GIMIC 2.0 users manual

Gauge-Including Magnetically Induced Currents

Jonas Jusélius

University of Tromsø
Department of Chemistry
N-9037 Tromsø

1 Introduction

This is the GIMIC program for calculating magnetically induced currents in molecules. For this program produce any kind of useful information, you need to provide it with an AO density matrix and three (effective) magnetically perturbed AO density matrices in the proper format. Currently only recent versions of ACES2 and Turbomole can produce these matrices, but Dalton is in the works. If you would like to add your favourite program to the list please use the source, Luke.

- For instructions how to compile and install this program refer to the INSTALL file in the top level directory.
- For more detailed information on how to use the program read the documents found in the Documentation directory. Note that the manual is obsolete.
- There is an annotated example input in the examples/ directory.
- For information on command line flags available run: '`gimic -help`'

The following features have been implemented in the program

- Current densities in 2D or 3D
- The modulus of the current
- The divergence of the current (this is useful for checking gauge invariance vs. gauge independence)
- Vector representation of the current in 2D or 3D
- Integration of the current flow through defined cut-planes in molecules
- Open-shells and spin currents
- Parallel execution through MPI (optional)

GIMIC has so far been interfaced to ACES2 and Turbomole. A small utility program to extract the AO density and perturbed densities from ACES2 calculations are included in the GIMIC source distribution. Turbomole 5.10 and newer also has the GIMIC interface built in.

2 Installation

GIMIC is written in pure Fortran 90/95, and thus requires a good F95 compiler to compile. GIMIC has been compiled and tested to work with the following compilers:

- GNU gfortran (4.2)

- g95 (0.9)
- Intel ifort (please note that ifort 9.0 does not work if optimizations are enabled)
- Portland pgf90

GIMIC uses the standard GNU autoconf generated 'configure' scripts to examine your system and pick sensible defaults for most build variables. For a complete list of available configuration options run

```
$ ./configure --help
```

It's recommended that GIMIC is properly installed after compilation, although not strictly necessary. Here /opt/gimic will be used as the install path. To configure and build gimic without parallel capabilities run

```
$ ./configure --prefix=/opt/gimic
$ make install
```

Then simply add /opt/gimic/bin to your path (or make the appropriate links in /opt/bin), and you should be up and running.

If you are not happy with the default compiler picked up by configure you can override the default by doing

```
$ FC=myfavouritef90 ./configure --prefix=/opt/gimic
```

If you want to build the ACES2 interface (xcpdens) you first need to build ACES2, and then run

```
$ ./configure --prefix=/opt/gimic --with-aces2=/path/to/ACES2/lib
```

If configure can't find the BLAS library (only needed for xcpdens), you need to specify where to look for it add the following flag to configure:

```
--with-blas-dir=/path/to/lib
```

If this does not work for some reason you can specify exactly how to link against BLAS on your system:

```
--with-blas='-L/path/to/my/blas -lfooblas -lwhatever'
```

Good luck!

3 Usage

To run GIMIC three files are needed:

1. A file containing the effective one-particle density, and the magnetically perturbed densities in AO basis
2. A MOL file, with information on molecular geometry and basis sets
3. A GIMIC input file

The following sections explains how to obtain the density file and the MOL file using either ACES2 or Turbomole.

3.1 Running ACES2

Using ACES2, the special driver script '`xgimic2.sh`' must be used to run the NMR shielding calculation. Modify the script to suit your needs (and set the paths correctly). If the NMR calculation is done with symmetry, the MOL file must be converted to C1 symmetry using the script `MOL2mol.sh`, prior to running GIMIC.

Example ZMAT:

```
CO2
O     2.14516685791074   0.00000000000000      0.00000000000000
C     0.00000000000622   0.00000000000000      0.00000000000000
O    -2.14516685791393   0.00000000000000      0.00000000000000

*ACES2(CALC=CCSD,BASIS=tzp,UNITS=BOHR
COORD=CARTESIAN
MEMORY=250000000
REFERENCE=RHF
SYMMETRY=ON
PROPERTY=NMR
MULTIPLICITY=1
CHARGE=0
SCF_MAXCYC=200,CC_MAXCYC=150,CC_EXPORDER=40
CC_CONV=10,SCF_CONV=10,LINEQ_CONV=10,CONV=10
LINEQ_EXPAN=30)
```

Run ACES2 via xgimic2.sh to produce the XDENS file:

```
$ xgimic2.sh --cc >aces2.out &
```

Convert the symmetry adapted MOL file to C1 symmetry:

```
$ MOL2mol.sh
```

The new MOL file is now called mol.

3.2 Running Turbomole

Starting with Turbomole 5.10, the GIMIC interface is part of the official distribution. To produce the necessary files to run GIMIC, you first need to optimize the wavefunction/density of the molecule, before running the `mpshift` program to produce the perturbed densities. Before you run `mpshift` you need to edit the `control` file and add the `$gimic` keyword. When the calculation has finished run the `turbo2gimic.py` script (distributed with GIMIC) to produce the `mol` and `XDENS` files.

3.3 Running GIMIC

To run gimic you need to have at least three files: The gimic input file (gimic.inp), the compound density file (XDENS) and the compound basis set and structure file (mol). Copy the example gimic.inp (in the `examples/` directory) to your work directory, edit to your needs, and execute

```
$ gimic [--mpi] [gimic.inp] >gimic.out
```

Before doing the actual calculation it might be a good idea to check that the grids are correct, run:

```
$ gimic --dryrun
```

and examine the .xyz files that GIMIC produces. If they look ok, simply run

```
$ gimic
```

If you want to run the parallel version, there is a wrapper script called '`qgimic`' (see `qgimic -help` for a list of command line options) to produce a generic run script for most queueing systems. Eg. to set up a parallel calculation with 8 CPUs, 1 h time and 200 MB memory to be run in `/work/slask`

```
$ qgimic -n 8 -t 01:00 -m 200 /work/slask
```

This produces a 'gimic.run' file. Edit this file and make sure it's ok, and then submit it to the queueing system:

```
$ qsub gimic.run
```

4 The GIMIC input file

The GIMIC input file is parsed by the getkw input parser, which defines a grammar based on sections and keywords in a recursive manner. The input consists of sections containing keywords and/or other sections, and so on. The input is in principle line oriented, but lines may be continued using a '

' at the end of a line. Furthermore, blanks and tabs are insignificant, with the exception of strings. Lines may be commented until end-of-line with a hash sign (#).

Sections are delimited by an opening '' and closing '', and may have a keyword argument enclosed between '(' and ')'.

Keywords come in two different types; simple keywords consisting of integers, reals or strings (enclosed in " "), and array keywords. Array keywords are enclosed in '[' ']' and elements – integers, reals or strings – are delimited by ','.

4.1 Keywords

The top level section defines a few global parameters:

dryrun=off Don't actually calculate anything. Good for tuning grids, etc. Can also be specified on the command line.

mpirun=off [boolean] Run in parallel mode.

title Useless keyword, but since every program with a bit of self respect has a title, GIMIC also has one...

basis=mol Name of the MOL file (eg. MOL or mol or whatever)

density=XDENS Name of the density file (eg. XDENS)

spherical=off Use spherical cartesians (i.e. 5d/7f/10g...). This is usually handled automagically. Experts only.

debug=1 Set debug level. The higher the number, the more useless output one gets.

diamag=on Turn on/off diamagnetic contributions

paramag=on Turn on/off paramagnetic contributions

openshell=false Open-shell calculation

screening=off Use screening to speed up calculations

screen_thrs=1.d-8 Screening threshold

show_up_axis=true Mark the "up" axis in .xyz files

calc=[cdens,...] This keyword determines what is to be calculated, and in what order. Possible options are: 'cdens' – calculate current densities, 'integrate' – integrate the current flow through a cut-plane, 'divj' – calculate the divergence of the current. Each of these options have their own respective sections to specify options and grids.

4.2 The current density

Section: cdens

jtensor=JTENSOR Name of output file containing the current tensors

jvector=JVECTOR Name of output file containing the current vectors

magnet=]0.0, -1.0, 0.0[Vector which specifies the direction of the magnetic field.

magnet_axis=z] Specify the magnetic field along a defined axis. Valid options are: i,j,k or x,y,z or T. "i,j,k" are the directions of the basis vectors defining the computational grid after any Euler rotation. "x,y,z" are the absolute fixed laboratory axis. "T" is used for integration and specifies the direction which is orthogonal to the molecular plane, but parallel to the integration plane.

scale_vectors=1.0 Scaling factor for plotting purposes.

diamag=on Annihilate the diamagnetic contribution to the current. Experts only.

paramag=on Annihilate the paramagnetic contribution to the current. Experts only.

grid(std) [subsection] Grid to be used for calculating the currents. See the "Grids" section for a description of how to specify grids.

plot(on) [subsection] Produce files suitable for plotting with 'gnuplot' or 'gopenmol'

vector=JVEC File to contain the current vector field (gnuplot friendly)

modulus=JMOD File to contain the modulus of the current density (gnuplot friendly)

nvector=NJVEC File to contain the normalized current vector field (gnuplot friendly). Mostly useful for debugging purposes.

gopenmol=jmod.plt File to contain the current density in a gopenmol friendly format.

4.3 Integration

Section: integral

magnet_axis=T] Specify the magnetic field along the direction which is orthogonal to the molecular plane, but parallel to the integration plane.

magnet=]0.0, -1.0, 0.0[Vector which specifies the direction of the magnetic field.

modulus=off Calculate the mod(J) integral, this is useful to verify that the actual integration grid is sensible in "tricky" molecules.

tensor=off Integrate the tensor components

interpolate=off If a calculation has been preformed on a even spaced grid, generate a grid suitable for Gaussian integration by doing Lagrange interpolation

lip_order=5 Polynomial order of the Lagrange Interpolation Polynomials

grid(bond) [subsection] Grid to be used for calculating the currents. See the "Grids" section for a description of how to specify grids.

4.4 The divergence of the current field

Subsection: divj

magnet=]0.0, -1.0, 0.0[Vector which specifies the direction of the magnetic field.

magnet_axis=z] Specify the magnetic field along a defined axis. Valid options are: i,j,k or x,y,z or T. "i,j,k" are the directions of the basis vectors defining the computational grid after any Euler rotation. "x,y,z" are the absolute fixed laboratory axis. "T" is used for integration and specifies the direction which is orthogonal to the molecular plane, but parallel to the integration plane.

gopemol=divj.plt Filename of gOpenMol plot

grid(std) [subsection] Grid to be used for calculating the currents. See the "Grids" section for a description of how to specify grids.

4.5 The electronic density

The GIMIC program can also produce plots of the electronic density. This code is very rudimentary currently, and cannot produce densities of specific MOs or ranges of MOs.

Section: edens

density='EDENS' Filename which contains AO density. XDENS is fine usually.

density_plot='edens_plt.txt' File name of density plot

gopemol=edens.plt Filename of gOpenMol plot

grid(std) [subsection] Grid to be used for calculating the currents. See the "Grids" section for a description of how to specify grids.

5 Grids

There are two principal types of grids; the simple 'std (or base)' grid, which is defined by a pair of (orthogonal) basis vectors, and the 'bond' grid which is mostly useful for defining cut-planes through bonds for integration. There is also a third grid type 'file', which specifies a file containing gridpoints For the exact format of this file please refer to the source in grid.f90. Furthermore there are two types of grids, evenly spaced or with grid points distributed for Gauss-Legendere or Guass-Lobato quadrature. This is specified with the 'type=even|gauss|lobato' keyword. When a quadrature grid is specified the order of the quadrature must also be specified with the 'gauss_order' keyword. The number of grid points in each direction is specified either explicitly using either of the array keywords 'grid_points' or 'spacing'. If the chosen grid is not a simple even spaced grid, the actual number of grid points will be adjusted upwards to fit the requirements of the chosen quadrature.

The shape of the grid can also be modified by the 'radius' key, which specifies a cutoff radius. This can be useful for integration. Sometimes it's practical to be able to specify a grid relative to a well know starting point. The 'rotation' keyword specifies Euler angles for rotation according to the x->y->z convention. Note that the magnetic field is not rotated, unless it is specified with 'magnet_axis=i,j or k'.

GIMIC automatically output a number of .xyz files containing dummy points to show how the grids defined actually are laid out in space.

5.1 Basic grids

The 'std' grid is defined by giving an 'origin' and two orthogonal basis vectors 'ivec' and 'jvec' which define a plane. The third axis is determined from $\mathbf{k} = \mathbf{i} \times \mathbf{j}$. The array 'lengths' specifies the grid dimensions in each direction.

grid(std):

type=even

origin=[-8.0, -8.0, 0.0] Origin of grid

ivec=[1.0, 0.0, 0.0] Basis vector i

jvec=[0.0, 1.0, 0.0] Basis vector j (k = i x j)

lengths=[16.0, 16.0, 0.0] Lenthts of (i,j,k)

spacing=[0.5, 0.5, 0.5] Spacing of points on grid (i,j,k)

grid_points=[50, 50, 0] Number of gridpoints on grid (i,j,k)

rotation=[0.0, 0.0, 0.0] Rotation of (i,j,k) -> (i',j',k') in degrees. Given as Euler angles in the x->y->z convention.

5.2 Bond grids

The 'bond' type grids define a plane through a bond, or any other defined vector. The plane is orthogonal to the vector defining the bond. The bond can be specified either by giving two atom indices, 'bond=[1,2]', or by specifying a pair of coordinates, 'coord1' and 'coord2'. The position of the grid between two atoms is determined by the 'distance' key, which specifies the distance from atom 1 towards atom 2. For analysing dia- and paramagnetic contributions, the positive direction of the bond is taken to be from atom 1 towards atom 2. Since one vector is not enough to uniquely defining the coordinate system (rotations around the bond are arbitrary), a fixpoint must be specified using either the 'fixpoint' atom index or the 'fixcoord' keyword. This triple of coordinates is also used to fix the direction of the magnetic field when the 'magnet_axis=T' is used.

The shape and size of the bond grid can be specified in two ways. The origin of the grid is fixed to the center of the bond, and all specifications are relative to this origin. The first method specifies lengths in four directions (they can be negative, as long as they pairwise sum up to a positive number):

```
up=5.0
down=5.0
in=1.5
out=5.0
```

The other way to specify the shape is using the 'height' and 'width' keywords, which specify intervals relative to the origin:

```
height=[-5.0, 5.0]
width=[-1.5, 5.0]
```

grid(std):

type=gauss|lobato Use uneven distribution of grid points for quadrature

bond=[1,2] Atom indices for bond specification

fixpoint=5 Atom index to use for fixing the magnetic field and grid orientation

coord1=[0.0, 0.0, 3.14] Coordinate of atom 1

coord2=[0.0, 0.0, -3.14] Coordinate of atom 2

fixcoord=[0.0, 0.0, 0.0] Fixation coordinate

distance=1.5 Place grid 'distance' between atoms 1 and atom 2

gauss_order=9 Order for Gauss quadrature

spacing=[0.5, 0.5, 0.5] Spacing of points on grid (i,j,k) (approximate)

grid_points=[50, 50, 0] Number of grid points on grid (i,j,k) (approximate)

up=4.0 Grid size in **i** direction

down=4.0 Grid size in $-$**i** direction

in=1.0 Grid size in $-$**j** direction

out=6.0 Grid size in **j** direction

height=[-4.0, 4.0] Grid size relative to grid center

width=[-1.0, 6.0] Grid size relative to grid center

radius=3.0 Create a round grid by cutting off at radius

rotation=[0.0, 0.0, 0.0] Rotation of (i,j,k) -> (i',j',k') in degrees. Given as Euler angles in the x->y->z convention.