A very short introduction to orbkit

February 4, 2014

orbkit is a Python program package for the generation of molecular orbitals and densities from output files of MOLPRO, TURBOMOLE, Gamess US, and Gaussian. A list of currently implemented features (and limitations) can be found in the Overview to check the suitability of orbkit for your problem. An instruction for the easy usage of orbkit and the functional principles are described in the Manual section. The several functions can be separately used and can be easily implemented in your own program.

The orbkit support team, Axel, Gunter, and Vincent, welcomes every new user and will be available to answer your questions. For any change requests, do not hesitate to contact the orbkit support team.

1 Overview

1.1 Features of orbkit

orbkit enables the user the calculation of atomic orbitals, molecular orbitals and densities on an arbitrary regular grid. The grid creator function generates a user defined grid which can be specified in different ways. The grid can be centered to a selected atom and the origin.

By default, orbkit loads the required data from Molden files which can be created with MOLPRO or TURBOMOLE. Other readable output formats are Gaussian (.fchk or .log) or Gamess US output files.

The centerpiece of orbkit, the orbkit_core.py python module, offers a variety of calculation options. The user can choose the computation and the storage of all or just selected molecular orbitals. The storage of all atomic orbitals is also a possible feature. The density can be calculated from all or just from several molecular orbitals. The selection of the molecular orbitals is realized by an external file. Example files, for the usage of the major features of orbkit, are available in the program package.

orbkit can construct different types of output files with the help of the output Creator (orbkit_output.py). The output formats include cube files, HDF5 files, and ZIBAmira Mesh files. orbkit can create a ZIBAmira network which enables the easy depiction of the densities or molecular orbitals with ZIBAmira.

A detailed description of the Python modules of orbkit are listed in the Function Reference section.

1.2 Limitations of orbkit

One limitation of orbkit is the ability to calculate only s, p, d, f and g atomic orbitals (Molden file limitation).

For calculation of the atomic orbitals and molecular orbitals, orbkit requires cartesian gaussians. The usage of spherical gaussian functions is not yet possible. The usage of cartesian gaussian functions is standard for the Molden file format, the Gaussian Formatted Checkpoint File (.fchk), and the Gamess US output. For Gaussian standard output (.log), this has to be specified in the route section with: `6D 10F`

2 Manual

`orbkit` can be very easily operated via the terminal. For advanced users, it is recommended to use `orbkit` as a Python function. Please note that the graphical user interface has been removed for the time being and the option `--no_display` is not necessary.

2.1 Usage via the Terminal

For an overview of the available features, a general help text is shown in the terminal by using the command:

```
python orbkit_main.py -h
```

For the normal calculation of the density from all molecular orbitals on the standard grid, the following command must be entered in the console:

```
python orbkit_main.py -i inputfile.molden -o outputfile --no_display
```

The first two terms start the Python program `orbkit_main.py`. This module requires an input file defined with `-i` and the filename and needs an output file which can be specified by `-o` followed by the filename. The file extension of the output will be automatically created by the program and does not need to be defined. The command `--no\_display` suppresses the execution of the graphical interface.

By default, `orbkit` divides the 3-dimensional grid in 2-dimensional slices. The atomic orbitals, the molecular orbitals, and the density are calculated for each slice. At the end of the calculation, the density is reassembled and stored in an output file. In the default setting, `orbkit` performs the density calculation by starting four subprocesses, which are distributed over the existing CPUs. The number of subprocesses can be modified with the subsequent command:

```
--numproc=4
```

For the modification of the grid, one can transfer the modified grid to the program via an external file and add a special related command in the terminal:

```
python orbkit_main.py -i inputname -o outputname --read_grid=new_grid.csv --no_display
```

An example of such an external text file can be found in the program package and has to be constructed as follows:

```
xmin=,-7.5,xmax=,7.5,Nx=,151
ymin=,-7.5,ymax=,7.5,Ny=,151
zmin=,-7.5,zmax=,7.5,Nz=,151
```

A special feature of the grid creator is the centering of the grid to a specific atom and the origin which can be performed by adding the two commands seen below:

```
-c --atom=A_NUM
```

`A_NUM` defines the number of the atom to which the grid is centered and corresponds to the numbering in output file of the respective quantum chemistry program.

For the calculation of the density, `orbkit` takes only into account the occupied molecular orbitals by default. To perform the density calculation with a selected set of molecular orbitals, the command:

```
--mo_list=MO_List
```

can be used. The selection can be specified by an external list `MO_List` with the structure:

```
1.1 2.1 1.3
1.1 4.1
4.1 2.3 2.1
```

or

```
1 2 3
3 4
```

Every row signifies a new calculation of the density from the molecular orbitals specified in this row. On the one hand, the orbitals can be defined by the symmetry and on the other, by the order in the respective output file. In the same manner, the computation and storage of a selected set of molecular orbitals can be invoked by

```
--calc_mo=MO_List
```

Here, the density computation is omitted.

In `orbkit`, the output can be issued as standard Gaussian cube files, HDF5 files or ZIBAmira Mesh files. Additionally, `orbkit` can directly create a ZIBAmira network for the easy depiction in ZIBAmira. The network is called as follows

```
--hx_network
```

and includes a simple color map creator. The cube file (file extension: .cb) which is a normal text file and contains the important informations of the calculation is generated by default. For large systems, the cube file requires a lot of space on the hard drive and plenty of time for its creation. In contrast, the HDF5 file format (file extension: .h5) which is a hierarchical data format can store and organize large amounts of numerical data. More informations about this file format can be found at:

http://en.wikipedia.org/wiki/Hierarchical_Data_Format

Another advantage of this file format is the easy readability with Matlab or Python. Example files for loading HDF5 files via Matlab or Python are available in the program package. With the JAVA program, HDFVIEW, the user can easily load and read the HDF5 files. The command for the creation of HDF5 files reads as follows

```
--hdf5
```

The last available data format are the ZIBAmira Mesh files (file extension: .am) which can be directly loaded in ZIBAmira and is generated with the subsequent command

```
--amira
```

2.2 Usage as Python Function

The first step of the usage of `orbkit` as a Python function is the import of the `orbkit_main.py` program as a Python module with the subsequent Python code line:

```
import orbkit_main
```

An example program code where `orbkit` is called as a function can be found in the `orbkit_run_from_fcn.py` file in the program package. Again, an exemplary calculation of a water molecule with a HDF5 output file serves for the explanation of the program code.

The name of the input file and the output file have to be defined with:

```
in_fid = 'h2o.md'
out_fid = 'h2o'
```

After the initialization of `orbkit` with the following code line:

```
orbkit_main.init()
```

the user can modify the settings for the investigated system. For the water molecule, the input file, the output file and the output file format have been selected, and the grid has been modified by an external file. The following code lines perform these changes:

```
orbkit_main.options.hdf5 = True
orbkit_main.options.csv_grid = './Tab.csv'
orbkit_main.options.filename = in_fid
orbkit_main.options.outputname = out_fid
```

All available features can be found in `orbkit_main.py` in the class cOptions. In the last step, the user has just to run `orbkit` as follows

```
orbkit_main.main()
```

Other example files can be found in the `orbkit_module_example.py` file and `orbkit_derivative_example.py` file in the program package.

3 Required Python Modules for `orbkit`

For a proper execution of `orbkit`, the following Python modules are required:

3.1 General Modules

- os - Miscellaneous operating system interfaces (http://docs.python.org/2/library/os.html)
- sys - System-specific parameters and functions (http://docs.python.org/2/library/sys.html)
- re - Regular expression operations (http://docs.python.org/2/library/re.html)
- time - Time access and conversions (http://docs.python.org/2/library/time.html)
- copy - Shallow and deep copy operations (http://docs.python.org/2/library/copy.html)
- optparse - Parser for command line options (http://docs.python.org/2/library/optparse.html)
- csv - CSV File Reading and Writing (http://docs.python.org/2/library/csv.html)
- multiprocessing - Process-based "threading" interface (http://docs.python.org/2/library/multiprocessing.html)

3.2 Special Modules

- scipy - Library of algorithms and mathematical tools (http://en.wikipedia.org/wiki/SciPy)
- gtk - GTK+ graphical user interface library (http://en.wikipedia.org/wiki/PyGTK)
- h5py - Interface to the HDF5 binary data format (https://code.google.com/p/h5py/)
- numpy - Library of high-level mathematical functions (http://en.wikipedia.org/wiki/NumPy)

4 Variables

`geo_info` → only required for creation of output file. python list (strings); dimensionality: (number of atoms ×3)

- 1st element: atom symbol
- 2nd element: atom number (according to output file, i.e., starting from 1)
- 3rd element: nuclear charge of atom

`geo_spec` → python list (floates); dimensionality: (the number of atoms ×3)
`ao_spec` → python list of python dictionaries (number of dictionaries corresponds to number of contracted Gaussians):

- `coeffs`: numpy array, 0th element: coefficient, 1st element: exponent; dimensionality: (number of primitives ×2)
- `pnum`: number of primitives (integer)
- `type`: type of the atomic orbital (string)
- `atom`: atom number (according to output file, i.e., starting from 1) (integer)

`mo_spec` → python list of python dictionaries (number of dictionaries corresponds to number of molecular orbitals):

- `coeffs`: numpy array (vector); dimensionality: (number of atomic orbitals ×1)

- `energy`: energy of molecular orbital (float)
- `occ\_num`: occupation of molecular orbital (float)
- `sym`: symmetry of molecular orbital (string)

`ao_list` → numpy array containing all atomic orbitals on the specified grid
`mo_list` → numpy array containing all molecular orbitals on the specified grid
`rho` → global numpy array in `orbkit_main.py` containing the density