SDGMS Manual version 1

Jordan Burkhardt

December 2024

Contents

1 Introduction

SDGMS was developed by Jordan Burkhardt at the Wanlu Li lab in the UCSD Nano-engineering department in 2024 as part of a master's thesis[4]. This software package employs a unique searching procedure which seeks to reduce the number of geometry optimizations required to discover the global minimum geometry. This method also employs a seed generation method to jumpstart the search, in addition to filtering techniques to prevent searching of unreasonable structures. Parameters to control all aspects of search can be specified by the user and are explained in this document.

2 General Usage

To perform global minimum searching 4 input files are required. The first of which is the SDGMS input file, which will be explained in further sections. Input files for the accompanying *ab initio* software for both single point and geometry optimizations must be provided. Finally a python script which interfaces SDGMS with the *ab initio* software must be provided, except for usage with the ADF[3, 6] and Gaussian 16[2, 1] softwares. This script will also be explained in a further section.

To run SDGMS all required input files must be placed in the directory where the search will be run. Then the program can be executed with the following syntax.

```
SDGMS inp.txt >> searchProgress.txt
```

Here, the SDGMS input file is inp.txt and the runtime output is redirected to the file searchProgress.txt although this is unnecessary as search results will be output to file names specified in inp.txt regardless.

3 SDGMS input file

Complete instructions for how to construct a SDGMS input file are included here, in addition to explanations of the various input parameters. Below is a complete input deck to run a global minimum search of the LaB_4^- molecule using ADF as the quantum chemistry package.

```
composition
2 La
4 B
end
xyzinp input
xyzout LaBout.xyz
rva
3,1 4,0
1,6,0
```

```
2,3,0
end
keep -1
calculator ADF
energyFile adfE.run
optimizationFile adfO.run
timeOutE 35
timeOutO 125
time 47.5
nopt 10
stepsize 0.4
uphill 20
branches 35
criteria 30
unique 0.07
nad 2
```

3.1 Composition block

The composition block specifies the composition of the search molecule. For syntax the word composition should be used as the first line, and the keyword end will end the block. For each line in between composition and end, the quantity of each element is specified. Below, the composition of LaB_4^- is specified. Please note that the charge of the molecule is specified within the *ab initio* calculators input deck.

```
composition
2 La
4 B
end
```

This section is required and there is no default value.

3.2 Input geometry for *ab initio* softwares

During runtime SDGMS will produce a folder titled calc$n$ for each quantum chemistry calculation that must be performed. n is the index of the quantum chemistry calculation. A copy of the quantum chemistry package input file is made inside the calculation folder, in addition to the input geometry. The name of this file is specified as follows:

```
xyzinp input
```

This command produces the file input.xyz as the input geometry for the quantum chemistry package. The keyword xyzinp is used to specify the name of the xyz file that the quantum chemistry package is looking for, in this case input.xyz. the .xyz ending is automatically added to whatever filename is provided. This should match the name provided in the quantum chemistry input files. For softwares that cannot take a geometry from an xyz file, the interface script is expected to use the contents of input.xyz to modify the quantum chemistry package input files. For the gaussian 16 software, this is handled automatically without the use of a python interface script.

The default value is *input.xyz*

3.3 Search results geometry

The xyzout keyword

```
xyzout [filename]
```

specifies the name of the primary output file for the geometry search with the preceeding useage. After the calculation has terminated, all local minima are sorted by energy and then saved into an xyz file where each frame corresponds to a local minima. The first frame is the global minima. The time it took to reach the global minima is included as part of the frame comment.

Additionally, after seed generation, the energy of each seed is calculated. Then these seeds are sorted and saved into a folder with the title $energy_sorted_[filename]$. The structures resulting from the geometry optimization of these seeds are saved in $optimized_[filename]$.

The default value is *out.xyz*

3.4 Seed generation

A brief explanation of the seed generation procedure is required to understand how to format the input section for the seed generator. A seed is generated for a molecule of n atoms by creating a series of polygons, each with e_i edges. An atom is placed at the edge of each polygon. All possible orderings of polygons (including placing polygons inside of eachother) and all possible distributions of atoms (and edges with no atoms) are generated when limitations are not placed on the seed search (i.e. these orderings are not random). Therefore, to create a seed, the number of polygons and the size of each polygon must be specified. Instead of specifying the size of each individual polygon in the seed, permitted sizes are specified in the SDGMS input. Then possible combinations of these permitted sizes are determined by the seed generator. Acceptable combinations follow the relation:

$$n_{atoms} + n_{holes} = \sum_{i=0}^{n_{polygons}} e_{a_i} \tag{1}$$

In this relationship, e is a vector of allowed polygon sizes. a is a list of indeces, length $n_{polygons}$ where a_i is the index of e such that e_{a_i} is the size of the ith polygon. SDGMS will determine this order. It is up to the user to ensure that there is some ordering of the given e, number of polygons, $n_{polygons}$, number of atoms, n_{atoms} and number of holes n_{holes} satisfies the relationship.

These inputs are specified in the rva input block as follows:

```
rva
3,1 4,0
1,6,0
2,3,0
end
```

The keyword *rva* specifies the beginning of the input block. Following this each line refers to a combination of input values for seed generation that the user wants to explore. In each of these, the first number specified is the number of rings. Then e, the list of allowed polygon sizes, is specified with individual list elements specified by a space. Finally another comma separates this segment from the number of holes which is the final value. Holes allow for no atom to be placed at a polygon edge. This number specifies the total number of holes in the generated seeds.

For the first line here we see that 3 polygons are requested, with allowed sizes of 1 and 4 edges and 0 holes. We can find a way to satisfy the relationship as follows:

$$e = 1, 4 \tag{2}$$

$$a = 0, 1, 0 \tag{3}$$

$$\sum_{i=0}^{n_{polygons}} e_{a_i} = 1 + 4 + 1 = 6 = n_{atoms} + n_{holes} \tag{4}$$

For a further example we can see seeds generated for the B_6^- anion search from the following input block:

```
rva
2,3,0
2,5 1,0
1,6,0
end
```

Here we can see that the line which specified polygons for sizes 5 and 1, combinations placing the 1 edged polygon inside the 5 edged polygon and ontop of the 5 edged polygon were created. Additionally SDGMS uses values provided in other input parameters to find permited bond distances in seeds.

Just a single set of inputs could create an excessively large amount of seeds for even a small molecule. Therefore the number of seeds should be reduced before single point calculations are performed on the seeds. The input keyword (*keep*) for this will be specified in the next section. For now it is important to note that when such a keyword is used, it reduces the amount of seeds we keep. If SDGMS created all seeds first, and then randomly deciding to keep a specified number of them, this would be computationally expensive for time and memory. Therefore before seed generation occurs, the number of possible seeds from the input are determined, and random indices within this range are

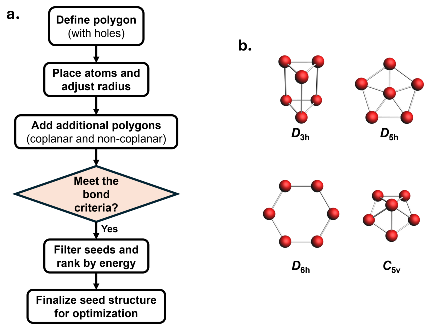
a.
Define polygon (with holes)
Place atoms and adjust radius
Add additional polygons (coplanar and non-coplanar)
Meet the bond criteria?
Yes
Filter seeds and rank by energy
Finalize seed structure for optimization
b.
D_{3h}
D_{5h}
D_{6h}
C_{5v}

Figure 1: a) A flowchart of the seed generator. b) Example seeds created by the above input block.

selected. Only seeds that would be generated from the said indices are then generated, greatly reducing time and memory.

There is no default value and this must be specified.

3.5 keep keyword

The number of seeds to perform single point calculations on, among all possible seeds that could be generated in the rva block is specified here. This is done with the keep keyword that is followed by the number of seeds (n). The value of -1 for n indicates keeping all seeds.

```
keep n
```

The default value is 0.

3.6 *Ab initio* package specification

Initially SDGMS was developed with compatibility for gaussian 16 and adf built in, however due to concerns of differing behavior for these packages among environments and the wide variety of available quantum chemistry packages, a different functionality was introduced to allow the user to generate a python script which bridges SDGMS with the desired quantum chemistry package. An example python script for useage with the CP2k package[5] is provided in the SDGMS github and later in this document. The calculator is specified with the calculator keyword. When a python script is being used the value P should be entered. For gaussian the keyword should be gaussian and for adf the keyword should be ADF.

```
calculator P
```

It is strongly recommended to use a custom python script even if the user wants to use SDGMS along with Gaussian or ADF.

The default value is P.

3.7 Quantum chemistry package single point, and geometry optimization input file specification

SDGMS needs to know the names of the single point input file and the geometry optimization file in order to copy them into the calc*n* folder. Here *n* is the calculation index. These are specified as followed:

```
energyFile cp2kE.inp
optimizationFile cp2kO.inp
```

Here the single point files name is cp2kE.inp, and the geometry optimization files name is cp2kO.inp.

There are no default values and these must be specified. If left blank the program may still work, but copying the files over will be left to the python script interface.

3.8 Quantum chemistry package interface script

Because SDGMS is written in C++ and it would be more difficult for users to learn the code base and modify the C++ source code, SDGMS can use a python file to communicate with the quantum chemistry package. The requirements for this python file, and a sample will be provided in later sections. This section is only to specify the files name.

```
pythonFile script.py
```

Here *script.py* is the name of the python script. This is also the default value.

3.9 Timers

3 timers are used by SDGMS. The first of which is the overall time allowed for the calculation, as many super computers have a time limitation for tasks. When this time is reached, SDGMS will finish up and print results. It will also create an output file to continue the current search from this point in a later task. This is specified in hours and is as follows:

```
time 47.5
```

Here the calculation is allowed to run for 47 and a half hours, which is also the default value.

The next timers are maximum limit for a single point and geometry optimization calculations. For both single point and geometry optimizations, it is important to limit the time given to the calculation so that it can just give up when this time is reached, rather than stalling the search.

```
timeOutE 35
timeOutO 125
```

For a single point calculation timeOutE is the keyword, and timeOutO for geometry optimizations. These are specified in minutes.

If not specified, no limits will be placed on *ab initio* calculator run time.

3.10 Number of optimizations on seeds

After seeds are procedurally generated, their energies are calculated and then the seeds are ordered. Then a geometry optimization is performed on the seeds. This gives SDGMS the best start possible. You probably want to limit the number of geometry optimizations as these are costly. This is done with the nopt keyword.

```
nopt x
```

Here x should be replaced with an integer, for the desired number of optimiations. As with the *keep* keyword, if you want to optimize all seeds put the value of x as -1.

The default value is -1.

3.11 Search parameters

Here parameters relevant to the search itself are explained. The search procedure is also explained so that the user may be informed when modifying the values. At first SDGMS will add all of the optimized seeds to a stack. The top value of the stack is popped and then b directions are explored from this geometry. In each of these directions, the geometry is perturbed by a direction vector. Depending on the input, this direction vector may displace multiple atoms or an atom in just a single x,y or z direction. Each displacement must have a total displacement size, d. At each new geometry, some filters are used to determine if this geometry is reasonable and unique. If the new geometry is accepted then a single point calculation is used to evaluate the geometry. This will continue until the energy decreases, indicating the local minimum well has been escaped. To enhance efficiency the user may want to cap the number of uphill steps, which can also be specified via an input parameter.

3.11.1 Step size

The step size of displacements is specified with the keyword stepsize as below:

```
stepsize 0.4
```

Here a stepsize of 0.4 angstroms is specified. This is also the default value.

3.11.2 uphill steps

The number of uphill steps is specifed with the keyword uphill:

```
uphill u
```

Replace u with an integer for the number of uphill steps. The default value is 20.

3.11.3 Branches

The number of branches, or directions that each local minima should be explored from is specified by the branches keyword as follows:

```
branches 35
```

Here 35 branches are specified. The default value is 6.

3.11.4 Direction vectors

The user may decide to displace a single atom for each step, or as many atoms as they desire. An alternative option exists to procedurally explore each direction vector (x,y,z, -x,-y,-z). This option is slower but is completely exhaustive of the search space given a small enough step size.

To specify n atom (or group of atoms) to be displaced the keyword nad should be used as follows:

```
nad n
```

To specify single directions be exhaustively explored in the alternative approach, they keyword sad should be used as follows:

```
sad
```

It is also possible to move atoms together as groups. This can be done using the nad keyword where n specifies the number of groups to be moved rather than atoms. Atoms can be combined into groups with the assignment keyword.

```
assignment 0 1 2 3
```

The preceeding useage of the assignment keyword assigns atoms of a 4 atomed molecule to their own groups. If the user wanted to group the first two atoms they would change the indeces as follows:

```
assignment 0 0 1 2
```

By default a unique index will be given to each atom. The indices must be given in order from 0 to n-1 or else errors may be encountered.

3.12 filtering criteria

Various filters are used to evaluate uniqueness of geometries such that the same geometry is not explored more than once, and also to ensure that a geometry is reasonable before quantum chemistry calculations.

Reasonability is determined by calculating the bond distances between each atom. SDGMS contains a list of expected covalent radii for each element.

$$d \geq (r_{e1} + r_{e2})(1 - c\%) \tag{5}$$

The following relationship is applied to each pair of atoms where r_{en} is the covalent radii of the element of atom n in the pair, d, is the interatomic distance between the two atoms, and $c\%$ is the covalent criteria percentage provided by the user as follows:

```
criteria c%
```

here c% is replaced by a double representing the desired covalent criteria percentage. This ensures that no set of two atoms are too close to eachother. Additionally the following relationship is used to determine if their is a bond between two atoms:

$$d \leq (r_{e1} + r_{e2})(1 + c\%) \tag{6}$$

All identified bonds are stored as an edge in a graph where each atom is a node. Finally it is checked that no atoms have been displaced from the molecule during search by ensuring each node is connected in the graph.

Additionally there is a uniqueness criteria, which is used to determine if two geometries are identical and therefore the new geometry should not be evaluated (as it has already been evaluated). The method used to determine the distance between two structures is detailed in the original publication, but it allows for the exact distance between two structures to be approximated in angstroms. The user provided uniqueness criteria is a cutoff for at what distance the structures should be identified as unique.

```
unique d
```

Here d should be replaced by a double value in angstroms. The default value is 30.

3.13 Continuing a previous calculation

Calculations can be continued from 3 separate points. The first point is after single point calculation of the seeds, the second is after geometry optimization of the seeds, and the third is at some point during search when it was cut short by the time limit.

To continue the search after single point calculation of the seeds add the following line to the input file:

```
seFile $energy_sorted_[filename]$
```

$energy_sorted_[filename]$ would be the name of the xyz file containing the energy sorted seeds from a previous calculation. This could be replaced by an .xyz file as long as the energy of each seed is written in the frame comment in the same way that SDGMS does.

```
oeFile $optimized_[filename]$
```

The above input specifies to continue after geometry optimization of seeds.

There are no default values.

3.14 random seed

In c++ a seed is used for random number generation. By default this will be the time that the calculation started, but this can be changed with the keyword rseed followed by an integer for the random seed.

The default value is 0. If 0 is given this will specify the time of the calculations start.

4 Python script interface

The interface python script must interface the SDGMS package with the chosen quantum chemistry package. SDGMS will run the interface script in three different ways depending on what is desired. The requirements for each useage case are presented below.

4.1 Optimization calculations

```
Python myfile.py O dirName
```

O specifies geometry optimization, dirName specifies the directory to perform the calculation in, relative to the directory python was executed from a file input.xyz will already be in the directory. this is the input geometry for the DFT calculator. the optimization input file for the ab initio calculator is also aready in the directory without any modifications.

The output must be a file "opt.xyz" in the directory. If the geometry optimization failed, the first line must be the string "-1". If the geometry optimization succeeded the file should contain the geometry in .xyz file format. The second line must contain the frame number, followed by 3 spaces, and then the energy of the geometry.

4.2 Single point calculations

```
Python myfile.py E dirName
```

E specifies an energy calculation. Requirements are the same as in input 1 except now the output must be named e.txt and contain the string "F" if the calculation failed. Otherwise the output should be a single line containing the energy.

4.3 Cancelling the calculation

```
Python myfile.py cancel
```

This will use the pkill command to shut down the ab initio calculator. For example when using gaussian the python file should submit the following command line input: "pkill g16"

4.4 Example script for interface with CP2K

Below is an example of the script interfacing with CP2K:

```
import subprocess
import sys
import time
import os
OFN = "cp2kO.inp"
EFN = "cp2kE.inp"

print("PYTHON: opened")

def CP2K(OFN,EFN,typ,CN,geomXYZ):
    print("Python: method")
    #OFN is optimization file, EFN is energy file,
    #geomXYZ is the geometry XYZ file name,
    #CN is calculation number
    com = 'mkdir ' + str(CN)
    subprocess.Popen(com,shell=True)
    runcom = ''
    if typ == "O":
        com = 'cp ' + OFN + ' ' + str(CN) + '/' + OFN
        subprocess.Popen(com, shell=True)
        runcom = 'cd ' + str(CN) + '\nnohup cp2k.psmp -i ' + OFN + ' -o cp2k.out &'
    elif typ == "E":
        com = 'cp ' + EFN + ' ' + str(CN) + '/' + EFN
        subprocess.Popen(com, shell=True)
        runcom = 'cd ' + str(CN) + '\nnohup cp2k.psmp -i ' + EFN + ' -o cp2k.out &'
    com = 'cp ' + geomXYZ + ' ' + str(CN) + '/' + geomXYZ
    subprocess.Popen(com, shell=True)
    #submit the command
    subprocess.Popen(runcom,shell=True)
    #wait for it to finish

    finished = False
    fail= False
    import time

    start_time = time.time()

    started = False
    while not started:
        if (time.time() - start_time) > 120:
            print("WARNING: cp2k did not start")
            started = True
            fail = True
        #for some reason this keeps triggering even though cp2kout is opened
        if os.path.exists(str(CN) + "/cp2k.out"):
            started = True
            fail = False
            print("PYTHON: cp2k started")
    if fail == False:
        while not finished:
            with open(str(CN) + "/cp2k.out", 'r') as file:
                for line in file:
                    if line[:41] == " The number of warnings for this run is :":
                        finished = True
                    elif "[ABORT]" in line:
                        finished = True
                        fail = True
                file.close()
    print("PYTHON: cp2k has finished")
    #the calculation is finished, analyze.
    with open( str(CN) + "/cp2k.out", 'r') as file:
        for line in file:
            if line[:47] == " ENERGY| Total FORCE_EVAL ( QS ) energy (a.u.):":
                energy = line[47:].strip()
            elif "[ABORT]" in line:
                fail = True
        file.close()
    if fail == True:
        if typ == "O":
            file = open(str(CN) + '/opt.xyz', 'w')
            try:
                file.write("-1")
            finally:
                file.close()
        elif typ == "E":
            file = open( str(CN) + '/e.txt', 'w')
            try:
                file.write("F")
            finally:
                file.close()
    else:
        if typ == "O":
            com = 'mv ' + str(CN) + '/*-pos-1.xyz ' + str(CN) + '/outxyz.xyz'
            subprocess.run(com,shell=True)

            with open( str(CN) + "/outxyz.xyz", 'r') as file:
                lines = []
                natoms = 0
                nl = False
                skipLine = False
                for line in file:
                    if natoms == 0:
                        natoms = line.strip()
                        nl = True
                    elif line.strip() == natoms:
                        lines = []
                        nl = True
                    else:
                        nl = False
                    if nl:
                        lines.append(natoms)
                    elif line[:4].strip() == "i =".strip():
                        lines.append(’Frame 0 ,Energy: ’ + energy + ’ , time: -1’)
                    else:
                        lines.append(line)
                file.close()

            file = open(str(CN) + ’/opt.xyz’, ’w’)
            try:
                for line in lines:
                    file.write(line.strip() + ’\n’)
            finally:
                file.close()
        elif typ == "E":
            file = open( str(CN) + ’/e.txt’, ’w’)
            try:
                file.write(energy)
            finally:
                file.close()

command = sys.argv[1]
if command == "cancel":
    com = ’pkill cp2k.psmp’
    subprocess.run(com,shell=True)
else:
    CP2K(OFN,EFN,sys.argv[1],sys.argv[2],"input.xyz")
```

This script can be used as a starting point for your own script. While creating a script for a calculator like VASP, the user may want to include extra features, such as copying the various VASP input files into the calc*n* folder themself.

This manual was written in 3 hours.

References

[1] MJEA Frisch. gaussian 09, revision d. 01, gaussian. *Inc, Wallingford CT*, 201, 2009.

[2] Trucks G. W. Schlegel H. B. Scuseria G. E. Robb M. A. Cheeseman J. R. Scalmani G. Barone V. Mennucci B. Petersson G. A. Nakatsuji H. Caricato M. Li X. Hratchian H. P. Izmaylov A. F. Bloino J. Zheng G. Sonnenberg J. L. Hada M. Ehara M. Toyota K. Fukuda R. Hasegawa J. Ishida M. Nakajima T. Honda Y. Kitao O. Nakai H. Vreven T. Montgomery J. A. Jr. Peralta J. E. Ogliaro F. Bearpark M. Heyd J. J. Brothers E. Kudin K. N. Staroverov V. N. Kobayashi R. Normand J. Raghavachari K. Rendell A. Burant J. C. Iyengar S. S. Tomasi J. Cossi M. Rega N. Millam M. J. Klene M. Knox J. E. Cross J. B. Bakken V. Adamo C. Jaramillo J. Gomperts R. Stratmann R. E. Yazyev O. Austin A. J. Cammi R. Pomelli C. Ochterski J. W. Martin R. L. Morokuma K. Zakrzewski V. G. Voth G. A. Salvador P. Dannenberg J. J. Dapprich S. Daniels A. D. Farkas Ö. Foresman J. B. Ortiz J. V. Cioslowski J. Fox D. J. Frisch, M. J. Gaussian 16.

[3] A.J. Atkins J. Autschbach O. Baseggio D. Bashford A. Bérces F.M. Bickelhaupt C. Bo P.M. Boerrigter C. Cappelli L. Cavallo C. Daul D.P. Chong D.V. Chulhai L. Deng R.M. Dickson J.M. Dieterich F. Egidi D.E. Ellis M. van Faassen L. Fan T.H. Fischer A. Förster C. Fonseca Guerra M. Franchini A. Ghysels A. Giammona S.J.A. van Gisbergen A. Goez A.W. Götz J.A. Groeneveld O.V. Gritsenko M. Grüning S. Gusarov F.E. Harris P. van den Hoek Z. Hu C.R. Jacob H. Jacobsen L. Jensen L. Joubert J.W. Kaminski G. van Kessel C. König F. Kootstra A. Kovalenko M.V. Krykunov P. Lafiosca E. van Lenthe D.A. McCormack M. Medves A. Michalak M. Mitoraj S.M. Morton J. Neugebauer V.P. Nicu L. Noodleman V.P. Osinga S. Patchkovskii M. Pavanello C.A. Peeples P.H.T. Philipsen D. Post C.C. Pye H. Ramanantoanina P. Ramos W. Ravenek M. Reimann J.I. Rodríguez P. Ros R. Rüger P.R.T. Schipper D. Schlüns H. van Schoot G. Schreckenbach J.S. Seldenthuis M. Seth J.G. Snijders M. Solà M. Stener M. Swart D. Swerhone V. Tognetti G. te Velde P. Vernooijs L. Versluis L. Visscher O. Visser F. Wang T.A. Wesolowski E.M. van Wezenbeek G. Wiesenekker S.K. Wolff T.K. Woo A.L. Yakovlev .J. Baerends, T. Ziegler. Adf 2024.1, scm, theoretical chemistry.

[4] W.L. Li J. Burkhardt. Structure search with the guaranteed escape algorithm. 2025.

[5] Thomas D. Kühne, Marcella Iannuzzi, Mauro Del Ben, Vladimir V. Rybkin, Patrick Seewald, Frederick Stein, Teodoro Laino, Rustam Z. Khaliullin, Ole Schütt, Florian Schiffmann, Dorothea Golze, Jan Wilhelm, Sergey Chulkov, Mohammad Hossein Bani-Hashemian, Valéry Weber, Urban Borštnik, Mathieu Taillefumier, Alice Shoshana Jakobovits, Alfio Lazzaro, Hans Pabst, Tiziano Müller, Robert Schade, Manuel Guidon, Samuel Andermatt, Nico Holmberg, Gregory K. Schenter, Anna Hehn, Augustin Bussy, Fabian Belleflamme, Gloria Tabacchi, Andreas Glöß, Michael Lass, Iain Bethune, Christopher J. Mundy, Christian Plessl, Matt Watkins, Joost VandeVondele, Matthias Krack, and Jürg Hutter. CP2k: An electronic structure and molecular dynamics software package - quickstep: Efficient

and accurate electronic structure calculations. *The Journal of Chemical Physics*, 152(19):194103, May 2020.

[6] G. te Velde, F. M. Bickelhaupt, E. J. Baerends, C. Fonseca Guerra, S. J. A. van Gisbergen, J. G. Snijders, and T. Ziegler. Chemistry with adf. *J. Comput. Chem.*, 22(9):931–967, 2001.