-Andzelm, Y. Sakai and H. Tatewaki
in *Gaussian Basis Sets For Molecular Calculations* ed. S. Huzinaga
(Elsevier, Amsterdam, 1984)
- [14] J. C. Slater, *Quantum Theory of Molecules and Solids*, vol.4
(McGraw-Hill, New York, 1974)
- [15] S. H. Vosko, L. Wilk and M. Nusair, *Can. J. Phys.* **58** (1980) 1200
- [16] A. D. Becke, *Phys. Rev. A* **38** (1988) 3098
- [17] C. Lee, W. Yang and R. G. Parr, *Phys. Rev. B* **37** (1988) 785;
B. Miehlich, A. Savin, H. Stoll and H. Preuss, *Chem. Phys. Lett.* **157** (1989) 200
- [18] The P86 functional contains a local part – J. P. Perdew and A. Zunger,
Phys. Rev. **B23** (1981) 5048 – and a nonlocal part – J. P. Perdew,
Phys. Rev. **B33** (1986) 8822
- [19] J. P. Perdew, in *Electronic Structure of Solids*, ed. P. Ziesche and H. Eschrig
(Akademie, Berlin, 1991)
- [20] N. C. Handy and A. J. Cohen, *Mol. Phys.* **99** (2001) 403
- [21] A. D. Becke, *J. Chem. Phys.* **98** (1993) 5648
- [22] W-M Hoe, A. J. Cohen and N. C. Handy, *Chem. Phys. Lett.* **341** (2001) 319
- [23] J. P. Perdew, K. Burke and M. Ernzerhof, *Phys. Rev. Lett.* **77** (1986) 3865
- [24] A. D. Becke, *J. Chem. Phys.* **107** (1997) 8554

-
- [25] F. A. Hamprecht, A. J. Cohen, D. J. Tozer and N. C. Handy, *J. Chem. Phys.* **109** (1998) 6264
- [26] P. J. Wilson, T. J. Bradley and D. Tozer, *J. Chem. Phys.* **115** (2001) 9233
- [27] A. D. Doese and N. C. Handy, *J. Chem. Phys.* **114** (2001) 5497
- [28] P. J. Wilson, R. D. Amos and N. C. Handy, *Chem. Phys. Lett.* **312** (1999) 475
- [29] A. D. Becke, *J. Chem. Phys.* **98** (1993) 1372
- [30] R. H. Hertwig and W. Koch, *Chem. Phys. Lett.* **268** (1997) 345
- [31] J. Baker and P. Pulay, *J. Chem. Phys.* **117** (2002) 1441
- [32] J. Baker and P. Pulay, *J. Comput. Chem.* **24** (2003) 1184
- [33] P. Pulay, *Chem. Phys. Lett.* **73** (1980) 393; *J. Comput. Chem.* **3** (1982) 556
- [34] V. R. Saunders and I. H. Hillier, *Int. J. Quant. Chem.* **7** (1973) 699
- [35] J. J. P. Stewart, P. Császár and P. Pulay, *J. Comput. Chem.* **3** (1982) 227
- [36] S. F. Boys, in *Quantum Theory of Atoms, Molecules and the Solid State*, ed. P. O. Löwdin (Academic Press, New York, 1968) p. 253
- [37] J. Pipek and P. G. Mezey, *J. Chem. Phys.* **90** (1989) 4916
- [38] A. V. Mitin, J. Baker, K. Wolinski and P. Pulay, *J. Comput. Chem.* **24** (2003) 154
- [39] (a) L. Füsti-Molnar and P. Pulay, *J. Chem. Phys.* **117** (2002) 7827;
(b) L. Füsti-Molnar, *J. Chem. Phys.* **119** (2003) 11080
(c) J. Baker, L. Füsti-Molnar and P. Pulay, *J. Phys. Chem. A* **108** (2004) 3040
- [40] B. I. Dunlap, *Phys. Rev. A* **42** (1990) 1127
- [41] K. Eichkorn, O. Treutler, H. Öhm, M. Häser and R. Ahlrichs, *Chem. Phys. Lett.* **240** (1995) 283
- [42] M. Beyer and H. Clausen-Schaumann, *Chem. Rev.* **105** (2005) 2921
- [43] K. Wolinski and J. Baker, *Mol. Phys.*, **107** (2009) 2403
- [44] J. Baker and K. Wolinski, *J. Comput. Chem.* **32** (2011) 43
- [45] J. Baker and K. Wolinski, *J. Molec. Modelling* in press
- [46] R. Ditchfield, *J. Chem. Phys.* **65** (1976) 3123
- [47] K. Wolinski, J. F. Hinton and P. Pulay, *J. Am. Chem. Soc.* **112** (1990) 8251
- [48] G. Rauhut, S. Puyear, K. Wolinski and P. Pulay, *J. Phys. Chem.* **100** (1996) 6310
- [49] G. Schreckenbach and T. Ziegler, *J. Phys. Chem.* **102** (1995) 606
- [50] G. Magyarfalvi and P. Pulay, *J. Chem. Phys.* **119** (2003) 1350
- [51] V. G. Malkin, O. L. Malkina, M. E. Casida and D. R. Salahub, *J. Am. Chem. Soc.* **116** (1994) 5898
- [52] P. v. R. Schleyer, C. Maerker, A. Dransfeld, H. Jiao and N. J. R. v. E. Hommes, *J. Am. Chem. Soc.* **118** (1996) 6317
- [53] K. Wolinski, *J. Chem. Phys.* **106** (1997) 6061
- [54] J. R. Cheeseman, M. J. Frisch, F. J. Devlin and P. J. Stephens *Chem. Phys. Lett.* **252** (1996) 211
- [55] S. Grimme, *J. Chem. Phys.* **118** (2003) 9095
- [56] Y. Jung, R. C. Lohan, A. D. Dutoi and M. Head-Gordon, *J. Chem. Phys.* **121** (2004) 9793
- [57] K. Wolinski and P. Pulay, *J. Chem. Phys.* **118** (2003) 9497
- [58] R. P. Steele, R. A. DiStasio, Y. Shao, J. Kong and M. Head-Gordon, *J. Chem. Phys.* **125** (2006) 074108
- [59] P. Pulay, S. Saebo and K. Wolinski, *Chem. Phys. Lett.* **344** (2001) 543
- [60] J. Baker and P. Pulay, *J. Comput. Chem.* **23** (2002) 1150
- [61] J. Baker, K. Wolinski and P. Pulay, *to be published*

-
- [62] S. Saebo, J. Baker, K. Wolinski and P. Pulay, *J. Chem. Phys.* **120** (2004) 11423
- [63] J. A. Pople, J. S. Binkley and R. Seeger, *Int. J. Quant. Chem., Suppl.* **Y-10** (1976) 1
- [64] K. Raghavachari and J. A. Pople, *Int. J. Quant. Chem.* **14** (1978) 91
- [65] S. F. Boys, *Proc. Roy. Soc. (London)* **A200** (1950) 542; *ibid.* **A201** (1950) 125
- [66] W. Meyer, *J. Chem. Phys.* **58** (1973) 1017
- [67] J. A. Pople, M. Head-Gordon and K. Raghavachari, *J. Chem. Phys.* **87** (1987) 5968
- [68] J. Cizek, *Advances in Chemical Physics*, Ed. P. C. Hariharan, vol.14, p. 35 (Wiley, New York, 1969)
- [69] A. R. Ford, T. Janowski and P. Pulay, *J. Comput. Chem.* **28** (2007) 1215
- [70] T. Janowski, A. R. Ford and P. Pulay, *J. Chem. Theory Comput.* **3** (2007) 1368
- [71] T. Janowski and P. Pulay, *J. Chem. Theory Comput.* **4** (2008) 1585
- [72] T. Janowski and P. Pulay, *Chem. Phys. Lett.* **47** (2007) 27
- [73] M. Pitonak, T. Janowski, P. Neogrady, P. Pulay and P. Hobza, *J. Chem. Theory Comput.* **5** (2009) 1761
- [74] R. S. Mulliken, *J. Chem. Phys.* **23** (1955) 1833
- [75] P. O. Löwdin, *Phys. Rev.* **97** (1955) 1474
- [76] J. Baker, *Theoret. Chim. Acta* **68** (1985) 221
- [77] U. C. Singh and P. A. Kollman, *J. Comput. Chem.* **5** (1984) 129
- [78] C. M. Breneman and K. B. Wiberg, *J. Comput. Chem.* **11** (1990) 361
- [79] J. Cioslowski, *J. Am. Chem. Soc.* **111** (1989) 8333
- [80] V. A. Rassolov and D. M. Chipman, *J. Chem. Phys.* **105** (1996) 1470
- [81] B. Wang, J. Baker and P. Pulay, *Phys. Chem. Chem. Phys.* **2** (2000) 2131
- [82] (a) A. Klamt and G. Schüürmann, *J. Chem. Soc. Perkin Trans. 2* (1993) 799
(b) J. Andzelm, C. Kölmel and A. Klamt, *J. Chem. Phys.* **103** (1995) 9312
(c) A. Klamt and V. Jonas, *J. Chem. Phys.* **105** (1996) 9972
(d) K. Baldrige and A. Klamt, *J. Chem. Phys.* **106** (1997) 6622
- [83] (a) A. Klamt, *J. Phys. Chem. A* **99** (1995) 224
(b) A. Klamt, V. Jonas, T. Burger and J. C. W. Lohrenz, *J. Phys. Chem. A* **102** (1998) 5074
- [84] E. Hückel, *Z. Phys.* **70** (1931) 204; *ibid.* **76** (1932) 628
- [85] J. A. Pople and D. L. Beveridge, *Approximate Molecular Orbital Theory* (McGraw-Hill, New York, 1970)
- [86] J. A. Pople and G. A. Segal, *J. Chem. Phys.* **43** (1965) S136; *ibid.* **44** (1966) 3289
- [87] J. A. Pople, D. L. Beveridge and P. A. Dobosh, *J. Chem. Phys.* **47** (1967) 2026
- [88] M. J. S. Dewar and N. C. Baird, *J. Chem. Phys.* **50** (1969) 1262
- [89] M. C. Zerner, G. H. Loew, R. F. Kirchner and U. T. Mueller-Westerhoff, *J. Am. Chem. Soc.* **102** (1980) 589
- [90] M. J. S. Dewar, *A Semiempirical Life* (ACS, Washington, 1992)
- [91] R. C. Bingham, M. S. J. Dewar and D. H. Lo, *J. Am. Chem. Soc.* **97** (1975) 1285
- [92] M. J. S. Dewar and W. Thiel, *J. Am. Chem. Soc.* **99** (1977) 4899
- [93] M. J. S. Dewar, E. Zoebisch, E. F. Healy and J. J. P. Stewart, *J. Am. Chem. Soc.* **107** (1985) 3902
- [94] J. J. P. Stewart, *J. Comput. Chem.* **10** (1989) 209, 221; *ibid.* **11** (1990) 543
- [95] W. Thiel and A. A. Voityuk, *J. Am. Chem. Soc.* **100** (1996) 616
- [96] M. Clark, R. D. Cramer and N. van Opdenbosch, *J. Comput. Chem.* **10** (1989) 982
- [97] A. K. Rappé, C. J. Casewit, K. S. Colwell, W. A. Goddard and W. M. Skiff, *J. Am. Chem. Soc.* **114** (1992) 10024

-
- [98] J. Baker, A. Kessi and B. Delley, *J. Chem. Phys.* **105** (1996) 192
- [99] (a) P. Pulay, G. Fogarasi, F. Pang, J. Boggs, *J. Am. Chem. Soc.* **101** (1979) 2550;
(b) G. Fogarasi, X. Zhou, P. W. Taylor, P. Pulay, *J. Am. Chem. Soc.* **114** (1992) 8191
- [100] P. Pulay and G. Fogarasi, *J. Chem. Phys.* **96** (1992) 2856
- [101] J. Baker, *J. Comput. Chem.* **18** (1997) 1079
- [102] J. Baker, *J. Comput. Chem.* **13** (1992) 240
- [103] J. Baker and P. Pulay, *J. Comput. Chem.* **21** (2000) 69
- [104] J. Baker, *J. Comput. Chem.* **7** (1986) 385
- [105] P. Császár and P. Pulay, *J. Mol. Struct. (Theochem)* **114** (1984) 31
- [106] M. Svensson, S. Humbel, R. D. J. Froese, T. Matsubara, S. Sieber and K. Morokuma, *J. Phys. Chem.* **100** (1996) 19357
- [107] S. Dapprich, I. Komáromi, K. S. Byun, K. Morokuma and M. J. Frisch, *J. Mol. Struct. (Theochem)* **461** (1999) 1
- [108] K. Fukui, *J. Phys. Chem.* **74** (1970) 4161
- [109] K. Ishida, K. Morokuma and A. Komornicki, *J. Chem. Phys.* **66** (1977) 2153
- [110] M. W. Schmidt, M. S. Gordon and M. Dupuis, *J. Am. Chem. Soc.* **107** (1985) 2585
- [111] (a) C. E. Blom and C. Altona, *Mol. Phys.* **31** (1976) 1377
(b) C. E. Blom, L. P. Otto and C. Altona, *Mol. Phys.* **32** (1976) 1137
(c) C. E. Blom and C. Altona, *Mol. Phys.* **33** (1977) 875; *ibid.* **34** (1977) 1137
- [112] P. Pulay, G. Fogarasi, G. Pongor, J. E. Boggs and A. Vargha, *J. Am. Chem. Soc.* **105** (1983) 7037
- [113] J. Baker, A. A. Jarzecki and P. Pulay, *J. Phys. Chem. A* **102** (1998) 1412
- [114] P. Pulay and F. Török, *Acta. Chim. Acad. Sci. Hung.* **47** (1965) 273
- [115] J. Baker, *J. Chem. Phys.*, **125** (2006) 14103