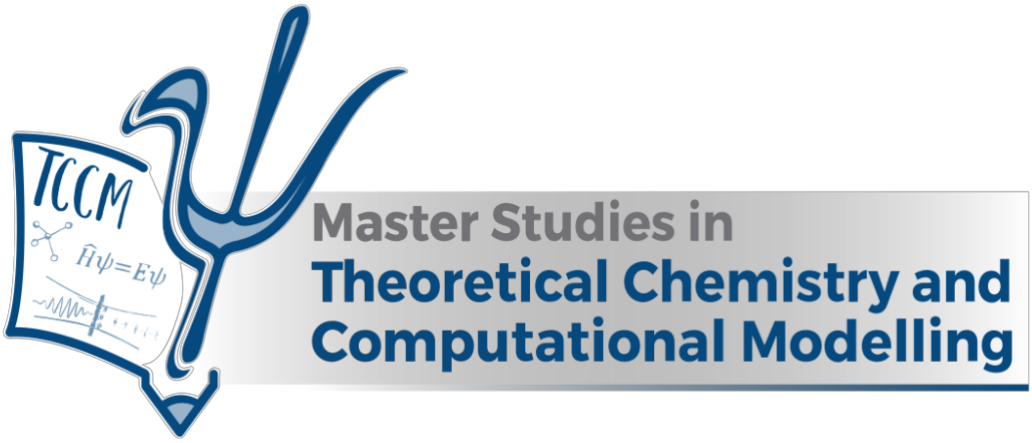
TCCM: Distance Learning Programming Project

September 2023

Jeremy HARVEY

Workbook and instructions



TCCM: Distance Learning Programming Project 2023

TUTOR PRESENTATION – JEREMY HARVEY

Work address: Department of Chemistry, KU Leuven, Celestijnenlaan 200F, B-3001 Leuven, Belgium
ORCID 0000-0002-1728-1596, jeremy.harvey@kuleuven.be; https://chem.kuleuven.be/en/research/qcpc/tcc

ACADEMIC QUALIFICATIONS

Doctorate in Sciences 1995, Supervisor: Heinz G. Viehe, Thesis title: *Syntheses and reactions involving thionium and sulfonium ions. The effect of electronegativity on the stability of substituted alkanes;* **Licence in Philosophy** 1994; **Licence in Chemistry** 1990. All at UC Louvain, Belgium.

PRESENT AND PREVIOUS APPOINTMENTS

- **Professor of Quantum Chemistry**, KU Leuven (Oct 2014 –)
- **Professor of Chemistry**, University of Bristol (2008 – 2014)
- **Temporary Lecturer, Lecturer, then Reader** in Chemistry, University of Bristol (1999 – 2008)
- **Postdoc**, Hebrew University of Jerusalem, and UC Irvine, with Prof. R. B. Gerber (1997 –1998).
- **Postdoc**, TU Berlin, with Prof. Helmut Schwarz (1995 –1997).

RESEARCH SUMMARY

After his PhD devoted to studying organic reaction mechanisms using experimental methods, Jeremy Harvey moved into computational chemistry as a postdoc and as a faculty member. His research focuses on the use of state-of-the-art quantum chemical methods to yield insight into various chemical problems, working often in collaboration with experimental colleagues. Most of his research uses existing quantum-chemical and molecular dynamics codes, though his work typically also involves some development of new methodology.

He is best known internationally for his research on spin-forbidden chemical reactions, combining accurate state-of-the-art quantum chemistry and advanced statistical rate theories to gain a detailed understanding of such reactions. Another major area of research concerns computational studies of catalysis and biocatalysis, including organometallic chemistry and metalloenzymes. A third notable area is the study of the dynamics of complex reactions, both in the gas phase and in solution, using the empirical valence bond (EVB) method in order to efficiently describe chemical reactivity.

Dr. Harvey has published almost 300 papers with over 14,000 citations excluding self-citations (h-index: 72). He is regularly invited to speak in international conferences, and has received several awards including the 2006 Corday-Morgan Medal of the UK Royal Society of Chemistry and the 2009 Dirac Medal of the World Association of Theoretical and Computational Chemists (WATOC), which is awarded once per year worldwide to the outstanding theoretical and computational chemist below age 40. In 2022, he was elected to membership of the International Academy of Quantum Molecular Science (IAQMS).

POSITIONS OF RESPONSIBILITY

JNH is or has been a member of several editorial advisory boards (*Chemical Science*, *Theoretical Chemistry Accounts*, *Journal of Coordination Chemistry*), a referee for many journals, and referee or panel member for grant funding bodies. He is or has been a member of external review committees for among others the Chemistry Department at the Ecole Normale Supérieure, Paris, the Max Planck Institut für Kohlenforschung, Müllheim, and the Institute of Organic Chemistry and Biochemistry, Prague. He has organised several conferences and has served as examiner for doctoral degrees in multiple European countries. Since August 2020 he is chair of the Department of Chemistry at KU Leuven.

SELECTED PUBLICATIONS

1) *Mechanism of the highly effective peptide bond hydrolysis by MOF-808 catalyst under biologically relevant conditions*, D. Conic, K. Pierloot, T. N. Parac-Vogt and J. N. Harvey, *Phys. Chem., Chem. Phys.*, 2020, **22**, 25136.
2) *Computational insight into the origin of unexpected contrast in chiral markers as revealed by STM*, A. Sanz-Matias, O. Ivasenko, Y. Fang, S. De Feyter, K. Tahara, Y. Tobe and J. N. Harvey, *Nanoscale*, 2018, **10**, 1680-1694.
3) *The Dynamics of the Reaction of FeO^+ and H_2: A Model for Inorganic Oxidation*, S. Essafi, D. P. Tew and J. N. Harvey, *Angew. Chem. Int. Ed.*, 2017, **56**, 5790 – 5794.
4) Vibrational *relaxation and micro-solvation of DF after F-atom reactions in polar solvents*, G. T. Dunning, D. R. Glowacki, T. J. Preston, S. J. Greaves, G. M. Greetham, I. P. Clark, M. Towrie, J. N. Harvey and A. J. Orr-Ewing, *Science*, 2015, **347**, 530 – 533.
5) *Assembly-line Synthesis of Organic Molecules with Tailored Shapes,* M. Burns, S. Essafi, J. R. Bame, S. P. Bull, M. P. Webster, S. Balieu, J. W. Dale, C. P. Butts, J. N. Harvey and V. K. Aggarwal, *Nature*, 2014, **513**, 183 – 188.
6) *Understanding the determinants of selectivity in drug metabolism through modeling of dextromethorphan oxidation by cytochrome P450,* J. Oláh, A. J. Mulholland and J. N. Harvey, *Proc. Natl. Acad. Sci. USA,* 2011, **108**, 6050 – 6055.

Distance Learning Module: Coding of the Semiempirical GFN1-xTB Method: TCCM

Jeremy Harvey, September 2023

Overall Goal. The aim of this course is to improve your skills for coding relevant for various computational chemistry problems by writing a code for performing semiempirical DFT calculations using the GFN1-xTB method. Unlike previous programming exercises, this involves a *medium-size* problem in which you write a whole stand-along program. Most computational chemistry coding instead involves contributing to a larger code base together with others, but it is useful to have built an entire code yourself on at least one occasion.

During the class, you will learn or reinforce your knowledge of a programming language (Python, Fortran, C, C++, or Julia, or…), and you will implement and test a code for performing (part of the) calculations for the GFN1-xTB method. For this exercise, the basic philosophy is that you should **write your own code from start to finish**, *i.e.* starting from nothing (**no copying** of even basic code from elsewhere – though see below the note on use of artificial intelligence), and that you should *progress* your skills. More experienced students should try to write more advanced code, but less experienced students can just write 'basic' code and still get a good grade.

The task this year is quite significant, and accordingly the detailed instructions include possible stages along the way: It is not necessary to perform all stages in order to pass the unit. All the required theory and all the key equations needed are provided as a document that is part of this manual. This document should be read with care, in particular because it also specifies more precisely exactly what the expected functionality of the code should be. Students who only accomplish part of the overall coding task should be able to pass the unit, perhaps with a slightly lower grade than those who complete the whole of task, but the emphasis is on learning new skills.

Working Method. This course is taught by distance learning, meaning you will be working on your own, with support through videoconferencing, an online exchange forum, and email. Like the other modules, the course is supposed to correspond to **6 ECTS**. One ECTS is supposed to correspond to something between 25 and 30 hours, so in principle, you can expect to invest **roughly** 150-180 hours on the course. This includes all reading, coding, videoconferencing, use of the forum, email. I know, this is a lot of time – roughly four full weeks of 40h/week or 20 days of 8h/day. Another way of measuring is that the total amount of work done should be about the same as for each of the other optional modules, or half the effort invested in the main Intensive Course. I suggest that it can be helpful to keep a **diary** listing the hours you have worked on the course, and a brief description of what you were doing in each case. In any case, I suggest you make a note every week of the progress you have made during that week. After an introductory videoconference, you will largely work on your own. However, I will be there to talk to you by videoconferencing if you hit problems that cannot be addressed in other ways. There will be occasional online code feedback/troubleshooting sessions.

I suggest that the first avenue for seeking help should be by asking questions addressed to the **Forum**, and all students are encouraged to participate actively by providing answers as well as questions in a constructive and supportive way. Be kind! Not everyone has the same knowledge, experience or insight. Also, be honest: there are no stupid questions. If you don't know or understand something, make an effort to find out or understand, but then if you still do not understand, ask a question – most likely the other students do not know either. I certainly will not give 'bad' grades to students who ask simple questions. Almost the opposite: **engagement with the forum** will be one (small) criterion for marking, meaning that non-participation will be considered a weak point. However, *quality* of contributions is much more important than *quantity*, and *respect*

for others is also important: you should not aim to answer every question, leaving spaces for others is important also. A few thoughtful and useful contributions is much better than a flood.

Programming language and computing framework. The code can *a priori* be written in any programming language of your choice. However it needs to be rapidly testable on my computer, so in practice I recommend using one of either Python, Fortran (90 or later), C, C++ or Julia. You may use another language if you really want to, but you should first consult with me. In case you use C++ you should avoid the use of unnecessary libraries, especially those with multiple dependencies that might be difficult to install. Again here, if in doubt, check with me. With Fortran or C you will typically not need to use any non-standard functions.
You should program, compile and test code on machines available to you locally. It is anticipated that most development can be done on a laptop, but you are encouraged to test on at least one other machine so as to ensure that your code is not dependent on some particular feature of your local environment. We are not seeking to develop a very high-performance code, so there is no need for access to very high-performance clusters. There is no need for parallelization. The calculations for the molecules provided take only a few seconds at most.

Personal work. The course relies in part on an honour code: the work you present should be **yours**, so you should be able to **fully explain every single line of code**. Of course, you will end up making extensive use of resources such as Stack Exchange (or even ChatGPT, see below). You may also end up getting suggestions from other students and tutors. This is normal and completely accepted. However, the code should ultimately be **yours**: you should be able to **fully explain all of it**. Note that in case more than one student from the same university takes the course, you should **not** turn this into a group project so your codes should end up being different (of course you can talk to each other). Too great similarity between the codes of students (whether in the same university or not) will be considered a negative point. Finally, the code should recognizably be based on the notes provided, so it should be clear how each equation leads to a line of code.

Artifical intelligence. By now everyone has heard of ChatGPT and its many relatives. These can also be used to help design computer code. I am not fundamentally opposed to you using these tools to help with this project. However, you **must still understand the code produced**, and it must complete the task as I have set it. Use with care!!

Assessment. At the end of the course, **your code should be handed in (= emailed to me, or a link to github)**: a code for GFN1-xTB calculations on arbitrary closed-shell molecules containing H, C, N and/or O atoms. I will test the code using a new .xyz file for a different molecule than those provided – it will have the same format as the examples (coordinates in Å). In case the whole task is not completed, the assessment can instead be done for code that carries out part of the calculation. Students should keep me informed if they feel they are not going to complete the full task. The course will be assessed based on the work during the working period, and on the code. You will be asked to **explain the code in a one-to-one video call** with me, at the end of the course. As well as from the forum and the coordinator and instructors, you may seek support from others and from your tutors. But remember: *every single line of code should be written by you and fully understood*. The work in the course will receive a single grade expressed as a score out of 10, with 10 representing a perfect piece of work (quite rare), 9 being an excellent grade, 8 very good, 7 good, 6 acceptable, and 5 is the lowest pass grade. The mark will be awarded based on (a) engagement with the forum: did you participate with high-quality (not quantity!!) questions and answers? Were you respectful to other students? Did you demonstrate learning?; (b) mainly: the program produced: does it do what is asked? Can you **fully** explain it?

Timeframe. Given the lack of classroom sessions, I can in principle be flexible about the timeframe. However, the basic rule is: you should try to finish the module by the end of December 2023. You are **very** strongly encouraged to finish by the end of January 2024. Any student wishing to finish later should contact me and explain their planned timetable. The aim is that you fit the work on this module around work you do for other modules, and work on your thesis.

TCCM: Distance Learning Unit
Notes on Programming Languages

Introduction

In this course, one of the main outcomes is writing some of your own code, in Python, Fortran, C++ and/or another language, perhaps Julia. The intention is to build a code starting from zero, so you also need to conceive the architecture of the code yourself. These notes provide a very short summary concerning programming in Python, Fortran, C++.

Programming Language

There are many different programming languages. These are of variable levels of usefulness for carrying out numerical computations relevant to chemistry. One of the most commonly used languages nowadays is Python. The versatility and ease of use of this language make it very attractive. For many programming tasks where efficiency is not critical, it is an excellent choice: modern computers are so fast that there are a huge number of problems which are too demanding to be solved numerically using paper and pencil or semi-automated techniques like spreadsheets, but which upon suitable coding become trivial to deal with using even very modest computer resources. Hence Python is probably the most commonly used language in scientific programming nowadays. However, in its native form it is rather inefficient for heavy numerical computations. Rather standard libraries such as NumPy, however, can be used to carry out many standard tasks in a more efficient way, so provided you use these, it is possible to write efficient code.

Two other programming languages have been used very heavily in scientific computing and quantum chemistry: Fortran and C++. These are compiled languages, where a text **source code** is written, then **compiled** to yield a (binary) executable code. Provided the algorithm of interest is implemented in a reasonable way, Fortran or C++ yield quite efficient code. There is a lot of knowledge about how to obtain very efficient code, but this is not the focus of the present exercise.

You should pick a language for your coding yourself. My suggestion is one of Python (**with** NumPy) C, C++, Fortran (in its modern dialects Fortran 90 or later, **not** Fortran 77), or Julia (see https://julialang.org/). You can also pick another language if you wish, but please in that case consult with me first. There are **many** tutorials available for programming in different languages, available online. I will suggest a procedure for learning to program in a separate document.

Linear Algebra and other routines

When carrying out quantum-chemical calculations as here, there are a number of numerical tasks that need to be performed which are not always standard in all programming languages. First, you need to evaluate a number of standard functions such as exp. These are usually available, but you may need to include libraries in some cases.

Another frequently required property is the ability to multiply matrices, and to diagonalize them (obtain eigenvalues and eigenvectors). Again, in some languages, this can be done directly or by including standard libraries. For example, matrix diagonalization can be done in Julia with the LinearAlgebra library. Python and in particular NumPy has an extensive 'linalg' linear algebra library. In Fortran90, matrix multiplication can be done in standard Fortran, but diagonalisation is not directly available. However, the dsyev routine from the LAPACK library can be called. On many systems, LAPACK routines are available with appropriate compiler flags – on a Mac laptop, the "-framework accelerate" compiler flag signals that LAPACK libraries are needed. On other systems, model Fortran code for dsyev can be obtained from the LAPACK website and simply included in the code.

Distance Learning Module
TCCM, J Harvey, September 2023

Some Tips on Programming

A medium-sized programming project like the one involved in this course can seem a bit overwhelming at first sight, since you need to learn the programming language, to understand the problem, *and* design then write a program to carry out the calculations. Fortunately, the task is not *too* large – my own code, that has all the required functionality *and* produces all the debugging output, has less than 1000 lines of Fortran code. Still, this is not nothing, and some careful strategy is needed in order to approach the task in the best way (or at least in a reasonably good way – there are many good approaches, but there are even more bad ones). Here I give some tips that I find helpful.

1. **Small Steps**
   The best way to keep up momentum and insight for a programming project, especially at the beginning, is to make *small steps*. Write your first code – something like "Hello, World", compile it, then check that it works. If it does, great!! Otherwise see points 7 and 8, below. Once it works, think about the next step. You might try to write a code that reads in two numbers and prints out their sum. Write that, compile it, check it. Then maybe write a code that reads in (x,y,z) coordinates for two points in space, calculates the distance between them, then writes it out. Compile that, check it. Then you are ready to start work on the actual problem, but do that in parts also!! Start with just the E_{rep} (eq. 9) and move *slowly* onwards. With each small task, which could **take less than an hour**, you can build up your confidence and your skills. If you instead set yourself a big goal, you may find it hard to work out how to start, and end up losing a lot of time. My very rough rule of thumb is that when starting out you should have a successful compile-test step at least **every two hours or so**. If you spend more than this between successful compile-test steps, you are trying to make too big steps.

2. **Keep your work**
   I recommend frequently storing examples of successful code. This can be formalized using tools such as github, but at this stage, you can do it almost as well by just making frequent copies of your code (every time something works, keep it, and make subsequent changes in a new copy of the source file), making frequent new directories, giving source code files informative names, and *not throwing files away or overwriting them*.

3. **Plan**
   For each piece of code, write down (or think through carefully) what you want it to do, then sketch (in your head or on paper) the steps it needs to take. This can be formalized by writing down a logic flowchart for your algorithm (see https://en.wikipedia.org/wiki/Flowchart) but equally you can just keep it simple as a set of bullet points or even a mental plan. Another thing to think about is to list the variables and arrays that you will need to use in your code.

4. **Modularize**
   Remember that the computer is there to help you. If you find yourself typing the same thing multiple times, consider that perhaps you should write a loop, a function, or a subroutine, so as to streamline the algorithm. Code that is a bit modular in this way will be easier to read by others (and yourself). It may also be more efficient.

5. **Keep it simple**
   It is very tempting to go overboard in terms of modularizing, making everything very beautiful in terms of coding technique. But this course has a fairly modest goal, and in practice, it is unlikely that the code you write is going to be used and extended by other users and programmers – so remember that you do not need to do things perfectly. For example, the code needs some flexibility (e.g. it should be able to work for different molecules), but it does not require flexibility in every respect. Each year, in this module, some students make the mistake of neglecting this principle and end up investing way too much time in the course – remember, there are other courses to follow, the masters thesis to do, and so on…

6. **Learn the basics**
   Every programming language provides tools to do a number of basic things. Make sure you have learned about these concepts in tutorials or books: variables, types and declarations; arrays; loops; flow control ("if" statements); functions and subroutines, including *intrinsic* functions. If you use a compiled language, you may also want to learn about compiler flags and code optimization, and about the use of makefiles.

7. **Embrace debugging**
   Debugging is an essential part of coding and has been ever since the early days (see https://www.computerhistory.org/tdih/september/9/). In fact, despite all the accumulated expertise related to programming techniques, in many (or most!) projects, more time will be spent on debugging code than on writing new code. So don't get frustrated if your code does not work the first time, and instead learn to enjoy the challenge! There are many ways to debug, but they all rely on the same basic techniques: looking at what has changed since your code last worked, checking that the code actually does what you intend it to do (it can be very instructive simply to check that variables contain the things that you think they should, for which purpose you can add lots of 'print' statements that output variables at key points in the code), and that it flows in the way you think it should (a print statement saying "I just entered function xxxxx" can be really revealing, if it turns out that your code never actually does enter that function….). Some people prefer to use IDEs (https://en.wikipedia.org/wiki/Integrated_development_environment) for code development; these do indeed streamline some of the debugging procedure, so they can be quite helpful (though for a medium project like this one, my view is that they are not *necessary*). If you cannot solve a bug after some period of time, consider that you may have done something conceptually wrong, so look back at tip 3 on planning.

8. **Ask for help, and give it.**
   If you don't know how to solve a given problem, ask! It could be a fellow-student, or a professor, or the forum. You can in principle post anonymously on the forum, but there is little point because the aim of the forum is not to penalize ignorance at the start of the course, but instead to help everyone make progress. Also, if someone gives outstanding advice to you, don't hesitate to also put it on the forum and thereby share it with others.

Writing a Code to Perform Semi-empirical DFT Calculations

Jeremy Harvey

October 11, 2023

Abstract

This document lays out the basic background needed to construct a program that performs semi-empirical Density Functional Theory (DFT) calculations for molecular systems. All the basic equations needed to perform the corresponding coding are provided for the specific type of semi-empirical method chosen, the so-called 'GFN1-xTB' method (the eXtended Tight-Binding method parameterized to describe Geometries, vibrational Frequencies, and Non-covalent interactions, in its first parametrization). The aim of this document is to briefly introduce the method, and to provide a short but fully self-contained set of equations and parameters as a reference for writing a working code, applicable to closed-shell neutral molecules containing only H, C, N and O atoms. For more details on the method, see the provided references.

1 Introduction

The development of modern quantum-chemical methods based on DFT or on correlated *ab initio* quantum-chemical methods, together with the availability of ever more powerful computers, has led to a golden age of computational quantum chemistry, with the accurate prediction of properties for molecular systems containing many hundreds of atoms, or for periodic systems with repeating units containing a similar number of atoms, seeming to have become almost trivial. However, standard Kohn-Sham DFT and correlated *ab initio* methods do remain computationally demanding, and in cases where many calculations of the energy are needed, such as when exploring large libraries of molecules or when performing dynamics simulations, it remains highly advantageous to be able to use more computationally efficient methods. These typically arise as approximations to DFT or *ab initio* methods through a process of simplification and parameterization, with parameters chosen so as to reproduce as closely as possible a range of target properties for a range of target compound types, as derived from either experiment or accurate computation. Early examples of such methods used parametrization to experiment, and hence these types of approach are referred to as 'semi-empirical' methods. Modern semi-empirical methods such as GFN1-xTB are quite accurate and computationally efficient, but also quite complicated, with many components to the energy function and many parameters, making implementation not completely trivial. The present self-contained notes should nevertheless make it feasible to implement part of or the whole of the GFN1-xTB method for simple closed-shell molecules.

2 Kohn-Sham Density Functional Theory

The GFNn-xTB methods are derived from DFT. In Kohn-Sham DFT, the ground-state electronic energy (in atomic units) is given as a functional $E[\rho]$ of the density ρ, and the density is itself expressed in terms of a set of n spatial 'Kohn-Sham' orbitals ψ_i^{KS}:

$$\rho(\mathbf{r}) = \sum_{i=1}^{n} n_i \left(\psi_i^{\mathrm{KS}}(\mathbf{r})\right)^2 \mathrm{d}\mathbf{r} \tag{1}$$

In this expression, n_i is the occupation of Kohn-Sham orbital ψ_i (usually 0, 1 or 2).

The energy is given by the following expression, where $E^H[\rho]$ is referred to as the 'Hartree energy' corresponding to the density:

$$\begin{aligned} E[\rho] &= \sum_{\mathrm{A}=1}^{N_N} \sum_{\mathrm{B}>\mathrm{A}}^{N_N} \int \frac{Z_\mathrm{A} Z_\mathrm{B}}{r_\mathrm{AB}} + \sum_{\mathrm{A}=1}^{N_N} \int \frac{-Z_\mathrm{A}}{r_\mathrm{A}} \rho(\mathbf{r})\, \mathrm{d}\mathbf{r} \\ &+ \sum_{k=1}^{n} \left\langle \psi_k^{\mathrm{KS}}(\mathbf{r}) \right| \frac{-\nabla^2}{2} \left| \psi_k^{\mathrm{KS}}(\mathbf{r}) \right\rangle + \frac{1}{2} \iint \frac{\rho(\mathbf{r}_1)\rho(\mathbf{r}_2)}{r_{12}}\, \mathrm{d}\mathbf{r}_1\, \mathrm{d}\mathbf{r}_2 + E_{xc}[\rho] \\ &= E_{NN} + E^H[\rho] + E_{xc}[\rho] \end{aligned} \tag{2}$$

In the above equation, N_N is the number of atoms in the system, Z_A and Z_B are the charges (in atomic units) of the nuclei of the A'th and B'th atoms, r_AB is the distance between the nuclei of atoms A and B, E_{NN} is the overall nucleus-nucleus repulsion energy, r_A is the distance between the point in space corresponding to vector $\mathbf{r}$ and the position of the A'th nucleus, r_{12} is the distance between the points $\mathbf{r}_1$ and $\mathbf{r}_2$, and $E_{xc}[\rho]$ is the exchange-correlation functional. Most exchange-correlation functionals can be computed as an integral over space of the exchange-correlation potential ν_{xc}, $E_{xc}[\rho] = \int \rho(\mathbf{r})\nu_{xc}(\mathbf{r})\, \mathrm{d}\mathbf{r}$, where ν_{xc} is a local function of the density (and perhaps of its derivatives). Some advanced functionals that attempt to capture London dispersion interactions instead express $E_{xc}[\rho]$ in terms of a semi-local exchange correlation potential $\nu_{xc}^{NL}(\mathbf{r}, \mathbf{r}')$.

Setting the requirement that the above energy must be a minimum with respect to variations of the orbitals, while also requiring the orbitals to be orthonormal, leads to the Kohn-Sham equations, a set of pseudo-one-electron equations whose solutions are the Kohn-Sham orbitals:

$$\hat{h}^{\mathrm{KS}} \psi_k^{\mathrm{KS}}(\mathbf{r}) = \epsilon_k^{\mathrm{KS}} \psi_k^{\mathrm{KS}}(\mathbf{r}) \tag{3}$$

In this expression, $\hat{h}^{\mathrm{KS}}$ is the Kohn-Sham operator, that includes terms relating to the kinetic energy, the Coulomb energy associated with electron-nuclei interactions, the Coulomb energy associated with mean electron-electron interaction, and the exchange-correlation potential. The Kohn-Sham operator depends on the orbitals, so that the equation needs to be solved self-consistently, with an initial guess of the orbitals being used to construct a first guess of the Kohn-Sham operator, which in turn yields improved orbitals, and so on.

For practical solution of these equations, the molecular orbitals ψ_k are expanded as linear combinations of atomic orbitals ϕ_μ:

$$\psi_k(\mathbf{r}) = \sum_{\mu=1}^{n_{basis}} c_{\mu k} \phi_\mu(\mathbf{r}) \tag{4}$$

To solve the Kohn-Sham equations 3, they are multiplied to the left by an arbitrary basis function ϕ_ν, and integration over $\mathbf{r}$ is performed, changing the equations 3 into a set of linear equations with unknowns $c_{\mu k}$, which can be written in matrix form as:

$$\mathbf{H}^{\mathrm{KS}} \mathbf{C} = \mathbf{S} \mathbf{C} \boldsymbol{\epsilon} \tag{5}$$

Where $\mathbf{H}^{\mathrm{KS}}$ is a matrix of integrals $H_{\mu\nu}^{\mathrm{KS}} = \langle \phi_\mu | \hat{h}^{\mathrm{KS}} | \phi_\nu \rangle$, $\mathbf{C}$ is the matrix of coefficients from eq. 4, $\mathbf{S}$ is a matrix of overlap integrals $S_{\mu\nu} = \langle \phi_\mu | \phi_\nu \rangle$ between the basis functions, and $\boldsymbol{\epsilon}$ is a diagonal matrix with Kohn-Sham orbital energies. Iterative solution of the Kohn-Sham equations within a basis set requires computing the overlap integrals, making an initial guess of the coefficients $\mathbf{C}$, using this to construct a first guess for the Kohn-Sham operator and hence obtain estimates of the Kohn-Sham matrix elements $H_{\mu\nu}^{\mathrm{KS}}$, solving matrix equation 5 to obtain a first estimate of the orbital energies $\boldsymbol{\epsilon}$ and a refined set of orbital coefficients $\mathbf{C}$, and then iterating using the new values for $\mathbf{C}$ until convergence is reached.

3 Tight-Binding Theory

There are many variants of tight-binding theory. These notes specifically describe the GFN1-xTB form of tight-binding. Like many tight-binding methods, it sets out to approximate the energy expression

of eq. 2. One key step in this approximation is to write the energy expression and the density in terms of a 'reference' density ρ_0 (which will be a superposition of neutral spherical atomic densities) and a density difference $\Delta\rho$ (confined to the valence electrons, with the core electron density assumed not to vary), where:

$$\rho(\mathbf{r}) = \rho_0(\mathbf{r}) + \Delta\rho(\mathbf{r}) \tag{6}$$

With this expression for the density, we can re-write the equation for the energy, eq. 2, as a formal Taylor expansion with respect to $\Delta\rho$ around ρ_0, as follows:

$$E[\rho] = E^{(0)}[\rho_0] + E^{(1)}[\rho_0, \Delta\rho] + E^{(2)}[\rho_0, (\Delta\rho)^2] + E^{(3)}[\rho_0, (\Delta\rho)^3] + \ldots \tag{7}$$

The GFNn family of tight-binding methods contain terms up to third-order in this expansion. We now describe each of the terms, and their mapping onto the different energy contributions from eq. 2. Note that in the GFNn-xTB family of methods, as in all semi-empirical methods, there are a large number of fitted parameters, which means that the 'physical' interpretation of the different energy terms should be taken with a grain of salt: in the end, the fitting ensures that the overall energy for each system used to parameterize the model agrees as well as possible with the benchmark data used to fit the method. The individual terms do not need to match exactly the part of the energy that they are named after, and hence the components of the energy may not match those obtained by accurate *ab initio* methods, in those cases where such reference values are available. Nevertheless, the functional form of the different terms needs to be carefully chosen in order to obtain a satisfactory fit, and hence the choice of the terms included (as well as the fitting) is the key factor that determines the accuracy of a given semi-empirical method. This document does not set out to justify the choices made for GFN1-xTB nor to compare it to other methods.

4 The GFN1-xTB Model

The implementation of the above formalism in GFN1-xTB is described as follows, in terms of the different order of energy contributions. The coupling of the first-, second- and third-order terms through a Fock-matrix formalism is also described. In each section, all the parameters needed to implement the method are given.

4.1 The Zero-th order Energy

The zero-th order energy $E^{(0)}[\rho_0]$ is given as the Kohn-Sham energy of the reference density. This reference density is formed by superimposing neutral, spherically symmetric densities for each of the atoms making up the system. When all atoms are infinitely separate from each other, this leaves only 'intra-atomic' energy contributions $E_A[\rho_0^A]$ for each atom, expressed as in eq. 2. These atomic energies do not need to be explicitly evaluated, because the choice is made to express the zeroth-order energy for a given non-separated set of positions of the atoms relative to the energy in the separated-atom limit. So the zero-th order energy is zero for uncharged systems when the atoms are all infinitely far from each other. As the atoms are neutral and spherical, and the density does not change when atoms move, all the energy terms corresponding to E^H in eq. 2 are zero for all values of the coordinates – there are no electrostatic interactions. However, E_{xc} is not zero: the electron clouds of the different atoms overlap, and this leads to changes in the exchange-correlation energy with respect to the separated limit. In the GFNn-xTB methods, this change is broken up into two contributions: an exchange-repulsion term E_{rep} (due to the Pauli repulsion arising when electrons occupy the same volume in space) and a dispersion-like attractive term $E_{\text{disp}}^{\text{D3}}$:

$$E^{(0)}[\rho_0](\mathbf{r}) - E^{(0)}[\rho_0](\mathbf{r} = \infty) = E_{\text{rep}}(\mathbf{r}) + E_{\text{disp}}^{\text{D3}}(\mathbf{r}) \tag{8}$$

In this equation, we have included explicitly the energy for the separated atoms, $E^{(0)}[\rho](\mathbf{r} = \infty)$, which could be included as an atom-specific parameter based on experiment or accurate computation. As the zeroth-order energy is computed in all cases relative to the zeroth-order energy of the separated atoms, this term can be neglected, and it will henceforth be omitted from all energy expressions.

Both $E_{\text{rep}}(\mathbf{r})$ and $E_{\text{disp}}^{\text{D3}}(\mathbf{r})$ are given as analytical expressions over the coordinates of the atoms, with a number of atom-specific parameters.

4.2 The Zero-th Order Repulsion Energy

The expression for $E_{\text{rep}}(\mathbf{r})$ is relatively simple, written as a sum over all atom pairs of terms depending on the corresponding atom-atom distances r_{AB}:

$$E_{\text{rep}}(\mathbf{r}) = \sum_{\text{A}=1}^{N_N} \sum_{\text{A}>\text{B}}^{N_N} \frac{Z_{\text{A}}^{\text{eff}} Z_{\text{B}}^{\text{eff}}}{r_{\text{AB}}} \exp\left(-\sqrt{\alpha_{\text{A}}\alpha_{\text{B}}}\,(r_{\text{AB}})^{k_f}\right) \tag{9}$$

In this expression, there are two types of atom-specific parameters, the 'effective charges' $Z_{\text{A}}^{\text{eff}}$, respectively equal to 1.116244, 4.428763, 5.498808, and 5.171786 atomic units for H, C, N and O, and the scaling terms α_{A}, which are respectively 2.209700, 1.281954, 1.727773, and 2.004253 bohr^{-1} for H, C, N and O. Based on the functional form, one might think that this energy term corresponds to a Coulombic repulsion term (inverse dependence on r_{AB}), but in fact the physical origin of this term is exchange-repulsion linked to the Pauli principle, which like Coulombic repulsion increases at short internuclear distance. Note also the presence of the exponential term that rapidly makes the term vanish for larger interatomic distances. The exponent k_f in the exponential term is fixed to a value of 1.5 for all atom types in the GFN1-xTB approach.

4.3 The Zero-th Order Dispersion Energy

The dispersion term $E_{\text{disp}}^{\text{D3}}(\mathbf{r})$ is also a sum over atom pairs, which is obtained from the dispersion correction term for DFT suggested by the Grimme group, using the 'damping term' suggested by Becke and Johnson, and leaving out the three-body term. It provides an overall attractive energy contribution depending on the inverse sixth and eighth powers of the interatomic distance:

$$E_{\text{disp}}^{\text{D3}}(\mathbf{r}) = \sum_{\text{A}<\text{B}}^{N_N} \sum_{n=6,8}^{N_N} \frac{-s_n C_n^{\text{AB}}}{r_{\text{AB}}^n} f_{\text{damp,BJ}}^{(n)}(r_{\text{AB}}) \tag{10}$$

The overall scaling factors s_6 and s_8 adopt the values $s_6 = 1$ and $s_8 = 2.4$. The C_6 and C_8 coefficients for each pair of atoms have a complicated dependence on the nature of A and B, and on the structure. The dependence on structure arises because the polarizability of an atom is taken as being a function of the bonding environment of the atom, and hence of the 'coordination number' of the involved atom(s), CN^{A} and CN^{B}. This dependence, and the expression for the coordination numbers themselves, are given below. First, however, the link between C_6 and C_8 for a given atom pair AB is given. The C_8 coefficient is simply a multiple of the C_6 coefficient, as shown below. The parameters Q_{A} and Q_{B} are atomic parameters that are related to the nuclear charge and to the spatial extent of the atomic electron density. For H, C, N, and O, these parameters are respectively equal to 4.029450, 9.640579, 7.353601 and 6.726848.

$$C_8^{\text{AB}} = 3C_6^{\text{AB}}\sqrt{Q_{\text{A}}Q_{\text{B}}} \tag{11}$$

The term $f_{\text{damp,BJ}}^{(n)}(r_{\text{AB}})$ in eq. 10 is a so-called 'damping function'. Due to the inverse power dependence on r_{AB}, the dispersion energy would go to ∞ for $r_{\text{AB}} \to 0$ in the absence of this term. To moderate this, the damping function tends very rapidly towards zero for small r_{AB}. The shape of this function used in XTB is that suggested by Becke and Johnson, and is of the form:

$$f_{\text{damp,BJ}}^{(n)}(r_{\text{AB}}) = \frac{(r_{\text{AB}})^n}{(r_{\text{AB}})^n + (a_1 r_{\text{AB},0} + a_2)^n} \tag{12}$$

The parameters $a_1 = 0.63$ and $a_2 = 5.0$ are fixed parameters for all atom pairs. The 'minimal radius' or damping constant $r_{\text{AB},0}$ is obtained as follows:

$$r_{\text{AB},0} = \sqrt{\frac{C_8^{\text{AB}}}{C_6^{\text{AB}}}} = (9Q_{\text{A}}Q_{\text{B}})^{1/4} \tag{13}$$

The C_6 coefficients are complicated functions of molecular structure, based on taking weighted average of coefficients applicable to given atom pairs in different environments, weighted based on the similarity of the structure being treated (as measured by atomic coordination numbers) to a set of reference structures. The first step in computing C_6 for a given structure and a given pair of atoms is to work out the 'coordination numbers' of the involved atoms. These are designed to correspond roughly to the usual chemical interpretation of a coordination number, but vary smoothly with structure. The definition of the coordination number for a given atom A is as follows:

$$\mathrm{CN}^{\mathrm{A}} = \sum_{\mathrm{B}\neq\mathrm{A}}^{N_N} \left[1 + \exp\left(-k_{\mathrm{CN}} \left\{ \frac{R^{\mathrm{disp}}_{\mathrm{A,cov}} + R^{\mathrm{disp}}_{\mathrm{B,cov}}}{r_{\mathrm{AB}}} - 1 \right\} \right) \right]^{-1} \tag{14}$$

Calculating this term requires a set of atom-specific parameters $R^{\mathrm{disp}}_{\mathrm{A,cov}}$ (scaled covalent radii), which are given in terms of *unscaled* covalent radii with values $R^{\mathrm{disp}}_{\mathrm{H,cov,unsc}} = 0.32$, $R^{\mathrm{disp}}_{\mathrm{C,cov,unsc}} = 0.75$, $R^{\mathrm{disp}}_{\mathrm{N,cov,unsc}} = 0.71$ and $R^{\mathrm{disp}}_{\mathrm{O,cov,unsc}} = 0.63$, all in Å, with $R^{\mathrm{disp}}_{\mathrm{A,cov}} = 4/3 R^{\mathrm{disp}}_{\mathrm{A,cov,unsc}}$, and the constant k_{CN} which is set to be equal to 16.

Once the coordination numbers for all atoms are determined, the pairwise coefficients C_6^{AB} from eq. 10 are given as a weighted average with weights L_{ij} of a set of element-pairwise reference values $C^{\mathrm{AB}}_{6,\mathrm{ref},ij}$, that serve as parameters. The weighted average is given by:

$$C_6^{\mathrm{AB}}(\mathrm{CN}^{\mathrm{A}}, \mathrm{CN}^{\mathrm{B}}) = \frac{\sum_{i=1}^{N_{\mathrm{A}}} \sum_{j=1}^{N_{\mathrm{B}}} C^{\mathrm{AB}}_{6,\mathrm{ref},ij}(\mathrm{CN}^{\mathrm{A}}_i, \mathrm{CN}^{\mathrm{B}}_j) L_{ij}}{\sum_{i=1}^{N_{\mathrm{A}}} \sum_{j=1}^{N_{\mathrm{B}}} L_{ij}} \tag{15}$$

The weights are products of the weights for each of the atoms; in turn, these are Gaussian functions of the 'distance' between the actual coordination numbers of the atoms, CN^{A} and CN^{B}, and the corresponding reference coordination numbers $\mathrm{CN}^{\mathrm{A}}_i$ and $\mathrm{CN}^{\mathrm{B}}_j$. The parameter k_L is equal to 4.

$$L_{ij} = L_i \times L_j = \exp\left(-k_L \left[\mathrm{CN}^{\mathrm{A}} - \mathrm{CN}^{\mathrm{A}}_i \right]^2 \right) \times \exp\left(-k_L \left[\mathrm{CN}^{\mathrm{B}} - \mathrm{CN}^{\mathrm{B}}_j \right]^2 \right) \tag{16}$$

When parameterizing the dispersion term, for each atom, a certain number of 'reference' coordination numbers were chosen, and then the C_6 reference coefficients were calculated for each pair of atoms and for each pair of reference coordination numbers using a quantum-chemical method. This process leads to a very large database of reference coefficients $C^{\mathrm{AB}}_{6,\mathrm{ref},ij}$ which can be obtained from the Grimme website. For 94 elements, there are $94 \times (94+1)/2 = 4465$ pairs of elements, and each element has up to $N_A^{\mathrm{max}} = 5$ different reference coordination numbers CN^A_i, so there are up to $4465 \times 5 \times 5 = 22325$ values of $C^{\mathrm{AB}}_{6,\mathrm{ref}}$ in the database. For the present purposes, as we are only going to write a code for treating compounds with H, C, N and O atoms, there are just 10 sorts of atom pairs (H,H), (H,C), (H,N), (H,O), (C,C), (C,N), (C,O), (N,N), (N,O) and (O,O), and hence there are fewer values that are needed. Still, there are too many values to list here – the parameters are provided separately, in the file `parameters.dat`.

In principle, there are contributions to the dispersion energy from higher-order terms: changes in electron population on the atoms lead to changes in their polarizability, which should also impact on the C_6 and C_8 coefficients. Some of these changes are approximately described through the dependence on the coordination number, and all other such changes are neglected.

4.4 The First-Order Energy

The first-order energy term arises due to the small shifts in electron populations that arise from electron sharing, i.e. polar covalent bonding. The charge shifts are described in terms of a set of molecular orbitals ψ_i, which we will assume to be either doubly occupied or empty, as we will only consider closed-shell systems with an even number of electrons n occupying $n/2$ orbitals. The orbitals will be represented in terms of linear combinations of atomic basis functions, using a modest-sized basis set with s and p functions, see eq. 17. The basis functions belong to *shells*: a single s basis function forms a shell on its own; the set of three p_x, p_y, and p_z functions, with the same spatial

dependence although along different axes, collectively form one shell. As will be seen below, some terms in the energy expression are calculated based on which shell a basis function 'belongs' to.

$$\psi_i(\mathbf{r}) = \sum_{\mu=1}^{n_b} c_{\mu i}\phi_\mu(\mathbf{r}) \tag{17}$$

Depending on the shape of each of the molecular orbitals, a larger or smaller electron density on each atom and for each atomic orbital will arise. These changes in population of the different atomic orbitals lead to partial charges on the atoms (described using Mulliken populations), and hence to a change in energy. This change in energy is calculated using an extended Hückel-type Hamiltonian ($\hat{H}_0$), details of which will be given below, and gives the energy $E^{(1)}_{\mathrm{EHT}}$:

$$E^{(1)}_{\mathrm{EHT}} = \sum_{i=1}^{n/2} 2\,\langle\psi_i|\hat{H}_0|\psi_i\rangle \tag{18}$$

The main contribution to the Hückel-type Hamiltonian is a set of atom- and shell-specific orbital energies H^l_A, with more negative values corresponding to more electron-attracting atomic orbitals (similar to more negative values of the atomic energy α in Hückel theory). The same H^l_A terms contribute to off-diagonal elements (playing a role similar to β in Hückel theory), which are also scaled by the overlap between basis functions, so that only atomic orbitals positioned close in space to each other are coupled when forming molecular orbitals. The values of H^l_A (or more precisely of matrix elements $\langle\phi_m e|\hat{H}_0|\phi_m e\rangle$ with respect to the basis functions) reflect the effects of electron kinetic energy, electron attraction to the nucleus, electron-electron mean Coulomb repulsion and exchange-correlation, all within a given atom.

4.5 Second- and Third-Order Energies

The second and third-order energies arise due to interactions between the charge variations on individual atoms. The charge variations involved are associated with the same Mulliken atom- and shell-populations that arise from polar covalent bonding and the shape of the molecular orbitals as discussed for the first-order term. This leads to changes in the overall electrostatic energy, both within atoms and between atoms. The second-order energy in GFN1-xTB is written as shown below, where the first summation runs over all pairs of atoms, while the second and third summations run over shells (e.g. the s and p shells) for each atom. q^l_{A} denotes the charge in shell l on atom A, and $\gamma_{AB,ll'}$ is an element of the kernel describing charge-charge interactions in terms of a modified form of Coulomb's law, the details of which will be shown below. As the charges are expressed per *shell*, and hence are spherically symmetric on a given atom, the Coulomb-like interaction is isotropic.

$$E^{(2)}_{\gamma} = \frac{1}{2}\sum_{\mathrm{A,B}}^{N_N}\sum_{l\in\mathrm{A}}\sum_{l'\in\mathrm{B}} q^l_{\mathrm{A}} q^{l'}_{\mathrm{B}}\gamma_{\mathrm{AB},ll'} \tag{19}$$

This second-order term describes a large part of the changes in electron-electron repulsion and exchange-correlation energy that arise due to changes in electron density arising from chemical bonding. It should also be noted that the charges q^l_{A} on each atom are computed from the coefficients $c_{\mu i}$ defining the shape of the molecular orbitals. While the Hückel Hamiltonian on its own expresses the preference for the electron density to locate in different shells, based on their energy levels H^l_{A}, the term above accounts for shell-shell attraction (q^l_{A} and $q^{l'}_{\mathrm{B}}$ of same sign) and repulsion (opposite sign) effects and hence also affect the shape of the orbitals. The presence of these terms, as well as those of third-order as described below, means that the shape of the molecular orbitals needs to be determined self-consistently, by guessing an initial set of orbital shapes, calculating the energy, then varying the orbital shapes in order to minimize the energy. The details of how this is done is described below, in section 4.7.

While electron-electron repulsion is of course a two-body effect, the second-order term above sets out to describe many complicated within- and between-atom effects that are not exactly described

by the chosen functional form. Higher accuracy is obtained when including a third-order term given as a third-order polynomial of the charges, as given below. This term has a fairly simple expression, as it depends only on the total charges of atoms q_A (and not on the charges of individual shells) and furthermore only depends on the within-atom interaction. The parameters Γ_A are element-specific parameters, equal to 0.000000, 0.1053856, 0.0042507, and -0.0005102 for H, C, N, and O, respectively.[1]

$$E_\Gamma^{(3)} = \frac{1}{3}\sum_\mathrm{A}^{N_N}(q_\mathrm{A})^3\Gamma_\mathrm{A} \quad ; \quad q_\mathrm{A} = \sum_{l\in\mathrm{A}} q_\mathrm{A}^l \tag{20}$$

Taken together, the first-order, second-order, and third-order energies of eqs. 18, 19 and 20 form what we will refer to as the *electronic energy* (in truth the repulsion and dispersion energies are also 'electronic', so another name could also be used...) E_elec. Note also that we will sometimes omit the subscripts EHT, γ and Γ.

$$E_\mathrm{elec} = E_\mathrm{EHT}^{(1)} + E_\gamma^{(2)} + E_\Gamma^{(3)} = E^{(1)} + E^{(2)} + E^{(3)} \tag{21}$$

4.6 Fock Matrix Formalism for the First-, Second- and Third-Order Energy

In the two previous sections, the first-, second- and third-order energies are introduced. The sum of these energies needs to be minimized with respect to the coefficients of the atomic orbitals for each of the molecular orbitals, while requiring the molecular orbitals to remain orthogonal. As in Hartree-Fock theory or Kohn-Sham DFT, this leads to a set of one-electron equations in terms of a 'Fock' operator $\hat{F}$. Diagonalizing the matrix representation of this Fock operator with respect to the basis functions yields the MOs, and needs to be done in a self-consistent way due to the coupling of the first-, second- and third-order terms. The detailed expressions for the values of the Fock matrix elements are now given.

First of all, some comments on the basis set of functions ϕ_μ. These are a set of contracted Gaussian functions, fitted to exponential functions, and for most atoms, a minimal basis set for only the valence atomic orbitals is included. The only exception is the hydrogen atom, for which two different basis functions are used, one 'normal' s function and one diffuse s' function. It should be noted that the atomic orbitals on different atoms are **not** considered to be orthogonal[2] (unlike the case for many other semi-empirical methods), so the overlap integrals $S_{\mu\nu}$ between different basis functions ϕ_μ and ϕ_ν need to be evaluated. These are the only integrals that need to be calculated – the contributions to the energy from kinetic energy, electron – nucleus interaction, and interelectron repulsion are handled in a different way. The details of the basis functions and of the procedure to calculate the overlaps is described below in detail, in section 4.8. Here we simply introduce the notation for these overlap integrals:

$$S_{\mu\nu} = \int \phi_\mu(\mathbf{r})\phi_\nu(\mathbf{r})\,\mathrm{d}\mathbf{r} = \langle\phi_\mu|\phi_\nu\rangle \tag{22}$$

Next we can write a general expression for the 'Fock' matrix elements over two basis functions, where ϕ_μ is on atom A and belongs to shell l, while ϕ_ν is on atom B and is in shell l':[3]

$$\begin{aligned} F_{\mu\nu} = \langle\phi_\mu|\hat{F}|\phi_\nu\rangle = \langle\phi_\mu|\hat{H}_0|\phi_\nu\rangle &- \frac{1}{2}S_{\mu\nu}\sum_C\sum_{l''}\left(\gamma_{\mathrm{AC},ll''} + \gamma_{\mathrm{BC},l'l''}\right)q_\mathrm{C}^{l''} \\ &- \frac{1}{2}S_{\mu\nu}\left(q_\mathrm{A}^2\Gamma_\mathrm{A} + q_\mathrm{B}^2\Gamma_\mathrm{B}\right) \end{aligned} \tag{23}$$

Next, we need to specify how each of these terms is calculated, from the coordinates of the atoms, and the parameters. We start with the first term in eq. 23, the matrix elements over the Hückel-type hamiltonian. Again we consider two basis functions, where ϕ_μ is on atom A and belongs to shell

[1] Note that the values of these parameters given in the original reference need to be divided by ten prior to use, this factor has been included here.

[2] The basis set **is** chosen so that the set of functions located on a given atom is orthonormal.

[3] Note that the minus sign for the two final terms in this equation is incorrectly shown as a plus sign in the original paper.

l, while ϕ_ν is on atom B and is in shell l'. A different expression is used for these matrix elements depending on whether the basis functions belong to shells situated on different atoms or not. First, we give the expression for matrix elements corresponding to basis functions situated on different atoms A $\neq$ B, hence $\mu \neq \nu$:

$$\langle\phi_\mu|\hat{H}_0|\phi_\nu\rangle = K_{\mathrm{AB}} k_{ll'} \frac{1}{2}(h^l_{\mathrm{A}} + h^{l'}_{\mathrm{B}}) S_{\mu\nu} \left(1 + k_{\mathrm{EN}} \Delta\mathrm{EN}^2_{\mathrm{AB}}\right) \Pi(r_{\mathrm{AB}}, l, l') \tag{24}$$

For basis functions on the same atom, basically the same expression is used, but with some small modifications. First, the only matrix elements to consider are diagonal elements $\langle\phi_\mu|\hat{H}_0|\phi_\mu\rangle$, since the basis functions on a given atom are chosen to be orthonormal, hence the $S_{\mu\nu}$ in eq. 24 is zero if $\mu \neq \nu$. Next, some of the scaling terms (K_{AB}, $k_{ll'}$) are omitted and others are by construction equal to one: $(1 + k_{\mathrm{EN}}\Delta\mathrm{EN}^2_{\mathrm{AB}})$ and $\Pi(r_{\mathrm{AB}}, l, l')$. We have:

$$\langle\phi_\mu|\hat{H}_0|\phi_\mu\rangle = h^l_{\mathrm{A}} \tag{25}$$

We now describe each of the ingredients of these matrix elements over the reference Hamiltonian $\hat{H}_0$ in detail:

- $K_{\mathrm{AB},ll'}$ is a scaling parameter which is equal to 1 for almost all atom shell pairs. For the cases we consider here (only molecules with H, C, N and O atoms), the only non-unity factors are $K_{\mathrm{HH},ss} = 0.96$ and $K_{\mathrm{HN},ss} = K_{\mathrm{HN},sp} = 1.04$. For all other atom and shell combinations, $K_{\mathrm{AB},ll'} = 1$. Note again that $K_{\mathrm{AB},ll'}$ is omitted for basis functions placed on the same atom.

- $k_{ll'}$ is a parameter that that scales the off-diagonal elements of the zeroth-order Hamiltonian, depending on the nature of the shells to which ϕ_μ and ϕ_ν belong, and more specifically on their angular momentum. For most types of shell pairs, this is simply given by $k_{ll'} = 1/2\,(k_l + k_{l'})$ with for example $k_s = 1.85$ for s shells and for p shells, $k_p = 2.25$. However, there are a number of special cases, and they are relevant for the molecules we wish to treat. First, for *sp* pairs, a slightly adjusted value is used, with $1/2\,(k_s + k_p) = 2.05$ but $k_{sp} = 2.08$. Next, the more diffuse s' basis function on hydrogen has a separate $k_{s'} = 2.85$. To avoid possible mistakes, here (and in the file with all parameters) we provide all relevant values for *pairs* of shell: $k_{ss} = 1.85$, $k_{sp} = 2.08$, $k_{pp} = 2.25$, $k_{ss'} = 2.35$, $k_{ss'} = 2.35$, $k_{s's'} = 2.85$, and $k_{s'p} = 2.55$. Note again that $k_{ll'}$ is omitted for basis functions on the same atom.

- The quantities h^l_{A} are effective atomic energy levels of the different shells. These are given in terms of element- and shell-specific base energy levels H^l_{A}, slightly scaled according to the coordination number of eq. 14 according to:

$$h^l_{\mathrm{A}} = H^l_{\mathrm{A}}(1 + k_{\mathrm{CN},l}\mathrm{CN}_{\mathrm{A}}) \tag{26}$$

  The parameters H^l_{A} corresponding to the unperturbed energy levels of the corresponding shells, are as follows: $H^s_{\mathrm{H}} = -10.923452$, $H^{s'}_{\mathrm{H}} = -2.171902$, $H^s_{\mathrm{C}} = -13.587210$, $H^p_{\mathrm{C}} = -10.052785$, $H^s_{\mathrm{N}} = -20.058000$, $H^p_{\mathrm{N}} = -12.889326$, $H^s_{\mathrm{O}} = -23.398276$, and $H^p_{\mathrm{O}} = -17.886554$, all in eV. The scaling constants $k_{\mathrm{CN},l}$ have the following values: $k_{\mathrm{CN},s} = k_{\mathrm{CN},s'} = 0.006$ and $k_{\mathrm{CN},p} = -0.003$.

- $S_{\mu\nu}$ is the overlap integral between basis functions, see eq. 22, whose calculation will be described below, section 4.8. It must be included in order to account for the fact that basis functions are not orthonormal.

- The term $\left(1 + k_{\mathrm{EN}}\Delta\mathrm{EN}^2_{\mathrm{AB}}\right)$ introduces some scaling of off-diagonal matrix elements based on $\Delta\mathrm{EN}_{\mathrm{AB}}$, the difference in Pauli electronegativity of the atoms. Importantly, this term is omitted if either or both of ϕ_μ and ϕ_ν is one of the diffuse s' shells on hydrogen. The electronegativities of H, C, N and O are respectively 2.20, 2.55, 3.04 and 3.44. The scaling constant $k_{\mathrm{EN}} = -0.007$.

- $\Pi(r_{\mathrm{AB}}, l, l')$ is a distance- and shell-dependent polynomial scaling function, expressed as:

$$\Pi(r_{\mathrm{AB}}, l, l') = \left\{1 + k^{\mathrm{poly}}_{\mathrm{A},l}\left(\frac{r_{\mathrm{AB}}}{R_{\mathrm{AB,cov}}}\right)^{1/2}\right\}\left\{1 + k^{\mathrm{poly}}_{\mathrm{B},l'}\left(\frac{r_{\mathrm{AB}}}{R_{\mathrm{AB,cov}}}\right)^{1/2}\right\} \tag{27}$$

The atom- and shell-specific coefficients $k^{\text{poly}}_{\text{A},l}$ are $k^{\text{poly}}_{\text{H},s} = k^{\text{poly}}_{\text{H},s'} = 0$, $k^{\text{poly}}_{\text{C},s} = -0.07082170$, $k^{\text{poly}}_{\text{C},p} = 0.00812216$, $k^{\text{poly}}_{\text{N},s} = -0.12745585$, $k^{\text{poly}}_{\text{N},p} = -0.01428367$, $k^{\text{poly}}_{\text{O},s} = -0.13729047$, and $k^{\text{poly}}_{\text{O},p} = -0.04453341$. The values of these parameters provided in the original paper need to be divided by 100 prior to use (this has already been done here). r_{AB} is the distance between the atoms on which the two shells are centred, and $R_{\text{AB,cov}}$ is the covalent radius for bonds between such atoms, equal to the sum of the covalent radii of the two atoms, which are given by a set of atom-specific parameters $R^{\Pi}_{\text{A,cov}}$ (with $R^{\Pi}_{\text{H,cov}} = 0.32$, $R^{\Pi}_{\text{C,cov}} = 0.75$, $R^{\Pi}_{\text{N,cov}} = 0.71$, and $R^{\Pi}_{\text{O,cov}} = 0.64$, all in Å. These radii should **not** be scaled by a factor 4/3 (unlike the $R^{\text{disp}}_{A,\text{cov}}$ used to compute coordination numbers in eq. 14). Also, pay attention to the fact that even prior to scaling, the value for oxygen is slightly different from the corresponding $R^{\text{disp}}_{A,\text{cov}}$ value.

Together with the energy expression of eq. 18, it can be seen that the negative shell level energies H^l_{A} are such that the reference Hamiltonian term returns a very negative energy, even for isolated atoms. Hence the GFN1-xTB energy for separated atoms is not equal to zero.

We now return to eq. 23 (reproduced for convenience below), and consider the last two terms.

$$\begin{aligned} F_{\mu\nu} = \langle\phi_\mu|\hat{F}|\phi_\nu\rangle = \langle\phi_\mu|\hat{H}_0|\phi_\nu\rangle &- \frac{1}{2}S_{\mu\nu}\sum_C\sum_{l''}\left(\gamma_{\text{AC},ll''} + \gamma_{\text{BC},l'l''}\right)q^{l''}_{\text{C}} \\ &- \frac{1}{2}S_{\mu\nu}\left(q^2_{\text{A}}\Gamma_{\text{A}} + q^2_{\text{B}}\Gamma_{\text{B}}\right) \end{aligned} \tag{28}$$

- These terms basically describe how the energy increases or decreases when adding electron density to the product of basis functions $\phi_\mu\phi_\nu$. This accounts for the second- and third-order energy contributions of eqs. 19 and 20. Due to normalization, the electron density associated with a product of basis functions is proportional to $S_{\mu\nu}$, hence the presence of this factor. This factor also means that these terms are largest for the diagonal elements $\langle\phi_\mu|\hat{F}|\phi_\mu\rangle$.

- q^l_{A} is the 'charge' of shell l of atom A, the difference between the number of electrons in that shell in the ground state of the neutral atoms, $n^l_{\text{A},0}$ (1 for the s orbital in hydrogen, 0 for the s' orbital, 2 for the s orbitals of C, N and O, and respectively 2, 3 and 4 for their p shells), and the actual number of electrons in the shell, given by the Mulliken population associated with the MO coefficients. This is most conveniently calculated using the *density matrix* $\mathbf{P}$ corresponding to the orbitals, as a double sum, first over all the basis functions ϕ_μ in shell l of atom A, and next over all other basis functions ϕ_ν (N_b is the total number of basis functions):

$$q^l_{\text{A}} = n^l_{\text{A},0} - \sum_{\mu\in l}\sum_{\nu}^{N_b} S_{\mu\nu}P_{\mu\nu} \tag{29}$$

  The elements of the density matrix are obtained as sums over the molecular orbital coefficients of eq. 17, including only the occupied molecular orbitals ($n_{\text{occ}} = n/2$):

$$P_{\mu\nu} = 2\sum_{k=1}^{n_{\text{occ}}} c_{\mu k}c_{\nu k} \tag{30}$$

- The Coulomb-law prefactors $\gamma_{\text{AB},ll'}$ from eq. 23 depend inversely on the distance between atoms A and B, but the dependence is not simply an inverse function $1/r_{\text{AB}}$, as a more complex dependence makes it possible to account at least approximately for the fact that one is not dealing with point charges. Unlike *ab initio* quantum chemical techniques or Kohn-Sham DFT, the Coulomb repulsion between electrons is not treated exactly through two-electron four-centre electron repulsion integrals. Instead, a simplified two-centre formalism is used, making xTB much faster than these other methods. Specifically, this Coulombic term is given by the Mataga-Nishimoto-Ohno-Klopman function:

$$\gamma_{\text{AB},ll'} = \left(\frac{1}{\left(r_{\text{AB}}\right)^{k_g} + \left(\eta_{\text{AB},ll'}\right)^{-k_g}}\right)^{1/k_g} \tag{31}$$

In this expression, $k_g = 2$ is a global parameter, and $\eta_{\mathrm{AB},ll'}$ is the average *hardness* of the two shells on the two atoms, defined below. Note that the sum in eq. 28 runs over *all* triples of atoms A, B, C, i.e. it includes the case where C = A or C = B, and/or A = B.

$$\frac{1}{\eta_{\mathrm{AB},ll'}} = \frac{1}{2}\left(\frac{1}{(1+\kappa_{\mathrm{A}}^{l})\eta_{\mathrm{A}}} + \frac{1}{(1+\kappa_{\mathrm{B}}^{l'})\eta_{\mathrm{B}}}\right) = \frac{1}{2}\left(\frac{1}{\eta_{\mathrm{A}}^{l}} + \frac{1}{\eta_{\mathrm{B}}^{l'}}\right) \tag{32}$$

Here η_{A} and η_{B} are element-specific hardness parameters, while κ_{A}^{l} and $\kappa_{\mathrm{B}}^{l'}$ are element- and shell-specific scaling parameters that adjust the atomic hardnesses to be shell-specific. The resulting shell-hardnesses $(1+\kappa_{\mathrm{A}}^{l})\eta_{\mathrm{A}}$ can also simply be written as η_{A}^{l}. Note that hardness is correlated with spatial extent: harder shells have more compact electron distributions, hence their interactions deviate less from the simple Coulomb-law dependence on $1/r_{\mathrm{AB}}$. The original paper provides the parameters η_{A} and κ_{A}^{l} (these latter further need to be scaled by a factor ten prior to use, which is not mentioned in the original paper), but instead here we provide values of η_{A}^{l}. These are: $\eta_{\mathrm{H}}^{s} = \eta_{\mathrm{H}}^{s'} = 0.470099$, $\eta_{\mathrm{C}}^{s} = 0.479988$, $\eta_{\mathrm{C}}^{p} = 0.4573719$, $\eta_{\mathrm{N}}^{s} = 0.476106$, $\eta_{\mathrm{N}}^{p} = 0.4911076$, $\eta_{\mathrm{O}}^{s} = 0.583349$, and $\eta_{\mathrm{O}}^{p} = 0.6052017$.

- The third-order term in equation 28 is straightforward to apply, and uses the atom-specific scaling factors Γ_{A} whose values are given above near eq. 20.

The contribution of the second- and third-order terms to the Fock matrix elements of eq. 23 can be rewritten in a simpler form in terms of *shell and atom level shifts* $\delta\epsilon_{\mathrm{A}}^{l}$ and $\delta\mathcal{E}_{\mathrm{A}}$, and the density matrix of eq. 30, defined as follows. These equations are easier to implement than those given above. Once again basis function ϕ_μ must be understood as belonging to shell l on atom A, while basis function ϕ_ν belongs to shell l' on atom B. The sum in the first equation runs over all shells B, i.e. it includes the case A = B.

$$\begin{aligned}
\delta\epsilon_{\mathrm{A}}^{l} &= \sum_{\mathrm{B}=1}^{N_N}\sum_{l'\in\mathrm{B}} \gamma_{\mathrm{AB},ll'} q_{\mathrm{B}}^{l'} \\
\delta\mathcal{E}_{\mathrm{A}} &= \Gamma_{\mathrm{A}} q_{\mathrm{A}}^{2} \\
F_{\mu\nu} &= \langle\phi_\mu|\hat{F}|\phi_\nu\rangle = \langle\phi_\mu|\hat{H}_0|\phi_\nu\rangle - \frac{1}{2}S_{\mu\nu}\left(\delta\epsilon_{\mathrm{A}}^{l} + \delta\epsilon_{\mathrm{B}}^{l'} + \delta\mathcal{E}_{\mathrm{A}} + \delta\mathcal{E}_{\mathrm{B}}\right)
\end{aligned} \tag{33}$$

4.7 Self-consistent Evaluation of the GFN1-xTB Energy

As already mentioned, the first-, second- and third-order energies of eqs. 18, 19 and 20 need to be evaluated self-consistently to yield the overall electronic energy of eq. 21. In practice, this means that the code for these parts should be structured as follows:

1. First one evaluates the elements of the overlap matrix $S_{\mu\nu}$, as described in section 4.8.

2. Next, one evaluates the *inverse square root* $\mathbf{S}^{-1/2}$ of the overlap matrix $\mathbf{S}$. This is needed in order to switch from expressions for the Fock and coefficient matrices $\mathbf{F}$ and $\mathbf{C}$ in terms of the non-orthogonal set of basis functions ϕ_μ, to matrices $\mathbf{F}'$ and $\mathbf{C}'$ expressed in terms of orthonormal basis functions. $\mathbf{S}^{-1/2}$ is obtained by first diagonalizing $\mathbf{S}$ i.e. finding the unitary matrix $\mathbf{L}$ such that $\mathbf{L}^{-1}\mathbf{S}\mathbf{L}$ (or equivalently $\mathbf{L}^{T}\mathbf{S}\mathbf{L}$, where the superscript 'T' denotes transposition) is a diagonal matrix Λ, then forming the diagonal matrix $\Lambda^{-1/2}$ (i.e. the matrix with as diagonal elements the inverse square roots of the diagonal elements of Λ), then using:

$$\mathbf{S}^{-1/2} = \mathbf{L}\Lambda^{-1/2}\mathbf{L}^{T} \tag{34}$$

3. Then one evaluates the elements of the Hückel-type Hamiltonian, $\langle\phi_\mu|\hat{H}_0|\phi_\nu\rangle$, using equations 24 to 27. Just as for $\mathbf{S}$ and $\mathbf{S}^{-1/2}$, the elements of this matrix do not depend on the atomic charges i.e. they do not depend on the shapes of the occupied molecular orbitals, so this matrix can be calculated once and for all at the start of the calculation.

4. At this point, one can calculate the Coulomb kernel matrix of elements $\gamma_{\mathrm{AB},ll'}$ using eq. 31. These do not vary with the charges, so can be computed once and for all prior to the self-consistent field iterations.

5. Next one calculates a first guess for the Fock matrix elements $F_{\mu\nu}^{(1)}$, based on eq. 33 and a first set of atom- and shell charges. These first guesses for the charges can in principle be calculated using some approximate electronegativity equalization approach, but an easier approach that is suitable here is simply to initially set all charges to zero. In that case, the first Fock matrix $\mathbf{F}^{(1)}$ is simply equal to $\mathbf{H}_0$.

6. The Fock matrix $\mathbf{F}$ can be diagonalized, i.e. one can compute the matrix $\mathbf{C}$ and diagonal matrix $\mathbf{E}$ such that the similarity transform of $\mathbf{F}$ by $\mathbf{C}$ is equal to $\mathbf{E}$. In fact, though, the diagonalization is performed in the orthogonalized basis, so it starts from matrix $\mathbf{F}'$ and yields $\mathbf{C}'$. $\mathbf{F}'$ is the Fock matrix in the orthonormal basis and is obtained from $\mathbf{F}$ as shown below, with $\mathbf{C}$ obtained from the coefficients in the orthogonal basis (contained in matrix $\mathbf{C}'$):

$$\mathbf{F}' = (\mathbf{S}^{-1/2})^T \mathbf{F} \mathbf{S}^{-1/2} \quad ; \quad \mathbf{F}'\mathbf{C}' = \mathbf{E}\mathbf{C}' \quad ; \quad \mathbf{C} = \mathbf{S}^{-1/2}\mathbf{C}' \tag{35}$$

7. With the resulting first guess for the set of coefficients $\mathbf{C}^{(1)}$, one can then construct a first guess for the density matrix $\mathbf{P}^{(1)}$ (using eq. 30, note that the $n/2$ occupied orbitals are those with the lowest energies, i.e. correspond to the columns of $\mathbf{C}$ mapping onto the lowest eigenvalues). The density matrix $\mathbf{P}^{(1)}$ enables calculation of the shell charges q_{A}^{l} and atomic charges q_{A} using eqs. 29 and 20, and then eq. 33 enables calculation of a new estimate for the Fock matrix, $\mathbf{F}^{(2)}$. This in turn yields $\mathbf{C}^{(2)}$, which yields $\mathbf{P}^{(2)}$, which can be used to obtain $\mathbf{F}^{(3)}$, and one goes on calculating $\mathbf{P}^{(k)}$ from $\mathbf{C}^{(k)}$, $\mathbf{F}^{(k+1)}$ from $\mathbf{P}^{(k)}$, and $\mathbf{C}^{(k+1)}$ from $\mathbf{F}^{(k+1)}$ until the successive values no longer change significantly.

8. To obtain convergence, one often needs to use *damping*, to prevent large changes in the charges being made at a given cycle, which can prevent identification of a self-consistent solution. In practice, once the charges $q_{\mathrm{A}}^{l,(k)}$ and $q_{\mathrm{A}}^{(k)}$ are computed from a given density matrix $\mathbf{P}^{(k)}$, these are compared to those obtained at the previous cycle, $q_{\mathrm{A}}^{l,(k-1)}$ and $q_{\mathrm{A}}^{(k-1)}$. If there is no large change, i.e. if the largest absolute change in q is smaller than 10^{-3} or so, then eq. 33 can simply be applied. In case larger changes are seen, it is best to scale the charge changes, using:

$$q_{\mathrm{A,damped}}^{l,(k)} = q_{\mathrm{A}}^{l,(k-1)} + \lambda \left(q_{\mathrm{A}}^{l,(k)} - q_{\mathrm{A}}^{l,(k-1)} \right) \tag{36}$$

   (and likewise for the atomic charges) prior to evaluating $\mathbf{F}^{(k+1)}$. The damping coefficient λ should be smaller than one; a value of 0.4 seems to work well.

9. At each cycle, the overall electronic energy $E_{\mathrm{elec}}^{(k)}$ of eq. 21, and its components $E_{\mathrm{EHT}}^{(1)}$ (eq. 18), $E_{\gamma}^{(2)}$ (eq. 19) and $E_{\Gamma}^{(3)}$ (eq. 20) can be calculated. The expression for the first-order energy of eq. 18 can be re-written in a more convenient form in terms of the density matrix:

$$E_{\mathrm{EHT}}^{(1)} = \sum_{\mu,\nu} P_{\mu\nu} \langle \phi_\mu | \hat{H}_0 | \phi_\nu \rangle \tag{37}$$

   It is important when evaluating the overall electronic energy $E_{\mathrm{elec}}^{(k)} = E_{\mathrm{EHT}}^{(1,k)} + E_{\gamma}^{(2,k)} + E_{\Gamma}^{(3,k)}$ during the SCF procedure that this is done in a consistent way. At the end of the SCF procedure, this makes little difference, but during the cycles, one should use charges for $E_{\gamma}^{(2)}$ and $E_{\Gamma}^{(3)}$ that result from the same density matrix as is used for $E_{\mathrm{EHT}}^{(1,k)}$.

4.8 Treatment of Basis Functions and Calculation of the Overlap Matrix

The basis set used here uses basis functions that are *contracted Gaussian functions* centred on individual atoms within the system treated. The most simple such functions are s-symmetric basis

functions, eq. 38:

$$\phi_\mu^s(\mathbf{r}) = \sum_{i=1}^{n_{c,\mu}} d'_{i\mu} N_{i\mu} \exp\left(-\zeta_{i\mu}|\mathbf{r} - \mathbf{r}_A|^2\right) \tag{38}$$

In this equation, $\mathbf{r}_A$ is the vector of spatial coordinates of nucleus A, on which the basis function is centred, $\mathbf{r}$ is the set of spatial coordinates for the point where the function is to be evaluated, $|\mathbf{r} - \mathbf{r}_A|$ is the distance between these two points, $n_{c,\mu}$ is the degree of contraction of basis function ϕ_μ, *i.e.* the number of primitive Gaussian functions used to generate the contracted function, and $\zeta_{i\mu}$ is the exponent of each primitive Gaussian – larger values of ζ correspond to 'tighter' primitives, smaller values correspond to more diffuse functions. $N_{i\mu}$ is a normalization coefficient, chosen so that the primitive Gaussian function is normalized (in the sense that $\int N_{i\mu}^2 \exp\left(-2\zeta_{i\mu}|\mathbf{r} - \mathbf{r}_A|^2\right) \mathrm{d}\mathbf{r} = 1$). The *contraction coefficients* $d'_{i\mu}$ are chosen when parameterizing the basis set, under the constraint that the overall function ϕ_μ is normalized, and are available in databases of basis sets. In order to make your program simpler, in this exercise, you will be provided with coefficients $d_{i\mu}$ that serve to at once contract and normalize the primitive basis functions i.e. that are chosen so that for each basis function $\phi_\mu(\mathbf{r}) = \sum_{i=1}^{n_{c,\mu}} d_{i\mu} \exp\left(-\zeta_i|\mathbf{r} - \mathbf{r}_A|^2\right)$, one has that $\int \phi_\mu^2(\mathbf{r}) \, \mathrm{d}\mathbf{r} = 1$. In these terms, the basis functions are expressed as in eq. 39 (where we introduce the notation $\phi_{i,\mu}^s(\mathbf{r})$ to refer to the i'th non-normalized primitive s Gaussian contributing to contracted basis function $\phi_\mu^s(\mathbf{r})$).

$$\phi_\mu^s(\mathbf{r}) = \sum_{i=1}^{n_{c,\mu}} d_{i\mu} \exp\left(-\zeta_i|\mathbf{r} - \mathbf{r}_A|^2\right) = \sum_{i=1}^{n_{c,\mu}} d_{i\mu} \phi_{i,\mu}^s(\mathbf{r}) \tag{39}$$

We will also use p basis functions. These will be *real* p functions, p_x, p_y and p_z. The expression for a p_x function is given below. A *shell* of p functions is a set of three functions, p_x, p_y and p_z, each of which has the same value of $n_{c,\mu}$, and the same sets of values of $\{\zeta_i\}$ and $\{d_{i\mu}\}$ (but of course the $(x - x_A)$ term is replaced by $(y - y_A)$ or $(z - z_A)$). Note that a contracted p_x function is a linear combination of primitive p_x functions only, and likewise for p_y and p_z.

$$\phi_\mu^{p_x}(\mathbf{r}) = \sum_{i=1}^{n_{c,\mu}} d_{i\mu}(x - x_A) \exp\left(-\zeta_i|\mathbf{r} - \mathbf{r}_A|^2\right) = \sum_{i=1}^{n_{c,\mu}} d_{i\mu} \phi_{i,\mu}^{p_x}(\mathbf{r}) \tag{40}$$

We will need to calculate overlap matrix elements between these functions, as already introduced in eq. 22, and shown again here:

$$S_{\mu\nu} = \int \phi_\mu(\mathbf{r}) \phi_\nu(\mathbf{r}) \, \mathrm{d}\mathbf{r} = \langle \phi_\mu | \phi_\nu \rangle \tag{41}$$

Referring back to the definition in eq. 39 of the s basis functions, we see that for overlap of one s basis function with another s basis function, this can be re-written as a sum of integrals of products of primitive Gaussian functions, eq. 42:

$$S_{\mu\nu}^{s,s} = \sum_{k=1}^{n_c} \sum_{l=1}^{n'_c} d_{k\mu} d_{l\nu} \int \exp\left(-\zeta_{k,\mu}|\mathbf{r} - \mathbf{r}_\mu|^2\right) \exp\left(-\zeta_{l,\nu}|\mathbf{r} - \mathbf{r}_\nu|^2\right) \mathrm{d}\mathbf{r} \tag{42}$$

The underlying overlap integrals between the primitive s Gaussian functions can be written as eq. 43:

$$S_{k,\mu,l,\nu}^{s,s} = \int \phi_{k,\mu}^s(\mathbf{r}) \phi_{l,\nu}^s(\mathbf{r}) \, \mathrm{d}\mathbf{r} = \int \exp\left(-\zeta_{k,\mu}|\mathbf{r} - \mathbf{r}_\mu|^2\right) \exp\left(-\zeta_{l,\nu}|\mathbf{r} - \mathbf{r}_\nu|^2\right) \mathrm{d}\mathbf{r} \tag{43}$$

In these equations, $\mathbf{r}_\mu$ refers to the coordinates of the atom on which basis function ϕ_μ^s is based, and $\mathbf{r}_\nu$ to the coordinates of the atom on which basis function ϕ_ν^s is based. It can be seen that the overall overlap integral of eq. 42 is a weighted sum of overlap integrals of primitive Gaussian functions. The integrand in eq. 43 is a product of two Gaussians, and that makes evaluation of the integral reasonably easy: the product of two Gaussians is itself a Gaussian, and the integral over all space of a Gaussian function centred at some arbitrary point $\mathbf{r}_0$ has a closed-form expression, eq. 44:

$$\int \exp\left(-\zeta|\mathbf{r} - \mathbf{r}_0|^2\right) \mathrm{d}\mathbf{r} = \left(\frac{\pi}{\zeta}\right)^{3/2} \tag{44}$$

The product of two Gaussian functions that is the integrand of eq. 42 can be re-written as the product of a Gaussian function with exponent ζ centred at $\mathbf{r}_P$ and a constant prefactor that depends on the distance between the two centres of the original Gaussian functions and on the harmonic mean ξ of the two exponents, eq. 45:

$$\exp\left(-\zeta_{k,\mu}|\mathbf{r}-\mathbf{r}_\mu|^2\right)\exp\left(-\zeta_{l,\nu}|\mathbf{r}-\mathbf{r}_\nu|^2\right)=\exp\left(-\xi|\mathbf{r}_\mu-\mathbf{r}_\nu|^2\right)\times\exp\left(-\zeta|\mathbf{r}-\mathbf{r}_P|^2\right) \tag{45}$$

Where $\mathbf{r}_P$, ζ, and ξ are defined in eq. 46:

$$\zeta=\zeta_{k,\mu}+\zeta_{l,\nu}\quad;\quad\xi=\frac{\zeta_{k,\mu}\zeta_{l,\nu}}{\zeta}\quad;\quad\mathbf{r}_P=\frac{\zeta_{k,\mu}\mathbf{r}_\mu+\zeta_{l,\nu}\mathbf{r}_\nu}{\zeta} \tag{46}$$

Combining equations 43, 44, 45 and 46 we can write closed-form expressions for the overlap integrals of primitive and contracted s Gaussian basis functions, eq. 47:

$$S^{s,s}_{k,\mu,l,\nu}=\exp\left(-\xi|\mathbf{r}_\mu-\mathbf{r}_\nu|^2\right)\left(\frac{\pi}{\zeta}\right)^{3/2}\quad;\quad S_{\mu,\nu}=\sum_{k=1}^{n_c}\sum_{l=1}^{n'_c}d_{k\mu}d_{l\nu}S^{s,s}_{k,\mu,l,\nu} \tag{47}$$

Evaluating all overlap integrals between s basis functions thus requires writing code with four nested loops, running over all distinct pairs of contracted basis functions (ϕ_μ,ϕ_ν) and over all pairs of primitive functions corresponding to each of the contracted functions, see eq. 39.

We also need to calculate integrals where one or both of the contracted functions is a p function. It turns out that the derivative of an s function with respect to x is a sum of an s function and a p_x function with the same exponent ζ, which after some algebra can be used to show that integrals over p functions can be written as combinations of integrals over s functions with the same exponent, multiplied by a polynomial function of the Cartesian coordinates. For convenience, we will introduce a new notation for the integrals, where a given integral over two primitive s functions $S^{s,s}_{k,\mu,l,\nu}$ is written as $(0||0)$, an integral over one primitive s function and one primitive p_x function is written as $(0||x)$, an integral over one primitive p_x function and one primitive p_z function is written as $(x||z)$ and so on. The expression for $(0||x)$ can be shown to be given by the expression below, where x_P is the x coordinate of $\mathbf{r}_P$ of eq. 46, and x_ν is the x coordinate of the atom where the second basis function is centred (the exponent $\zeta_{l,\nu}$ for the 's' function here should be taken as being the same as that of the actual p_x function being considered):

$$S^{s,p_x}_{k,\mu,l,\nu}=(0||x)=(x_P-x_\nu)(0||0) \tag{48}$$

Similar expressions are obtained for $(0||y)$, $(0||z)$, $(x||0)$, $(y||0)$ and $(z||0)$ by appropriate substitutions. For integrals over two primitive p functions, there are two cases. Where the two p functions involved correspond to different Cartesian coordinates, e.g. for $(y||z)$, we have two possible ways for expressing it in terms of integrals over s and p functions, as shown below. Which of these expressions is used is purely a question of convenience.

$$S^{p_y,p_z}_{k,\mu,l,\nu}=(y||z)=(y_P-y_\mu)(0||z)=(z_P-z_\nu)(y||0) \tag{49}$$

If both p functions involve *the same* Cartesian direction, e.g. $(z||z)$, there is an additional term (where ζ is given by eq. 46). Once again, one can freely choose to start by evaluating $(z||0)$ or $(0||z)$, whichever is most convenient. Since all nine integrals $(\alpha||\beta)$, where α and β are equal to x, y or z, are ultimately all constructed from the same integral $(0||0)$, it makes sense to evaluate all nine integrals in one go, so as not to have to recompute $(0||0)$ multiple times.

$$S^{p_z,p_z}_{k,\mu,l,\nu}=(z||z)=(z_P-z_\mu)(0||z)+\frac{1}{2\zeta}(0||0)=(z_P-z_\nu)(z||0)+\frac{1}{2\zeta}(0||0) \tag{50}$$

For both the s,p and p,p integrals, the integrals over primitive basis functions given by the above expressions need to be summed up for all combinations of primitives, as in eq. 47.

As a final comment on the overlap matrix elements, the resulting matrix $\mathbf{S}$ is symmetric, so that $S_{\mu\nu}=S_{\nu\mu}\ \forall\mu,\nu$. This symmetry can be exploited by computing the integrals only for *unique* pairs of shells, and assigning the final values to both $S_{\mu\nu}$ and $S_{\nu\mu}$.

5 Overall Coding Strategy

As mentioned before, while GFN1-xTB is a very efficient method, and the calculations on typical medium-sized molecules with less than 50 H, C, N and O atoms should take less than a few seconds, the method is quite complicated as it involves many terms. Here I briefly outline the strategy that you could follow to construct the desired code, and also set out the ground rules for what is expected, and lay out some of the choices you can make depending on the amount of time and experience that you have.

I provide for you four types of file: (a) A parameters file, named `parameters.dat`, with all the parameters needed to do calculations on closed-shell molecules containing only H, C, N and O atoms. The parameters have been described above, and the parameter file contains comments that cross-reference which equation a given parameter appears in. While some parameters are given in eV, all calculations should be performed using atomic units as all the equations above assume atomic units. (b) Files with coordinates for a set of simple sample molecules, with names `moleculename.xyz`. The coordinates are provided in $\mathring{A}$, but should be converted to atomic units (bohr) prior to performing calculations – once again, *all* calculations should use atomic units. The molecules include very small diatomic molecules (CO), simple hydrogen-containing molecules H_2O and CH_4, medium-sized molecules such as methanol and *sec*-butylamine, and the slightly larger molecule glucose, $C_6H_{12}O_6$. (c) Extensive output- and debugging files with names `moleculename.out`. These contain output from my code, with final energies but also many intermdiate quantities, often cross-referenced to the number of the equation in this document where they are described. The energies have been validated to energies obtained from the `xtb` code from the Grimme group. (d) Extended 'detailed input' files with names `moleculename.inp`, which contain some important pre-computed intermediate quantities, especially the overlap matrix elements $S_{\mu\nu}$ and the zeroth-order Hückel-like hamiltonian matrix elements $\langle\phi_\mu|\hat{H}_0|\phi_\nu\rangle$. If you wish, you can make the task simpler by aiming to perform calculations starting from these files, and not just from the `moleculename.xyz` files. As mentioned elsewhere, your choice should be motivated by your level of experience and interests.

I suggest writing a sequence of codes with increasing complexity, perhaps of the following natures. Keep each version of your code that works (either using a version manager such as in `github` or by making manual copies). The steps below that can be omitted in case you use the detailed input with intermediate quantities (from the `moleculename.inp` files) are mentioned in each case.

1. After you learn, if needed, the basic coding skills needed to write a simple program, I recommend you start by writing a code to read in the `moleculename.xyz` files, and then write code that calculates all the atom-atom distances r_{AB} for a given molecule. Check these against the debugging files.

2. Write code that reads in the first few lines of the `parameters.dat` file, with the $Z_{\mathrm{A}}^{\mathrm{eff}}$ and α_{A} parameters, and then computes the zero-th order repulsion energy of eq. 9. Again, debug this with the provided `moleculename.out` files. The obtained energy should agree to at least six decimal places for all the molecules if your code is correct.

3. Read in the parameters relating to the dispersion energy from the `parameters.dat` files, and calculate the coordination numbers (eq. 14), and debug/check those. Again agreement should be within many decimal places.

4. Complete the calculation of the dispersion energy, eqs. 10 to 16. This step is optional: you can choose to leave it until last or indeed omit it if you wish.

5. Write code that reads in properties of the basis set and allocates basis functions and shells to each atom. Be careful! The parameter file gives information about the basis set allocated to each **type of** atom, e.g. for all C atoms. If your molecule contains (for example) multiple C atoms, **each of them** needs to carry the same s and p shells (which differ only in where in space they are centred). In case you opted for using the detailed input files, all of the important shell- and basis-function properties are included in those files, so this step can be left out here (and simply replaced by appropriate reading in of the relevant properties.

6. Compute the overlap matrix elements, starting with the simple case where both basis functions are s functions, then move on to p functions. For the set of three overlap matrix elements $(0||x)$, $(0||y)$ and $(0||z)$ where the p functions belong to the same shell, calculate all three overlap matrix elements in one single call to a function, and likewise for the nine integrals of the type $(x||y)$. Note that for simple symmetric molecules, many overlap integrals are zero for symmetry reasons, which may be the case even if your code has a bug. Hence check your code works also for the slightly larger molecules before moving on. This step (which is not completely trivial) can be completely omitted if you opt for using detailed input, since you can simply read in the integrals. You will of course instead have to write code to read in the values from the `moleculename.inp` files, but that is much easier.

7. Evaluate the inverse square-root matrix $\mathbf{S}^{-1/2}$. The `moleculename.inp` files do not include, so this step cannot be omitted.

8. Evaluate the Hückel-type Hamiltonian matrix elements $\langle\phi_\mu|\hat{H}_0|\phi_\nu\rangle$ of eq. 24. Like point 6., evaluating these integrals is not very hard but it is not completely trivial either. Hence you have the option simply to read these files from the detailed input files.

9. Diagonalise the matrix $\mathbf{H}_0$, after first converting to the orthogonal basis. Efficient routines for matrix diagonalization are available in various programming languages. In Python, you can use the `numpy.linalg` library and specifically the `numpy.linalg.eigh` function. In Fortran, the LAPACK library is available on most machines and the routine DSYEV can be used to obtain eigenvectors and eigenvalues. The debugging files contain the results of this step so you can check everything is done correctly.

10. Then complete the last steps described in section 4.7, with a loop until you reach convergence. 9-10 cycles should be sufficient for most molecules described here. Remember that in the early stages of the SCF procedure, you will need to use 'damping' (eq. 36). You can check for convergence at each cycle by monitoring for negligible changes in charges (largest change in charge smaller than 10^{-5}) or in energy (energy change less than 10^{-6}). This last step is in reality several steps, but once you have reached this point, you should be able to complete these last parts fairly straightforwardly.

Once again, I understand that some students may not be able to complete all steps in the time available. You are free to make choices about which parts you do – above and elsewhere there are suggestions about possible options. Provided a serious attempt has been made, and that that the code written is fully understood, and can perform some of the work for an unseen molecule by reading in an arbitrary file `moleculename.xyz` or `moleculename.inp`, a pass grade can be obtained. Bear in mind how much time you have and make sensible choices. I do encourage you to try to reach as far as possible in the above sequence of steps. From experience in previous years, I believe that well over half the group should be able to complete the whole task, but I also expect that a significant minority will get most of the benefit of the course even if they leave out some parts. As a final comment: make sure your code outputs intermediate quantities so as to enable easy comparison with correct values in case part of your code does not work.

6 Acknowledgements

I wish to thank Stefan Grimme and the Grimme group for development of the GFNn-xTB family of methods and for making their code for this method available. I also wish to thank Hani Mustafa Hashim for helping with the development and testing of the code used to generate debugging informaton for the present exercise.

7 References

(a) A webpage with many useful tips on coding quantum chemistry: https://github.com/CrawfordGroup/ProgrammingProjects

(b) A general overview of the GFNn-xTB methods ($n = 0, 1, 2$): "Extended tight-binding quantum chemistry methods", C. Bannwarth, E. Caldeweyher, S. Ehlert, A. Hansen, P. Pracht, J. Seibert, S. Spicher and S. Grimme, *WIREs Comput. Mol. Sci.* 2021, **11** e1493, DOI: https://doi.org/10.1002/wcms.1493.

(c) What I refer to above as the 'original paper', which contains a systematic description of the GFN1-xTB method and its performance: "A Robust and Accurate Tight-Binding Quantum Chemical Method for Structures, Vibrational Frequencies, and Noncovalent Interactions of Large Molecular Systems Parametrized for All spd-Block Elements (Z = 1 – 86)", S. Grimme, C. Bannwarth and P. Shushkov, *J. Chem. Theory Comput.* 2017, **13**, 1989 – 2009, https://doi.org/10.1021/acs.jctc.7b00118.

(d) Description of the DFTD3 dispersion correction: "A consistent and accurate *ab initio* parametrization of density functional dispersion correction (DFT-D) for the 94 elements H-Pu", S. Grimme, J. Antony, S. Ehrlich and H. Krieg, *J. Chem. Phys.*, **2010**, *132*, 154104, https://doi.org/10.1063/1.3382344.

(e) Description of the 'Becke-Johnson' short-range dampiung function: "A post-Hartree-Fock model of intermolecular interactions: Inclusion of higher-order corrections", E. R. Johnson and A. D. Becke, *J. Chem. Phys.*, **2006**, *124*, 174104, https://doi.org/10.1063/1.2190220,andrefs.\therein..

(f) A method for evaluating integrals over basis functions (only the simplest case of basis function overlap integrals is needed for this project): "Efficient recursive computation of molecular integrals over Cartesian Gaussian functions", S. Obara and A. Saika, *J. Chem. Phys.*, **1986**, *84*, 3963 – 3974, https://doi.org/10.1063/1.450106.