MATHSCOUT Background and Tutorial

Michael P. Barnett[a*] and Joseph F. Capitani[b†]

[a]Meadow Lakes, Hightstown, NJ 08520,
[b]Joined Departments of Chemistry and Biochemistry,
Manhattan College/The College of Mount Saint Vincent,
Riverdale, NY 10471

May 28, 2007

Abstract

MATHSCOUT is a MATHEMATICA[‡] package to postprocess the output of other programs for scientific calculations. We wrote MATHSCOUT to import data from a major program for *ab initio* computational chemistry into MATHEMATICA, so that we could postprocess the chemical results. It can be used to import the output of many other packages that are used, *e.g.* in molecular dynamics, crystallography, spectroscopic analysis, metabolic and physiological modeling, meteorology and other areas of environmental science, cosmology and particle physics. MATHSCOUT assigns a name to each table and non-tabular datum that it extracts. This name is constructed mechanically from the identifier or phrase that precedes or follows or embeds the item in the output that MATHSCOUT processes. A selection of non-contiguous items, or all the items in a section of the file, or in the entire file are extracted using simple commands. So far, we have focused on our immediate needs to postprocess the output of the Gaussian[§] program. Calculations on several molecules that illustrate the usage of the package are presented here. MATHSCOUT is shortened to `msct` in the software.

1 Introduction

Recently, we used Gaussian 03 [4] to compute structural properties of a series of molecules containing 4-atom rings, and ran MATHEMATICA [18] scripts to compute puckering angles from data in the output [3]. The electronic cutting and pasting from Gaussian output to MATHEMATICA input was tedious but routine enough for mechanization to seem possible. We have taken time out from the substantive calculations to explore this issue.

Several authors have written programs over the past decade that act on the output of successive versions of the Gaussian package. The programs include

1. APOST-MS, APEX4, BO-VIR and other programs of Mayer and his associates that are discussed in [9, 10, 11] and papers cited therein. These programs can be downloaded from `occam.chemres.hu/programs`.
2. FREQCHK included in the Gaussian 03 distribution [4], see `www.gaussian.com`.

*michaelb@princeton.edu
†joseph.capitani@manhattan.edu
‡MATHEMATICA is a registered trademark of Wolfram Research Inc.
§GAUSSIAN is a registered trademark of Gaussian Inc.

3. the associated GAUSSVIEW of Gaussian Inc. [5], see `www.gaussian.com`,
4. GOPENMOL, maintained by Laaksonen [7], see `www.csc.fi/gopenmol`,
5. HYPERCHEM 7.5 for Windows, a product of Hypercube, Inc. [6], see `www.hyper.com`.
6. MOLDEN of Schaftenaar and Noordik [15], see `www.cmbi.ru.nl/molden`.
7. MOLEKEL of Flükiger, Portmann, and Lüthi[13] see `www.cscs.ch/molekel`,
8. PCMODEL, Serena Software [17], see `www.serenasoft.com`.
9. WEBMO version 6.0 of Polik and Schmidt [16], see `www.webmo.net`, and
10. JUMBO of Murray-Rust and his associates[12], see `www.ch.ic.ac.uk/omf/cml/doc`.

Some of these programs draw and render

1. ball, stick, wire, and CPK models (4–9),
2. electronic densities and electrostatic potentials (4–9),
3. potential energy surfaces and animations (4–9).

Some perform further calculations

1. to compute bond orders [9, 11], exact energy decomposition [8], decomposition for fuzzy atoms [14], and mass spectrometric cleavages [10],
2. to compute electron densities and, from these, electrostatic potential, distributed multipoles, and Laplacians [15],
3. to compute vibrational frequencies (3–9),
4. for bond order and multipole moment analysis (3–9),
5. in demonstrations of convergence behaviour (10).

The packages 1, 3 and 4 import Gaussian data from the "formatted checkpoint file" (FPCHK) whilst the packages 5, 7 and 9 import data from both the FPCHK and the LOG files. The package 6 uses the LOG file. Murray-Rust is developing a virtual consortium for quantum chemistry programs and "where possible will collaborate with the program author(s) to add CML as an output format" [12].

We have written MATHEMATICA scripts to extract data from the abundant information that is contained in the Gaussian output file as well as from the FPCHK file to meet a variety of needs. For example, whilst the checkpoint file provides the data for rendering many properties, data in the main output file of the scanning job is needed to render the plots of energy and the virial *versus* bond length in nitrogen shown in Figures 1 and 2.

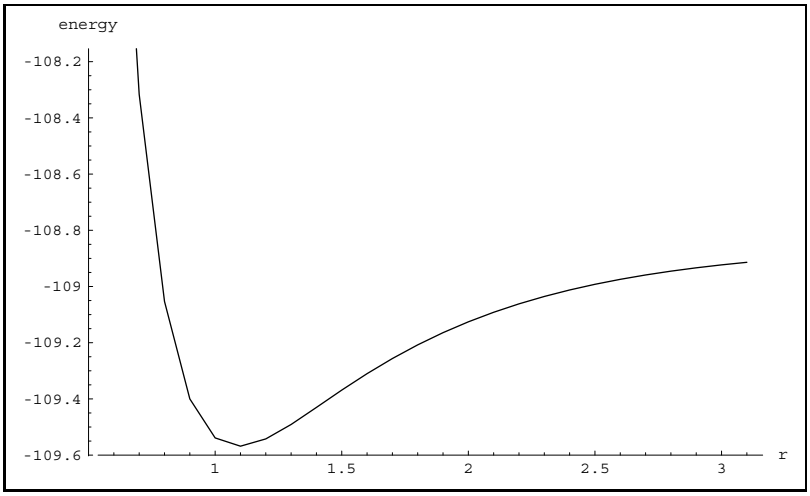

Figure 1. Energy *vs* internuclear distance in N_2.

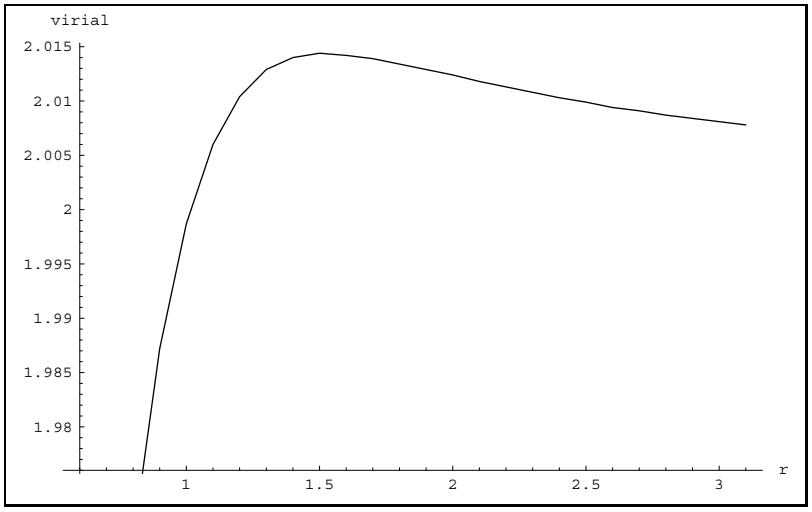

Figure 2. Virial *vs* internuclear distance in N_2.

The log file is needed, too, for the data that shows the convergence of the average zinc-oxygen distance in a calculation on $Zn(H_2O)_6^{++}$. Table 1 lists these averages and the spreads *i.e.* (max-min)/average.

iteration	1	2	3	4	5	6	7	8	9	10
average	2.544	2.509	2.461	2.414	2.369	2.327	2.280	2.214	2.148	2.121
spread	0.001	0.007	0.015	0.023	0.029	0.033	0.036	0.041	0.043	0.027
iteration	11	12	13	14	15	16	17	18	19	20
average	2.115	2.113	2.122	2.133	2.145	2.147	2.139	2.125	2.117	2.113
spread	0.014	0.018	0.012	0.009	0.021	0.029	0.040	0.029	0.012	0.008
iteration	21	22	23	24	25	26	27	28	29	30
average	2.116	2.119	2.122	2.122	2.120	2.115	2.118	2.117	2.117	2.117
spread	0.005	0.011	0.014	0.011	0.004	0.006	0.003	0.003	0.003	0.003

Table 1. Convergence of Zn-O distance in $Zn(H_2O)_6^{++}$ iteration.

The Gaussian output files contain a considerable variety of tables and individual data that are identified in several different styles. We have written some scanning utilities in MATHEMATICA to deal with this diversity. These comprise the MATHSCOUT package in the Computer Physics Communications Program Library. The scanning commands for a particular application are based on the structure of the lines that contain the data to be extracted. The keywords and phrases that accompany the data in the Gaussian (or other) output are coaleseced and cosmeticized mechanically into identifiers that conform to MATHEMATICA syntax. These identifiers are associated with the data and the tables in lists of MATHEMATICA `Equal` statements. Typical statements in these lists are

```
nuclearRepulsionEnergy == 9.2635178625
StandardBasis == "6-311G(2d,p) (5D, 7F)"
symbolicZmatrix == {{O}, {H, 1, r}, {H, 1, r, 2, a}}
initialValuesOfVariables == {r == 0.95, a == 105}
```

Several simple MATHEMATICA language devices are used to extract individual results from these lists. The full range of MATHEMATICA resources that perform mathematical operations and deal with character strings, files and graphics then

can be brought to bear on this imported information. The `==` (`Equal`) symbol is used instead of the `=` (`Set`) symbol to connect an identifier and its value. This allows multiple uses of the same identifier that can be distinguished by context, *e.g.* the lists in which they occur, or by tagging.

Other authors have incorporated the output of Gaussian runs in hundreds of research reports and in numerous teaching documents. Whilst some of these incorporations were made with the help of postprocessors, as mentioned above, it seems likely that electronic cutting and pasting was used for most. The white papers by Cheeseman, Frisch, Ochterski and unnamed authors on the Gaussian website `http://www.gaussian.com/g_whitepap/white_pap.htm` illustrate the use of Gaussian output to teach many chemical topics. Whilst the existing postprocessors render Gaussian output, and derive further electronic and molecular properties that are of common interest, the scope is circumscribed by the postprocessor design. In contrast, MATHSCOUT enables the interactive and batch production of an open-ended variety of conflations, combinations and derived quantities. These can be inspected on the screen, and rendered and typeset in publishable quality for distribution electronically and in print. Potential applications of MATHSCOUT include

1. mechanized documentation for teaching and research,
2. studies of convergence acceleration,
3. other systematic analyses of the behaviour of individual and related runs,
4. graphic display of numerical output,
5. exploration of models of molecular structure and behaviour that require further mathematical manipulation of the Gaussian results,
6. analysis of log and scan files that are archived in individual laboratories and in depositories.

Because all users of Gaussian are acquainted with items in the log file, we use examples of these to introduce the usage of MATHSCOUT. Successive sections of this tutorial cover the following material

- §2 describes the basic working procedure and some simple examples,
- §3 describes the extraction of data from lists and tables,
- §4 shows how multiple occurrences are extracted and distinguished, when (1) the number of occurrences does not vary and (2) when this number depends on the convergence of an iterative process,
- §5 explains the mechanical incorporation of verbatim displays of portions of text files in a LaTeX coded document, without manual cutting and pasting,
- §6 explains the scripts that produced Figures 1 and 2 and Table 1, to show how MATHSCOUT is used to render diagrams, and to typeset Gaussian output in tabular formats,

- §7 describes the component of our cyclobutane calculation that shows how a selection of data from Gaussian runs for several different molecules are combined,
- §8 describes the extraction of data from the checkpoint file using MATHSCOUT.

The accompanying User's Guide

1. provides a systematic description of the MATHSCOUT scanning commands,
2. describes the mechanized documentation in more detail than given here,
3. summarizes the extraction of all the data from the Gaussian log files for the water and zinc hydrate calculations,

MATHSCOUT scripts are written in a functional programming style that we have evolved for other applications of MATHEMATICA to computational chemistry and other fields [2]. The accompanying Development Guide

1. explains the implementation of MATHSCOUT,
2. shows how the command language can be extended very easily,
3. lists auxiliary files of test and demonstration material,
4. explains how these are used to ensure backward compatibility when the package is extended.

2 Some short examples

An extraction run begins by loading the MATHSCOUT package in a MATHEMATICA session. For simplicity, the input/output statements in the present tutorial refer to Gaussian output files in subdirectories of the directory containing the MATHSCOUT package `msct.m`. The setup stage for the water calculation is typical. The statement

```
waterLog = inputGaussianFile["water/water.log"];
```

1. stores the successive lines of the specified file as a list of character strings,
2. ensures that each "`=`" symbol in these is flanked by spaces,
3. removes leading, trailing and redundant spaces in each line,
4. removes periods from the ends of lines, and
5. removes null lines.

The displays in this tutorial show the lines of Gaussian files in the trimmed form that `inputGaussianFile` produces.

Note that when working directly with the unpacked MATHSCOUT package, the appropriately qualified file names are `"scanDem/water/water.log"` and `"scanDem/zincHydrate/zincHydrate.log"`.

Our first example shows the extraction of the single datum in the line that comprises the single element of the list

```
{Sum of electronic and thermal Free Energies = -76.445137}
```

The quote marks at the ends of each input line that has been loaded as a string are implied throughout this tutorial. A succession of lines of input is displayed as a list. The line that has just been displayed is converted to

```
{SumOfElectronicAndThermalFreeEnergies == -76.4451}
```

by the MATHSCOUT statement

```
extractDataFrom[waterLog,
 using[
 {skipTo["Free Energies"],
  extractCurrentLine,
  useEmbeddedEqual}]];
```

A preliminary inspection had shown that the free energy only appears in the one line of the log file that contains the character string "`Free Energies`". This kind of observation is needed to write all the scanning statements. Because the input line contains a single `=` symbol, MATHSCOUT coalesces the words before this symbol to form the name. Also, the `=` (`Set`) is changed to `==` (`Equal`). Correspondingly, the consecutive lines

```
{Thermal correction to Energy = 0.024185,
Thermal correction to Enthalpy = 0.025129,
Thermal correction to Gibbs Free Energy = 0.003701,
Sum of electronic and zero-point Energies = -76.427488,
Sum of electronic and thermal Energies = -76.424653,
Sum of electronic and thermal Enthalpies = -76.423709,
Sum of electronic and thermal Free Energies = -76.445137}
```

are converted to

```
{ThermalCorrectionToEnergy == 0.024185,
ThermalCorrectionToEnthalpy == 0.025129,
ThermalCorrectionToGibbsFreeEnergy == 0.003701,
SumOfElectronicAndZeroDashpointEnergies == -76.4275,
SumOfElectronicAndThermalEnergies == -76.4247,
SumOfElectronicAndThermalEnthalpies == -76.4237,
SumOfElectronicAndThermalFreeEnergies == -76.4451}
```

by the MATHSCOUT statements

```
extractDataFrom[waterLog,
 using[
{skipTo["Thermal"],
  extract[7 lines],
  useEmbeddedEqual}]];
```

The line

```
{Zero-point vibrational energy 56053.4 (Joules/Mol)}
```

needs a more detailed specification of the extraction than the earlier examples. An obvious description enumerates

1. the words that are coalesced to construct the name, and

2. the word that constitutes the value (or the words that constitutes the values when the named object is a list).

Once we had decided to use this convention, the coding to support the mnemonic `pairWords[{...}, ...]` was trivial. The complete MATHSCOUT statement is

```
zeroPoint =
extractDataFrom[waterLog,
 using[
{skipTo["Zero-point vibrational"],
  extractCurrentLine,
  pairWords[{1,2,3,5},4]}]];
```

This produces

```
{ZeroDashpointVibrationalEnergyJoulesSlashMol == 56053.4}
```

3 Lists and tables

Gaussian output contains many lists and tables. The styles that occur most often are typifed by

1. the Z-matrix, which is delimited by (i) its title and (ii) the identifier in the line that always follows the final line of the table,
2. the input orientation, that contains column headings set apart by lines of hyphens, with a line of hyphens at the end,
3. the optimized parameters, boxed by lines of hyphens and `!` marks,
4. the population analysis that is, essentially, a rectangular matrix that is folded to six items per line,
5. the distance matrix that is, essentially, a triangular matrix that is folded to five items per line,
6. the collection of multipole moments, that are free-format lists.

3.1 Extracting the Z-matrix

The Z-matrix is headed `Symbolic Z-matrix:` and it is followed by `Variables:`. For water, the relevant lines of the log file are

```
{Symbolic Z-matrix:,
Charge = 0 Multiplicity = 1,
O,
H 1 r,
H 1 r 2 a,
Variables:}
```

The matrix is converted to

```
{SymbolicZmatrix ==
{{O},
{H, 1, r},
{H, 1, r, 2, a}}}
```

by

```
symbolicZmatrix =
 extractDataFrom[waterLog,
 using[scanList[extractSymbolicZmatrix]]];
```

where

```
scanList[extractSymbolicZmatrix] =
{skipTo["Symbolic Z-matrix:"],
  makeNameFromCurrentLine,
  skipToNextLine,
  extractSuccessiveLinesBeforeNext["Variables:"],
  splitOnSpaces};
```

Lines 1–4 in this scan list are self-explanatory. The `splitOnSpaces` command constructs a MATHEMATICA list that consists of the items that are separated by spaces in the string that is split. Each item is converted to the MATHEMATICA representation of an integer, real number (preserving the accuracy) or symbol if it has the appropriate syntax. Otherwise, it is kept as a character string. The conversion of real numbers is suppressed for documentation, at times, by `preventConversion=True`. This suppresses lengthy strings of noise digits and built up powers of 10. Another use is mentioned at the end of §3.6.

3.2 Extracting the input orientation table

The following lines of data for the initial input orientation in the water file typify a tabular style that is used repeatedly in Gaussian output.

```
{Input orientation:,
---------------------------------------------------,
Center Atomic Atomic Coordinates (Angstroms),
Number Number Type X Y Z,
---------------------------------------------------,
1 8 0 0.000000 0.000000 0.000000,
2 1 0 0.000000 0.000000 0.950000,
3 1 0 0.917630 0.000000 -0.245878,
---------------------------------------------------}
```

These lines are converted to

```
{InputOrientation ==
{{1, 8, 0, 0.000000, 0.000000, 0.000000},
{2, 1, 0, 0.000000, 0.000000, 0.950000},
{3, 1, 0, 0.917630, 0.000000, -0.245878}}}
```

by

```
inputOrientation =
 extractDataFrom[waterLog,
 using[scanList[extractInputOrientation]]];
```

where

```
scanList[extractInputOrientation] =
{skipTo["Input orientation:"],
  makeNameFromCurrentLine,
```

```
  skip[4 lines],
  extractSuccessiveLinesBeforeNext["----"],
  splitOnSpaces,
  skipToNextLine};
```

The final command advances the scanning cursor for consistency with the needs of further commands when this scan list is incorporated in a longer list for a more extensive extraction.

3.3 Extracting the optimized parameter table

The optimized parameters in the water file typify another tabular style that is used in Gaussian output.

```
{---------------------------,
! Optimized Parameters !,
! (Angstroms and Degrees) !,
------------------------- -------------------------,
! Name Definition Value Derivative Info. !,
-----------------------------------------------------,
! R1 R(1,2) 0.9627 -DE/DX = 0.0 !,
! R2 R(1,3) 0.9627 -DE/DX = 0.0 !,
! A1 A(2,1,3) 103.9148 -DE/DX = 0.0001 !,
-----------------------------------------------------}
```

This is converted to

```
{OptimizedParameters ==
{{R1, R(1,2), 0.9627, -DE/DX, 0},
{R2, R(1,3), 0.9627, -DE/DX, 0},
{A1, A(2,1,3), 103.915, -DE/DX, 0.0001}}}
```

by

```
optimizedParameters =
 extractDataFrom[waterLog,
 using[scanList[extractOptimizedParameters]]];
```

where

```
scanList[extractOptimizedParameters] =
{skipTo["Optimized Parameters"] ,
  makeNameFromCurrentLine,
  skip[4 lines],
  extractSuccessiveLinesBeforeNext["----"],
  deleteWords[{1, 6, -1}]};
```

The commands in this scan list are largely self-explanatory. The `deleteWords` command in this script deletes the `!` and `=` symbols, and returns the input line as a list of strings. Further commands that coalesce descriptive phrases will be used, largely without comment, in the examples that follow in this section and in later sections. Whilst a list of these is needed to use MATHSCOUT to full advantage, scripts can be read without a glossary and the examples here can be used as prototypes for further work.

3.4 Extracting the population analysis

The population analysis illustrates the unfolding of a rectangular matrix. The relevant part of the log file for $Zn(H_2O)_6^{++}$ is

```
{Population analysis using the SCF density.,
...,
Condensed to atoms (all electrons):,
1 2 3 4 5 6,
1 Zn 10.413341 0.104376 -0.005140 -0.005895 0.103698 -0.005659,
<<17 lines>>,
19 H -0.005843 -0.000004 0.000000 0.000001 -0.000161 0.000004,
<<40 lines>>,
19,
1 Zn -0.005843,
<<17 lines>>,
19 H 0.322839,
Mulliken atomic charges:}
```

The 40 lines that are omitted from this display contain the data for the 19 atoms coordinated with atoms 7, ..., 12 and then with atoms 13, ..., 18. The input table is converted to

```
{populationAnalysisCondensedToAtoms ==
{{1, Zn, 10.413341, 0.104376, ..., -0.005159, -0.005843},
<<17 lines>>,
{19, H, -0.005843, -0.000004, ..., -0.021359, 0.322839}}}
```

by

```
unfilletedOutputForPopulationAnalysis =
extractDataFrom[
 zincHydrateLog, using[scanList[populationAnalysis]]];
```

where

```
scanList[populationAnalysis] =
{skipTo["Population analysis"]   ,
  skipToNext["Condensed to atoms"],
  skipToNextLine,
  extractSuccessiveLinesBeforeNext[
   "Mulliken atomic charges"] ,
  unfold,
  callTheResults[populationAnalysisCondensedToAtoms]};
```

3.5 Extracting the distance matrix

The extraction of a distance matrix shows how a triangular matrix is unfolded. In the zinc hydrate file, the matrix is

```
{Distance matrix (angstroms):,
1 2 3 4 5,
1 Zn 0.000000,
2 O 2.544496 0.000000,
3 H 3.057904 1.013377 0.000000,
4 H 3.057592 1.011392 1.639501 0.000000,
5 O 2.544353 3.612396 4.058512 3.398881 0.000000,
```

```
6 H 3.058750 3.707898 3.918671 3.456905 1.012637,
<<12 lines>>,
19 H 3.054884 4.476826 4.258097 5.006885 3.705885,
<<25 lines>>,
19 H 3.401702 3.668385 4.337178 4.218934 4.765916,
16 17 18 19,
16 H 0.000000,
17 O 4.415231 0.000000,
18 H 4.163579 1.013083 0.000000,
19 H 4.974219 1.012715 1.635881 0.000000,
Stoichiometry H12O6Zn(2+)}
```

The sequence of 25 lines that are omitted from this display consist of 15 lines that coordinate atoms 6,. . . , 10 with atoms 6, . . . , 19, and 10 lines that coordinate atoms 11,. . . , 15 with atoms 11, . . . , 19. The input matrix is converted to

```
{distanceMatrix ==
{{0.},
{2.5445, 0.},
{3.0579, 1.01338, 0.},
{3.05759, 1.01139, 1.6395, 0.},
{2.54435, 3.6124, 4.05851, 3.39888, 0.},
{3.05875, 3.7079, 3.91867, 3.45691, 1.01264, 0.},
{3.05567, 4.55317, 5.00138, 4.3901, 1.0122, 1.63995, 0.},
<<11 rows>>,
{3.05488, 4.47683, 4.2581, 5.00689, 3.70589, <<12 items>>, 3.48454, 0.}}}
```

by

```
unfilletedOutputForDistanceMatrix =
extractDataFrom[
 zincHydrateLog, using[scanList[distanceMatrix]]];
```

where

```
scanList[distanceMatrix] =
{skipTo["Distance matrix"] ,
  skipToNextLine,
  extractSuccessiveLinesBeforeNext[
   "Stoichiometry"],
  unfoldTriangular,
  callTheResults[distanceMatrix]};
```

3.6 Extracting the multipoles

Multipoles comprise a quasi-tabular body of data. The lines

```
{Dipole moment (field-independent basis, Debye):,
X = 0.0000 Y = 0.0000 Z = -1.9678 Tot = 1.9678,
Quadrupole moment (field-independent basis, Debye-Ang):,
XX = -7.2125 YY = -4.2883 ZZ = -6.0338,
XY = 0.0000 XZ = 0.0000 YZ = 0.0000,
<<7 lines>>,
Hexadecapole moment (field-independent basis, Debye-Ang**3):,
XXXX = -5.3379 YYYY = -6.0480 ZZZZ = -6.2481 XXXY = 0.0000,
<<2 lines>>,
XXYZ = 0.0000 YYXZ = 0.0000 ZZXY = 0.0000}
```

are converted to

```
{dipoles ==
 {X == 0, Y == 0, Z == -1.9678, Tot == 1.967},
quadrupoles ==
 {XX == -7.2125, YY == -4.2883, ZZ == -6.0338, XY == 0,
XZ == 0, YZ == 0},
tracelessQuadrupoles ==
 {XX == -1.3676, YY == 1.5566, ZZ == -0.1889, XY == 0,
XZ == 0, YZ == 0},
octapoles ==
 {XXX == 0, YYY == 0, ZZZ == -1.1326, XYY == 0,
XXY == 0, XXZ == -0.2851, XZZ == 0, YZZ == 0,
YYZ == -1.2209, XYZ == 0},
hexadecapoles ==
 {XXXX == -5.3379, YYYY == -6.048, ZZZZ == -6.2481, XXXY == 0,
XXXZ == 0, YYYX == 0, YYYZ == 0, ZZZX == 0,
ZZZY == 0, XXYY == -2.1694, XXZZ == -1.9818, YYZZ == -1.7308,
XXYZ == 0, YYXZ == 0, ZZXY == 0}}
```

by the MATHSCOUT statements

```
scanList[extractMultipoles] =
{skipTo["Dipole"], extractNextLine, concatenateTheExtract,
  useEmbeddedEqual, callTheResults[dipoles],
  skipToNextLine, extract[2 lines], concatenateTheExtract,
  useEmbeddedEqual, callTheResults[quadrupoles],
  skipToNextLine, extract[2 lines], concatenateTheExtract,
  useEmbeddedEqual, callTheResults[tracelessQuadrupoles],
  skipToNextLine, extract[3 lines], concatenateTheExtract,
  useEmbeddedEqual, callTheResults[octapoles],
  skipToNextLine, extract[4 lines], concatenateTheExtract,
  useEmbeddedEqual, callTheResults[hexadecapoles]};
```

The displayed values of the multipoles were produced using the `preventConversion` action mentioned at the end of §3.1. The zero values would have appeared as 10^{-5} otherwise.

4 Repetitive data

4.1 Fixed number of repetitions

The word "dipoles" actually occurs 5 times in the log file when jobs are run with a commonly used set of options. The lines that follow the first and second lines containing `Dipoles` are converted to

```
{dipoles[1] == {X == 0, Y == 0, Z == -1.9678, Tot == 1.9678},
dipoles[2] == {X == 0, Y == 0, Z == -1.989, Tot == 1.989}}
```

by

```
dipoles2 =
extractDataFrom[waterLog,
 using[
 {skipToNext["Dipole"],
```

```
  extractNextLine, useEmbeddedEqual,
  callTheResults[dipoles[1]],
  skipToNext["Dipole"],
  extractNextLine, useEmbeddedEqual,
  callTheResults[dipoles[2]]}]]
```

The line that follows the final line containing `Dipoles` is converted to

```
{dipoles[-1] == {X == 0, Y == 0, Z == -1.9678, Tot == 1.9678}}
```

by

```
dipoles3 =
extractDataFrom[waterLog,
 using[
 {skipToFinal["Dipole moment"],
  extractNextLine, useEmbeddedEqual,
  callTheResults[dipoles[-1]]}]]
```

4.2 Iterative data

The major part of a typical log file consists of successive iterations headed "Berny [Schlegel] optimization" followed, 3 lines later, by the iteration number in the context that

```
{Step number 1 out of a maximum of 20}
```

typifies. Gaussian writes a considerable amount of data in each iteration, and the analysis of trends in many of these data is useful. The extraction is very simple. For example, the value, threshold and convergence of the maximum and RMS force and displacement are written in each iteration and the final triple is displayed again in the summary at the end of the log file. The first line containing maximum force data in the water file is

```
{Maximum Force 0.012670 0.000450 NO}
```

This is extracted by

```
maxForceOne =
 extractDataFrom[waterLog,
  using[{skipToNext["Maximum Force"],
     extractCurrentLine, pairWords[{1,2}, {3,4,5}]}]]
```

Hence

```
{MaximumForce == {0.01267, 0.00045, NO}}
```

The statement

```
maximumForcesVerbose =
 extractDataFrom[waterLog,
 using[
 {exhaustively[
   {skipToNext["Maximum Force"],
     extractCurrentLine, pairWords[{1,2}, {3,4,5}]}]}]]
```

extracts all the lines containing "Maximum Force" from the log of the water calculation. This went through 3 iterations. Hence

```
{MaximumForce == {0.01267, 0.00045, NO},
MaximumForce == {0.001376, 0.00045, NO},
MaximumForce == {0.000091, 0.00045, YES},
MaximumForce == {0.000091, 0.00045, YES}}
```

This is condensed to

```
{MaximumForce ==
{{0.01267, 0.00045, NO},
{0.001376, 0.00045, NO},
{0.000091, 0.00045, YES}}}
```

by

```
maximumForcesConcise =
 maximumForcesVerbose //
  drop[-1] // coalesceUnderCommonName
```

In MATHEMATICA, the postfix notation `x // f` is synonymous with `f[x]`.

5 Embedding pieces of text files

To produce this tutorial, we typed a LaTeX file `tutorialRaw.tex` that contains a line of the form `\\!!`*specification of display*`!!\\` wherever a verbatim display is needed. This file can be typeset and proofed for overall content. Each specification conforms to LaTeX syntax, and is set on a separate line. Then executing `embedExtracts["tutorialRaw", "tutorial"]` converts `tutorialRaw.tex` to `tutorial.tex` that contains the appopriate verbatim sequence in place of each specification. An embed specification contains

1. the name of the file to be extracted,
2. either
   (a) the line number at which the extract begins, or
   (b) a string that starts the first line of the extract, or a string that starts a line which is close to the first line of the extract,
   (c) the notation *n*`[`*key*`]` when the extract begins with the *n*-th line that starts with the given key,
3. the offset from the line specified by a key, that can be omitted when zero (though the comma must be kept if there is a line count),
4. the number of lines to be extracted, unless these lines extend to the end of the file, in which case the number is omitted.

The direct use of these conventions is shown by the action of

```
\\!!"water/water.log", " Thermal", 6, 1!!\\
```

This extracts the same information as the first example in §2. The specification follows the line of `tutorialRaw.tex` that contains the end of the present sentence.

```
 Sum of electronic and thermal Free Energies=          -76.445137
```

This does *not* contain the braces or commas that occur in the pieces of input displayed in earlier sections. Also, the spacing is different, and the line is specified by offset from the first line that begins " Thermal", instead of being specified as the first line that contains "Free Energies". Correspondingly, the specification

```
\\!!"water/water.log", " Thermal", 7!!\\
```

embeds

```
 Thermal correction to Energy=                    0.024185
 Thermal correction to Enthalpy=                  0.025129
 Thermal correction to Gibbs Free Energy=         0.003701
 Sum of electronic and zero-point Energies=            -76.427488
 Sum of electronic and thermal Energies=               -76.424653
 Sum of electronic and thermal Enthalpies=             -76.423709
 Sum of electronic and thermal Free Energies=          -76.445137
```

The earlier displays were introduced by the following multi–step process.

1. Each piece of an input file that is needed is extracted from the list of strings returned by the function `inputGaussianFile` in the manner described at the start of §2. Elementary MATHSCOUT utilities extract the requisite line(s) as element(s) of a list.

2. The transformed information is constructed by an `extractDataFrom` statement, containing an explicit scan list or the name of a scan list that is defined separately.

3. When the input and / or output contains a long sequence of lines with similar structure, this is abbreviated, as in the population analysis of §3.4 and distance matrix of §3.5 by notations such as `<<`n `lines>>`. The User's Guide explains how this is done. The input lines and the transformed result are written as separate files. All of this information is referenced in `tutorialRaw.tex` by `\\!!...!!\\` statements. Each of these statements contains

   (a) the name of a small individual file of the kind just mentioned, or

   (b) `displays.in`, the name of the file that produced these.

6 Rendering and tabular typesetting

6.1 Constructing Figures 1 and 2

The data for Figure 1 is extracted from the nitrogen scan file by

```
energyTriples =
 extractDataFrom[nitrogenScan,
  using[
 {skipTo["Summary"], makeNameFromCurrentLine,
   skip[2 lines],
   extractSuccessiveLinesBeforeNext["---"],
   splitOnSpaces}]] // First
```

The `extractDataFrom` function returns its results in the form of a MATHEMATICA list, as mentioned earlier, even when the list contains just a single element, which happens here. The `First` function extracts this element. Hence

```
{SummaryOfThePotentialSurfaceScan ==
{{1, 0.6, -106.851},
{2, 0.7, -108.316},
<<23 triples>>,
{26, 3.1, -108.914}}}
```

The energy–radius pairs are extracted by

```
energy[r] = energyTriples // extractPart[2] //  toEachElement[drop[1]]
```

The functions in this example have self-explanatory names, and they are covered systematically, with other related commands, in the User's Guide. Hence

```
{{{0.6, -106.851}, <<24 pairs>>, {3.1, -108.914}}}
```

The data is plotted and the points are joined by the MATHEMATICA statement

```
curve1 = ListPlot[energy[r], PlotJoined -> True]
```

The axes are provided as arguments of the `Show` function that constructs a "graphics object" from the code for one or more plots.

```
fig1=
 Show[curve1, AxesOrigin -> {0.5, -109.6}, AxesLabel -> {"r", "energy"}]
```

Then the PostScript* encoding of Figure 1 is written by

```
Display["!psfix > " <> resultsDirectory <> "nitrogen01.ps", fig1]
```

The virial data is dispersed through the successive scan cycles, in lines that are typified by

```
{Convg = 0.1076D-08 -V/T = 1.9047}
```

The complete set is extracted by

```
virialLines =
 extractDataFrom[nitrogenScan,
  using[{exhaustively[
   {skipToNext["-V"], extractCurrentLine, pairWords[4, 6]}]}]] //
 tagConsecutively
```

Hence

```
{DashVSlashT[1] == 1.9047,
<<24 lines>>,
DashVSlashT[26] == 2.0078}
```

The r values in the `energy[r]` list are combined with the $-v/t$ values by

```
virial[r] =
{energy[r] // toEachElement[extractPart[1]],
  virialLines // toEachElement[extractPart[2]]} // Transpose
```

Hence

```
{{{0.6, 1.9047}, <<24 pairs>>, {3.1, 2.0078}}}
```

The plot is constructed and saved in PostScript by statements that are completely analogous to those for the energy plot. These new statements are constructed mechanically from the earlier statements by changing `1` to `2` in `fig1`, `curve1`, and `.01`, and `energy` to `virial`.

*POSTSCRIPT is a registered trademark of Adobe Inc.

6.2 Constructing Table 1

The distance matrices that are computed in the successive Berny iterations for zinc hydrate are found by

```
allDistanceMatrices =
extractDataFrom[zincHydrateLog,
 using[
 {exhaustively[
   {skipToNext["Distance matrix"] ,
  skipToNextLine,
  extractSuccessiveLinesBeforeNext[
   "Stoichiometry"],
  unfoldTriangular,
  callTheResults[distanceMatrix]}
 ]} ]] //
 tagConsecutively // drop[-2];
```

The `drop[-2]` drops the two repetitions of the final distance matrix in the summaries that follow the iteration at the end of the log file. Hence

```
{distanceMatrix[1] ==
{{{0.}, {2.5445, 0.}, {3.0579, 1.01338, 0.}},
<<15 rows>>,
{3.05488, 4.47683, 4.2581, <<12 items>>, 4.97422, 4.97422, 1.63588, 0.}},
<<28 matrices>>,
distanceMatrix[30] ==
{{{0.}, {2.11744, 0.}, {2.7904, 0.977739, 0.}},
<<15 rows>>,
{2.78308, 3.50259, 3.3594, <<12 items>>, 4.46074, 4.46074, 1.6095, 0.}}}
```

Data for zinc oxygen distances are extracted by

```
zincOxygenDistances =
 allDistanceMatrices //
  toEachElement[
   extractPart[2],
   extractParts[{2, 5, 8, 11, 14, 17}],
   toEachElement[extractPart[1]]];
```

Hence

```
{{2.5445, 2.54435, 2.5429, 2.5445, 2.54458, 2.54359},
<<28 sets>>,
{2.11744, 2.12005, 2.11419, 2.11516, 2.11575, 2.12012}}
```

Averages and spreads found using two simple utilities

```
averages =
 zincOxygenDistances //
  toEachElement[average, roundToDecimalPlace[3]]

spreads =
 zincOxygenDistances //
  toEachElement[percentageSpread, times[100], roundToDecimalPlace[3]]
```

Other statistics are computed by further simple procedures. The LaTeX coded table is constructed and saved as `igmTable1.tex` by another simple utility that acts on a list of formating items and data that are provided mnemonically.

```
writeLaTeXtable["igmTable1.tex",
 columnSpecification["|l|", 11 "r|"] ,
 hline, tabbedLine["iteration", integerList[1 - 10]]  ,
 hline, tabbedLine["average", averages // extractParts[1 - 10] ],
 hline, tabbedLine["spread", spreads // extractParts[1 - 10]],
 hline,
 hline, tabbedLine["iteration", integerList[11 - 20]],
 hline, tabbedLine["average", averages // extractParts[11 - 20]],
 hline, tabbedLine["spread", spreads // extractParts[11 - 20]],
 hline,
 hline, tabbedLine["iteration", integerList[21 - 30]],
 hline, tabbedLine["average", averages // extractParts[21 - 30]],
 hline, tabbedLine["spread", spreads // extractParts[21 - 30]],
 hline, hline]
```

This writes the file that is imported by

```
\input{igmTable1.tex}
```

in §1. The readability of expressions such as `3 "x"` for `{"x", "x", "x"}` and `integers[1 - 3]` for `{"1", "2", "3"}` that breach the default MATHEMATICA conventions are easy to code — see Users' Guide.

7 A stereochemical example — puckering in cyclobutane and its analogues

Recently, we reported some formulas that relate the puckering angles in 4-atom ring molecules, that we refer to collectively as cyclobutanes (CBs) [3]. We use Gaussian 03 to compute the molecular geometry and dipole moments of several CBs with and without substitutents and hetero-atoms in the ring. The new formulas give the puckering angles on the diagonals PR and QS in rings with the atomic labels shown in Figure 3.

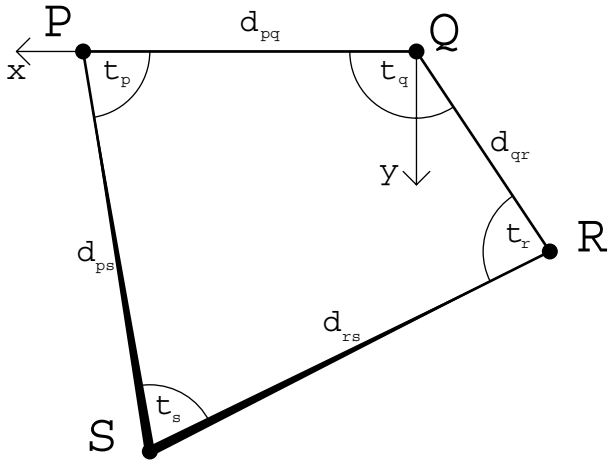

Figure 3. Notation for four-membered rings.

The final Z-matrices, tables of internal coordinates and dipole moments were

cut and paste from the output of the separate Gaussian runs into a working file. MATHEMATICA scripts were written that

1. extracted the relevant bond lengths and bond angles from a final Z-matrix, using a list that associated the labels P, Q, R, S in Figure 1 with the numerical labels in the Z-matrix and other tables in the Gaussian output,

2. applied the new formulas to compute the puckering angles,

3. computed the puckering angles directly from the internal coordinates,

4. compared the puckering angles found by these two routes, showing identity in all cases,

5. converted the numerical bond lengths, bond angles, puckering angles, and dipole moments to character strings expressing the required precision,and wrote the LaTeX code containg these that was typeset as Table 2.

	d_{12}	d_{23}	d_{34}	d_{14}	t_1	t_2	t_3	t_4	ϕ_{12}	ϕ_{23}	μ D
CB	1.55	1.55	1.55	1.55	88.57	88.57	88.57	88.57	25.48	25.48	0.00
*cis*1,3-Cl_2CB	1.54	1.54	1.54	1.54	86.78	89.70	86.78	89.70	27.94	28.65	2.07
*trans*1,3-Cl_2CB	1.54	1.55	1.55	1.54	88.11	89.27	88.11	89.67	23.33	23.60	1.14
*cis*1,3-Br_2CB	1.54	1.54	1.54	1.54	86.33	89.90	86.33	89.90	28.78	29.68	1.96
*trans*1,3-Br_2CB	1.54	1.55	1.55	1.54	87.85	89.32	87.85	89.81	24.09	24.45	1.15
silaCB	2.38	2.38	2.38	2.38	87.72	87.72	87.72	87.72	32.09	32.09	0.00
*cis*1,3-Cl_2silaCB	2.38	2.38	2.38	2.38	87.89	89.80	87.89	89.80	22.74	23.11	2.69
*trans*1,3-Cl_2silaCB	2.38	2.38	2.38	2.38	86.26	89.53	86.26	89.62	30.28	31.15	0.84
azetane	1.49	1.55	1.55	1.49	88.85	86.01	88.85	90.43	25.87	25.75	1.22
phosphetane	1.90	1.54	1.54	1.90	90.95	96.91	90.95	74.99	26.85	26.18	1.34
oxetane	1.45	1.54	1.54	1.45	91.81	84.76	91.80	91.64	0.11	0.11	1.87
thietane	1.86	1.54	1.54	1.86	91.28	96.83	91.27	76.58	21.70	21.18	2.03

Table 2. Geometry and dipole moments of cyclobutanes

The cutting and pasting was tedious. Now, we start with a directory `cbAll` that contains **the entire set** of CB output files. These have the extension `.LOG`. Then the following steps serve as a prototype for many other tabulations of data from the output of Gaussian runs that pertain to sets of related molecules or variations of externally determined parameters.

```
fullFileNames = FileNames["cbAll/*.LOG"]

fileNames =  fullFileNames // toEachElement[stringDrop[5]]

directoryNames = fileNames  // toEachElement[stringDrop[-4]]

directoryNames // toEachElement[stringPrepend["mkdir "], Run]

fileCount = fileNames // Length

{fileCount  * "mv cbAll/", fileNames,  fileCount * " ",
  directoryNames, fileCount * "/. ", } // Transpose //
  toEachElement[StringJoin, Run]
```

These, respectively

1. extract the list of names in the directory `cbAll` that have the extension .LOG, (actually `FileNames["cbAll/*"]` is adequate since the `.LOG` files are the only files in `cbAll`),
2. lop off the extension `.LOG`,
3. create subdirectories with these names, by using `Run` to play through to the Unix command line,
4. move each output file into the appropriate subdirectory — *e.g.* move `silaCB.LOG` to `silaCB/silaCB.LOG`, by playing through to the UNIX command line again, using `Run`.

Then

```
{directoryNames, fileCount * "/", fileNames} // Transpose //
   toEachElement[StringJoin, splitGaussianOutput]
```

extracts the data from each of the CB files, writing the output in the directory that contains the entire file. If necessary, the assignment of `fullFileNames` can be restricted to a subset of the files. `splitGaussianOutput` wraps a short script of MATHSCOUT primitives.

8 Using formated checkpoint files

Data is extracted from FPCHK files using the principles that have been applied to log and scan files in the preceding sections. After splitting an input line into a list of items that are separated by spaces, the MATHSCOUT utilities determine which have the syntax of numbers and convert these automatically. The `I` and `R` flags in the checkpoint files are redundant in this context. They are removed by `deleteIandRflags`, as in

```
waterCheckpoint =
 inputGaussianFile["GaussOut/water/water.chkpt"] // deleteIandRflags;
```

Then, for example

```
fromWaterCheckpoint =
 extractDataFrom[
  waterCheckpoint,
   using[{skipTo["atoms"],
   extractSuccessiveLinesIncludingNext["vectors"],
   putEqualAtSpace[-1]}]]
```

produces

```
{NumberOfAtoms == 3,
Charge == 0,
Multiplicity == 1,
NumberOfElectrons == 10,
NumberOfAlphaElectrons == 5,
NumberOfBetaElectrons == 5,
NumberOfBasisFunctions == 30,
NumberOfIndependantFunctions == 30,
NumberOfPointChargesInSlashMolSlash == 0,
NumberOfTranslationVectors == 0}
```

The methods of the earlier sections of this tutorial cover all the contingencies that arise in the checkpoint files.

References

[1] M. P. Barnett, Transformation of harmonics for molecular calculations. J. Chem. Inf. Sci. 43 (4) 1158–1165, 2003.

[2] M. P. Barnett, Mathscape amd molecular integrals, J. Symb. Comp. 42 (3) 265–289, 2007.

[3] M. P. Barnett; J. F. Capitani, Modular chemical geometry and symbolic calculation. Int. J. Quant. Chem. 106 (1) 215–227, 2006.

[4] M. J. Frisch; G. W. Trucks; H. B. Schlegel; G. E. Scuseria; M. A. Robb; J. R. Cheeseman; J. A. Montgomery; Jr.; T. Vreven; K. N. Kudin; J. C. Burant; J. M. Millam; S. S. Iyengar; J. Tomasi; V. Barone; B. Mennucci; M. Cossi; G. Scalmani; N. Rega; G. A. Petersson; H. Nakatsuji; M. Hada; M. Ehara; K. Toyota; R. Fukuda; J. Hasegawa; M. Ishida; T. Nakajima; Y. Honda; O. Kitao; H. Nakai; M. Klene; X. Li; J. E. Knox; H. P. Hratchian; J. B. Cross; C. Adamo; J. Jaramillo; R. Gomperts; R. E. Stratmann; O. Yazyev; A. J. Austin; R. Cammi; C. Pomelli; J. W. Ochterski; P. Y. Ayala; K. Morokuma; G. A. Voth; P. Salvador; J. J. Dannenberg; V. G. Zakrzewski; S. Dapprich; A. D. Daniels; M. C. Strain; O. Farkas; D. K. Malick; A. D. Rabuck; K. Raghavachari; J. B. Foresman; J. V. Ortiz; Q. Cui; A. G. Baboul; S. Clifford; J. Cioslowski; B. B. Stefanov; G. Liu; A. Liashenko; P. Piskorz; I. Komaromi; R. L. Martin; D. J. Fox; T. Keith; M. A. Al-Laham; C. Y. Peng; A. Nanayakkara; M. Challacombe; P. M. W. Gill; B. Johnson; W. Chen; M. W. Wong; C. Gonzalez; J. A. Pople, Gaussian 03, Revision B.04. Gaussian, Inc. Pittsburgh, PA, 2003.

[5] Gaussian Inc. Carnegie Office Park Building 6, Pittsburgh, PA 15106 USA. www.gaussian.com

[6] Hypercube, Inc. 1115 NW 4th Street, Gainesville, FL, 32601 USA. www.hypercube.com.

[7] L. Laaksonen, Center for Scientific Computing, Espoo, Finland. www.csc.fi/gopenmol.

[8] I. Mayer, An exact chemical decomposition scheme for the molecular energy. Chem. Phys. Letters 382 (3) 265–269, 2003.

[9] I. Mayer, Charge, bond order and valence in the *ab initio* SCF theory. Chem. Phys. Letters, 97, 270–274, 1983.

[10] I. Mayer; Á. Gömöry, Predicting primary mass spectrometric cleavages: a 'quasi–Koopmans' ab initio approach. Chem. Phys. Letters, 344, 543, 2001.

[11] I. Mayer; P. Salvador, Overlap populations, bond orders and valences for "fuzzy" atoms. Chem. Phys. Letters, 383, 368, 2004.

[12] P. Murray-Rust, Quantum Chemical Calculations `www.ch.ic.ac.uk/omf/cml/doc/examples/qchem.html`.

[13] S. Portmann; H. P. Luthi, Chimia 54, 766–770, 2000.

[14] P. Salvador; I. Mayer, Energy partitioning for "fuzzy" atoms. J. Chem. Phys. 120 (11) 5046–5052, 2004.

[15] G. Schaftenaar; J.H. Noordik, J. Comp. Aided Mol. Design, 14, 123-134, 2000.

[16] J. R. Schmidt; W. F., Polik, WebMO, version 6.0. WebMO, Holland, MI 49423 USA.

[17] Serena Software, Box 3076, Bloomington IN, 47402 USA.

[18] Wolfram, S. The Mathematica Book, 5th ed. Wolfram Media, 2003.