Geometry Optimization and Coordinate Systems

Jeremy Harvey

October 25, 2024

Abstract

This document lays out the basic background needed to construct a program to perform structural optimization (often referred to as 'geometry optimization') for molecular systems, using both Cartesian and internal coordinates. All the basic equations needed to perform the corresponding coding are provided, including the equations for a simple sample potential energy energy surface based on molecular mechanics. The aim of this document is to introduce the basic theory for geometry optimization, and to discuss the advantages and disadvantages of using Cartesian and internal coordinates for optimization.

1 Introduction

Many problems in computational chemistry and related fields involve geometry optimization (or more precisely structural optimization, though we will use the term 'geometry optimization' in this document). This is a specific example of a more general numerical problem, the optimization of a function to a minimum or maximum, which is also relevant in e.g. machine-learning model optimization. In the case of geometry optimization, the goal is to find the value(s) r_{opt} of the coordinate(s) for the atoms of the treated molecular or periodic system that yield the lowest potential energy. For most molecular systems, with $n_{\mathrm{at}} \geq 3$ atoms, the system has multiple degrees of freedom (the exception is diatomic molecules, that can be described using just one coordinate, the atom-atom distance r), so the coordinates $\mathbf{r}$ that need to be optimized form a vector[1] $\mathbf{r} \equiv (r_1, r_2, r_3, \ldots r_N)$ where N is the number of coordinates, and the potential energy $V(\mathbf{r})$ that needs to be optimized is a scalar function of many variables, $V : \mathbb{R}^N \rightarrow \mathbb{R}$. A minimum-energy structure is characterized by having a gradient of the energy V which is zero, i.e. the derivatives of the energy with respect to all the coordinates are zero:

$$\frac{\partial V(\mathbf{r}_{\mathrm{opt}})}{\partial r_i} = 0 \qquad \forall i \tag{1}$$

There are several sorts of minimum, so geometry optimization can have multiple forms. A *local* minimum-energy structure $\mathbf{r}_{\mathrm{local}}$ satisfies eq. 1, and it is also the case that all 'nearby' sets of coordinates have a higher potential energy. There may however be another minimum-energy structure $\mathbf{r}'$ with $V(\mathbf{r}') < V(\mathbf{r}_{\mathrm{local}})$. Finding the *global* minimum-energy structure $\mathbf{r}_{\mathrm{global}}$, which satisfies eq. 1 but furthermore lies lower in energy than all other local minima, is a hard problem that we will not tackle in this exercise.

Another comment to be made about the condition expressed by eq. 1 is that it is met by points other than minima, namely maxima and saddle-points of various orders. These points are distinguished by properties of the *second* derivatives of V with respect to the coordinates r_i, which form

[1] In this document, we use (usually lower-case) italic characters (e.g. r or x or V) to represent scalar quantities, while vectors are shown as (usually lower-case) bold characters (such as $\mathbf{x}$ or $\mathbf{r}$). Higher-order tensors and in particular matrices are shown as upper-case bold characters (e.g. $\mathbf{M}$). As scalar quantities, the *elements* of vectors or matrices are shown in italics (e.g. r_3 or M_{ij}).

the so-called Hessian matrix **H**:

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 V}{\partial r_1^2} & \frac{\partial^2 V}{\partial r_1 \partial r_2} & \cdots & \frac{\partial^2 V}{\partial r_1 \partial r_N} \\ \frac{\partial^2 V}{\partial r_2 \partial r_1} & \frac{\partial^2 V}{\partial r_2^2} & \cdots & \frac{\partial^2 V}{\partial r_2 \partial r_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 V}{\partial r_N \partial r_1} & \frac{\partial^2 V}{\partial r_N \partial r_2} & \cdots & \frac{\partial^2 V}{\partial r_N^2} \end{pmatrix} \tag{2}$$

At a minimum, all the eigenvalues of the Hessian are positive (depending on the choice of coordinates **r**, some may be equal to zero). At a maximum (which is of little relevance in computational chemistry), they are all negative, while at a saddle-point of order n, there are n negative eigenvalues and $N - n$ positive or zero ones. For the most part in this project, we will be interested in minima.

The topic for this exercise is twofold: first, we want to explore efficient algorithms for locating minima. Next, we want to discuss the impact that the choice of coordinates **r** used to describe the structure has on the speed of geometry optimization. As usual for this set of exercises, we try to gain a good understanding of the principles involved, and to provide the basis for an implementation for a 'real' problem. It should however be noted that the routines for geometry optimization that are commonly used in computational chemical software are usually quite complex; they incorporate many tests and failsafe procedures so that they work robustly and efficiently for a wide range of problems. Listing all such aspects would go beyond the scope of this exercise (and the knowledge of the author).

2 Algorithms for Geometry Optimization

There are many different algorithms that can be used to optimize structures. These can be classified in various different ways. One of these concerns the type of information about V that is computed at each point during the optimization:

1. Some algorithms use only the potential energy $V(\mathbf{r})$ itself. One very simple algorithm of this type is a form of Monte Carlo 'simulation': a starting structure is chosen, then a small random change is made, and the energy at the new point is evaluated. If it is lower than the energy at the starting point, the new point is retained, while if the energy increases, the new structure is discarded (this is effectively a Monte Carlo simulation with $T0$, at higher temperature, the algorithm will instead retain structures with higher energy than the previous one, albeit with a low probability). After many such trial steps, the energy will tend to reach a minimum.

2. Many optimization methods also make use of the *gradient* of the energy, the set of values $\partial V/\partial r_i$ for $i = 1 \ldots N$. This usually leads to much more efficient optimization methods, because it is often possible to calculate the gradient with little additional computational expense compared to calculating only V, while the gradient automatically provides information on a direction which if followed will lead to a decrease in energy. This is simply the direction opposite to the that of the gradient, $\mathbf{p} = -\partial V/\partial \mathbf{r}$. The *Steepest Descent* algorithm described below is a typical optimization method that uses only the gradient.

3. Second-order methods also make use of the Hessian matrix of second derivatives of the energy. This too can often be calculated at relatively low computational expense, though for quantum-chemical methods, the exact calculation of the Hessian is more demanding in terms of computer resources, and the code needed to compute it is also typically rather complicated. Still, where the Hessian is available, it contains much additional information compared to the gradient and energy. Multi-dimensional variants of the Newton method use the Hessian matrix.

4. While the gradient can be computed exactly for many methods, the Hessian is more demanding, as explained above. For this reason, geometry optimization with full computation of the Hessian matrix after each geometry change is not very commonly used. Insight into the second derivatives of the potential energy is however very useful for performing geometry optimization. For

this reason, a broad range of methods that include information on the Hessian in at least an approximate way, but without explicitly calculating it exactly, are commonly used. Such methods are referred to as quasi-Newton methods, and the so-called Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm that will be used here is one such method. This will be described in detail below.

2.1 Optimization in One Dimension

The problems that interest us relate to optimization of functions of many variables (remember that the coordinates $\mathbf{r}$ describing a molecule correspond to a vector), but it is useful to start by discussing optimization of a function of one variable – for example the bond length r in a diatomic molecule. We assume that we have some initial guess for the bond length, r_0, and we want to find r_{opt}, the equilibrium distance (or more precisely, the nearest *local* minimum). If as well as the target function $f(r)$, we know the first and second derivatives, then we can apply Newton's method. We start by assuming that a second-order Taylor expansion applies around r_0, so that:

$$\begin{aligned} f(r_0 + \Delta r) &= f(r_0) + \left.\frac{\mathrm{d}f}{\mathrm{d}r}\right|_{r_0} \Delta r + \frac{1}{2}\left.\frac{\mathrm{d}^2 f}{\mathrm{d}r^2}\right|_{r_0} \Delta r^2 + \mathcal{O}(\Delta r^3) \\ &= f(r_0) + f'(r_0)\Delta r + \frac{1}{2} f''(r_0)\Delta r^2 + \mathcal{O}(\Delta r^3) \end{aligned} \tag{3}$$

Obviously this expression is most accurate for small Δr; the leading-order additional term depends on the third power Δr^3. If we can neglect this term, then an extremum (which will usually be a minimum) of the function will occur when the first derivative with respect to r (or equivalently Δr) of the sum of the first three terms in the expansion is equal to zero:

$$\frac{\mathrm{d}}{\mathrm{d}\Delta r}\left(f(r_0) + f'(r_0)\Delta r + \frac{1}{2} f''(r_0)\Delta r^2 \right) = f'(r_0) + f''(r_0)\Delta r = 0 \tag{4}$$

In other words:

$$\Delta r_{\text{opt}} \approx -\frac{f'(r_0)}{f''(r_0)} \quad \text{and} \quad r_{\text{opt}} \approx r_0 - \frac{f'(r_0)}{f''(r_0)} \tag{5}$$

This is not an exact expression, unless f really is a quadratic function. For other cases, this equation can be used to provide a good estimate r_1 (improved compared to r_0) for the location of the extremum (the formula can predict either a maximum or a minimum, depending on the sign of $f''(r_0)$, but of course in geometry optimization, a minimum will most likely be what we are looking for). In turn, the same equation can be applied using r_1, $f'(r_1)$ and $f''(r_1)$ to yield a refined estimate r_2, and so on for r_k and r_{k+1}, with convergence to a minimum reached once r_k and r_{k+1} (and/or $f(r_k)$ and $f(r_{k+1})$) no longer differ from each other, and/or once the magnitude of the remaining gradient $f'(r_k)$ is small enough.

In case it is not possible to calculate the second derivative exactly, one can estimate it, by using the change in the first derivative for two values of r, r_{k-1} and r_k. This yields the following expression:

$$f''(r_k) \approx f''(r_{k+1}) \approx f''(r_{k-1}) \approx \frac{f'(r_k) - f'(r_{k-1})}{r_k - r_{k-1}} \Rightarrow r_{k+1} = r_k - f'(r_k)\frac{r_k - r_{k-1}}{f'(r_k) - f'(r_{k-1})} \tag{6}$$

In this case, the first two values of r to be used, r_0 and r_1, need to be obtained by making a small change in r based on some estimate of the second derivative. Note also that in case the function $f(r)$ is very "ill-behaved" (meaning that the second derivative is far from constant, so eq. 3 is only valid for very small values of Δr), the Newton method and the approximate form of eq. 6 may not lead to convergence and one may need to enforce some kind of maximum step size $r_{k+1} - r_k$ so as to avoid divergence.

2.2 Steepest Descent Optimization

We now consider optimization in many dimensions, e.g. optimization of the structure of a molecule with more than two atoms. As mentioned above, a simple algorithm for geometry optimization relies on using the gradient to identify the direction in which the potential energy decreases most steeply. More specifically, starting from a structure $\mathbf{r}_0$, a sequence of updated estimates of the minimum energy structure $\{\mathbf{r}_1, \mathbf{r}_2, \ldots\}$ are constructed, in each case by taking the nearest minimum along the direction given by the gradient at the previous point, *i.e.* one uses:

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha \left.\frac{\partial V(\mathbf{r})}{\partial \mathbf{r}}\right|_{\mathbf{r}_k} = \mathbf{r}_k + \alpha \mathbf{p}_k \tag{7}$$

In this equation, α is a positive scalar. As mentioned above, in steepest descent optimization, α is usually chosen so that $V(\mathbf{r}_k + \alpha \mathbf{p}_k)$ reaches a minimum along the direction $\mathbf{p}_k$. This is known as performing a **line search**. Finding such a local minimum along $\mathbf{p}_k$ requires multiple evaluations of the energy, and since evaluating the gradient often only costs a small amount of time compared to calculating the energy, it can be more efficient to simply perform a rough optimization along the direction $\mathbf{p}_k$ (possibly with just one step) before choosing a new direction to search along. Several recipes are available for defining a suitable "loosely" optimized outcome for a line search. These rules are generally expressed in terms of the step direction $\mathbf{p}$ and the gradient direction ∇V. In the case of steepest descent optimization, these directions are of course parallel, with $\mathbf{p} = -\nabla V$, but for some other optimization methods, they can be different. A commonly-used requirement for a rough line search is that α should satisfy the so-called "Wolfe rules", whereby:

$$\begin{aligned} V(\mathbf{r}_k + \alpha \mathbf{p}_k) &\leq V(\mathbf{r}_k) + c_1 \alpha \mathbf{p}_k \cdot \nabla V(\mathbf{r}_k) \\ -\mathbf{p}_k \cdot \nabla V(\mathbf{r}_k + \alpha \mathbf{p}_k) &\leq -c_2 \mathbf{p}_k \cdot \nabla V(\mathbf{r}_k) \end{aligned} \tag{8}$$

Where c_1 and c_2 are constants and $0 < c_1 < c_2 < 1$. The first of these conditions requires that the energy after the geometry update be 'significantly' lower than before it (it is always possible to guarantee that the energy will be lower if one takes a very small step – here one ensures that it drops 'enough'), with the stringency of this test depending on c_1. Here we will use a value of 0.1 for this constant. The second condition requires that the gradient along the search direction should become 'significantly' smaller. We will not use this second criterion here.

Regardless of whether or not a strict line search is performed, steepest descent optimization is a good way to loosely optimize a function, but for many problems, it is not an efficient way to perform full optimization, as depending on the initial structure and the degree of curvature of the surface in different directions, the optimization can tend to 'oscillate' and converge only slowly to the minimum. This is illustrated for a simple two-dimensional problem in figure 1.

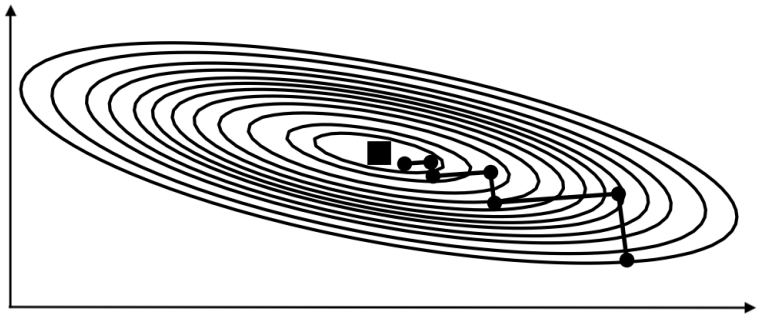

Figure 1: Oscillatory behaviour of the steepest descent optimization algorithm for a two-dimensional function shown as a contour plot. The starting structure $\mathbf{r}_0$ is connected to a set of six improved points (round dots, $\mathbf{r}_1 \ldots \mathbf{r}_6$)) reached by accurate line searches; the location of the minimum is shown by a square.

2.3 Multidimensional Newton Methods and the BFGS Algorithm

In the section on one-dimensional methods, we discussed the Newton method (eq. 5), where the first and second derivative can be combined to locate the minimum of a function if it is quadratic, or to obtain a good estimate of the minimum, in case the function is non-quadratic but one is close to the minimum. This can be extended to many-dimensional systems:

$$\mathbf{r}_{\text{opt}} \approx \mathbf{r}_0 + \Delta\mathbf{r} = \mathbf{r}_0 - \frac{\nabla V(\mathbf{r}_0)}{\mathbf{H}(\mathbf{r}_0)} = \mathbf{r}_0 - \mathbf{H}^{-1}(\mathbf{r}_0)\nabla V(\mathbf{r}_0) \tag{9}$$

Here $\mathbf{H}(\mathbf{r}_0)$ is the Hessian matrix of second derivatives of V with respect to the coordinates (eq. 2), and division by this matrix is equivalent to multiplying by its inverse.

If one is able to calculate the Hessian matrix, this multidimensional Newton method can be a very good way to optimize functions. However, in many applications in computational chemistry, calculating the Hessian is quite demanding, and it tends to have many small eigenvalues which make finding its inverse difficult. The BFGS algorithm, and related approaches, represent a means to sidestep both of these difficulties. The idea is basically to apply a multidimensional version of eq. 6, whereby the matrix of second derivatives is estimated by taking the change in the gradient of the function between two or more points. As the Hessian is a square matrix, with N^2 entries (though the Hessian is symmetric, so there are only $N(N+1)/2$ unique values in the matrix), while the gradient is a vector with N values, it is not possible to get an estimate of all elements of the Hessian from just two calculations of the gradient. Hence the approximate Hessian used will always be approximate. It is however possible to specify a way to approximate the *change* in the estimated Hessian based on the change in the gradient between two successive optimization points such that on the one hand the change in gradient is exactly accounted for, and on the other hand, the approximate Hessian is guaranteed to remain positive definite if it had this property to start with. Positive definite character for a matrix means that its eigenvalues are all positive, such that it can be inverted without problems. If the approximate Hessian is initially constructed as a diagonal matrix with positive diagonal elements, it will be positive definite.

The BFGS algorithm is one such optimal way to update the Hessian, while preserving positive-definite character and incorporating as much information as possible from the values of the gradients at successive steps during an optimization. This algorithm can be described as follows. One first chooses an initial structure $\mathbf{r}_0$, and makes an initial guess for the approximate Hessian, $\mathbf{B}_0 \approx \mathbf{H}(\mathbf{r}_0)$, then one iteratively carries out the following steps until convergence is reached (typically when the norm of the gradient drops below some pre-specified threshold):

1. Solve the multidimensional Netwon equation to define a search direction $\mathbf{p}_k$, where $\mathbf{B}_k\mathbf{p}_k = -\nabla V(\mathbf{r}_k)$.

2. Perform a line search along the direction $\mathbf{p}_k$, to yield an updated set of coordinates $\mathbf{r}_{k+1} = \mathbf{r}_k + \alpha_k\mathbf{p}_k$. In theory, one could simply use $\alpha_k = 1$, but it is often useful to instead perform a line search, at least approximately, as discussed above.

3. Define $\mathbf{s}_k = \alpha_k\mathbf{p}_k$ and hence $\mathbf{r}_{k+1} = \mathbf{r}_k + \mathbf{s}_k$.

4. Even if one does not perform a line search, it can be useful to check that the predicted step $\mathbf{s}_k$ is not too large. For example, one can define a maximum step length $\lambda_{\max}$ and check that $|\mathbf{s}_k| \leq \lambda_{\max}$, otherwise performing scaling $\mathbf{s}'_k = \mathbf{s}_k \times (\lambda_{\max}/|\mathbf{s}_k|)$.

5. Define $\mathbf{w}_k = \mathbf{B}_k\mathbf{s}_k$

6. Calculate the gradient at the new structure, and define: $\mathbf{y}_k = \nabla V(\mathbf{r}_{k+1}) - \nabla V(\mathbf{r}_k)$.

7. Calculate a new estimate of the Hessian, using:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \otimes \mathbf{y}_k}{\mathbf{y}_k \cdot \mathbf{s}_k} - \frac{\mathbf{w}_k \otimes \mathbf{w}_k}{\mathbf{s}_k \cdot \mathbf{w}_k} \tag{10}$$

In the final step, the update for $\mathbf{B}$, the symbol $\otimes$ denotes the *outer product* operation between two vectors. The outer product of two vectors $\mathbf{a}$ and $\mathbf{b}$ with respectively m and n elements is an $m \times n$ matrix $\mathbf{A}$ with elements A_{ij} equal to $a_i b_j$. The outer product can also be written as a matrix product of a vector with the transpose of a vector, (and the scalar product can be written as a matrix product of the transpose of a vector with a vector) so the above equation can also be written:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{w}_k \mathbf{w}_k^T}{\mathbf{s}_k^T \mathbf{w}_k} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k^T}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} \tag{11}$$

The second equality gives the BFGS Hessian update equation in a commonly-written form, which does not make use of the vector $\mathbf{w}$.

Under most conditions, the resulting matrix $\mathbf{B}_{k+1}$ will be positive definite and hence can be inverted. Some attention does need to be paid to how to treat approximate Hessians with very small positive eigenvalues. Also, the computational expense of inverting the approximate Hessian is not always negligible. For this reason, it is often useful to work with a reformulation of the algorithm in terms of the *inverse* of the approximate Hessian, $\mathbf{B}^{-1}$. In many introductions to the BFGS method, this is written as $\mathbf{H}$, but to avoid confusion with the Hessian, I will instead refer to this matrix as $\mathbf{M} \equiv \mathbf{B}^{-1}$. Step 1 above is then carried out by setting the search direction as $\mathbf{p}_k = -\mathbf{M}_k \nabla V(\mathbf{r}_k)$. And step 7 is replaced by the following update for the approximate inverse Hessian (where $\mathbf{v}_k = \mathbf{M}_k \mathbf{y}_k$):

$$\mathbf{M}_{k+1} = \mathbf{M}_k + \frac{(\mathbf{s}_k \cdot \mathbf{y}_k + \mathbf{y}_k \cdot \mathbf{v}_k)\mathbf{s}_k \otimes \mathbf{s}_k}{(\mathbf{s}_k \cdot \mathbf{y}_k)^2} - \frac{\mathbf{v}_k \otimes \mathbf{s}_k + \mathbf{s}_k \otimes \mathbf{v}_k}{\mathbf{s}_k \cdot \mathbf{y}_k} \tag{12}$$

This is the method you should implement and test in this programming exercise (using the modified version for steps 1 and 7 as just described).

3 Coordinate Systems and Geometry Optimization

Different coordinate systems can be used to represent molecular structures. One obvious choice is Cartesian coordinates, with the coordinate vector $\mathbf{r}$ then being $\mathbf{x} \equiv \{x_1, y_1, z_1, x_2, \ldots z_{n_{at}}\}$. This is a common choice for the coordinate system, but it is often not an ideal choice, as Cartesian coordinates are highly coupled. Consider a molecule with one bond A – B lying along the direction $x = y$, and assume that the current structure has a near-optimal distance r_{AB}. Assume that atom A needs to move due to interactions with other atoms in the molecule. A change in the x coordinate of atom A will lead to a change in r_{AB} unless the atom also undergoes an appropriate motion along the y direction. Hence most changes in one coordinate – here x_{A} – will be unfavourable unless they are accompanied by changes in other coordinates – here y_{A}.

So-called *internal* coordinates $\mathbf{q}$ are preferable in this regard: bond lengths, bond angles, and bond dihedrals are typically less coupled to one another than Cartesian coordinates are. Take a water molecule, described by $\{r_1, r_2, \theta\}$. A change in any of these three coordinates will have only modest coupling to changes in other coordinates. Working with less coupled coordinates tends to lead to faster convergence of the geometry optimization. One can never remove all coupling (for example non-bonded interactions such as dispersion and Pauli exchange-repulsion lead to some coupling), but internal coordinates are usually superior in this respect to Cartesian coordinates.

Internal coordinates have, however, a different problem: a natural choice of the set of coordinates $\mathbf{q}$ tends to be *redundant*, i.e. there tend to be more coordinates than degrees of freedom. Consider a molecule of methane CH_4. With its five atoms, there are intrinsically nine degrees of freedom ($3 \times 5 - 6$) corresponding to the internal arrangements of the atoms within the molecule. Yet a natural set of internal coordinates, the four C-H bond lengths and six H-C-H angles, has ten coordinates and hence overdetermines the structure.[2] This problem can be dealt with in two ways: either one picks a subset

[2] One can note that there are also 'too many' Cartesian coordinates in this case, namely 15 coordinates. However, these can all be freely varied and any choice of these Cartesian coordinates will yield a valid structure. The redundancy in the coordinates merely has as consequence that there exist multiple different choices for the coordinates that all correspond to the same 'internal' molecular structure, but rotated and/or translated with respect to one another. In contrast, if one assigns each of the six bond angles independently of each other, one can have a set of angles that are fundamentally incompatible with each other, with no corresponding possible structure.

of the possible internal coordinates that exactly spans the possible structures but without redundancy, or one picks a method to remove the redundancy as part of the optimization method. The former choice corresponds *e.g.* to the use of so-called Z-matrix coordinates, and will not be considered here. The latter choice is the one that will be used here, and the required algebra will now be described.

3.1 Working with Redundant Coordinate Sets

We consider a molecule described by the Cartesian coordinates $\mathbf{x}$ (a set with dimension $n_x = 3n_{at}$), and the n_q redundant internal coordinates $\mathbf{q}$. In our case, we will be working only with saturated hydrocarbons, so the latter set will include C-C and C-H bond distances, various bond angles (H-C-H, H-C-C, and C-C-C) and dihedral angles around C-C bonds. We will determine the set of coordinates $\mathbf{q}$ upon initializing the optimization, by defining all bonds, and we will then choose for $\mathbf{q}$ the redundant set with all bond lengths, all of the $6n_{\mathrm{C}}$ bonding angles defined at carbon atoms, and all of the $9n_{\mathrm{C\text{-}C}}$ dihedral angles around each of the $n_{\mathrm{C\text{-}C}}$ carbon-carbon bonds present. Typically, n_q is (much) larger than n_x. *E.g.* for two of the test set of molecules to be studied, ethane and methylcyclohexane, n_x is respectively 24 and 63, n_{C} is respectively 2 and 7, $n_{\mathrm{C\text{-}C}}$ is respectively 1 and 7, and n_q is respectively $7 + 12 + 9 = 28$ and $21 + 42 + 63 = 126$. In principle, the forcefield to be used can treat clusters containing multiple molecules (e.g. liquid methane). However, in that case, the set of internal coordinates to be used needs to include some intermolecular coordinates. No such case will need to be treated for this exercise, i.e. the code only needs to describe single molecule. The forcefield can also describe cyclic molecules such as cyclohexane. Some special treatment is needed to handle dihedral terms when using three-membered rings (cyclopropanes) but again, we will sidestep this problem by excluding cyclopropanes from the set of target molecules (cyclobutanes and larger cycles should be allowed for).

It will frequently be necessary to compute the internal coordinates from knowing the Cartesian coordinates. Formulas for doing this are provided further down.

Assume that an initial structure $\mathbf{x}$ is changed by a small (quasi-infinitesimal) amount $\delta\mathbf{x}$. Then $\mathbf{q}$ will change by a small amount $\delta\mathbf{q}$, and:

$$\delta\mathbf{q} = \mathbf{B}\delta\mathbf{x} \tag{13}$$

Here $\mathbf{B}$ is a $(n_q \times n_x)$ matrix containing the partial derivatives $\partial q_j/\partial x_i$, for which expressions will be given below. This matrix plays an important role in the theory of rotational and vibrational spectroscopy, of chemical reactions, and of geometry optimization, and is traditionally referred to as the **Wilson B Matrix**, named after the theoretical chemist E. Bright Wilson.[3]

This B matrix can be used to convert the gradient of the energy with respect to the internal coordinates $\mathbf{g}_q$ into the gradient in terms of the Cartesian coordinates, following:

$$\mathbf{g}_x = \mathbf{B}^T\mathbf{g}_q \tag{14}$$

Here $\mathbf{g}_q$ is the vector of length n_q with the gradient elements $\{\partial V/\partial q_j\}$ and $\mathbf{g}_x$ is the vector (of length n_x) of partial derivatives $\{\partial V/\partial x_i\}$. Typically when doing geometry optimization, one will wish to be able to perform also the reverse transformation, from Cartesian coordinates to internal coordinates. This is not trivial because the matrix $\mathbf{B}$ is not square and cannot simply be inverted. It *does* however have a *generalized* inverse, which we will write as $\mathbf{B}^{-1}$, and which is obtained as:

$$\mathbf{B}^{-1} = \mathbf{G}^{-}\mathbf{B} \tag{15}$$

$\mathbf{G}^{-}$ is also a generalized inverse of a matrix, in this case of the matrix $\mathbf{G}$, which is the matrix product of the $n_q \times n_x$ matrix $\mathbf{B}$ with its own transpose (which is an $n_x \times n_q$ matrix), so $\mathbf{G}$ is square and has dimensions $n_q \times n_q$:

$$\mathbf{G} = \mathbf{B}\mathbf{B}^T \tag{16}$$

Because the coordinate set $\mathbf{q}$ is redundant, $\mathbf{G}$ has a set of eigenvalues that are equal to zero. This explains why we need to take a *generalized* inverse of $\mathbf{G}$. In fact, $\mathbf{G}$ will typically have $3n_{at} - 6$

[3] Be careful not to confuse this rectangular matrix with the (square) approximate Hessian $\mathbf{B}$ used in BFGS theory.

non-zero eigenvalues, and $n_q - (3n_{at} - 6)$ zero eigenvalues, which can be written as:

$$\mathbf{V}^T\mathbf{G}\mathbf{V} = \begin{bmatrix} \mathbf{\Lambda} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{17}$$

Here $\mathbf{\Lambda}$ is a diagonal matrix containing the $3n_{at} - 6$ non-zero eigenvalues of $\mathbf{G}$, while the blocks $\mathbf{0}$ simply contain zeroes, and $\mathbf{V}$ is a matrix containing the eigenvectors of $\mathbf{G}$. The generalized inverse of $\mathbf{G}$ is then:

$$\mathbf{G}^- = \mathbf{V}\begin{bmatrix} \mathbf{\Lambda}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\mathbf{V}^T \tag{18}$$

An equivalent way of expressing this is to say that $\mathbf{G}^-$ is given as a sum of the outer products of the eigenvectors with non-zero eigenvalues of $\mathbf{G}$ with themselves, with each outer product being multiplied by the inverse of the corresponding eigenvalue.

The generalized inverse of $\mathbf{B}$ provides a means to convert $\mathbf{g}_x$ to $\mathbf{g}_q$:

$$\mathbf{g}_q = \mathbf{B}^{-1}\mathbf{g}_x = \mathbf{G}^-\mathbf{B}\mathbf{g}_x \tag{19}$$

To perform geometry optimization in terms of internal coordinates, one applies the set of steps defined in section 2.3, slightly adapted to the following form: First, one chooses an initial structure $\mathbf{x}_0$, and converts it to an initial structure $\mathbf{q}_0$. One also makes an initial guess for the inverse Hessian in the space of internal coordinates, $\mathbf{M}_{q,0}$. Then one iteratively carries out the following steps:

1. Calculate the gradient in Cartesian coordinates $\mathbf{g}_{x,k}$, the Wilson B matrix $\mathbf{B}_k$ and its generalized inverse $\mathbf{G}_k^-$, and hence, using eqs. 19, obtain the gradient $\mathbf{g}_{q,k}$. Note that although many of the different energy terms used in molecular mechanics depend on internal coordinates, one cannot simply perform differentiation of those expressions to obtain $\mathbf{g}_q$. This is because of energy terms that couple multiple internal energy coordinates, and especially the Lennard-Jones non-bonded terms, see below. Hence $\mathbf{g}_q$ must indeed be obtained from $\mathbf{g}_x$ and transformation thereof.

2. Obtain the search direction in internal coordinates $\mathbf{p}_{q,k}$, where $\mathbf{p}_{q,k} = -\mathbf{M}_{q,k}\mathbf{g}_{q,k}$.

3. Perform a line search along the direction $\mathbf{p}_{q,k}$, to yield an updated set of coordinates $\mathbf{q}_k + \alpha_k\mathbf{p}_{q,k}$. When using internal coordinates, it is often possible to carry out the so-called "full Newton step", i.e. to use $\alpha_k = 1$, without needing to perform a line search (though it may be desirable to scale the step if it is predicted to have a very large magnitude). For scaling, calculate the average length of $\mathbf{p}_{q,k}$, $l_{q,k} = \sqrt{(\sum_i p_{q,k}^2)/n_q}$, and make sure this is smaller than $\lambda_{\max}$, the maximum step length, otherwise set $\mathbf{p}_{q,k} = \mathbf{p}_{q,k} \times (\lambda_{\max}/l_{q,k})$.

4. Define $\mathbf{s}_{q,k} = \alpha_k\mathbf{p}_{q,k}$ and hence $\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{s}_{q,k}$.

5. Convert the obtained updated structure in internal coordinates $\mathbf{q}_{k+1}$ into an updated structure in Cartesian coordinates $\mathbf{x}_{k+1}$ (the details of how to perform this non-trivial step are given below).

6. Calculate the gradients $\mathbf{g}_{x,k+1}$ and $\mathbf{g}_{q,k+1}$ at the new structure,[4] and define: $\mathbf{y}_{q,k} = \mathbf{g}_{q,k+1} - \mathbf{g}_{q,k}$, and $\mathbf{v}_{q,k} = \mathbf{M}_{q,k}\mathbf{y}_{q,k}$):

7. Calculate a new estimate of the inverse Hessian, using:

$$\mathbf{M}_{q,k+1} = \mathbf{M}_{q,k} + \frac{(\mathbf{s}_{q,k}\cdot\mathbf{y}_{q,k} + \mathbf{y}_{q,k}\cdot\mathbf{v}_{q,k})\mathbf{s}_{q,k}\otimes\mathbf{s}_{q,k}}{(\mathbf{s}_{q,k}\cdot\mathbf{y}_{q,k})^2} - \frac{\mathbf{v}_{q,k}\otimes\mathbf{s}_{q,k} + \mathbf{s}_{q,k}\otimes\mathbf{v}_{q,k}}{\mathbf{s}_{q,k}\cdot\mathbf{y}_{q,k}} \tag{20}$$

This is repeated for enough cycles, until the norm of the gradient falls below some predefined threshold. The test can be performed for the gradient in Cartesian coordinates.

Step 5, where the new structure in internal coordinates is converted into a new structure in Cartesian coordinates, is not trivial. In fact, when using redundant internal coordinates, as we do

[4] For this purpose one should use the actual new structure $\mathbf{q}_{k+1}$ obtained as described below.

here, there is no guarantee at all that a given set of values of the internal coordinates $\mathbf{q}$ maps on to any possible set of Cartesian coordinates. This was already discussed above for the case of methane: in principle, step 4 can lead to a set of values of the six H-C-H angles that cannot be mapped onto any possible set of Cartesian coordinates. One typically finds that geometry update steps lead to values of $\mathbf{q}$ that do quite closely match a given set of coordinates $\mathbf{x}$, but this is usually not an exact match. Hence step 5 above could perhaps more accurately be described as finding the closest-matched set of Cartesian coordinates.

This search for the optimal updated Cartesian coordinates needs to be performed iteratively.

a. We define $\mathbf{s}_{q,k}^{(0)} \equiv \mathbf{q}_{k+1} - \mathbf{q}_k$ (see step 4 above) as the desired change to be carried out on the internal coordinates.

b. We define $\mathbf{x}_{k+1}^{(0)} \equiv \mathbf{x}_k$ (the previous structure) as our initial guess for the new Cartesian coordinates.

c. Then we iterate. At each iteration, $\mathbf{x}_{k+1}^{(j+1)} = \mathbf{x}_{k+1}^{(j)} + \mathbf{B}^T\mathbf{G}^{-}\mathbf{s}_{q,k}^{(j)}$ is a refined guess for the Cartesian coordinates.

d. These $\mathbf{x}_{k+1}^{(j+1)}$ can be used to compute the corresponding internals, $\mathbf{q}_{k+1}^{(j+1)}$.

e. $\mathbf{s}_{q,k}^{(j+1)} = \mathbf{q}_{k+1} - \mathbf{q}_{k+1}^{(j+1)}$ is now the new update that we want to carry out for the internal coordinates, and one can repeat from step c. above.

f. In ideal cases, this difference would become smaller and smaller until some $\mathbf{x}_{k+1}^{(j)}$ would be found which would correspond to a set of values $\mathbf{q}_{k+1}^{(j)}$ that are identical to the desired values $\mathbf{q}_{k+1}$. In practice, this will usually not occur, but in most cases, the set of Cartesian coordinates $\mathbf{x}_{k+1}^{(j)}$ will converge, i.e. the difference between them (measured e.g. as the largest absolute difference $|x_{k+1,i}^{(j+1)} - x_{k+1,i}^{(j)}|$) will tend to zero. Once this difference falls below some threshold, we can take the last set of coordinates $x_{k+1,i}^{(j+1)}$ as the new structure.

The approximate Hessian update step 7 above should be performed using the change in internal coordinates $\mathbf{s}_{q,k}$ that was actually performed, rather than the initially suggested change from step 4.

4 Evaluating the Potential Energy, its Gradient, and the B Matrix

In order to be able to do geometry optimization, one needs a function that returns the energy $V(\mathbf{r})$ and its gradient elements $\partial V/\partial r_i$. Our goal will be to perform geometry optimization using Cartesian coordinates but also in terms of internal coordinates, hence the Wilson B matrix will also be needed, and expressions for that are given also. Even if the geometry optimization is performed using Cartesian coordinates only, the gradient elements are computed using intermediate quantities that resemble the elements of the Wilson B matrix, so you should read the corresponding section even if you are not yet doing optimization with internal coordinates.

4.1 Molecular Mechanics Energy for Saturated Alkanes

For this exercise, we have chosen to focus on a simple energy function that is rapid to evaluate and simple to code, namely a basic molecular mechanics expression. For additional simplicity, we choose to focus only on the case of saturated alkanes (as discussed above, we exclude non-covalently bound *clusters* of multiple alkane molecules, and also exclude cyclopropane ring-containing molecules. The energy expression is:

$$V(\mathbf{r}) = \sum_{\text{bonds}} k_b(r_b - r_0)^2 + \sum_{\text{angles}} k_a(\theta_a - \theta_0)^2 + \sum_{\text{dihed}} A_\phi(1 + \cos n\phi) + 4\sum_{i,j} \epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] \quad (21)$$

In this expression, there are four types of terms: the first relate to stretching of bonds, with k_b being the corresponding force constant, r_0 the equilibrium distance, and r_b the actual distance. The second set of terms relate to bending occurring between two bonds to a same atom, with k_a, θ_0 and θ_a the force constant, the ideal angle, and the actual angle. The third term gives the dependence on the dihedral angle ϕ formed by four connected atoms A-B-C-D. The dihedral angle ϕ is the angle between the vectors normal to the planes ABC and BCD, and in some cases it may be useful to introduce an offset ϕ_0 so that the cosine term in eq. 21 is replaced by $\cos n(\phi - \phi_0)$, though in this exercise, we will not need to use such an offset. n is an integer that determines how often the dihedral energy term oscillates for a full rotation around the bond in question, while A_ϕ is the associated barrier height. Finally, the fourth term is a sum over all pairs of atoms *except* those (A-B) which are bonded to each other, or which are both bonded to a third atom C (A-C-B), which accounts for dispersion attractive interactions and for repulsive exchange interactions. This term is also referred to as describing the "van der Waals" interactions between non-bonded atoms, or as the "Lennard-Jones" term, as its use was introduced by John Lennard-Jones.

To be more specific, in this exercise we will consider only saturated hydrocarbons, so there will be only two types of atom, C and H, which reduces the number of parameters to be considered. We will also use a rather simple set of forcefield parameters, summarized in Table 1, corresponding to the so-called 'tiny' forcefield as available in the Tinker molecular mechanics package.

Table 1: Molecular mechanics parameters in the 'tiny' forcefield for saturated hydrocarbons.

	Atom(s)	Value	Units		Atom(s)	Value	Units
k_b	C-C	300.	kcal mol^{-1}Å^{-2}	r_0	C-C	1.53	Å
k_b	C-H	350.	kcal mol^{-1}Å^{-2}	r_0	C-H	1.11	Å
k_a	H-C-H	35.	kcal mol^{-1}radian^{-2}	θ_0	H-C-H	109.50	degrees
k_a	H-C-C	35.	kcal mol^{-1}radian^{-2}	θ_0	H-C-C	109.50	degrees
k_a	C-C-C	60.	kcal mol^{-1}radian^{-2}	θ_0	H-C-C	109.50	degrees
A_ϕ	X-C-C-X	0.3	kcal mol^{-1}	n	X-C-C-X	3	/
ϵ_i	H	0.03	kcal mol^{-1}	σ_i	H	1.20	Å
ϵ_i	C	0.07	kcal mol^{-1}	σ_i	C	1.75	Å
A_{ij}	H,H	4382.44	kcal mol^{-1}Å^{12}	B_{ij}	H,H	22.932	kcal mol^{-1}Å^6
A_{ij}	H,C	64393.99	kcal mol^{-1}Å^{12}	B_{ij}	H,C	108.644	kcal mol^{-1}Å^6
A_{ij}	C,C	946181.74	kcal mol^{-1}Å^{12}	B_{ij}	C,C	514.714	kcal mol^{-1}Å^6

The parameters for all dihedral angles around a central C-C bond are taken to be the same, irrespective of whether one has a sequence of atoms X-C-C-X with all carbon atoms (C-C-C-C) or with one or two hydrogens (H-C-C-C, C-C-C-H or H-C-C-H). It can be noted that the van der Waals parameters are not given in a form that can directly be used in eq. 21, as the ϵ and σ parameters are given as atomic parameters $\{\epsilon_i, \sigma_i\}$ rather than atom-pair parameters $\{\epsilon_{ij}, \sigma_{ij}\}$. This is common in the construction of forcefields, as there usually are far too many atom type pairs in order to be able to obtain parameters for each of them. Instead, atom type parameters are obtained, and for pairs of atom types, an average value is used. Two types of averaging are commonly used, one of which is the obvious arithmetic averaging. In the 'tiny' forcefield, a co-called 'geometric' averaging protocol is instead used:

$$\epsilon_{ij} = \sqrt{\epsilon_i \epsilon_j} \quad ; \quad \sigma_{ij} = 2\sqrt{\sigma_i \sigma_j} \tag{22}$$

This gives $\epsilon_{\mathrm{HH}} = 0.03$ kcal mol^{-1}, $\epsilon_{\mathrm{HC}} = 0.045826$ kcal mol^{-1}, and $\epsilon_{\mathrm{CC}} = 0.07$ kcal mol^{-1}, with $\sigma_{\mathrm{HH}} = 2.40$ Å, $\sigma_{\mathrm{HC}} = 2.8983$ Å, and $\sigma_{\mathrm{CC}} = 3.50$ Å. It is also possible to re-express the van der Waals interaction between two atoms and the associated parameters in yet another way, where $A_{ij} = 4\epsilon_{ij}\sigma_{ij}^{12}$ and $B_{ij} = 4\epsilon_{ij}\sigma_{ij}^{6}$ (these A_{ij} and B_{ij} values are also in Table 1):

$$V_{ij}^{\mathrm{vdW}} = \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} \tag{23}$$

4.2 Internal Coordinates in Terms of Cartesian Coordinates

The expressions above are in terms of bond distances and angles. It is convenient to have expressions for these internal coordinates in terms of the Cartesian coordinates of each of the atoms involved in the bond, bending angle, dihedral or pair of non-bonded atoms. This is also helpful when deriving the expressions for the gradient elements, below. The algebra involved is tedious enough that it is useful to provide all formulas here.

For the distance between two atoms A and B, as needed for the bond-stretching and van der Waals terms, one has:

$$r_{\mathrm{AB}} = \sqrt{(x_{\mathrm{A}} - x_{\mathrm{B}})^2 + (y_{\mathrm{A}} - y_{\mathrm{B}})^2 + (z_{\mathrm{A}} - z_{\mathrm{B}})^2} \tag{24}$$

For the angle θ formed by three atoms A, B, and C (where B is taken to be the central atom), one has the following, where $\mathbf{r}_{\mathrm{BA}}$ is the vector going from the position of atom B to that of atom A, with elements $\{x_{\mathrm{A}} - x_{\mathrm{B}}, y_{\mathrm{A}} - y_{\mathrm{B}}, z_{\mathrm{A}} - z_{\mathrm{B}}\}$, $\mathbf{r}_{\mathrm{BA}} \cdot \mathbf{r}_{\mathrm{BC}}$ is the dot product of the two vectors, and $r_{\mathrm{AB}} = r_{\mathrm{BA}}$ is the **norm** of r_{AB} *i.e.* the *distance* from A to B:

$$\theta = \arccos \frac{\mathbf{r}_{\mathrm{BA}} \cdot \mathbf{r}_{\mathrm{BC}}}{r_{\mathrm{AB}} r_{\mathrm{BC}}} \tag{25}$$

Consider four atoms A, B, C and D connected in that order. The corresponding dihedral angle ϕ along the bond B – C is the angle between the vectors that are orthogonal to the planes A-B-C ($\mathbf{t}$; we write the norm of a vector as $|\mathbf{t}|$ or simply t) and B-C-D ($\mathbf{u}$). One gets the following expression for the dihedral angle ϕ:

$$\mathbf{t} = \mathbf{r}_{\mathrm{AB}} \times \mathbf{r}_{\mathrm{BC}} \quad ; \quad \mathbf{u} = \mathbf{r}_{\mathrm{BC}} \times \mathbf{r}_{\mathrm{CD}} \quad ; \quad \phi = \arccos \frac{\mathbf{t} \cdot \mathbf{u}}{tu} \tag{26}$$

In this expression, the symbol $\times$ denotes the 'cross-product' or vector product of the two indicated vectors, or:

$$\mathbf{r}_1 \times \mathbf{r}_2 = \begin{pmatrix} y_1 z_2 - y_2 z_1 \\ z_1 x_2 - z_2 x_1 \\ x_1 y_2 - x_2 y_1 \end{pmatrix} \tag{27}$$

The simple arccos function in most programming languages typically returns a value of the angle between 0 and π – negative angles are hence not returned, as $\cos x = \cos(-x)$. If one wishes to compute a dihedral angle including its sign, which is needed to get gradient and B matrix elements correctly, a more complicated approach is needed. In that case, one can use the following expression based on the arctangent function with two arguments:

$$\begin{aligned} &\mathbf{t} = \mathbf{r}_{\mathrm{AB}} \times \mathbf{r}_{\mathrm{BC}} \quad ; \quad \mathbf{u} = \mathbf{r}_{\mathrm{BC}} \times \mathbf{r}_{\mathrm{CD}} \quad ; \quad \mathbf{v} = \mathbf{t} \times \mathbf{u} \\ &\cos\phi = \frac{\mathbf{t} \cdot \mathbf{u}}{tu} \quad ; \quad \sin\phi = \frac{\mathbf{r}_{\mathrm{BC}} \cdot \mathbf{v}}{r_{\mathrm{BC}} tu} \quad ; \quad \phi = \arctan(\sin\phi, \cos\phi) \end{aligned} \tag{28}$$

The two-argument arctan function in this expression is available in many programming languages (including Fortran, Python, Julia, Rust – sometimes it is called something like `atan2`), and returns an angle between $-\pi$ and π.

4.3 The Wilson B Matrix

The next set of expressions that we will want to use are those giving the derivative of the individual energy terms in eq. 21 with respect to the Cartesian coordinates. As we will see, these gradient terms are related to the elements of the Wilson B matrix (containing as elements derivatives of the internal coordinates with respect to the Cartesian coordinates), so we will first give expressions for these elements. This matrix has dimensions $n_q \times n_x$, and is quite sparse, since each internal coordinate typically only depends on 6 (bond length), 9 (bond angle) or 12 (dihedral) Cartesian coordinates.

We first consider the derivative of a bond length r_{AB} with respect to the Cartesian coordinates of the two atoms A and B involved. We have:

$$\frac{\partial r_{\mathrm{AB}}}{\partial x_A} = \frac{1}{2\sqrt{(x_{\mathrm{B}} - x_{\mathrm{A}})^2 + (y_{\mathrm{B}} - y_{\mathrm{A}})^2 + (z_{\mathrm{B}} - z_{\mathrm{A}})^2}} \times -2(x_{\mathrm{B}} - x_{\mathrm{A}}) = \frac{x_{\mathrm{A}} - x_{\mathrm{B}}}{r_{\mathrm{AB}}} \tag{29}$$

The expressions for the derivatives with respect to the y and z coordinates, and for those with respect to the coordinates of atom B, are similar. We can write the following compact expression for the vectors containing all such B-matrix elements:

$$\left\{\frac{\partial r_{\mathrm{AB}}}{\partial x_{\mathrm{A}}}, \frac{\partial r_{\mathrm{AB}}}{\partial y_{\mathrm{A}}}, \frac{\partial r_{\mathrm{AB}}}{\partial z_{\mathrm{A}}}\right\} = \frac{\mathbf{r}_{\mathrm{BA}}}{r_{\mathrm{AB}}} = -\left\{\frac{\partial r_{\mathrm{AB}}}{\partial x_{\mathrm{B}}}, \frac{\partial r_{\mathrm{AB}}}{\partial y_{\mathrm{B}}}, \frac{\partial r_{\mathrm{AB}}}{\partial z_{\mathrm{B}}}\right\} \tag{30}$$

For angles, a similar derivation can be performed, where the form of the expression obtained is different for the central atom B and the terminal atoms A and C. Here we give only the results:

$$\begin{gathered}
\mathbf{p} = \mathbf{r}_{\mathrm{BA}} \times \mathbf{r}_{\mathrm{BC}} \quad ; \quad \left\{\frac{\partial \theta_{\mathrm{ABC}}}{\partial x_{\mathrm{B}}}, \frac{\partial \theta_{\mathrm{ABC}}}{\partial y_{\mathrm{B}}}, \frac{\partial \theta_{\mathrm{ABC}}}{\partial z_{\mathrm{B}}}\right\} = \frac{-\mathbf{r}_{\mathrm{BA}} \times \mathbf{p}}{r_{\mathrm{AB}}^2 \cdot p} + \frac{\mathbf{r}_{\mathrm{BC}} \times \mathbf{p}}{r_{\mathrm{BC}}^2 \cdot p} \\
\left\{\frac{\partial \theta_{\mathrm{ABC}}}{\partial x_{\mathrm{A}}}, \frac{\partial \theta_{\mathrm{ABC}}}{\partial y_{\mathrm{A}}}, \frac{\partial \theta_{\mathrm{ABC}}}{\partial z_{\mathrm{A}}}\right\} = \frac{\mathbf{r}_{\mathrm{BA}} \times \mathbf{p}}{r_{\mathrm{AB}}^2 \cdot p} \quad ; \quad \left\{\frac{\partial \theta_{\mathrm{ABC}}}{\partial x_{\mathrm{C}}}, \frac{\partial \theta_{\mathrm{ABC}}}{\partial y_{\mathrm{C}}}, \frac{\partial \theta_{\mathrm{ABC}}}{\partial z_{\mathrm{C}}}\right\} = \frac{-\mathbf{r}_{\mathrm{BC}} \times \mathbf{p}}{r_{\mathrm{BC}}^2 \cdot p}
\end{gathered} \tag{31}$$

Finally, for dihedral angles, one obtains different expressions for the terminal atoms A and D and the 'internal' atoms B and C. The expressions make use of the vectors $\mathbf{t}$ and $\mathbf{u}$, respectively, already used above in eq. 28:

$$\begin{gathered}
\mathbf{t} = \mathbf{r}_{\mathrm{AB}} \times \mathbf{r}_{\mathrm{BC}} \quad ; \quad \mathbf{u} = \mathbf{r}_{\mathrm{BC}} \times \mathbf{r}_{\mathrm{CD}} \\
\left\{\frac{\partial \phi}{\partial x_{\mathrm{A}}}, \frac{\partial \phi}{\partial y_{\mathrm{A}}}, \frac{\partial \phi}{\partial z_{\mathrm{A}}}\right\} = \left(\frac{\mathbf{t} \times \mathbf{r}_{\mathrm{BC}}}{t^2 \cdot r_{\mathrm{BC}}}\right) \times \mathbf{r}_{\mathrm{BC}} \quad ; \quad \left\{\frac{\partial \phi}{\partial x_{\mathrm{D}}}, \frac{\partial \phi}{\partial y_{\mathrm{D}}}, \frac{\partial \phi}{\partial z_{\mathrm{D}}}\right\} = \left(\frac{-\mathbf{u} \times \mathbf{r}_{\mathrm{BC}}}{u^2 \cdot r_{\mathrm{BC}}}\right) \times \mathbf{r}_{\mathrm{BC}} \\
\left\{\frac{\partial \phi}{\partial x_{\mathrm{B}}}, \frac{\partial \phi}{\partial y_{\mathrm{B}}}, \frac{\partial \phi}{\partial z_{\mathrm{B}}}\right\} = \mathbf{r}_{\mathrm{AC}} \times \left(\frac{\mathbf{t} \times \mathbf{r}_{\mathrm{BC}}}{t^2 \cdot r_{\mathrm{BC}}}\right) + \left(\frac{-\mathbf{u} \times \mathbf{r}_{\mathrm{BC}}}{u^2 \cdot r_{\mathrm{BC}}}\right) \times \mathbf{r}_{\mathrm{CD}} \\
\left\{\frac{\partial \phi}{\partial x_{\mathrm{C}}}, \frac{\partial \phi}{\partial y_{\mathrm{C}}}, \frac{\partial \phi}{\partial z_{\mathrm{C}}}\right\} = \left(\frac{\mathbf{t} \times \mathbf{r}_{\mathrm{BC}}}{t^2 \cdot r_{\mathrm{BC}}}\right) \times \mathbf{r}_{\mathrm{AB}} + \mathbf{r}_{\mathrm{BD}} \times \left(\frac{-\mathbf{u} \times \mathbf{r}_{\mathrm{BC}}}{u^2 \cdot r_{\mathrm{BC}}}\right)
\end{gathered} \tag{32}$$

It should be noted that dihedral angles become undefined if either the three atoms A-B-C or the atoms B-C-D are collinear, as the vector products $\mathbf{t}$ or $\mathbf{u}$ are then undefined. This will not happen in the stiff hydrocarbon molecules studied here, but some care is needed to handle such cases correctly in more general geometry optimization problems.

4.4 The Gradient of the Energy

We close this section on the evaluation of the molecular mechanics forcefield by providing expressions for the derivative of the energy with respect to the Cartesian coordinates of the atoms. This is the Cartesian gradient of the energy already referred to above. Equation 21 provides the MM energy as a sum of terms, so the Cartesian gradient is also a sum of terms. Expressions for each of those terms can be obtained by applying the chain rule to eq. 21 (or possibly eq. 23 for the van der Waals/Lennard-Jones terms), making use of the partial derivatives that make up the Wilson B matrix, from the previous section. As already mentioned, the Wilson B matrix is highly sparse, which means that each individual energy contribution from eq. 21 only contributes to the gradient of a few atoms. Still, there are many energy terms, and there are several such terms for each atom. It can be helpful in the context of coding and debugging to break up the overall gradient into contributions from each type of energy. We re-write eq. 21 as:

$$V(\mathbf{r}) = V_{\mathrm{stretch}} + V_{\mathrm{bend}} + V_{\mathrm{torsion}} + V_{\mathrm{vdW}} \tag{33}$$

And then likewise write for the Cartesian gradient:

$$\mathbf{g}(\mathbf{r}) = \frac{\partial V(\mathbf{r})}{\partial \mathbf{x}} = \mathbf{g}_{\mathrm{stretch}} + \mathbf{g}_{\mathrm{bend}} + \mathbf{g}_{\mathrm{torsion}} + \mathbf{g}_{\mathrm{vdW}} \tag{34}$$

Debugging output is provided to assist with testing code, with numerical values of these different components of the gradient given for several reference molecular structures.

Each of the above gradients is a sum of terms. For the bond stretching terms, one has the following expression:

$$\mathbf{g}_{\text{stretch}}(\mathbf{r}) = \frac{\partial V_{\text{stretch}}}{\partial \mathbf{x}} = 2 \sum_{i=1}^{n_{\text{stretch}}} k_{b,i}(r_{b,i} - r_{0,i}) \frac{\partial r_{b,i}}{\partial \mathbf{x}} \tag{35}$$

This can also be written in terms of the contribution from the i'th bond, assumed to be between atoms A and B, to the gradient on atoms A and B, whose elements we will write as e.g. $\partial V^{(i)}_{\text{stretch}} \big/ \partial x_A$:

$$\left\{ \frac{\partial V^{(i)}_{\text{stretch}}}{\partial x_{\text{A}}}, \frac{\partial V^{(i)}_{\text{stretch}}}{\partial y_{\text{A}}}, \frac{\partial V^{(i)}_{\text{stretch}}}{\partial z_{\text{A}}} \right\} = 2k_{b,i} \left(r_{b,i} - r_{0,i}\right) \frac{\mathbf{r}_{\text{BA}}}{r_{\text{AB}}} = - \left\{ \frac{\partial V^{(i)}_{\text{stretch}}}{\partial x_{\text{B}}}, \frac{\partial V^{(i)}_{\text{stretch}}}{\partial y_{\text{B}}}, \frac{\partial V^{(i)}_{\text{stretch}}}{\partial z_{\text{B}}} \right\} \tag{36}$$

For bending terms, one has the following expression, which can again be combined with eq. 31 to yield the desired Cartesian gradient terms.

$$\mathbf{g}_{\text{bend}}(\mathbf{r}) = \frac{\partial V_{\text{bend}}}{\partial \mathbf{x}} = 2 \sum_{i=1}^{n_{\text{bend}}} k_{a,i}(\theta_{a,i} - \theta_{0,i}) \frac{\partial \theta_{a,i}}{\partial \mathbf{x}} \tag{37}$$

The derivative of an individual dihedral angle torsion term (here, for example, for a central atom) is given as follows, with the required element for the chain rule to be taken from eq. 32.

$$\frac{\partial}{\partial x_{\text{C}}}(A_\phi(1 + \cos n\phi)) = -nA_\phi \sin n\phi \frac{\partial \phi}{\partial x_{\text{C}}} \tag{38}$$

One final point can be made about the contribution of the Lennard-Jones terms to the $\mathbf{g}_{\text{vdW}}$ component of the gradient. At first sight, each term within eq. 23 appears to yield the following sort of derivative expression:

$$\frac{\partial V^{\text{vdW}}_{ij}}{\partial x_i} = \frac{\partial}{\partial x_i} \left(\frac{A_{ij}}{r^{12}_{ij}} - \frac{B_{ij}}{r^{6}_{ij}} \right) = \left[\frac{-12A_{ij}}{r^{13}_{ij}} + \frac{6B_{ij}}{r^{7}_{ij}} \right] \times \frac{\partial r_{ij}}{\partial x_i} \tag{39}$$

However, using eq. 29, this can be re-written as:

$$\frac{\partial V^{\text{vdW}}_{ij}}{\partial x_i} = \left[\frac{-12A_{ij}}{r^{13}_{ij}} + \frac{6B_{ij}}{r^{7}_{ij}} \right] \times \frac{x_i - x_j}{r_{ij}} = (x_i - x_j) \left[\frac{-12A_{ij}}{r^{14}_{ij}} + \frac{6B_{ij}}{r^{8}_{ij}} \right] \tag{40}$$

This new expression is more convenient to evaluate, as it contains only even powers of r_{ij}. Starting from the coordinates $\{x_i, y_i, z_i\}$ and $\{x_j, y_j, z_j\}$ of the two atoms, only subtraction and multiplication are needed to compute the gradient terms when written as in eq. 40, with no need to evaluate a square root. When taking into account the large number of Lennard-Jones energy contributions involved in molecular mechanics energies of larger molecules, this leads to faster evaluations, especially with older computers and compilers for which the square root operation was rather slow.

5 Planning and Writing the Code

In this section, I suggest a set of steps to follow in order to write code(s) to optimize the structure of a saturated hydrocarbon using the 'tiny' molecular mechanics forcefield. As mentioned elsewhere in this manual, you may choose to stop at some or other point along the sequence of steps suggested. The suggestions below also include some suggestions about code structure.

1. The starting structure is provided as a modified "mol2" file, as a list of atoms (with coordinates) and a list of bonds. The first code you write should simply read in the coordinates and the list of bonds, and then print them out again, to check that they were read in correctly. The coordinates can be stored in an array of floating point numbers, probably best as a vector of length n_x, though a two-dimensional $(3, n_{at})$ array could also be used. The list of bonds can be stored in an integer array, with dimensions $(2, n_{bonds})$.

2. A second step can be to calculate all bond lengths, and calculate the bond stretching energy. For this, you also need parameters: equilibrium bond lengths and force constants. Note that for this exercise, we will always be using the same parameters, those from the 'tiny' forcefield (Table 1), so you can 'hardwire' the parameters (include them in the code) if you wish. Alternatively, a file with the parameters is provided. If you do 'hardwire' the parameters, I suggest you start from the atom-type-specific σ_i and ϵ_i parameters, and then apply eqs. 22 and 23 to convert to σ_{ij} and ϵ_{ij} or A_{ij} and B_{ij}. That way you will have the same exact values that I used (the values in the Table above are rounded off) and so you should get almost exactly the same outputs as me.

3. The third step is to build lists with the identifying properties of all internal coordinates: this means that one needs to store the identities of each of the atoms characterizing each of the bond stretches, angle bends, and torsions. You can store these as separate lists of integers (2 for bond stretches, 3 for bond angles, 4 for torsions) or as one list including also an "internal type" identifier. To generate the list, you need to think a little bit about how to convert the provided list of bonds into lists of internals. For each carbon atom, the six X-C-Y angles correspond to bending terms, and for each C-C bond, the nine X-C-C-Y dihedrals correspond to torsions. To make it easier to generate the list of internals, I have structured all the provided mol2 files to contain all C atoms first in the list of coordinates, and to list all C-C bonds first in the list of bonds. I also provide the number of C atoms n_{C} and the number of C-C bonds $n_{\mathrm{C\text{-}C}}$. I remind you that in case one wishes to treat non-covalent assemblies, e.g. a cluster fromed from multiple methane molecules, the set of all bond stretches and angle bends (and torsions) does not fully account for structure, and inter-molecular coordinates would be needed also. In this exercise, we will assume that we work only with single molecules, so the set of bonds, angles and torsions is sufficient. For each provided test file, sample output files provide lists of coordinates so you can check and debug your work. Some of you may also decide to perform this step (as well as the following steps 4. to 6.) initially only for the case of methane. Methane has no C-C bonds and also has no Lennard-Jones energy. Also, the minimum-energy structure has an energy very close to zero. That makes it simpler to code the energy and gradient expressions whil you are still in the debugging phase – the additional terms can be added later.

4. Then calculate the magnitude of each bond angle and torsion angle, and evaluate the stretching, bending, torsion, and Lennard-Jones energies. For all provided test files, the correct values of these energy components are given, again for debugging purposes. For the Lennard-Jones energy, remember that it should be omitted if the two atoms involved form a bond, or if they are both involved in a bond angle A-B-C. For the present small molecules you could simply make a square matrix of integers with dimension n_{at} which specifies whether a given atom pair should contribute to the Lennard-Jones energy.

5. Now you should calculate the gradient of the energy with respect to the Cartesian coordinates. For debugging purposes, it is useful to calculate separately the gradient of the bond stretching energy, the angle bending energy, the torsional energy, and the Lennard-Jones energy. Reference values of these gradient components are given for all of the test molecules.

6. Once you are able to calculate the gradient correctly, you should implement your first geometry optimization routine. I suggest to start with the BFGS algorithm, with Cartesian coordinates. You will need to build the initial approximate inverse hessian $\mathbf{M}$. For this, I suggest setting the diagonal elements to $1/300$ Å^2 (kcal mol^{-1})$^{-1}$ and all off-diagonal elements to zero. I further suggest carrying out a rough line search at each cycle. First set α to 0.8, then check that the energy at the new structure satisfies the first of the two Wolfe conditions (using $c_1 = 0.1$). If not, decrease α by multiplying it by 0.8, until the condition is met. The sample output files contain a log of the energies obtained at each cycle and during the line searches when performing these in the way just described. The *overall* geometry optimization can be taken to be converged when the RMS gradient reaches some or other threshold value. I used values of the order of 0.001 kcal mol^{-1} Å^{-1}, which is sufficient to obtain a well-converged final energy. Sample files

contain final energy and structure, again for testing and debugging purposes. As detailed below, I also provide the values of the various terms relevant to the BFGS algorithm for each geometry update cycle.

7. The final step is to construct a code to perform geometry optimization in internal coordinates. The first thing to do here is to build the Wilson B Matrix, and its generalized inverse $\mathbf{G}^{-}$. Samples of these matrices are provided for debugging. To calculate $\mathbf{G}^{-}$, you will need to diagonalize $\mathbf{G}$. This can be done with suitable library functions in Python or Julia. If you code in Fortran, the easiest way to do this will be to call the LAPACK routine `DSYEV`. Then you can compute the gradient in internal coordinates – again, sample values are provided.

8. Once you have been able to compute the gradient in internal coordinates, you can write a BFGS optimizer. Here too, you need an initial guess for the inverse Hessian. I suggest setting this to 1/600 Å^2 (kcal mol^{-1})$^{-1}$ for the diagonal elements corresponding to bond stretches, 1/150 radian^2 (kcal mol^{-1})$^{-1}$ for bending terms, and 1/80 radian^2 (kcal mol^{-1})$^{-1}$ for torsions. All non-diagonal elements can be zero. For this code, there is no need to implement line-searching, though it is wise to test the norm of the step to be taken, $\mathbf{s}_k$, by taking its root-mean-square ($\sqrt{\mathbf{s}_k \cdot \mathbf{s}_k / n_q}$). If it is larger than 0.02 (the units here are hard to define, as bond lengths are in Angstrom and angles in radians), scale $\mathbf{s}_k$ so its RMS is equal to 0.02. Sample outputs are provided. You should find improved convergence in terms of numbers of steps needed to reach a given RMS gradient threshold. Your code may however take longer to run, as the algebra involved in the conversion between Cartesian and internal coordinates takes a significant portion of the time for this problem compared to the 'core' task of computation of the energy and gradient with respect to Cartesian coordinates, which is relatively undemanding with molecular mechanics. For QM methods, the time taken to obtain the energy and gradient almost always dominates, so routines using internal coordinates are almost always more efficient.

6 Files Provided

I will provide for you the files needed to code and debug your code. The files provided include:

(a) A file with parameters, `tiny.parameters`. This simply contains the forcefield parameters, in case you decide not to hardwire them.

(b) A set of files *moleculename*`.mol2` (where *moleculename* is either methane, ethane, methylpropane, butane, methylcyclohexane, pinane, or cholestane, by increasing order of size. These files contain the following information, with the following structure:

- line 1: n_{at} followed by n_{bonds} followed by the number of carbon atoms, and then the number of C-C bonds, followed by a bunch of input fields (mostly zeroes) which have no relevance here. For example, in methylcyclohexane C_7H_{14}, this comes out as 21 21 7 7 (coincidentally the number of C atoms matches the number of C-C bonds and the number of atoms matches the number of bonds). In ethane, it comes out as 8 7 2 1.
- lines 2 to $1 + n_{at}$: details of each atom in the molecule, with one atom per line. For each atom, first the x, y and z coordinates, in Å, then the atom label, then a set of input fields (again mostly zeroes) that can be ignored. Although this is not a requirement of the `mol2` file format, I will always put the carbon atoms first.
- lines $2 + n_{at}$ to $1 + n_{at} + n_{bonds}$ contain details of the bonds present in the forcefield, with one bond per line. As for the atoms, I list C-C bonds first. In each line, there is the index of the first atom in the bond (with a numbering system running from 1 to n_{at}), then the index of the second atom, then a '1' to indicate it is a single bond (we only handle single bonds here), then a set of input fields, all just '0', which are not important here.

(c) In the assessment, I will provide you with a new file *moleculename*`.mol2` and check that your code generates the correct output for that case.

(d) A set of files *moleculename*`.out1` with basic output for debugging purposes, covering roughly steps 1. to 5. in the previous section:

- line 1: an echo of the total number of atoms.
- Then the coordinates and labels of the atoms.
- The the total number of bond stretching, bending, and torsion coordinates.
- Then the total molecular mechanics energy at the input structure, followed by its stretching, bending, torsion, and vdW components (all expressed in kcal mol^{-1}).
- A list of the bonds present in the structure, as well as their actual length in the input structure, given in Å. The contribution to the stretching energy from that bond is also provided.
- A list of the bond angles present, with their magnitudes in radian and in degrees. The contribution to the bending energy is also given.
- A list of the bond torsions, giving the dihedral angles in radian then degrees, and the contribution to the torsional energy.
- A list of vdW energies for all unique pairs of atoms. For atoms that do not contribute, the value is zero.
- A set of lines with the x, y and z components of the gradient of the overall MM energy with respect to the Cartesian coordinates of each atom.
- Four more sets of lines with the x, y and z components of the gradient of the energy with respect to the Cartesian coordinates of each atom, with respectively the stretching energy gradient, bending energy gradient, torsional energy gradients, and vdW energy gradient.

(e) A set of files *moleculename*`.out2` with output related to geometry optimization in Cartesian coordinates, for debugging purposes, covering step 6. in the previous section. These files rapidly become very large as the molecules contain more atoms, so I provide them only for methane, ethane, n- and iso-butane, and methylcyclohexane.

- First of all, some general info about the system and the energy at the initial structure is given.
- Then I give the target RMS gradient value (in kcal mol^{-1} Å^{-1}) used for the optimization.
- Then I give the initial gradient vector, and the initial guess for the inverse Hessian.
- Then I give information at each cycle of the optimization: First, the predicted structure change $\mathbf{p}_k = -\mathbf{M}_k \nabla V(\mathbf{r}_k)$ (see above eq. 12), then details of the line search as discussed in section 2.2, giving the values of α that are tested, and the resulting energies $V(\mathbf{r}_k + \alpha \mathbf{p}_k)$, the updated structure $\mathbf{r}_{k+1} = \mathbf{r} + \alpha \mathbf{p}_k$ obtained when α has been chosen (I don't give all the intermediate structures generated during the line searchk but you will need to generate these!), the energy before and after the change in structure, and the RMS magnitude of the gradient at the new structure, the gradient at the new structure, and the updated guess for the inverse Hessian from eq. 12.
- Once the GMS gradient falls below the required threshold, the optimization stops. At that point the final energy and the final structure are printed one more time.

(f) A set of files *moleculename*`.out3` with output related to geometry optimization in internal coordinates, for debugging purposes, covering steps 7. and 8. in the previous section. Here there are a lot of intermediate quantities, so these files are only provided for the smaller molecules, up to methylcyclohexane. These files contain:

- First of all, some general info about the system and the energy at the initial structure is given.
- Then I give the target RMS gradient value (in kcal mol^{-1} Å^{-1}) used for the optimization. This target RMS gradient value is for the *Cartesian* gradient.

- Before starting the iterations, it is wise to check that one can form various quantities correctly. So I first of all print these at the first structure. First, I give the Wilson B matrix (where the order of the internal coordinates is: first the bond lengths, then the bond angles, then the dihedrals), then the product of **B** with itself, **G**, then the eigenvalues of **G**, then the generalized inverse $\mathbf{G}^-$ of eq. 18.
- At this point the geometry optimization proper can start. The output starts by reminding you of the maximum average step length. Then at each cycle the key information is given:
- The predicted step $\mathbf{p}_{q,k} = \mathbf{s}_{q,k} = \mathbf{M}_{q,k}\mathbf{g}_{q,k}$ is obtained. As mentioned, for this problem, it turns out that there is no need to do a line search, so $\alpha_k = 1$.
- At this point, we need to generate an updated structure in terms of Cartesian coordinates. As explained above, this needs to be done iteratively, and the details of the steps performed are now given:
  - First of all, I give the threshold for considering the fitting has converged, the maximum change in a Cartesian coordinate between $\mathbf{x}_{k+1}^{(j)}$ and $\mathbf{x}_{k+1}^{(j+1)}$.
  - Then I give the initial change to the Cartesian coordinates, $\mathbf{B}^T\mathbf{G}^-\mathbf{s}_{q,k}^{(1)}$.
  - Then at each iteration, I give the new set of internals $\mathbf{q}_{k+1}^{(j)}$, and the difference between these and the desired new set of internals, $\mathbf{s}_{q,k}^{(j+1)} = \mathbf{q}_{k+1} - \mathbf{q}_{k+1}^{(j+1)}$.
  - This then yields a new set of Cartesians $\mathbf{x}_{k+1}^{(j+1)}$.
  - I report how much these new Cartesians differ from those in the previous iteration.
  - Once this difference falls below the threshold, we can exit the mini-iterations. The new internals are Cartesians are the last ones reported.
- After these iterations to work out the best-fit new Cartesian coordinates, I provide these new Cartesian coordinates, and also provide the corresponding internal coordinates $\mathbf{q}_{k+1}$ - note that it is these coordinates which are used to determine $\mathbf{s}_{q,k}$ for the Hessian update procedure.
- At this new structure, I provide the new Wilson B matrix and its generalized inverse, and calculate the energy and gradient, and convert the gradient to internal coordinates, which is also given. I then give the updated approximate inverse Hessian in the space of internal coordinates, $\mathbf{M}_{q,k+1}$ (eq. 20).
- Then one can check for convergence of the geometry optimization, and possibly start a new cycle.
- After the geometry optimization has converged, the final energy and structure are given.

7 References

(a) The Tinker software package, from which the 'tiny' forcefield used here is taken, can be found here: https://dasher.wustl.edu/tinker/

(b) There is a nice introduction to the BFGS method here.

(c) The use of internal coordinates for geometry optimization has been suggested by several authors. The method closest to that suggested here was published by Peng, Ayala, Schlegel and Frisch in the paper *Using Redundant Internal Coordinates to Optimize Equilibrium Geometries and Transition States*, *Journal of Computational Chemistry*, 1996, **17**, pp. 49 – 56, available here from the journal webpage, or here from Prof. Schlegel's webpage.

(d) The Wilson B matrix is discussed in several places. This paper: V. Bakken and T. Helgaker, *J. Chem. Phys.*, 2002, **117**, 9160, contains a convenient summary of the expressions for the B matrix elements.