Computing for Science (CFS) Ltd.,
CCLRC Daresbury Laboratory.

Generalised Atomic and Molecular Electronic Structure System

G A M E S S - U K

USER'S GUIDE and REFERENCE MANUAL

Version 8.0 June 2008

PART 11. FORMATTED OUTPUT from GAMESS-UK

M.F. Guest, J. Kendrick, J.H. van Lenthe and P. Sherwood

Contents

1 The Punch Directive

This chapter provides a description of the way formatted output may be obtained from GAMESS–UK for use by other programs. Output is controlled by the Class 1 PUNCH directive followed by associated keywords, which determine what is to be written out. The output is written on FORTRAN unit 58. Under control of the rungamess script, the punchfile is saved in the job submission directory with the filename extension .pun. Otherwise, it may directed to a specific file in the manner used for the other fortran streams (See the Machine Specific Sections later in the manual).

Specification of the PUNCH directive without any keywords is equivalent to the directive

```
PUNC GEOM CONN OCCU VECT BASI TYPE LEVE
```

(where we use the shortest valid abbreviations for the keywords)

First, we provide an overview of the structure of the punchfile, and then describe each the arrangement and format used for each type of data. For jobs with multiple RUNTYPE directives, the placement of the PUNCH directive is important, and multiple PUNCH directives may be used, see section 4.

2 Punch File Structure

The GAMESS–UK punchfile consists of records which are grouped into blocks, each block being preceded by a block header. The header serves to identify the block, specify its length (in records) and to provide information to be used when the block is read. Within a block, all records are written with the same FORTRAN format. The formats used by GAMESS–UK when writing each type of block are described below. All information in the header takes the form of variable assignments, each of which has the syntax VAR = DATA where VAR is the name of the header variable, and DATA is the associated value. Two such variables appear in the header of every block, regardless of the contents of the block. These are "block" (which is set to a character string defining the type of data) and "records" (which is set to an integer specifying the number of data records in the block). The value of the "block" variable will be used in this chapter to refer to blocks of that particular type, and will be denoted by a **bold** font. If the format used to write out the data within the block differs from the default for the given block type (i.e. that described in this chapter) the variable "f_format" is set to a character string containing the FORTRAN format used. The number of additional variables set will depend on the type of data block, and where appropriate these are described below.

The block header is usually contained on a single record. In cases where continuation lines are needed the continuation character backslash (\) will be placed at the end of all incomplete lines.

It should be noted that the case of the punchfile will depend on the machine specific version of GAMESS–UK in use. Currently, it will be lower case for all UNIX machines, and uppercase for the IBM 3090 and Cray implementations.

As a simple example, the following punchfile results from running a default (STO3-21G) basis set calculation on formaldehyde (see Part 2 of this manual) with the PUNCH directive added.

Table 1: Keywords of the PUNCH Directive

keyword	Data
TITLE	title
COORDINATES	coordinates (single-point or at the end of optimisation)
INITIAL	coordinates before optimisation
OPTIMISE	coordinates during optimisation
CONNECTIVITY	atom connectivity
BASIS	basis set
OVERLAP	overlap matrix
VECTORS	MO vectors
OCCUPATIONS	MO occupations
EIGENVALUES	scf orbital energies
SCFENERGY	scf energy
GVB	gvb pair coefficients
VIBRATIONAL	vibrational frequencies
NORMAL	vibrational normal modes
INFRARED	infra-red intensities
RAMAN	raman intensities
GRID	grid data from graphics directive
GSYM	symmetry data for grid points
MULLIKEN	mulliken population analysis
LOWDIN	lowdin population analysis
SPIN	spin densities at the nuclei
PDC	potential derived charges
GRADIENT	gradients of the energy w.r.t. nuclear positions
TRANSFORM	the rototranslation matrix used to reorient the molecule
SECDER	cartesian second derivatives
DMA	distributed multipole analysis
TYPE	class of calculation
LEVEL	level of calculation
HIGH	request higher precision format

The output is in lower case as it was generated on a UNIX machine.

```
block = calculation_type records = 1
single_point
block = calculation_level records = 1
rhf
block=fragment records=0
block = coordinates records =     4 unit = au
  c             0.0000000      0.0000000      0.9998722
  o             0.0000000      0.0000000     -1.2734689
  h             0.0000000      1.7650653      2.0942591
  h             0.0000000     -1.7650653      2.0942591
block = connectivity records =    3
    1    2
    1    3
    1    4
block = basis records =  21
c          1  s   1   1      172.256000       0.061767
c          1  s   1   2       25.910900       0.358794
c          1  s   1   3        5.533350       0.700713
c          1  s   2   1        3.664980      -0.395897
c          1  s   2   2        0.770545       1.215840
c          1  p   3   1        3.664980       0.236460
c          1  p   3   2        0.770545       0.860619
c          1  s   4   1        0.195857       1.000000
c          1  p   5   1        0.195857       1.000000
o          2  s   1   1      322.037000       0.059239
o          2  s   1   2       48.430800       0.351500
o          2  s   1   3       10.420600       0.707658
o          2  s   2   1        7.402940      -0.404453
o          2  s   2   2        1.576200       1.221560
o          2  p   3   1        7.402940       0.244586
o          2  p   3   2        1.576200       0.853955
o          2  s   4   1        0.373684       1.000000
o          2  p   5   1        0.373684       1.000000
h          3  s   1   1        5.447178       0.156285
h          3  s   1   2        0.824547       0.904691
h          3  s   2   1        0.183192       1.000000
block = occupations records =    22 index =   1
  2.000000
  2.000000
  2.000000
  2.000000
  2.000000
  2.000000
  2.000000
  2.000000
  0.000000
  0.000000
  0.000000
  0.000000
  0.000000
  0.000000
  0.000000
  0.000000
```

```
  0.000000
  0.000000
  0.000000
  0.000000
  0.000000
  0.000000
block = vectors records =    88 index =   1 elements =  22
  -0.000199  -0.000556   0.000000   0.000000   0.002516  -0.018231
   0.000000   0.000000   0.013900  -0.983533  -0.098056   0.000000
   0.000000  -0.004511   0.047908   0.000000   0.000000   0.013258
   0.000801  -0.000450   0.000801  -0.000450

... etc ...

   0.001786  -0.103327   0.000000   0.000000   0.276317  -1.150242
   0.000000   0.000000   0.780772   0.062516  -1.711541   0.000000
   0.000000  -0.253675   2.543902   0.000000   0.000000   0.952243
   0.074248  -0.016513   0.074248  -0.016513
```

3 Punch Keywords and Output Data Formats

3.1 TITLE

This keyword requests a single block of type **title**, containing a the job title, written as a single record with format (a80).

3.2 COORDINATES, INITIAL and OPTIMISE

These keywords cause cartesian coordinates to be written to the punchfile. For a single-point calculation the COORDINATES keyword should be used to punch a single data block of type **coordinates**.

The progress of geometry optimisation or saddle-point calculations may be monitored using the INITIAL and OPTIMISE keywords. The INITIAL keyword requests that the starting geometry be punched in a block of type **initial_coordinates**. The OPTIMISE keyword requests that a block of type **update_coordinates** will be punched at every point of the optimisation pathway, including the initial and final points. The header variable "index" will be set to the position of the structure in the sequence. If the COORDINATES keyword is also supplied, a **coordinates** block will also be punched out, containing the geometry of the last point.

All three types of **coordinates** blocks have the same format:

item	format
atom label	2x,a8
x,y,z	3(3x,f12.7)

3.3 CONNECTIVITY

This keyword requests a single block containing a list of connected atom pairs. The contact list is calculated using an internal table of approximate covalent radii.

Each atom pair is written as a single record with format (2i5) The coordinates and atom ordering are those used in the **coordinates** block. Each pair is listed once only, with the lower atom number first. The number of pairs is given by the header variable "records".

3.4 BASIS

This keyword results in a single data block being punched, which contains a definition of the basis set in terms of the contracted Gaussian primitives. Each line describes a single primitive function, with contents and format as shown in the following table.

item	format
atom tag	a8,1x
atom index	i3,1x
type (s,p,d or f)	a2,1x
shell index	i3,1x
primitive index	i3,1x
exponent	f15.6
coefficient	f15.6

The atom index is incremented each time a new atom type is started. The shell index, which is set to 1 at the start of each atom type, is incremented for each shell (The s and p components of GAUSSIAN-type sp shells are treated as independent shells). Likewise, the primitive index is set to 1 at the start of each shell. The exponents are given in atomic units. The coefficients are used to expand the contraction in terms of primitives which are themselves normalised - the normalisation factors are *not* included in the coefficients.

3.5 OVERLAP

This keyword request the overlap (or one-electron integral) matrix in the AO basis. A single block of type **overlap** is written. The AO ordering follows from the order of atoms in the **coordinates** block, and the ordering of contracted basis functions on a given atom type follows that written to the **basis** block. The header variable "elements" is set to the number of AO functions (nf), and the matrix can then be read with a FORTRAN loop structure of the form

```
   do 100 j = 1,nf
        read(npun,1000)(q(i,j),i=1,num)
100  continue
1000 format(6f11.6)
```

3.6 VECTORS, EIGENVALUES, OCCUPATIONS and SCFENERGY

These keywords request the output of one or more sets of MO vectors and associated information about the wavefunction. The four keywords each give rise to one block for each wavefunction requested, the types being **vectors**, **eigenvalues**, **occupations** and **scf_energy** respectively. The keywords may optionally be followed by one or more integers, specifying which sets of vectors are required by the section numbers of the dumpfile sections on which they are stored. The section selections on each keyword are combined, and the same list of sections is used for output of all four block types. It is thus not possible to specify the sections independently for any of the four blocks, although if any the keywords is omitted no blocks of that type will be written. The block header variable "index" is set to a different integer value (counting from 1) for each set of vectors punched. If the section specification is omitted, the program will attempt to determine a suitable set (or sets) from the data given on the ENTER directive, but at present there are a number of run types for which this has not been implemented.

The vectors are output in the AO basis, and should be read from a FORTRAN program using a loop structure similar to that given above for the **overlap** block. The value of nf (the number of AO function and the number of vectors punched) should again be taken from the value of the header variable "elements". The fastest varying index (i) is the MO (vector) index, and j is the AO (basis) index.

The SCFENERGY keyword results in a **scf_energy** energy block for each vectors section requested, containing a single record with the SCF energy written with format (f15.8).

The EIGENVALUES and OCCUPATIONS keywords both give rise to blocks (**eigenvalues** and **occupations**) respectively, containing one record per MO, in format (f10.6). The eigenvalues are in Hartrees, and the occupations in electrons.

3.7 GVB

The GVB pair information associated with a set of canonicalised orbitals may be punched using the GVB keyword. Only one set of GVB pair information is stored on the dumpfile, that associated with the second entry on the ENTER directive. This section must be one of those for which a **vectors**, **scf_energy**, **eigenvalues** or **occupations** block has been requested.

There is one record per gvb pair, arranged as follows

item	format
orbital indices	2i5
pair coefficients	2f12.6

3.8 GRID and GSYM

The GRID keyword requests output of numerical data arrays constructed using the directives described in Part 8 (under control of RUNTYPE ANALYSE, GRAPHICS) and written to the

dumpfile. Both arrays of point coordinates (ie grids without associated data values, generated by the GDEF directive) and arrays of data calculated at specified points (from the CALC directive) may be output. The GRID keyword must be followed by at least one integer value, specifying the dumpfile section from which the data is to be taken. Up to 10 grids may be requested. The GSYM keyword requests that symmetry equivalences between the points also be computed and written out.

The punchfile will contain a number of data blocks for each grid requested.

The **data** block is empty, and is used signify the beginning of set of data block.

The **grid_title** block simply contains one record - the title, as specified on the GRAPHICS directive.

The **grid_axes** block contains one record for each axis of a regular grid specification. The contents of each of the records are as follows.

item	format
number of data points along the axis	1x,i3
axis ranges	2f11.6
axis type	i2
axis unit	a3
axis name	1x,a5

In general, the axis range fields hold the minimum and maximum values of the parameter corresponding to the axis. In the grids written by GAMESS–UK this is always a distance, and the minimum and maximum values are set to 0.0 and the length of the plot edge (in aus) respectively. The axis type is currently always 0 (denoting a regularly spaced axis), and axis unit is the string ' au'.

The **grid_mapping** block serves to define the 3D coordinates of the grid. There is one record for each axis, containing two coordinate triples. Like **grid_axes** This block only appears for regular grids.

item	format
x,y,z coords of start of axis	3f10.6
x,y,z coords of end of axis	3f10.6

The **grid_points** block contains the coordinates in 3D space of each point on an irregular grid. Each record contains the x, y, and z coordinates of a point on the grid, in format 3f11.6. Irregular grids are generated by the GAMESS–UK 3D contouring routines.

Certain irregular grids, such as those generated by the GAMESS–UK contouring routines have the property that although the points do not lie regularly in 3D space, the space they occupy may be divided into cells (in general parallelepiped shaped) with either 1 or 0 points in each. This property arises simply because the contouring routines start by dividing space in this way, and performing an interpolation calculation, where appropriate, in each cell. The **grid_indices** block contains one record per grid point, specifying the cell concerned by 3 integer indices

(format 3i5)

The data itself is written out in a **grid_data** block using format (f11.6). It may be read using FORTRAN code of the following type. np1 and np2 are the numbers of points along the axes, obtained from the **grid_axes** block, and nel is the number of elements associated with each data point (1 = scalar data, 3 = vector etc) obtained from the value of the header variable "elements". Note that the fastest varying index (apart from the loop over elements) is that corresponding to the *first* axis.

```
      do 100 j = 1,np2
         do 100 i = 1,np1
            read(npun,1000)(q(k,i,j),k=1,nel)
 100 continue
1000 format(3f11.6)
```

The **grid_data** block will be empty ("elements"= 0) if the dumpfile section requested on the punch directive contained a grid definition (from a GDEF directive) rather than an array of values (from CALC).

The **grid_symmetry** block will be included if the GSYM keyword is given. For each point there will be a record containing a single integer (format (1x,i8)) corresponding to the index in the list of points of the first point which is symmetry equivalent to the current point.

3.9 MULLIKEN and LOWDIN

These keywords request that the respective population analyses (by atomic orbital and reduced to atoms) and calculated charges be written out. For UHF wavefunctions, the alpha and beta components are written out separately. The blocks generated by the MULLIKEN keyword are as follows, the block type naming should be self-explanatory.

block type	Notes
mulliken_ao_populations	
mulliken_atom_populations	
alpha_mulliken_ao_populations	UHF only
alpha_mulliken_atom_populations	UHF only
beta_mulliken_ao_populations	UHF only
beta_mulliken_atom_populations	UHF only
mulliken_atomic_charges	

The data in each block is arranged one number per record, with format (f10.6). The atom ordering follows that of the **coordinates** block, while the AO ordering on a given centre follows the arrangement of contracted functions given in the **basis** block.

The blocks generated by the LOWDIN directive are precisely analogous with corresponding names.

3.10 SPIN

The spin densities at the nuclei are output in response the SPIN keyword. A single block **spin_densities** is written, with one record per atom in format (f10.6).

3.11 PDC

This keyword requests that the results of a potential derived charges calculation be written to the punchfile. A single data block (**potential_derived_charges**) is generated, with one record per atom in format f10.6.

3.12 VIBRATIONAL and NORMAL

These keywords control the output of blocks of types **vibrational_frequencies** and **normal_coordinates** respectively, which contain the results of the normal coordinate analysis which is performed under RUNTYPE FORCE or RUNTYPE HESSIAN. The **vibrational_frequencies** block contains the frequencies (in wavenumbers) for each mode with a real frequency. There is one value per record, format (f16.6). One **normal_coordinates** block is written out for each mode with a real frequency, the block variable "index" identifies the corresponding value in the **vibrational_frequencies** block. Each record contains an atom symbol, followed by the normal coordinate for that atom, the format (2x,a8,3(f12.7)) is the same as that for the spatial coordinates.

3.13 INFRARED and RAMAN

These keywords control the output of block of the types **infrared_intensity** and **raman_intensity** respectively, which contain the intensities computed under RUNTYPE HESSIAN, RUNTYPE INFRARED or RUNTYPE RAMAN. A separate block is written for the intensity of each vibrational mode with one value per record, format (f16.6).

3.14 GRADIENT and TRANSFORM

These keywords are used when the gradient of the energy with respect to the nuclear coordinates is required. GRADIENT requests the output of a **gradients** block, each record of which contains the derivative of the energy with respect to the cartesian coordinates of one atom, format (2x,3f15.7).

Since the molecule may well have been reoriented by GAMESS–UK as part of the point group symmetry analysis, these gradients will have to be transformed if they are required to correspond to the orientation of the molecule as it was input to GAMESS–UK, and account must also be taken of the reordering of atoms. The TRANSFORM keyword requests a block of type **tr_matrix**, containing a rotation and translation matrices (denoted here as R and T respectively) which may be used to construct the gradient in the original, input frame. There are 3 records, record number i containing R(i,1), R(i,2), R(i,3), T(i) in format (2x,4f15.7). Taking

the coordinates **c** in the GAMESS–UK coordinate system (i.e. after symmetry adaption, as found in the **coordinates** block) the transformation R**c**-T yields the coordinates of the atom as it was input to GAMESS–UK, and R**g**, (where **g** is the gradient from the **gradients** block) is the gradient in the initial coordinate system. Note that the record structure written out in the **tr_matrix** block is suitable for inclusion (without the block header) after the ORIENT directive.

The TRAN keyword also results in the punching of a **point_group** block, which contains a single record containing a point group symbol and the order of the principle rotation axis (format (1x,a8,2x,i2)). Note that the point group symbol matches that printed out by GAMESS–UK in the listing (one of c1, cs, ci, cn, s2n, cnh, cnv, dn, nh, dnd, cinfv, dinfh, t, th, td, o, oh, i or ih) The symbol and axis order are also required as input to the ORIENT directive.

3.15 SECDER

The cartesian second derivative matrix is punched out in a block named **cartesian_hessian**. One element is written per line, with format f15.7. Only the *analytic* second derivatives can be punched in this way.

3.16 DMA

The punch DMA option will result in the output of the sites and magnitudes of the poles from a DMA analysis. It results in a single block **dma_sites**, containing one record per site, of the following structure:

item	format
site label	a8,2x
x,y,z coordinates	3f10.6
highest pole (lmax)	i4
relative radius	f10.6

The poles are then punched out, using one block of type **dma_pole** for each pole. The block header defines a variable "sites" indicating the location of the pole in the list of DMA sites. The components of the pole are printed, one per line, in the sequence q00, q10, q11c, q11s, q20 etc, using a format e16.16.

3.17 TYPE and LEVEL

The punch facilities controlled by these keywords have not yet been fully implemented. A brief description is included here for completeness only. We do not recommend using the output from these directives at the present time

These keywords cause information concerning the mode of calculation to be provided, based on

the RUNTYPE and SCFTYPE specifications. The TYPE keyword causes a **calculation_type** block to be written containing one of the following strings

```
single_point
geometry_optimisation
saddle_point_search
force_constants
```

The string is left justified in a field of 25 characters.

The LEVEL keyword causes a **calculation_level** block to be punched containing a single line. The first field will normally be the scf type, with the exception that direct scf is classified as rhf and casscf is classified as mcscf. This string is left justified in a field of 8 characters. This field may be followed by one of the following, depending on the run type.

```
 plus ci
 plus gf
 plus tda
```

3.18 HIGH

The default format statements described above have been chosen to provide sufficient precision for most applications that need to analyse the GAMESS–UK wavefunction or geometry, without resulting in punchfiles of unreasonable length. However, sometimes data in the punchfile is needed to higher accuracy. For some data items (currently the **coordinates, grid_mapping, grid_points, grid_data gradients normal_coordinates, vibrational_frequency, tr_matrix, potential_derived_charges, mulliken.. lowdin.. and spin_densities** blocks) higher precision output is available. The HIGH keyword should then appear on the PUNCH directive, before the keyword requesting the data that is required to higher. The effect of the HIGH keyword does not carry over to subsequent PUNCH directives. The following example file might be used if the coordinates were required to higher precision.

```
TITLE
SCF, PUNCH COORDINATES
PUNCH HIGH COOR
ZMAT ANGS
O
H 1 1.0
H 1 1.1 2 109.0
END
RUNTYPE SCF
ENTER
```

The resultant punchfile is given below.

```
block = fragment records = 0
block = coordinates records =     3 unit = au \
 f_format = "(2x,a8,3(3x,f22.14))"
  o               0.00000000000000         0.00000000000000        -0.21947397179724
  h               0.00000000000001         1.53845580221546         0.87789588718897
  h               0.00000000000000        -1.53845580221546         0.87789588718897
```

Note that if the punchfile is to be read by a FORTRAN program, with a fixed format read, the value of the "f_format" variable on the header must be extracted and passed to the FORTRAN read. FORTRAN- and C-callable routines to simplify this operation are available from the authors.

The line length of high format punched output will, in general, exceed 80 characters.

4 Punchfiles from Multiple Runtype Jobs

When an input file contains multiple RUNTYPE directives, care should be taken with the placing of the PUNCH directive. An error will occur if you request data that is not generated during current job step. It is sometimes possible to punch data from an earlier job step, but only if the data is resident on the dumpfile (e.g. the vectors).

A simple example is shown below, where an SCF calculation is followed by two wavefunction analysis steps: a graphical analysis calculation and a distributed multiple analysis. Here, none of the SCF results are needed, so it is possible to omit the PUNCH directive from the first job step

```
title
h2o 6-31g**
adapt off
accuracy 19 8
zmat angs
o
x 1 1.0
h 1 ho 2 hox
h 1 ho 2 hox 3 180.0
variables
ho 0.967
hox 53.8
end
basis 6-31g**
enter 1
runtype analyse
punch coordinates grid 100
graphics
gdef
type 2d
points 50
calc
type density
section 100
vectors 1
enter 1
```

```
runtype analyse
punch dma
dma
vectors 1
enter 1
```