A Guide to Evclid, the Geometric Calculator

Hanno Essén
Royal Institute of Technology
Department of Mechanics
SE-100 44 Stockholm, Sweden

Revised September 2001

Contents

List of Figures

List of Tables

Preface

This guide describes how to use the computer program EVCLID, a geometric calculator and editor intended mainly for use by quantum chemists. It also presents the theoretical concepts that underlie the main part of the program. Some of these concepts are new and in this sense part of this guide is simply a scientific publication. In particular the concept of five geometric coordinates, arising from the three geometric parameters (distance, angle, dihedral angle), is new.

The first chapter gives the general ideas and principles needed to use the program. The second chapter gives an overwiew of the available commands and how they interact. The final chapter presents the geometric ideas and definitions on which the program is based. This chapter should be readable even to someone without access to the program. Its later part, however, consists of example runs.

The program EVCLID is now approximately twenty years old and has been revised several times. The present (1992) version differs from the previous in the following additions: z-matrix editing, improved automatic z-matrix construction, possibility to pass reading of input temporarily to file, possibility to assign input lines to variables, and, finally, an improved save-facitlity. During the work I have discovered two bugs and fixed them: The reading of chemical symbols (charge) from z-matrix files was, unintentionally, format sensitive. Reflection in an arbitrary plane defined by a normal vector did not work properly unless the normal vector was a unit vector.

Since the 1992 version two new features have been added. In a subcommand to `ige` (input geometric parameters) one can specify a new input point to have equal distances to two, three, or four other already entered points. In a subcommand to `ipo` (input polygon or polyhedron) one can also get truncated versions of the platonic polyhedra, for exampel the famous football polyhedron with 60 vertices. A few new references have also been added in the bibliography.

I would like to thank users of the program that have encouraged me with their comments and advice. For several years Dr. Mats Svensson, formerly of the Department of Physics, University of Stockholm, has been especially helpful. I also would like to thank Göran Gustafssons Stiftelse (foundation) for providing excellent computer facilities.

Nowadays there are several commerical programs with graphical interfaces that are simpler to use than EVCLID, but there are many features in EVCLID that are not available in those. Nor do those programs have as deep an understanding of Euclidean geometry built in as EVCLID has. Ideally, of course, a graphical interface should be added to EVCLID, but that is still in the future.

Hanno Essén

KTH, Stockholm, Sweden
September, 2001

Chapter 1

Introduction

1.1 What does EVCLID do?

EVCLID is a geometric calculator. Just as ordinary calculators work on real numbers, EVCLID handles points of three-dimensional space. When you start to run the program you must always begin by supplying it with some *input*, i.e. a set of points. This can be done in several ways. They can be entered in terms of coordinates, which may be Cartesian, cylindrical, or spherical. They can be read from a file that may have a number of different formats. They can be specified as the corners of a regular polygon or polyhedron of desired size and type. Last, but not least, one can tell the program where a point is by giving three geometric parameters (distance, angle, or dihedral angle) which define the position of the point in terms of previously entered points. This is the most advanced and unique feature of the EVCLID program.

All the different input methods result in storage of the Cartesian coordinates of the points in the central point-storing array of the program. The points are identified and referred to through their row number in this array. One can also, if one wishes, store an atomic number (an integer between -9 and 99) for each point. This can be useful if the points correspond to the positions of atoms in a molecule.

When one has stored points in the program one can do various things with them. One can *display* their coordinates (Cartesian, cylindrical, or spherical) or various combinations of geometric parameters, which give their relative positions in a coordinate independent way, on the terminal screen. Or one can name and create a file on which one can *output* such data.

EVCLID can perform *geometric* operations (translations, rotations, reflections, inversions, and scalings) numerically on all or a subset of the stored points (or on copies of them). The program also can perform an automatic transformation to a natural coordinate system defined by the (atomic number) inertia tensor of the point set (molecule).

Finally one can *edit* the array of points by copying them, by removing some of them or by renumbering them to some new more desirable order. In this way one can organize one's calculations more efficiently.

Of course you don't need a manual to use the EVCLID program. The purpose of this text is to tell you why and when you should use the program in the first place and to give you some organizational, geometrical and logical ideas that you need to use it creatively. A large number of *help* texts, contained in the program and displayed on command, explain the details needed to run the program.

To summarize, the basic features of the EVCLID program can be classified under the following headings:

- Input of
    1. Coordinates or Geometric parameters (distance, angle, dihedral angle) from keyboard or file.
    2. Regular polygon or polyhedron.
    3. Atomic numbers.
- Display and Output of
    1. Coordinates.
    2. Geometric parameters.
- Geometric operations:
    1. Translations and Rotations
    2. Reflections, Inversions, and Scalings
    3. Transformation to principal axes system
- Edit features:
    1. Copying
    2. Removal
    3. Renumbering
- Help online

1.2 When and how to use EVCLID

The users of EVCLID are assumed to have some problem or problems involving geometric calculations. Such problems are most likely to arise when working with molecular geometries in connection with some kind of quantum chemical calculations. For example, setting up the geometry of a dimer by copying, translating and rotating the original molecule, was one of the early applications. Geometry optimization and search for reaction paths are applications that are eminently suited to EVCLID. This is the type of problems that the author of the program was faced with and most users of it also. In particular EVCLID communicates well with the quantum chemistry program Gaussian 90 [1], [2] (and earlier versions) since one can use it to read, produce, edit, and write 'z-matrices', the coordinate independent geometry input to that program.

The chemical background explains why EVCLID refers to points as *'atoms'*. There may, of course, arise other situations in which it can be useful. If nothing else one can learn much geometry by playing with it. In fact one of the main difficulties in using the program has nothing to do with EVCLID but relates to the geometry that underlies it. In particular the concept of a dihedral angle takes some getting used to. One should always, at least in more complicated situations, use paper and pencil to draw pictures alongside with the calculations. EVCLID cannot do your geometric thinking but it will make the corresponding calculations trivial.

The best way to learn about the *basic* use of the program is through examples and some can be found below. Yet, examples should be tried at the computer rather than given in the manual. Also, just one example may be misleading since the number of different ways the program can be used is almost infinite. The purpose of this 'manual' is to give the general ideas and rules that can make the program into an effective problem solving tool.

1.3 About the Program and its Name

EVCLID was first conceived in 1982 [3] and the early versions were called 'Euclid' one of which has been available from QCPE (Quantum chemistry program exchange) [4]. The name was changed after a big revision in 1989 [5] when the `ige` command (input of geometric parameters) was rewritten to be able to handle all five ('V' in roman notation, hence the 'V' in EVCLID) geometric coordinates instead of only the three simple ones as was the case before. Also the spelling with a 'u' had already been used in the computer world and the 'v' spelling is the one the Romans, who did not have the letter 'u' in their alphabet, used.

The program is mainly written in Fortran 77 and the source code is currently over 6300 lines and uses approximately 215 K-bytes of memory. The Fortran input formats for numbers are not much used, instead EVCLID reads character-strings. The program searches these strings for commands or numbers. Numbers are converted to integers or double precision reals. The structure is simple: the main program handles the commands and calls the appropriate subroutines, one for each command. The subroutine corresponding to the command `xyz` is called 'xyzcom'. There are 26 of these but there are many more subroutines, over 70, many of which are common to several commands. The author has written most of these but some (the linear equations solving and 'amoeba' routines) are adapted from the book

Numerical Recipes by Press et al. [6].

1.3.1 Portability

The program is highly portable though some features depart from the ANSI standard (Fortran 77), or are not clearly defined by the standard. It compiles and runs with no changes on a VAX, a Sun and an Alliant. It also compiles and runs well on MacFortran/020 except for the save options (see page 18), which do not work properly. It is only possible to do one save. MacFortran also requires that one replaces the strings `read(5` and `write(6` by `read(*` and `write(*` respectively.

The deviations from the ANSI standard consist of use of lower case letters and of the use of the $ format control specifier which is used to write a prompt on the same line as the input is read. This, however, only occurs once in the source file and is easily removed if necessary.

To run properly Evclid needs a terminal with screen of minimum 24 lines 80 characters wide. The number of points (atoms) it can handle is, at the time of writing, arbitrarily set to 121 but this number is easily changed with a few replacements at the beginning of the source file as explained there. It should not be greater than 999 since then some output formats must be changed too.

It is the authors wish that the program and information about it should be freely available to anyone who might have use for it. Contact the author for free copies.

1.4 General Principles

The most basic rule is this: the program asks you to type text or enter numbers by issuing prompt-lines and when you are finished typing the input line you tell the computer this by hitting the (carriage) return key [1] (sometimes indicated with a symbol like ↩).

- **Rule: Finish input lines with 'return'**

On the next few pages the basic rules for running the program are given in 'bulleted' **bold face** type lines like the above. For a quick overview it is enough to read them. Information about exceptions to these rules is usually given in the fine print near it.

On a reasonably fast and available computer input to Evclid should lead to immediate response. I.e. the program tells you what it has done or that it didn't like your input and then what it wants you to do. It is thus highly interactive and tries to be user friendly.

[1] The experienced user can type several input lines as a single line with semicolons (`;`) instead of returns.

Type	x	yz-alternatives							
Input	`i`	`ge`	`co`	`po`	`an`	`sa`	`cf`	`zm`	`h`
Display	`d`	`co`	`zm`	`ge`	`h`				
Output	`o`	`zm`	`cf`	`sa`	`h`				
Geometric	`g`	`pr`	`tr`	`ro`	`sc`	`re`	`in`	`h`	
Edit	`e`	`mo`	`or`	`pe`	`rr`	`rm`	`cp`	`zm`	`h`
Help	`h`		`i`	`d`	`o`	`g`	`e`	`h`	

Table 1.1: In this table the two letters yz are abbreviations as follows: (`i d o`) `ge` =Geometric parameters, `co` =Coordinates, `po` =Polygon or Polyhedron, `an` =Atomic Number, `sa` =Save, `cf` =Coordinate File, `zm` =Z-Matrix. (`g`) `pr` =Principal axes, `tr` =Translate, `ro` =Rotate, `sc` =Scale, `re` =Reflect, `in` =Invert. (`e`) `mo` =Move, `or` =Order, `pe` =Permute, `rr` =Remove & Renumber, `rm` =Remove, `cp` =Copy, `zm` =Z-Matrix.

1.4.1 On Commands in General

A command is a three letter abbreviation xyz which tells the program what you wish to do. The first letter, x, is an abbreviation for the general type of option and is one of the letters

- $x =$ `i`, for Input commands
- $x =$ `d`, for Display commands
- $x =$ `o`, for Output commands
- $x =$ `g`, for Geometric commands
- $x =$ `e`, for Edit commands
- $x =$ `h`, for Help commands

The next two letters, yz, are an abbreviation of the specific option of this type you wish to use and the different possibilities are given in table 1.1. (A more extensive summary is given on page 17.) In the case $x =$ `h` only one more letter, `i, d, o, g, e`, or `h`, is needed and the result is the display of text (maximum one screen) which informs you of the various available commands for each corresponding general type. Thus if you type `hi` and the hit the return key in response to the 'Type: command' prompt (the 'top level' of the program) you will be given information of the various available input commands `i`yz, for example that the command `ico` means that you wish to input coordinates. If you type `he` you can, among other things, read that `epe` means that you wish to edit the points by permuting them. To get a complete command menu (at the top level), just type `h`. The help text then displayed also gives a brief summary of the basic notation. The two forms `h`x and x`h` are equivalent so you don't have to remember the order.

- **Rule: A command results in a prompt-line**

As a rule each command results in a prompt line 'Enter: ...' in response to which the user is expected to enter input and push return whereupon the program proceeds with some action or computation. The *exceptions* to this rule are, apart from the help-commands, the two commands `isa` and `osa`, input from and output to save-file, which result in immediate action. The prompt-lines remind you of what input the command will accept but if you don't understand the notation you can again type `h` to get a help text explaining the possible input to this command. At the end of the help text the 'Enter: ...' prompt reappears so that you can type your input while studying the text. When you have typed your input and finished it with return the program takes some action. For most commands you then get the top level prompt 'Type: command' again but others assume that you wish to do more of the same and thus reissues the 'Enter: ...' prompt. If you don't want to do more of the same there are, in general two alternatives. Either you can return to the top level by returning a blank line.

- **Rule: Back to top level with an empty line**

Or you can type a command *xyz* to go directly to the corresponding new prompt. *Exceptions* are the prompts resulting in display or output for which return of a blank line result in the default display or output. When the input from or output to file prompts 'Type: file-name' have been issued you cannot give a command (in this case there is a risk for ambiguity). With this exception there is the:

- **Rule: One can always give a command**

It doesn't matter what the prompt is as long as it is not for a file-name. If the prompt is 'Enter: ...' the command must be given in its full three letter form. At the top level when the 'Type: command' prompt is seen one can use short forms for the full commands. These are given in the corresponding help texts (displayed on command `h`*x*).

- **Rule: Top level accepts abbreviated commands**

For many commands the two letter abbreviation is sufficient. Also, for each main type ($x =$ `i, d, o, g, e, h`) there is a default specific option *yz* which is selected if only the first letter of the full command is given. This default command is the one listed first in each line of the menu in table 1.1. Finally there are a few exceptional one letter 'aliases' for commands that can be used at top level. These are

1. `c` for input of coordinates `ico`.
2. `s` which stands for `isa` if the program is empty and for `osa` if it contains points (input from and output to save-file).

3. $\hookleftarrow$ (just return of an empty line) stands for `dco` i.e. display coordinates.

4. `r` for the edit option remove `erm`.

A full list of abbreviations is given on page 17.

Finally it is important to know about the `quit` command which is used to terminate the EVCLID run.

- **Rule: Terminate the run with `quit`**

If you have done some work with the program that you might need in the future, be sure to store it away with one of the output options before typing `quit` since then all input will vanish. This command is also accepted by all prompts (including the file-name prompts so you can't have a file named 'quit').

1.4.2 On Prompt-lines and Input

There are only three different types of prompt-lines issued by EVCLID. These are listed here:

△ 1. The top level prompt for a command which also gives some accounting information about the contents of the central point array:

```
Type:command   (no of atoms:6)  (input charge:2)                       ?xyz?
Evc>>
```

△ 2. The prompt for input of arguments to the command `xyz` typically looks like this:

```
Enter: arg1 arg2 (arg3...)   (& argi)             (h = help)             *xyz*
xyz>>
```

△ 3. File-name prompts issued by `icf, izm, ocf,` and `ozm` for the name of input and output files:

```
Type: file-name          ( ... )     (h = help)                          f*xyz*
file>
```

In general each command corresponds to one prompt of type 2 in this list. However, two commands have optional input-modes which you can get to by giving a proper response to the first prompt-line. These are `ico` where you will be prompted for Cartesian coordinates but can change this to prompts for cylindrical or spherical coordinates, and `ian`, input of atomic numbers, where you are prompted for atom number and charge but can get a prompt for charge only. The general rules for responding to the typical prompt-line are:

- **Arguments are separated with blanks at input**
- **Numbers must be given in decimal notation (not exponential)**
- **Angles are always given (and displayed) in degrees**
- **At least one argument is always needed (exceptions: display and output)**
- **Optional arguments are in parenthesis in the prompt-line**
- **The number of input numbers determines action of** EVCLID
- **One input line is sufficient (exception:** `ige`**)**
- **One can always type** `h` **for help**

EVCLID studies your input line and takes action according to its findings. First it looks for a (three letter) *xyz* command or `quit`. Leading and trailing blanks are ignored. If there is no command in the input line it is searched for input numbers which are counted and assigned to variables according to the specific convention for the corresponding command. In some input lines EVCLID also looks for special symbols and perhaps numbers that follow this symbol. So, giving one, two, or three numbers may mean entirely different things to the program. If you are uncertain, type `h` and study the help text before typing input. Normally the program does not check that your input numbers are of correct type (integer or float). This is convenient since you don't have to put a decimal point on floats that do not have anything beyond the decimal point. On the other hand, when integers are expected 1.4 can be taken to mean 1 and 4 (two numbers) or it may be truncated to 1. EVCLID always tells you what it has done so you will normally quickly find out if your input was interpreted as you intended.

1.4.3 Notation

The number of the array row in which the Cartesian coordinates of an atom are stored is usually referred to as the *'atom number'*. In prompt-lines and displays the symbol for one such number is `n`. When two such atom numbers are of interest the notation is normally `m n` and when several are needed it is `n1 n2 n3` . . . Thus the notation is:

`n`	for one atom number,
`m n`	for two atom numbers,
`n1 n2 n3` . . .	for three or more atom numbers.

When the position of an atom is given by means of reference to other atoms, `i j k` are used for the numbers of the *reference atoms*. The notation is thus:

`i j k`	for reference atom numbers.

The most frequently occurring symbols in the argument lists of prompt-lines are `nf` and `nl`. These are used to specify what subset, or *range*, of entered atoms (points) that you wish to operate on. The 'name' of a point as far as EVCLID is concerned is the number `n` at which it is stored where: $1 \leq$ `n` $\leq n_{max}$ (currently $n_{max} = 121$). A sub-range `nf, nl` is given when something is to be restricted to atoms with numbers `n` fulfilling: `nf` $\leq$ `n` $\leq$ `nl`. Thus `nf` is the first number in the range and `nl` is the last. Note that you can choose atom numbers `n` freely at input, there is no need for a range to be full of input atoms. Thus the notation is

`nf nl`	for the first and last atom numbers of sub-range of atoms.

When a prompt-line looks as follows: '`Enter: nf (nl)` ...' and you do not intend to specify a sub-range, just enter 1 and all atoms will be affected by the action of the command. The optional `nl` is automatically chosen to include all your input.

The following notation is also used consistently by EVCLID:

`Z`	atomic numbers (nuclear charge in units of proton charge),
`ct`	coordinate type
`r`	the distance between two points
`rh`	the distance between a point and the z-axis
`tht`	un-signed angles (take values between $0°$ and $180°$)
`phi`	signed angles (take values between $-180°$ and $180°$)

The coordinate type symbol `ct` takes the values `x, c,` and `s` for Cartesian, cylindrical, and spherical coordinates respectively. The symbol `r` may be a distance between two atoms (as in `r(m,n)`) or the spherical coordinate r, i.e. the distance between an atom and the origin. `rh` stands for the greek letter ρ and is one of the cylindrical coordinates (the radius of the cylindrical coordinate suface). `tht` denotes the greek letter θ and stands both for bond angles (as in `tht(i,j,k)`) and for the spherical coordinate θ, i.e. the angle between the position vector of a point and the z-axis. Finally `phi`, which stands for the greek letter ϕ, is either a dihedral angle or the azimuthal angle in spherical coordinates.

1.4.4 Making Command Macros

If you are an experienced user of EVCLID and find that you frequently type the same sequence of input you may be interested in some shortcut. One possibility is to do as follows: write the sequence of input in a file named *filename* and, when EVCLID prompts you for input, type: `<<`*filename*. The program will then read input lines from the file *filename* instead of from the terminal. When the file has been read the prompting will come back to the terminal, unless, of course, there is a `quit`-command in the file.

If you are using a UNIX-system you can make EVCLID read its input from a file by giving the command `evclid <`*filename* to the operating system (you might of course use som abbreviation such as, for example, `evc` instead of the full name of EVCLID). Then, however, EVCLID will *only* read that file. With the 'double arrows' (`<<`) you temporarily pass the control of the program to a file and then get it back.

Note that instead of 'new-lines' (carriage returns) between the input lines you can have semicolons. In this way the file can be written somewhat more compactly. This can also be useful when typing input from the terminal. If you, for example, wish to read the help-text explaining the input accepted by the command `xyz`, just type `xyz;h`. Note that use of the semicolons or of command 'macro' files requires that you can anticipate the responses of EVCLID correctly.

EVCLID allows you to assign (or 'store') an input line, or part of an input line, to a variable. This is done as follows: type

`$`*i*= *string*

(and then return), where $i = 0, 1, 2, \ldots, 9$. For the rest of the session you can then simply type `$`*i* instead of typing *string*. The *string* may not contain any new-lines or semicolons, i.e. it must be a command, a filename, an argument, or a sequence of arguments. This feature is useful if there is some long number, or numbers, that you anticipate that you, otherwise, would have to type several times.

It is also possible to use this feature to make command-'macros' with arguments since it allows a command file, *filename*, to contain strings like, for example, `$3`. If you have assigned an appropriate value, *string*, to `$3` before running the file (with the command `<<`*filename*) all occurrences of `$3` will be replaced by *string* before the commands are processed. If you have forgotten your assignments they will be listed if you type the command `$*`.

The services mentioned in this section are provided by the subroutine `readin` which is EVCLID's 'user interface'. In summary:

- **Separate input lines on a single line with semicolons (;)**
- **Pass reading of input lines to a file by** `<<`*filename*
- **Assign (part of) input line to variable by** `$i=`*string* (`i`=0...9)
- **Use** `$i` **instead of** *string* **in input line**
- **List stored assignments by** `$*`

Chapter 2

Overview

2.1 Top Level Help

The command **hh** given at the top level of the program results in the appearance a list of the top level help commands as well as some general explanations on the form:

```
  BASIC HELP ABOUT EVCLID
 The program works on an array of Cartesian coordinates of
points. One can *Input  points to this array in various ways
and once this has been done one can *Display  properties of
the point set on the screen or *Output  such to file. One can
manipulate the point set with *Geometric operations or with
*Edit options (copy, remove, renumber). At the top level you
can get *Help with one of the following commands:
 h    for command menu           hi   for input-commands
 hd   for display-commands       ho   for output-commands
 hg   for geometric-commands     he   for edit-commands
 hh   for the present text
 Once you have selected and given a command:xyz you will get
a prompt something like:
Enter:a1 (a2)... etc.  (& a..)                (h = help)                 *xyz*
which tells you what arguments the command expects. Now you
can enter: h for an explanation of what the argument symbols
stand for. This information is maximum one screen long and
then the prompt for input,  Enter:. etc., appears again.
Note that the number of arguments is important and deter-
mines the action taken. Input lines always end with return.
Arguments in parenthesis are optional.
```

The command menu (`h`) is essentially the same as table 1.1. The other five top level help texts are given below with some comments.

2.1.1 Input Commands

The following information appears when you type `hi` (or `ih`) at the top level:

```
         INPUT FROM TERMINAL
   Input of atoms from the terminal can be done using:
 Geometric parameters (distances, angles, dihedral angles)
 referencing already entered atoms;     command:ige  (i)
 or using Coordinates;                  command:ico  (c)
   Make Polygon or Polyhedron with      command:ipo  (ip)
 Atomic Numbers can be entered with     command:ian  (ia)
   When you have entered the command a prompt-line will appear
 which tells you what input will be accepted. If you still
 don't understand you can type:h for an explanation.
                           (Short command-form in parenthesis)
         INPUT FROM FILE
   If the command:s (or:osa) has been given earlier to Save in-
 put this input will come back with     command:isa  (s)
 When the program is empty the command:s is taken to mean:isa
 otherwise it means:osa. Note that the command:isa removes all
 current contents before it puts back what was stored at save.
   To read files containing:
 charges and Cartesian coordinates;     command:icf
 charges and Z-matrix;                  command:izm  (iz)
   The z-matrix concept is explained if you give command:dzm
 and then type:h. (See also: ho for information on save.)
```

The command `ige` is the most advanced feature of EVCLID and there is a lot of further help available after you have given the command. Note that though normally the geometric parameters must reference already entered atoms one can use this option from scratch; the first two atoms can be specified without reference to anything.

The easiest way to generate points for testing the program with is to use the command `ipo` which allows you to have the corners (vertices) of a regular polygon or polyhedron (one of the five Platonic bodies) generated automatically. Also truncated polyhedra are available.

Note that the commands `isa` (=`s` when the program is empty) and `osa` are unique in that they do not result in a prompt-line but in immediate action. If there is input `s` will mean `osa` and results in immediate storage of the current memory contents of the program at the end of the save file. The command `isa`, when there is input, will result in a prompt which asks you which save you wish to read in (the saves are numbered in the order that you have made them). All the current input is then removed and replaced by that which existed at the time of the save in question.

2.1.2 Display Commands

Display is the word for output to the standard output which will normally be the terminal screen. When you have given some input or read something from a named file (or the save file) you naturally wish to look at it and see if it is what it should be. This is done with the display options which thus normally will be among the most used features of the program. The following information is called up with command `hd`:

```
           DISPLAY OPTIONS
Display of Coordinates,          command:dco  ( )
Display of Z-matrix,             command:dzm  (dz)
Display of Geometric parameters, command:dge  (dg)
                          (Short command form in parenthesis)
  After the command has been typed a prompt line will appear.
No input is needed for the standard (default) display, just
press return and the information will appear on the screen.
As usual you can always type:h for help.
  For coordinates the default is Cartesian coordinates
(else you must give coordinate type:ct  as c or s).
  A z-matrix is a set of 3N-6 geometric parameters that give
the shape of the N-atom set in a coordinate independent way.
  Lists of geometric parameters running over all pairs of
atoms are generated with:dge. The default parameter type is
distance. Give the command and type:h for further information.
  One can always give a range: nf,nl to limit display.
To cancel display you must type:q (just return will not work).
```

Note that the display commands are different from most others in that their prompt-lines only demand optional input. To get the default display you only have to press return ($\hookleftarrow$). Only one screen of information at the time is displayed; even if there is much input you never get a lot of information scrolling past.

Once you have entered atomic numbers for your atoms these are also displayed together with the other information (by all three options). If atomic numbers Z, also referred to as charge, have not been entered they are not displayed but have the value one. If you do not want atomic numbers in the display (or otherwise) you can remove them with `ian`.

Note that at top level a push of the return key is the short form for `dco` and since the resulting prompt does not require any input for the default display i.e. Cartesian coordinates, two returns are enough to get a list of your current input.

Display of geometric parameters (`dge`) will produce a list of the distances between the atoms if no input is given. Giving the number of one atom results in a list of the angles at that atom; giving two atom numbers a list of dihedral angles between planes intersecting at the line between them.

2.1.3 Output Commands

The top level information about output commands that typing `ho` produces looks as follows:

```
        OUTPUT TO FILE
  Output options are     command:ozm  (o)    for Z-matrix
 and                     command:ocf  (oc)   for Coordinates
                     (short command form in parenthesis)
  You will be asked for a name for the output file the first
 time the command is given (it must be a new name).
  The lines written to the file are also displayed on the
 screen. Each time the commands are called and lines appear on
 the terminal screen corresponding lines are added at the end
 of the output file (because no rewind is done).
  In a subsequent run the output files can be read back in
 with: izm  or: icf  but the standard restart should be Save.
        ABOUT SAVE
  To save all your input to the program at a given time,
 just type: s  When there are entered atoms this is short for
                         command:osa  (s)    for Save
 The output is written to file: ecldsv.dat and can be read back
 in later in the same or a subsequent run with command:isa.
 Each time you give the command s (or osa) output is written to
 the end of the file ecldsv.dat. When the program is empty  s
 will mean  isa  and the output of the last save is read in.
 Otherwise isa issues a prompt which asks which save you wish.
```

Note that the command `osa`, or just `s` if atoms have been entered, does not result in a prompt-line but results in the immediate writing of current contents to the end of a file called 'ecldsv.dat'. EVCLID first counts the old saves in the file and notifies you of the number of the save you just did.

The other two options will prompt you for a (new) file-name the first time the commands are given. Next time the commands are given it is assumed that you wish to add more output at the end of the file. If this is not the case you can close it. You are then prompted for a new file-name. Once a file-name has been established you get a prompt of the same type as the display prompts i.e. no input is needed; return of a blank line results in output of all entered atoms. If you change your mind and don't want to write anything (yet) to the file you have to type `q` to cancel (skip) output.

The files created with the output options will end up in the directory from which you are running the program. Naturally you must run the program from the same directory if you wish to read them with input commands later.

2.1.4 Geometric Commands

The top level help text on the geometric commands is displayed on command `hg` and looks as follows:

```
  The default GEOMETRIC option is a transformation to a
natural molecular coordinate-system: the origin is posi-
tioned at the center of charge, and the axes are made Princi-
pal axes i.e. eigenvectors of the (charge-)inertia tensor.
In a symmetric molecule this system is often a natural sys-
tem. A symmetry-axis becomes z-axis.
               Principal axes      command:gpr  (g)
  The continuous transformations translation, rotation and
scaling are optionally performed on a copy. The commands are:
               Translation         command:gtr  (gt)
               Rotation            command:gro  (gr)
               Scaling             command:gsc
  It is also possible to perform reflection or inversion.
These are performed on a copy unless otherwise requested.
The commands for these operations are:
               Reflection          command:gre
               Inversion           command:gin
  All these options accept input of  nf, nl  the range to
be affected by the operation. In: gpr  one can also, if
desired, specify a (different) range: ndf, ndl  to define
the new origin and axes. Further Help available after the
appropriate command has been given.
```

Note that `gtr` and `gro` allow you to specify a translation or rotation explicitly in terms of numerical input or (often more interesting) in terms of some desired result that the transformation is to have on your entered points. Also `gpr` can be used for doing rotations: if you make the `gpr`-transformation defined by three atoms the plane defined by these becomes rotated to the xy-plane. These rotations are always around an axis through the origin (which `gpr` puts at the center of charge). If you wish to rotate around another point you have to combine translation and rotation with edit commands (see next section).

When the center of charge or charge 'inertia tensor' is calculated any charges (atomic numbers) that are zero or negative do not contribute. (The program allows atomic numbers `Z` to have values from -9 to 99.)

Scaling means multiplying the input (Cartesian coordinates) with a numerical factor. This is equivalent to a change of length unit. The factor required for changing between atomic units (bohrs) and ångströms (1 Å=10^{-10} m) is known by the program can be obtained by typing a special symbol.

2.1.5 Edit Commands

The command `he` given at the top level produces following text:

```
                EDIT OPTIONS

 The available Edit options can be summarized as follows:

 EDIT OF ATOMS (and charges):
     (O) RENUMBERING (permutational) options
 Move (with automatic renumbering)    command:emo  (e)
 Order (ct or an)                     command:eor  (eo)
 Permute                              command:epe  (ep)
     (-) REMOVING options
 Remove & Renumber (to close gap)     command:err  (er)
 Remove (i.e. Delete)                 command:erm  (r)
     (+) ADDING option
 Copy (a given range)                 command:ecp  (ec)

 EDIT OF Z-MATRIX
 Z-matrix (input, display, remove)    command:ezm  (ez)

   When the command has been given a prompt will (as usual)
 appear. For all these commands one can then type:h for Help
 to get an explanation of the input and its effect.
```

These options allow you to rearrange your file of atoms in various ways that can simplify the handling of calculations. They work a bit like a line oriented text editor. If you have entered atomic numbers these are, of course, also renumbered and copied together with the coordinates of the corresponding atom. When an atom is removed with `erm` its charge is not changed so that if you then put it back in a new position, with `ige` or `ico`, it still has the old atomic number.

Combinations of edit and geometric options extend the capabilities of the calculator. For example, one can rotate around a point which is not at the origin, in spite of the fact that `gro` only rotates around an axis through the origin. Do as follows: 1. copy the point with `ecp`, 2. translate all points except the copy so that the desired point coincides with the origin (using `gtr`), 3. rotate all points except the copy as you intended (using `gro`), 4. translate all points with the vector from the point at the origin to the copy point (again using `gtr`). 5. remove the copy point (with `erm`).

The z-matrix requires that the position of each atom is specified using reference atoms. These are automatically calculated by the program unless a z-matrix file has been read, in which case the reference atoms are stored. With `ezm` you can input, display, or remove such stored information.

2.2 EVCLID Command Table

Type	Command	Abbrev.	Description
Input	`ige`	`i`	input of geometric parameters
	`ico`	`c`	input of coordinates
	`ipo`	`ip`	input of polygon or polyhedron
	`ian`	`ia`	input of atomic numbers
	`isa`	(`s`)	input of save-file (`s` if empty)
	`icf`		input of coordinate file
	`izm`	`iz`	input of z-matrix file
	`ih`		input help
Display	`dco`	↩	display of coordinates
	`dzm`	`dz`	display of z-matrix
	`dge`	`dg`	display of geometric parameters
	`dh`		display help
Output	`ozm`	`o`	output of z-matrix file
	`ocf`	`oc`	output of coordinate file
	`osa`	(`s`)	output of save-file (`s` if input)
	`oh`		output help
Geometric	`gpr`	`g`	principal axes transformation
	`gtr`	`gt`	translation
	`gro`	`gr`	rotation
	`gsc`		scaling
	`gre`		reflection
	`gin`		inversion
	`gh`		geometric help
Edit	`emo`	`e`	move (with renumbering if needed)
	`eor`	`eo`	order
	`epe`	`ep`	permute
	`err`	`er`	remove & renumber (to close gap)
	`erm`	`r`	remove
	`ecp`	`ec`	copy
	`ezm`	`ez`	z-matrix
	`eh`		edit help
Help	`h`		command menu
	`h`x		help about x, same as x`h`
	`hh`		basic help
	`quit`		terminate run

Note that the abbreviations can be used only at top level. The reason for this is that one and two letter commands have special meanings in many of the sub-levels. In particular the letter `h` gives a different help text for each sub-level (command).

2.3 On Save and Safety

When a lot of work has been done on a geometry with the program one is naturally anxious not to loose it. EVCLID is constructed to protect your data from minor mistakes; erroneous input will in most cases not terminate the run but lead to error messages. To make a safety copy of what one has stored in the program memory one only needs to give the command `s` (at top level, otherwise the full name, `osa`, must be used). If there is anything to *save* this command results in immediate writing of the memory contents to the end of a file called `ecldsv.dat`. You are notified of how many saves you have done to this file.

If the program is empty, as it is when you have just started it, the command `s` is interpreted as `isa` (input save). This leads to immediate reading into memory of the last save done. If this is not the save you wish, just give the command `isa` again and you will get a prompt that allows you to give the number of the save (on the file `ecldsv.dat`) that you wish. Whenever the program is not empty `isa` will give this prompt. You can thus keep a small library of 'molecules' for quick access on the save file.

Since every save is added at the end, the file `ecldsv.dat` will eventually become very long. You can then delete it if you don't need it anymore. If you have done this EVCLID will start a new file next time you do a save. Another possibility is to remove things from it by editing it. Each save ends with a dashed line so, by removing such a line and everything above it up to (but not including) the next dashed line, you get a new, shorter save file that can be understood by the program. You may not understand the file `ecldsv.dat` yourself though; it is not formatted to be readable but to be accurate and compact.

The commands that can be dangerous to your work are the following:

- `quit` since this command immediately terminates the run.
- `isa` since this command will put back the memory contents as they where at the time of the save (and remove everything else). [1]
- `erm` and `err` since these removing commands obviously enable you to wipe out whatever work you have done.

Attempts to over-write already entered atoms with new by means of the input options (except `isa`) lead to warnings. If you are uncertain about the outcome of a command you can perform it on a copy or give the command `s` before it is performed. If you don't like the result, all you have to do is type `isa` and read in the save that you made.

[1] If you want to merge two molecules you must read one from a named file (using `icf` or `izm`) since you can then choose to store what is read at free atom numbers, in response to the prompt line. This will not affect any previous contents of the program.

Erroneous use of the geometric operations can, of course, mess up the input so, in case of doubt, do a save first. The display (and output) options, on the other hand, are ‘passive’ and are always safe to use.

2.3.1 Memory Organisation

EVCLID’s memory consists of three somewhat independent parts. The main part is the array containing the Cartesian coordinates (x, y, z) of the atoms. A separate integer vector stores the atomic numbers, Z, of the atoms. These are all initialized to 1 and if all have the value 1 it is assumed that the user is not interested in atomic numbers. These will then not be displayed. The atomic numbers (or charges) can be changed with `ian` (and with `ipo`). They are recorded when a coordinate or z-matrix file is read.

The edit options which renumber atoms will also renumber their atomic numbers. On the other hand, removing an atom does not remove its charge. This has the advantage that if you remove an atom temporarily just to change its position it will still have the atomic number you may have given it.

There is a third storage for z-matrix reference atoms i, j, k which is even more independent. If you read a z-matrix file, the reference atoms are stored so that when you later display or output a z-matrix it will be the one you read. (If no such reference atoms are stored they are calculated according to a nearest neighbour principle by the program.) If you wish to change the contents of this part of the memory you must use the command `ezm` (edit z-matrix).

When you make a save (`osa`) all three parts of the memory are stored if they have input. If you, on the other hand, output atoms to a coordinate file, the z-matrix reference atoms, if any, will naturally not be recorded.

The top level prompt line informs you of the contents of these three parts of the memory. Typically it might look as follows:

```
Type:command   (no of atoms:8)  (input charge:2) (z-mtx rows:2)          ?xyz?
```

Note that only a charge differing from 1 (i.e. a non-hydrogen atom) is counted as an input charge. The number of z-matrix rows should normally be, at least, two less than the number of atoms since the first and second atoms do not require referencing.

Chapter 3

Geometric Parameters

3.1 Distance, Angle, and Dihedral Angle

The shape of an $N-$atom molecule is usually given in a coordinate independent way in terms of (minimum) $3N - 6$ geometric parameters which can chosen among the *three* types:

distance between atoms i and j,

$$0 \leq r(i,j), \tag{3.1}$$

angle (or *bond angle*) at j between the line segments j-i and j-k

$$0 \leq \theta(i,j,k) \leq 180^\circ, \tag{3.2}$$

dihedral angle (or *torsion angle*) between the two planes defined by i-j-k and j-k-l meeting at the line j-k and counted positive from the plane containing i to that containing l according to the right hand rule around the j-k axis oriented from j to k,

$$-180^\circ < \phi(i,j,k,l) \leq 180^\circ. \tag{3.3}$$

The sign convention for dihedral angles is shown geometrically in figure 3.1. It is in agreement with the convention used in chemistry [8] and crystallography [9].

This sign convention can be equivalently expressed as follows: ϕ is positive if atom l is on the positive side of the oriented triangle $i \to j \to k \to i$, otherwise negative. Which of the two is more convenient depends on personal taste and geometric situation. The importance of the sign can be appreciated from the fact that the specifications of the geometries of the two enantiomers of a chiral molecule may differ only in such a sign.

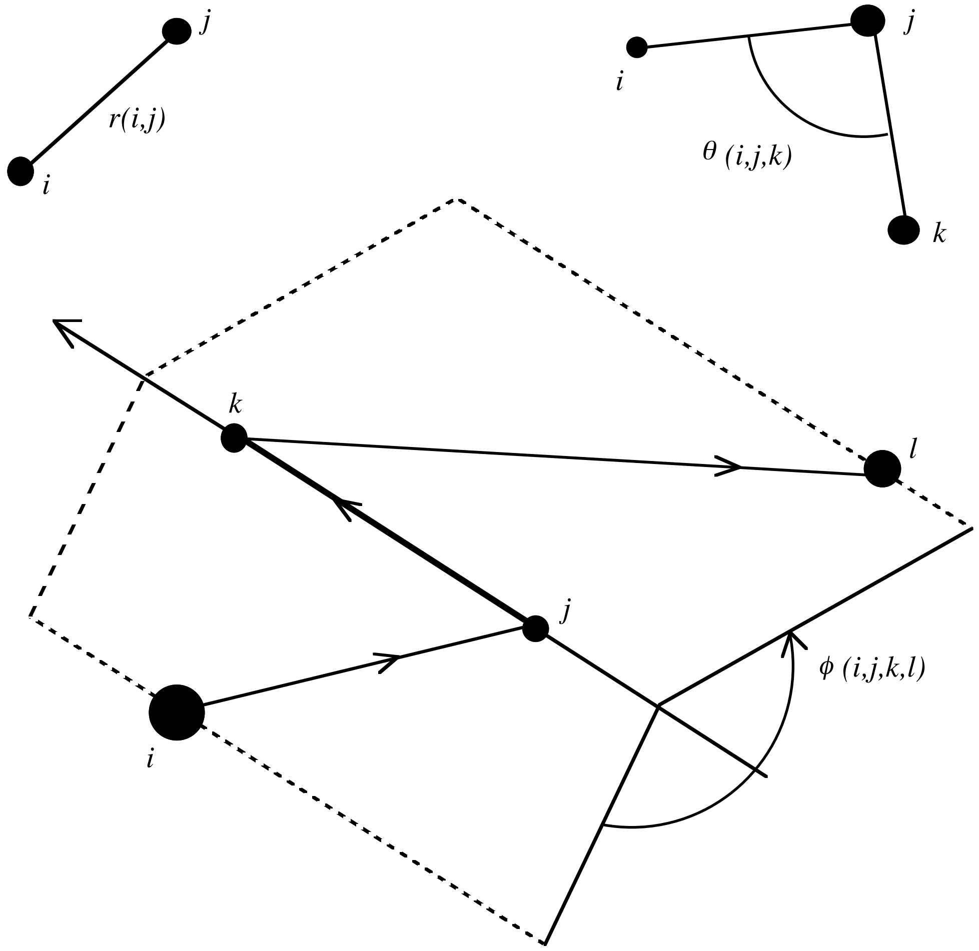

Figure 3.1: Familiarity with the three geometric parameters that are shown in this figure is imperative for the effective use of the EVCLID program. Note the sign, or orientation, convention for the dihedral angle indicated by the arrows. It can be expressed as follows: if the thumb of the right hand points from j to k the fingers curl in the positive direction from the plane containing i to the plane containing l. If this is the 'short' way from i to l the angle is positive and $< 180°$, as in the figure. Otherwise the angle is either greater than $180°$ or negative. In the program the notation for θ is `tht` and that for ϕ is `phi`.

Assuming that one knows the Cartesian coordinates of the atoms in some coordinate system these definitions can be conveniently translated into explicit formulas. Let $\mathbf{a}_i$ be the position vector of atom i, then we have:

$$r(i,j) \equiv |\mathbf{a}_j - \mathbf{a}_i|, \tag{3.4}$$

$$\mathbf{b}_{ij} \equiv (\mathbf{a}_j - \mathbf{a}_i)/r(i,j), \tag{3.5}$$

$$\theta(i,j,k) \equiv \arccos(\mathbf{b}_{ji} \cdot \mathbf{b}_{jk}), \tag{3.6}$$

$$\mathbf{b}_{ijk} \equiv (\mathbf{b}_{ij} \times \mathbf{b}_{ik})/\sin\theta(k,i,j), \tag{3.7}$$

$$\phi(i,j,k,l) \equiv \operatorname{sign}(\mathbf{b}_{ij} \cdot \mathbf{b}_{jkl}) \cdot \arccos(\mathbf{b}_{ijk} \cdot \mathbf{b}_{jkl}). \tag{3.8}$$

It is important to notice the sign attached to the dihedral angle. Though it is not often discussed one must make a distinction between signed and un-signed angles. Dihedral angles belong to the first category and the angles of equation (3.6) to the second, un-signed. Signed angles require that a positive orientation for rotations in the plane of the angle is defined (see for example Pedoe [7] for a discussion of this).

The formulas just given determine the relative geometric parameters of atoms in molecules provided the Cartesian coordinates are known. These formulae are used by EVCLID to calculate geometric parameters for display (by the commands `dzm` and `dge`). The much more difficult problem of finding the Cartesian coordinates when the position of an atom is given by geometric parameters referencing atoms with known positions, is also solved by EVCLID. This is discussed in the next section.

3.2 The Five Geometric Coordinates

Since space is three dimensional three *coordinates* will, in general, specify the position of a point in space. Any coordinate system will require some *reference point* (for example the 'origin') and *reference directions* (the axes). The coordinate system is then some rule that, at least locally, gives a one to one mapping between the points of space and three numbers. Well known examples are Cartesian (or rectilinear) coordinates, cylindrical and spherical coordinates.

Instead of having *one* reference point and axes one can define a coordinate system using *three* reference points. As long as three points are not on the same line they can be used to define a coordinate system. This means that in any non-linear molecule one can take three suitable atoms to define a coordinate system and give the positions of other atoms in the molecule in terms of this system. In this way one gets a, so called, *coordinate independent* description of the geometry of the molecule. One still needs coordinates but these are not arbitrary; instead they refer to a system that is fixed in (and by) the molecule.

We now assume that we have three atoms, i, j and k, which define a coordinate system or, equivalently, whose positions we consider given. How can one then give 'coordinates' for another atom n? If we wish to use the geometric parameters (distance, angle, dihedral angle) discussed in the previous section it turns out that there are *five* different 'coordinates'. These five are given in table 3.1. Only these five qualitatively different positions for the unknown atom number n exist because of the permutational symmetries

$$r(i,j) = r(j,i), \tag{3.9}$$

$$\theta(i,j,k) = \theta(k,j,i), \tag{3.10}$$

and

$$\phi(i,j,k,l) = -\phi(l,j,k,i) = -\phi(i,k,j,l), \tag{3.11}$$

which hold according to the definitions of these quantities.

The reason that one must consider the five 'rules' of table 3.1 for finding a number that describes the position of atom n, as different coordinates is that they correspond to different surfaces. The points of space that have the same value for some coordinate lie on a (two-dimensional) surface, the *coordinate surface*. For Cartesian coordinates, for example, these surfaces are planes. For the five specifications listed in the table one gets the surfaces given in the middle column of table 3.1. It is easy to understand that keeping a distance fixed corresponds to allowing n to lie on a sphere with that distance as radius. The other surfaces are a bit more difficult to visualize, see figure 3.2, but only the 'fifth surface' is really hard.

Algebraic equations for the surfaces on the general form $f(\mathbf{r}) = 0$ can easily be derived by means of vector calculus. Using the notation of formulae 3.4 - 3.8, and the obvious shorthand r_{ni} for $r(n,i)$ etc., the explicit formulae for the functions f are as follows:

$$\begin{aligned}
f_1(\mathbf{r};\ \mathbf{a}_i, r_{ni}) &= |\mathbf{r}-\mathbf{a}_i| - r_{ni}, && (3.12)\\
f_2(\mathbf{r};\ \mathbf{a}_i, \mathbf{b}_{ij}, \theta_{nij}) &= (\mathbf{r}-\mathbf{a}_i)\cdot\mathbf{b}_{ij} - |\mathbf{r}-\mathbf{a}_i|\cos\theta_{nij}, && (3.13)\\
f_3(\mathbf{r};\ \mathbf{a}_i, \mathbf{b}_{ij}, \mathbf{b}_{ijk}, \phi_{nijk}) &= (\mathbf{r}-\mathbf{a}_i)\cdot[\,\mathbf{b}_{ijk}\cos\phi_{nijk} && (3.14)\\
&\quad +(\mathbf{b}_{ijk}\times\mathbf{b}_{ij})\sin\phi_{nijk}\,], &&\\
f_4(\mathbf{r};\ \mathbf{a}_i, \mathbf{a}_j, \theta_{jni}) &= (\mathbf{r}-\mathbf{a}_i)\cdot(\mathbf{r}-\mathbf{a}_j) && (3.15)\\
&\quad -|\mathbf{r}-\mathbf{a}_i||\mathbf{r}-\mathbf{a}_j|\cos\theta_{jni}, &&\\
f_5(\mathbf{r};\ \mathbf{a}_i, \mathbf{b}_{ik}, \mathbf{b}_{ij}, \phi_{knij}) &= [(\mathbf{r}-\mathbf{a}_i)\times\mathbf{b}_{ik}]\cdot[(\mathbf{r}-\mathbf{a}_i)\times\mathbf{b}_{ij}] && (3.16)\\
&\quad -|(\mathbf{r}-\mathbf{a}_i)\times\mathbf{b}_{ik}||(\mathbf{r}-\mathbf{a}_i)\times\mathbf{b}_{ij}|\cos\phi_{knij}. &&
\end{aligned}$$

The equations for the surfaces corresponding to dihedral angles (3.14 and 3.16) must be supplemented with conditions that the point is in the correct half-space.

Type no	Notation	Surface	Degree	Topology
1	$r(n,i)$	Sphere	2	compact
2	$\theta(n,i,j)$	Cone	2	infinite
3	$\phi(n,i,j,k)$	Half-plane	1	infinite
4	$\theta(j,n,i)$	Torus (rotated circle arc)	4	compact
5	$\phi(k,n,i,j)$	'Hyperbolic cone'	4	infinite

Table 3.1: The five geometric coordinates (for atom n in terms of reference atoms i, j, k)

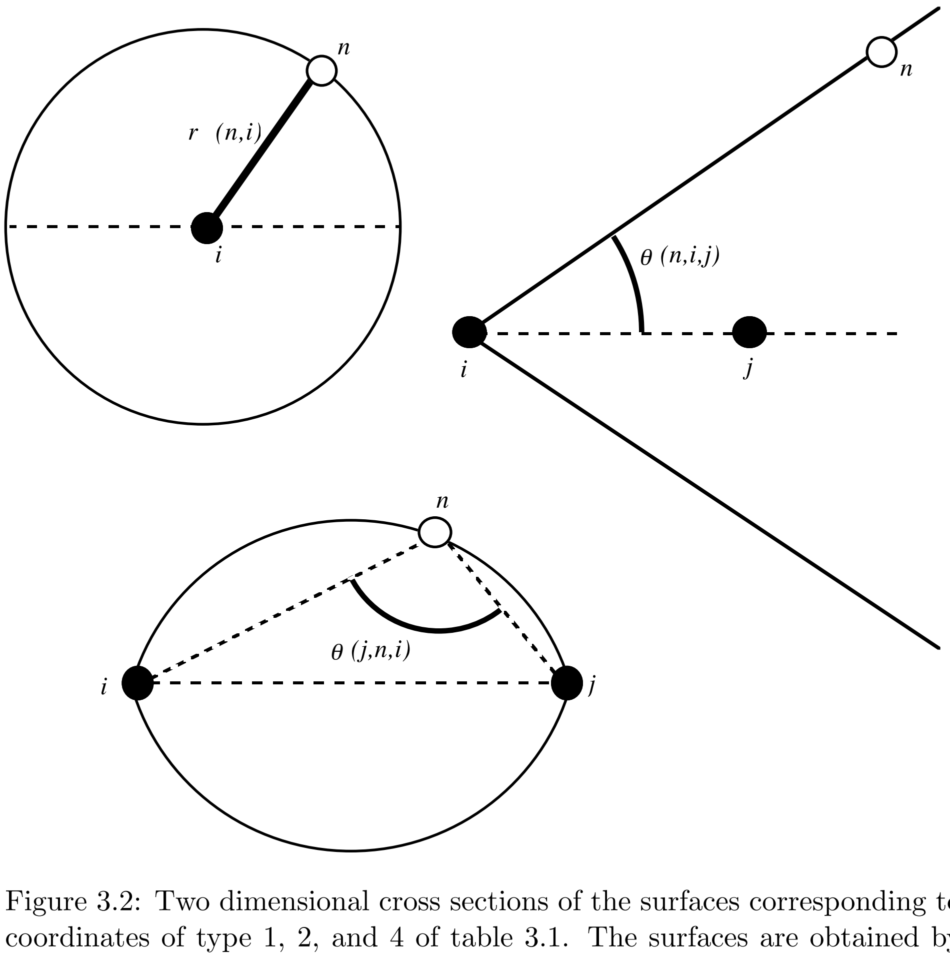

Figure 3.2: Two dimensional cross sections of the surfaces corresponding to coordinates of type 1, 2, and 4 of table 3.1. The surfaces are obtained by rotating around the horizontal dashed line and correspond to a sphere, a cone and a rotated arc of a circle. For the case that $\theta(j,n,i) = 90°$ this last surface is also a sphere since then the arc will be a semi-circle.

3.2.1 Numerical Calculation of Point of Intersection

When three geometric coordinates for a point are given three surfaces on which the point must lie have been defined. The position of the point must then be a common point of all three surfaces. In general this will be a point where the three surfaces intersect.

The Cartesian coordinates for a point specified by three geometric coordinates must thus be calculated as the point of intersection of three surfaces of the types given by equations 3.12 - 3.16. This entails finding the root(s) $\mathbf{r} = \mathbf{a}_n$ of the, in general, non-linear system of equations:

$$f_{\alpha_\lambda}(\mathbf{r};\, \mathsf{C}_\lambda) = 0, \qquad \lambda = 1, 2, 3. \tag{3.17}$$

Here $\alpha_\lambda \in \{1, 2, 3, 4, 5\}$ and C_λ stands for the collection of constants that determine the shape, size, position, and orientation of the surface. In an earlier treatment of this problem by the author [3], which only treated the three first of the five coordinates, it was found that the most stable and accurate way to solve this system is to look for the minima of the sum of the squares to the three surface functions

$$F(\mathbf{r}) = \sum_{\lambda=1}^{3} f^2_{\alpha_\lambda}(\mathbf{r};\, \mathsf{C}_\lambda). \tag{3.18}$$

A minimum of this function, $F(\mathbf{r})$, is easily found by searching for a zero of its gradient by means of the Newton-Raphson method:

$$\mathbf{r}_{\nu+1} = \mathbf{r}_\nu - \mathbf{H}(\mathbf{r}_\nu)\mathbf{g}(\mathbf{r}_\nu), \tag{3.19}$$

where $\mathbf{g}(\mathbf{r})$ is the gradient of $F(\mathbf{r})$ and $\mathbf{H}$ is the inverse of its Hessian, see Dahlquist and Björck [10]. EVCLID uses analytic formulae for the gradient and Hessian of the function $F(\mathbf{r})$, calculated on the basis of the formulae 3.12 - 3.16, to perform this root search. Very near the point of intersection one can switch to a more direct method based on linearization of the system 3.17. In some cases a specification might involve surfaces that are tangent rather than intersecting. In this case EVCLID resorts to the brute force 'Amoeba' method adopted from Press et al. [6], so that an existing root will nearly always be found.

3.3 Specifying the Relative Positions of Atoms

3.3.1 The First Three Atoms

When one specifies the positions of atoms in a molecule in a coordinate independent way the position of one atom must remain completely unspecified. We will refer to this atom as the first atom, or atom number 1. The second atom (atom number 2) can only have a given distance, $r(2, 1)$, to the first specified. To get a Cartesian coordinate system defined by the molecule it is convenient to let the first atom define the origin and the second the direction of the x-axis. EVCLID uses this convention; when the program is empty `ige` (input of geometric parameters) prompts for one geometric parameter, which must be a distance. On the terminal this looks as follows:

```
Type:command      (h = help,  hh = basic help)                          ?xyz?
Evc>> ige
Enter: ge i j ( k ( l))    1 line                       (h = help)      *ige*
ige1> 2.81  2 1
  1         0.000000000       0.000000000       0.000000000
  2         2.810000000       0.000000000       0.000000000
```

The second atom is thus taken to define the positive direction of the x-axis. (It is not necessary to give the first two atoms the numbers 1 and 2, any numbers will do). EVCLID always echos the number and the Cartesian coordinates of stored atoms when they are entered by means of `ige` or `ico`.

We now assume that two atoms have been specified in the above manner. The six different specifications possible for the third atom are shown in figure 3.3 and tabulated in table 3.2. By convention the third atom is taken to define the positive y-axis, that is it will get a positive y-coordinate in the molecule define coordinate system (unless it is on line with 1 and 2). EVCLID will accept any of the six types of specification. Typically this might look as follows:

```
Enter: ge i j ( k ( l))    2 lines                      (h = help)      *ige*
ige1> 2.81  3 1
ige2> 60.0  3 1 2
  3         1.405000000       2.433531385       0.000000000
```

The notation should be clear: the first line means that $2.81 = r(3,1)$ and the second that $60^\circ.0 = \theta(3,1,2)$. Here we used the specification of type 2 in table 3.2. This is the preferred specification because it always give a unique root. Specifications that may have two roots (number 3 and 4) should, of course, be avoided unless you really know what you are doing. One can ask EVCLID to check if there is a second root (see the help-text `hx` in the `ige` command).

One should note that the third atom must not lie on the same line as the first two if it is to define a direction for the y-axis. Once a y-axis has been defined a (molecule defined) coordinate system is completely determined. Any subsequent atoms that you enter must be placed in this coordinate system and must thus be specified with three parameters (coordinates). `ige` allows you to put the third atom on the same line as the first two. In that case a fourth atom, not on the line, must be treated in a special way so that its third parameter is suppressed (see the help-text `hx` in the `ige` command). This might look as follows

1.	$r(3,1)$	$r(3,2)$	1 root if $r(3,1) + r(3,2) > r(1,2)$, else 0
2.	$r(3,1)$	$\theta(3,1,2)$	1 root, always
3.	$r(3,1)$	$\theta(3,2,1)$	1 root if $r(3,1) > r(1,2)$, else 2 or 0
4.	$r(3,1)$	$\theta(1,3,2)$	1 root if $r(3,1) < r(1,2)$, else 2 or 0
5.	$\theta(3,1,2)$	$\theta(3,2,1)$	1 root if $\theta(3,1,2) + \theta(3,2,1) < 180°$, else 0
6.	$\theta(3,1,2)$	$\theta(1,3,2)$	1 root if $\theta(3,1,2) + \theta(1,3,2) < 180°$, else 0

Table 3.2: The six different specifications of the third atom (3) with respect to atoms 1 and 2. Only the roots on one side of the line 1-2 are counted. See figure 3.3.

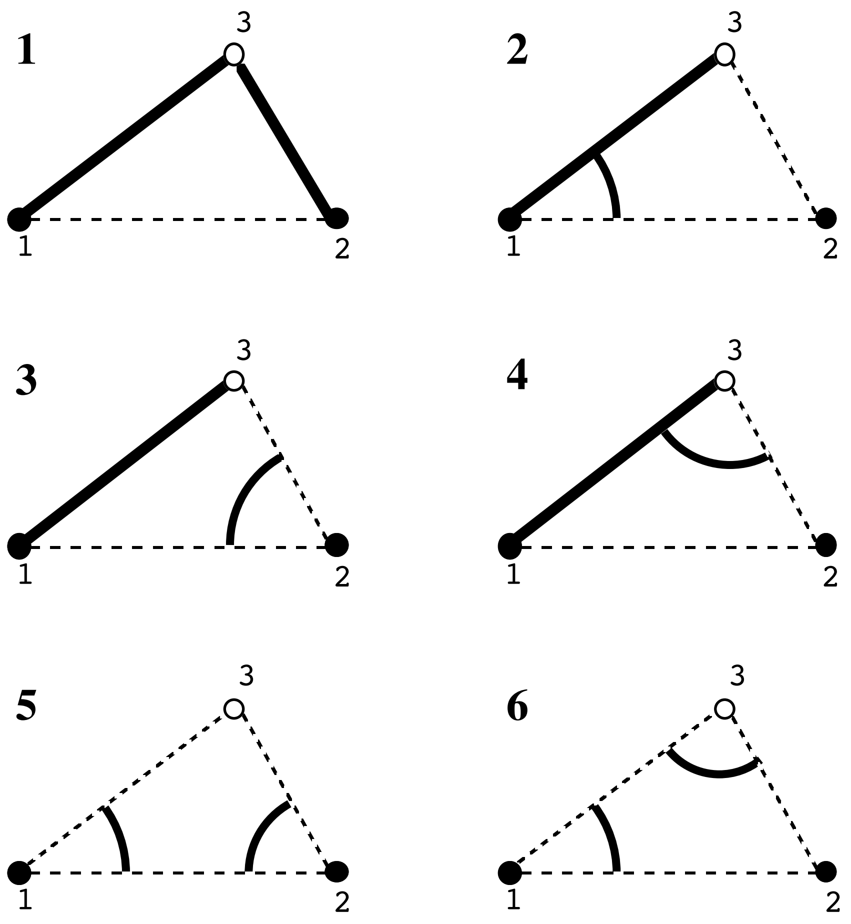

Figure 3.3: The six different ways in which the (two-dimensional) position of a third point (3) can be specified with respect to two given points (1 and 2) using distances and angles. Cases which arise simply by permuting atoms 1 and 2 have not been counted as different. The corresponding algebraic notation is given in table 3.2.

```
Evc>> ige
Enter: ge i j ( k ( l))    1 line                          (h = help)     *ige*
ige1> 1.5  2 1
  1         0.000000000      0.000000000      0.000000000
  2         1.500000000      0.000000000      0.000000000
Enter: ge i j ( k ( l))    2 lines                         (h = help)     *ige*
ige1> 180. 3 2 1
When angle 0 or 180 two parameters suffice.
ige2> 1.5  3 2
  3         3.000000000      0.000000000      0.000000000
Enter: ge i j ( k ( l))    3 lines                         (h = help)     *ige*
ige1> 1.5  4 1
ige2> 90.0  4 1 2
ige3> d
  4         0.000000000      1.500000000      0.000000000
```

The letter `d` is a command that tells `ige` that this atom is to be treated as if it was the third atom. We now proceed to the general case when three coordinates must be given.

3.3.2 The General Case

The specification of the position of an atom in general may be such that two of the parameters reference two atoms in such a way that one of the two-dimensional (third atom) cases of table 3.2 arise. The purpose of the third parameter (or coordinate) is then to fix the orientation of the triangle of atoms around the axis defined by the two reference atoms. See figure 3.4. The most 'direct' way of doing this is tho use a dihedral angle which measures the angle between two planes that meet at the line defined by the two reference atoms of the triangle. This is case number v in figure 3.4. If we start EVCLID from scratch again input of the first four atoms might then look as follows:

```
Enter: ge i j ( k ( l))    1 line                          (h = help)     *ige*
ige1> 1.5  2 1
  1         0.000000000      0.000000000      0.000000000
  2         1.500000000      0.000000000      0.000000000
Enter: ge i j ( k ( l))    2 lines                         (h = help)     *ige*
ige1> 45.0  1 3 2
ige2> 90.0  3 1 2
  3         0.000000000      1.500000000      0.000000000
Enter: ge i j ( k ( l))    3 lines                         (h = help)     *ige*
ige1> 1.5  4 1
ige2> 90.0  2 1 4
ige3> 90.0  3 1 2 4
  4         0.000000000      0.000000000      1.500000000
```

Here the specification of atom three is of the type 6 of table 3.2. Note the dihedral angle $90^\circ.0 = \phi(3,1,2,4)$ in the last input line. The opposite sign would have given a point with negative z-coordinate. Since it is common to make sign-errors, and since it is fairly common that one, in fact, wants both atoms, there is a special command that will produce the point corresponding to the opposite sign of the last entered dihedral angle. To get it, just type the letter `a`:

```
Enter: ge i j ( k ( l))    3 lines                          (h = help)     *ige*
ige1> a
  0         0.000000000       0.000000000      -1.500000000
  Atom number: n=  4  already stored; to delete old
and store new position, confirm this n, else change it:
Enter: n                                        (n=0 to cancel)
#n?>> 5
  5         0.000000000       0.000000000      -1.500000000
```

EVCLID finds the new point but does not know what number it should be stored as. If you made a sign error you can type `4` and have the old one replaced. If you, on the other hand, want both, just type `5`, as in the example, (or any free number) to store the new point.

3.3.3 Multiple Roots and the Orientation Convention

As indicated in figure 3.4 there are eight qualitatively different ways of choosing a third parameter when the first two belong to one of the six two-dimensional sub-cases of table 3.2. This gives 48 different combinations. There are also some three-dimensional specifications which do not contain a two-dimensional sub-case. There is thus a very large number of possibilities, but it is not wise to use them all indiscriminately. The question of the uniqueness of the roots should considered. The root is unique in the example above and is so in general for case v of figure 3.4 with the dihedral angle $\phi(4,1,2,3)$ as third parameter. The specifications corresponding to i and ii of figure 3.4 can also be quite useful even though they give two roots: the point 4 in the figure and its mirror image below the plane of the 1-2-3 triangle.

To get around this one must use a sign, or orientation, convention. In EVCLID this convention is that, whenever there is no dihedral angle in the specification, the root is assumed to lie on the positive side of the oriented triangle[1] of reference atoms. This means that the root is sought on the side of the triangle that the thumb of the right hand points to when the fingers curl round the vertices in numerical (increasing) order. This is the side of point 4 in figure 3.4. Should you wish the other point you just give

[1]If there are more than three reference atoms the first three are taken to define the triangle.

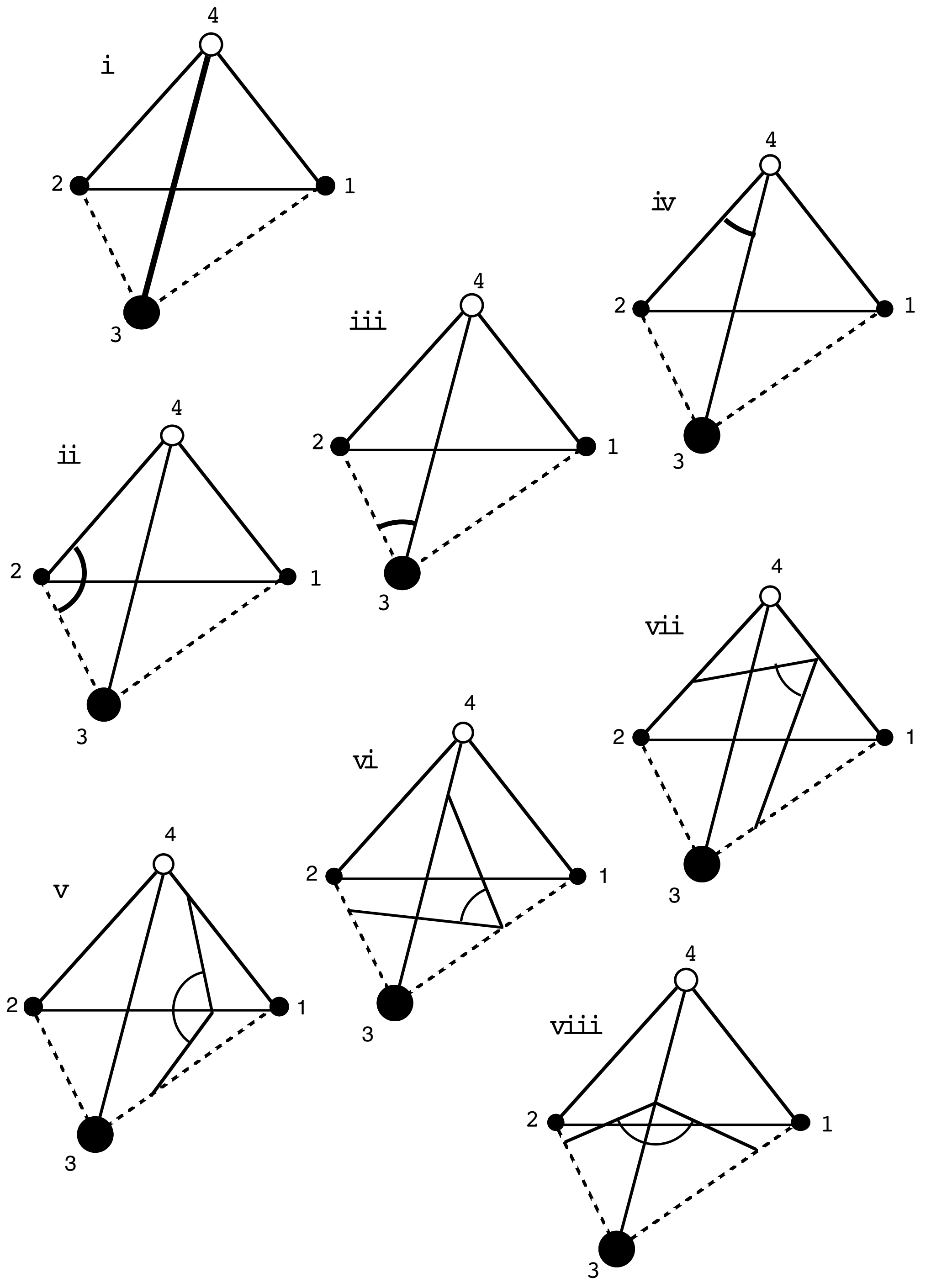

Figure 3.4: The eight different ways in which the (three-dimensional) position of a fourth point (4) can be specified when the triangle 1-2-4 already is given by one of the six two-dimensional sub-cases of figure 3.3.

a minus-sign to one of the parameters. If you got the wrong point you can get the other one by typing the letter `b`. This is illustrated in the following example.

```
Type:command      (h = help,  hh = basic help)                           ?xyz?
Evc>> ipo
Enter: name  (  size (  Z))                (h = help)                    *ipo*
ipo>> 3  1.5
  2-dimensional   3-gon in range:  1 -  3
Type:command   (no of atoms:3)                                           ?xyz?
Evc>> dco;
Enter:(ct)   (nf (nl))                         (h=help, q=skip)          *dco*
               n       x                y                z
               1      0.86602540      0.00000000      0.00000000
               2     -0.43301270      0.75000000      0.00000000
               3     -0.43301270     -0.75000000      0.00000000
Type:command   (no of atoms:3)                                           ?xyz?
Evc>> dco; c
Enter:(ct)   (nf (nl))                         (h=help, q=skip)          *dco*
               n      rh              phi              z
               1      0.86602540      0.00000000      0.00000000
               2      0.86602540    120.00000000      0.00000000
               3      0.86602540   -120.00000000      0.00000000
Type:command   (no of atoms:3)                                           ?xyz?
Evc>> ige
Enter: ge i j ( k ( l))    3 lines                        (h = help)     *ige*
ige1> 1.5  4 1; 1.5  4 2; -1.5  4 3
  4         0.000000000      0.000000000     -1.224744871
Enter: ge i j ( k ( l))    3 lines                        (h = help)     *ige*
ige1> b
  0         0.000000000      0.000000000      1.224744871
  Atom number: n=  4  already stored; to delete old
and store new position, confirm this n, else change it:
Enter: n                                       (n=0 to cancel)
#n?>> 4
  4         0.000000000      0.000000000      1.224744871
```

Here the input option `ipo` (input of regular polygon or polyhedron) first generates an equilateral triangle of side length 1.5. Then the Cartesian and the cylindrical coordinates are displayed to check it. After that `ige` is invoked to enter a fourth point that completes a regular tetrahedron. Three distances (1.5) are given. Instead of returning each line separately they are separated with semicolons (;). The minus sign on the last distance causes the root to be the one ‘below’ the xy-plane. The command `b` then gives the one above the plane.

3.3.4 Midpoint or Equal Distances Specification

As a subcommand to `ige` you can specify a new point to be located so that it has equal distances to two, three, or four previoulsy entered points. This is useful to get a new point (atom) at the midpoint of some previously entered atoms. To do this, give the command `ige`. When the prompt appears you put the lower case letter `r` first in the input line. Then the number of the new atom you wish to add and after that the numbers ofthe atoms to which it is to have equal distances. For example

```
Evc>> ige
Enter: ge i j ( k ( l))    3 lines                         (h = help)     *ige*
r 9 1 3
```

will put the new atom 9 at the midpoint between atoms 1 and 3.

In the case of two reference atoms the result is simply a new atom at the midpoint of the two existing atoms. In the case of three reference atoms the new point will be at the center of the unique circle that goes through the three atoms. It is thus in the same plane as these. The three atoms may not be on a line since then this point is not well defined.

In the case of four reference atoms the new point will be at the center of the sphere defined by the four atoms. These may then, of course, not be in the same plane.

3.3.5 The Z-Matrix

In the quantum chemistry program Gaussian 90 [1] (and earlier and later versions) there is a special form for the geometry input called a 'z-matrix'. The z-matrix specifies the positions of the atoms relative to each other according to a scheme that corresponds to type 2 of figure 3.3 and then case v of figure 3.4. This means that the position of atom n is given by three geometric parameters as follows:

$$r(n,i),\ \theta(n,i,j),\ \phi(n,i,j,k). \tag{3.20}$$

Here i, j, k are reference atoms with known positions. This scheme, which was originally suggested by Eyring [11], has the advantage that it always gives a unique root (position) as mentioned above. It also has the advantage that one can write down the fairly simple analytic formula

$$\begin{gathered}\mathbf{r}_n = \mathbf{a}_i + r(n,i)\,\{\cos\theta(n,i,j)\,\mathbf{b}_{ij} + \\ + \sin\theta(n,i,j)\,[\cos\phi(n,i,j,k)\,(\mathbf{b}_{ijk}\times\mathbf{b}_{ij}) - \sin\phi(n,i,j,k)\,\mathbf{b}_{ijk}]\,\}\end{gathered} \tag{3.21}$$

for the position vector of atom n. Here the notation of equations 3.4 - 3.8 has been used. The z-matrix has advantages over coordinates when it comes to geometry optimizations, though this has been questioned in recent times, see Baker and Hehre [12]. Conceptually such a coordinate independent description is superior to using Cartesian coordinates and once one has got used to z-matrices, or geometric parameters in general, they do convey much more immediate and useful information about the molecule. Their usefulness in molecular dynamics has been discussed by Noid et al. [13].

EVCLID can be used to produce, read, write, and edit z-matrices. As an example, let us input the methane molecule CH_4 and display its z-matrix. First we put a carbon atom at the origin using `ico`, then we give it charge $Z = 6$, then we use `ipo` to make the tetrahedron (t) of hydrogens (with radius 1.093 Å).

```
Evc>> ico
Enter:(n) x  y  z      or: ct      (h = help)                         *ico*
ico>> 0 0 0
   1         0.000000000      0.000000000      0.000000000
Enter:(n) x  y  z              (h = help)                            x*ico*
ico>> ian; 6
Enter: Z   or: n1 Z1 ( n2 Z2..( n5 Z5))   or: nf nl Z     (h = help)     *ian*
Atom no:  1   given  Z= 6  i.e. is a  C -atom.
All entered atoms have now been given charge.
Type:command   (no of atoms:1)  (input charge:1)                      ?xyz?
Evc>> ipo
Enter: name  (  size (  Z))                (h = help)                 *ipo*
ipo>> t  -1.093
  3-dimensional   4-gon in range:  2 -  5
Type:command   (no of atoms:5)  (input charge:1)                      ?xyz?
Evc>> dzm;
Enter: ( nf ( nl ( i j k)))   (a) (o) (#)    (h=help, q=skip)         *dzm*
Symbol  Z      n,   i,   r(n,i)    ,  j, tht(n,i,j),  k, phi(n,i,j,k)
 C1     6      1
 H1     1      2    1   1.0930000
 H2     1      3    1   1.0930000    2  109.47122
 H3     1      4    1   1.0930000    2  109.47122    3   120.00000
 H4     1      5    1   1.0930000    2  109.47122    3  -120.00000
Type:command   (no of atoms:5)  (input charge:1)                      ?xyz?
Evc>> dzm
Enter: ( nf ( nl ( i j k)))   (a) (o) (#)    (h=help, q=skip)         *dzm*
dzm>> a
Symbol  Z      n,   i,   r(n,i)    ,  j, tht(n,i,j),  k, tht(n,i,k),     sign
 C1     6      1
 H1     1      2    1   1.0930000
 H2     1      3    1   1.0930000    2  109.47122
 H3     1      4    1   1.0930000    2  109.47122    3   109.47122       -1
 H4     1      5    1   1.0930000    2  109.47122    3   109.47122        1
```

Note that the first atom is unspecified, for the second the distance to the first is given, for the third a distance an angle. Only the fourth and fifth atoms are given by three parameters. The CHC-angle 109°.47 is the famous 'tetrahedral angle' which has the exact expression $2\arcsin\sqrt{2/3}$ (see for example Gillespie and Hargittai [14]). The second z-matrix, obtained by typing an `a` in response to the `dzm`-prompt-line, is a so called 'alternative' z-matrix in which the dihedral angle has been replaced by the bond angle $\theta(n,i,k)$. This corresponds to case ii of figure 3.4. As discussed above the root is then no longer unique but becomes so if one specifies on which side of the oriented triangle $i-j-k$ it is. This is done by the `-1` and `1` in the rightmost column; `1` means that it is on the positive side and vice versa.

The reference atoms i, j, k for atom n are determined as follows. To find i the four (if there are that many) nearest neighbours of atom n, with numbers less than n are found. The nearest is chosen as reference atom i unless it is a hydrogen $Z = 1$; these are avoided. If all four are hydrogens the nearest is taken. If several are equally near the one with the lowest number is taken. To find j the four nearest atoms to i are found. Hydrogens are avoided as well as reference atoms that make the angle $\theta(n,i,j)$ zero or 180°. Finally, for atom k hydrogens are not avoided[2] but the angle $\theta(i,j,k)$ should not be zero or 180°.

To get a good automatic z-matrix one should give low numbers to atoms with many bonds. If the automatic z-matrix is unsuitable for some reason one can edit the z-matrix using the `ezm`-command. This might look as follows.

```
Type:command   (no of atoms:5)  (input charge:1)                           ?xyz?
Evc>> ezm
Enter: n i j k (a)    or: x (n)(#)   to edit z-matrix   (h = help)       *ezm*
ezm>> 3 2 1
Z-matrix reference atoms for atom   3  stored
Enter: n i j k (a)    or: x (n)(#)   to edit z-matrix   (h = help)       *ezm*
ezm>> 5 1 2 3 a
Z-matrix reference atoms for atom   5 stored
Enter: n i j k (a)    or: x (n)(#)   to edit z-matrix   (h = help)       *ezm*
ezm>> dzm;
Enter: ( nf ( nl ( i j k)))   (a) (o) (#)    (h=help, q=skip)            *dzm*
Symbol  Z      n,   i,   r(n,i)    ,  j, tht(n,i,j),  k, ang(n,i,(j,) k) sign
 C1     6      1
 H1     1      2    1   1.0930000
 H2     1      3    2   1.7848615    1   35.26439
 H3     1      4    1   1.0930000    2  109.47122    3   120.00000
 H4     1      5    1   1.0930000    2  109.47122    3   109.47122        1
Type:command   (no of atoms:5)  (input charge:1) (z-mtx rows:2)         ?xyz?
```

[2]The middle reference atoms i and j should have two bonds but there is no reason for k to have that.

Here we have forced the third atom to have $i = 2$ and $j = 1$, and we have forced the fifth atom to be specified in the alternative way. When there are stored z-matrix rows these will be used by `dzm` (and `ozm`). If you want to look at the calculated z-matrix you can type the character `#`.

With the command `ozm` you can output a z-matrix to a file and get a format suitable for the Gaussian-programs. This looks as follows:

```
Evc>> ezm
Enter: n i j k (a)    or: x (n)(#)   to edit z-matrix   (h = help)       *ezm*
ezm>> r
All z-matrix information removed
Enter: n i j k (a)    or: x (n)(#)   to edit z-matrix   (h = help)       *ezm*
ezm>> ozm
Type: file-name  (format)   for z-matrix output     (h = help)          f*ozm*
file> methane.zm
Enter: ( nf ( nl ( i j k)))  (a)   (/=close)   (h=help, q=skip)          *ozm*
ozm>>
C
H    1  1.0930000
H    1  1.0930000   2   109.47122
H    1  1.0930000   2   109.47122   3   120.00000
H    1  1.0930000   2   109.47122   3  -120.00000
```

Here we first removed the stored z-matrix information (rows), since it wasn't particularly useful. The output on the screen is the same as the output to the file. EVCLID can also read files produced in this way. When this is done the Cartesian coordinates are calculated using formula 3.21 and stored. Also the reference atoms i, j, k are stored so that if you display or output a z-matrix it will be exactly the one that was read. To display the calculated (automatic) z-matrix, all you have to do is type `dzm;#`, but if you wish to output it you must first go to `ezm` and remove the stored information.

EVCLID can also read, so called symbolic z-matrices used in the Gaussian programs. For example if the file `methane.zm` is changed to look as follows

```
C
H    1  rCH
H    1  rCH   2   109.47122
H    1  rCH   2   109.47122   3   120.00000
H    1  rCH   2   109.47122   3  -120.00000

rCH 1.093
```

it can still be read by EVCLID (for further examples, see below). This looks as follows.

```
Type:command      (h = help,  hh = basic help)                         ?xyz?
```

```
Evc>> izm
Type: file-name         for z-matrix input   (h = help)                 f*izm*
file> methane.zm
Enter: nf (nl)    for input atoms from z-matrix-file    (h = help)       *izm*
izm>> 1
   5 atoms read from z-matrix file: methane.zm
Type:command   (no of atoms:5)  (input charge:1) (z-mtx rows:3)          ?xyz?
Evc>> dzm;
Enter: ( nf ( nl ( i j k)))   (a) (o) (#)    (h=help, q=skip)            *dzm*
Symbol  Z      n,   i,   r(n,i)    ,  j, tht(n,i,j),  k, phi(n,i,j,k)
 C1     6      1
 H1     1      2    1   1.0930000
 H2     1      3    1   1.0930000    2  109.47122
 H3     1      4    1   1.0930000    2  109.47122    3   120.00000
 H4     1      5    1   1.0930000    2  109.47122    3  -120.00000
Type:command   (no of atoms:5)  (input charge:1) (z-mtx rows:3)          ?xyz?
Evc>> ezm;d
Enter: n i j k (a)    or: x (n)(#)   to edit z-matrix   (h = help)       *ezm*
  n,    i,   j,   k,  alt
  3     1    2
  4     1    2    3
  5     1    2    3
```

To summarize, the following commands concern themselves with z-matrices: `izm`, `dzm`, `ozm`, and `ezm`. For more complete information, study the help texts of these commands.

3.3.6 Some Final Examples

Assume that we know that the C_2H_4-molecule has CH-distance 1.086, CHC-angle 117.3, and CC-distance 1.337 and that we want its Cartesian coordinates (or z-matrix). If we use the `ige`-command of Evclid we find that it is easy to input the first carbon and its two hydrogens, but then we miss the HCC-angle. There are several ways around this. One is, of course, to use a calculator to find the angle from the formula $[360 - \theta(\mathrm{HCH})]/2$ but this would not work in the similar case that arises for the C_2H_6-molecule. Once the first carbon and its hydrogens have been entered one has an isosceles triangle, for the case of C_2H_4 (and a regular trigonal pyramid in the case of C_2H_6). This isosceles triangle (or pyramid) is placed ‘obliquely’ in the Cartesian coordinate system. By means of the `gpr`-command (transformation to principal axes system) one can use Evclid to ‘straighten out’ such oblique objects. This is illustrated in the example below. After `gpr` has straightened the isosceles triangle `gtr` is used to move the origin to the carbon atom. It is then trivial to enter the second carbon atom by means of Cartesian coordinates.

```
Evclid, a geometric calculator and editor by H. Essen            version:1992
Type:command      (h = help,  hh = basic help)                          ?xyz?
Evc>> ige
Enter: ge i j ( k ( l))    1 line                        (h = help)    *ige*
ige1> 1.086  2 1
  1         0.000000000      0.000000000      0.000000000
  2         1.086000000      0.000000000      0.000000000
Enter: ge i j ( k ( l))    2 lines                      (h = help)     *ige*
ige1> 1.086  3 1
ige2> 117.3  3 1 2
  3        -0.498093416      0.965038315      0.000000000
Enter: ge i j ( k ( l))    3 lines                      (h = help)     *ige*
ige1> gpr
Enter: nf ( nl ( ndf ( ndl)))   to transform        (h = help)          *gpr*
gpr>> 1
 Translation done;  distance:  0.376672
 Degeneracy:  0
Transformation done.
Type:command   (no of atoms:3)                                          ?xyz?
Evc>> dco
Enter:(ct)   (nf (nl))                        (h=help, q=skip)          *dco*
dco>>
              n       x              y              z
              1      0.00000000     -0.37667155      0.00000000
              2      0.92744958      0.18833577      0.00000000
              3     -0.92744958      0.18833577      0.00000000
Type:command   (no of atoms:3)                                          ?xyz?
Evc>> gtr
Enter: vector (# dist)     (% nf nl)    (+ = copy)        (h = help)     *gtr*
gtr>> 1
Translation done; distance:   0.3767   vector=(    0.0000    0.3767    0.0000)
Type:command   (no of atoms:3)                                          ?xyz?
Evc>>
Enter:(ct)   (nf (nl))                        (h=help, q=skip)          *dco*
dco>>
              n       x              y              z
              1      0.00000000      0.00000000      0.00000000
              2      0.92744958      0.56500732      0.00000000
              3     -0.92744958      0.56500732      0.00000000
Type:command   (no of atoms:3)                                          ?xyz?
Evc>> ico
Enter:(n) x  y  z      or: ct      (h = help)                           *ico*
ico>> 0 -1.337 0
   4         0.000000000     -1.337000000      0.000000000
```

```
Enter:(n) x  y  z               (h = help)                           x*ico*
ico>> emo
Enter: nnew  n   or: nfnew  nf  nl    to move atom or range  (h = help)  *emo*
emo>> 2 4
    Input atom number:  4        with contents:  1 atom
 is now renumbered to:  2
As a result the range:  2 -  3  has become
renumbered and is now:  3 -  4  It contains:  2 atoms
Type:command   (no of atoms:4)                                          ?xyz?
Evc>> ian; 1 2 6;
Enter: Z   or: n1 Z1 ( n2 Z2..( n5 Z5))   or: nf nl Z     (h = help)     *ian*
Range   1 -  2   given  Z= 6  i.e. are  C -atoms.
Enter: Z   or: n1 Z1 ( n2 Z2..( n5 Z5))   or: nf nl Z     (h = help)     *ian*
Type:command   (no of atoms:4)  (input charge:2)                        ?xyz?
Evc>> gpr
Enter: nf ( nl ( ndf ( ndl)))   to transform        (h = help)           *gpr*
gpr>> 1 4 1 2
 Translation done;  distance:  0.668500
 Degeneracy:  2
Transformation done.
Type:command   (no of atoms:4)  (input charge:2)                        ?xyz?
Evc>> d;
Enter:(ct)   (nf (nl))                         (h=help, q=skip)         *dco*
Symbol   Z           n       x               y               z
 C1      6           1      0.00000000      0.00000000     -0.66850000
 C2      6           2      0.00000000      0.00000000      0.66850000
 H1      1           3      0.92744958      0.00000000     -1.23350732
 H2      1           4     -0.92744958      0.00000000     -1.23350732
Type:command   (no of atoms:4)  (input charge:2)                        ?xyz?
Evc>> gre
Enter: plane   or: x  y  z  (% nf nl)  (/ = no copy)  (h = help)        *gre*
gre>> z %3 4
Reflected atoms in range:  5 -  6
Type:command   (no of atoms:6)  (input charge:2)                        ?xyz?
Evc>> dzm;
Enter: ( nf ( nl ( i j k)))   (a) (o) (#)    (h=help, q=skip)           *dzm*
Symbol  Z      n,   i,   r(n,i)   ,  j, tht(n,i,j),  k, phi(n,i,j,k)
 C1     6      1
 C2     6      2    1   1.3370000
 H1     1      3    1   1.0860000    2  121.35000
 H2     1      4    1   1.0860000    2  121.35000    3   180.00000
 H3     1      5    2   1.0860000    1  121.35000    3     0.00000
 H4     1      6    2   1.0860000    1  121.35000    3   180.00000
Type:command   (no of atoms:6)  (input charge:2)                        ?xyz?
```

```
Evc>> dco;
Enter:(ct)   (nf (nl))                        (h=help, q=skip)          *dco*
Symbol   Z           n       x               y               z
 C1      6           1      0.00000000      0.00000000     -0.66850000
 C2      6           2      0.00000000      0.00000000      0.66850000
 H1      1           3      0.92744958      0.00000000     -1.23350732
 H2      1           4     -0.92744958      0.00000000     -1.23350732
 H3      1           5      0.92744958      0.00000000      1.23350732
 H4      1           6     -0.92744958      0.00000000      1.23350732
```

In this example we have also used the edit-command `emo` to move the second carbon-atom so that it becomes atom number 2 (instead of 4) and the geometric option `gre` (reflect) to create the two last hydrogens.

The following is an example of input of C_2H_6. Instead of using `gpr` to straighten the trigonal pyramid the second carbon atom is entered by means of a specification that uses two geometric coordinates of type 5 of table 3.1 (when such exotic specifications are used the result should be checked). Note that the z-matrix depends on the charge of the atoms; before atomic numbers have been entered, by means of `ian`, the z-matrix does not reflect the symmetry of the molecule.

```
Type:command      (h = help,  hh = basic help)                          ?xyz?
Evc>> ige
Enter: ge i j ( k ( l))    1 line                        (h = help)     *ige*
ige1> 1.093  2 1
  1         0.000000000      0.000000000      0.000000000
  2         1.093000000      0.000000000      0.000000000
Enter: ge i j ( k ( l))    2 lines                       (h = help)     *ige*
ige1> 1.093  3 1; 109.1  3 1 2
  3        -0.357649164      1.032829161      0.000000000
Enter: ge i j ( k ( l))    3 lines                       (h = help)     *ige*
ige1> 1.093  4 1; 109.1  4 1 2; 109.1  4 1 3
  4        -0.357649164     -0.502332312      0.902440205
Enter: ge i j ( k ( l))    3 lines                       (h = help)     *ige*
ige1> 1.534  5 1; 120.  2 5 1 3; 120.  4 5 1 2
  5        -0.520629899     -0.731245162     -1.243937708
Enter: ge i j ( k ( l))    3 lines                       (h = help)     *ige*
ige1> emo; 2 5
Enter: nnew  n   or: nfnew  nf  nl   to move atom or range  (h = help)  *emo*
    Input atom number:  5        with contents:  1 atom
 is now renumbered to:  2
As a result the range:  2 -  4  has become
renumbered and is now:  3 -  5  It contains:  3 atoms
Type:command   (no of atoms:5)                                          ?xyz?
Evc>> gpr; 1 5 1 2
```

```
Enter: nf ( nl ( ndf ( ndl)))   to transform        (h = help)           *gpr*
 Translation done;  distance:  0.767000
 Degeneracy:  2
Transformation done.
Type:command   (no of atoms:5)                                           ?xyz?
Evc>> gin; 3 5
Enter: nf  (nl)    (/ = no copy)  to invert    (h = help)                *gin*
  3 inverted atoms in range:  6 -   8
Type:command   (no of atoms:8)                                           ?xyz?
Evc>> ;
Enter:(ct)   (nf (nl))                        (h=help, q=skip)           *dco*
              n       x               y               z
              1       0.00000000      0.00000000      0.76700000
              2       0.00000000      0.00000000     -0.76700000
              3       1.02812436      0.00000000      1.13795729
              4      -0.51406218      0.89038181      1.13795729
              5      -0.51406218     -0.89038181      1.13795729
              6      -1.02812436      0.00000000     -1.13795729
              7       0.51406218     -0.89038181     -1.13795729
              8       0.51406218      0.89038181     -1.13795729
Type:command   (no of atoms:8)                                           ?xyz?
Evc>> ;c
Enter:(ct)   (nf (nl))                        (h=help, q=skip)           *dco*
              n       rh             phi              z
              1       0.00000000      0.00000000      0.76700000
              2       0.00000000      0.00000000     -0.76700000
              3       1.02812436      0.00000000      1.13795729
              4       1.02812436    120.00000000      1.13795729
              5       1.02812436   -120.00000000      1.13795729
              6       1.02812436    180.00000000     -1.13795729
              7       1.02812436    -60.00000000     -1.13795729
              8       1.02812436     60.00000000     -1.13795729
Type:command   (no of atoms:8)                                           ?xyz?
Evc>> dzm;
Enter: ( nf ( nl ( i j k)))   (a) (o) (#)    (h=help, q=skip)            *dzm*
              n,   i,   r(n,i)   ,  j, tht(n,i,j),  k, phi(n,i,j,k)
              1
              2    1   1.5340000
              3    1   1.0930000    2  109.83994
              4    1   1.0930000    3  109.10000    2  -120.44902
              5    1   1.0930000    3  109.10000    4  -119.10196
              6    2   1.0930000    1  109.83994    3   180.00000
              7    2   1.0930000    6  109.10000    1   120.44902
              8    2   1.0930000    6  109.10000    7   119.10196
```

```
Type:command   (no of atoms:8)                                        ?xyz?
Evc>> ian; 1 2 6;
Enter: Z   or: n1 Z1 ( n2 Z2..( n5 Z5))   or: nf nl Z     (h = help)     *ian*
Range   1 -  2   given  Z= 6  i.e. are  C -atoms.
Enter: Z   or: n1 Z1 ( n2 Z2..( n5 Z5))   or: nf nl Z     (h = help)     *ian*
Type:command   (no of atoms:8)  (input charge:2)                      ?xyz?
Evc>> dzm;
Enter: ( nf ( nl ( i j k)))   (a) (o) (#)    (h=help, q=skip)            *dzm*
Symbol  Z      n,   i,   r(n,i)    ,  j, tht(n,i,j),  k, phi(n,i,j,k)
 C1     6      1
 C2     6      2    1   1.5340000
 H1     1      3    1   1.0930000    2  109.83994
 H2     1      4    1   1.0930000    2  109.83994    3   120.00000
 H3     1      5    1   1.0930000    2  109.83994    3  -120.00000
 H4     1      6    2   1.0930000    1  109.83994    3   180.00000
 H5     1      7    2   1.0930000    1  109.83994    3    60.00000
 H6     1      8    2   1.0930000    1  109.83994    3   -60.00000
Type:command   (no of atoms:8)  (input charge:2)                      ?xyz?
Evc>> dge; <1.6
Enter:(-i) (m (n))   (< max-dist.)   (% nf nl)     (h=help, q=skip)      *dge*
   i    j          r(i,j)         charges   symbols
   1    2          1.534000000      6- 6    C1-C2
   1    3          1.093000000      6- 1    C1-H1
   1    4          1.093000000      6- 1    C1-H2
   1    5          1.093000000      6- 1    C1-H3
   2    6          1.093000000      6- 1    C2-H4
   2    7          1.093000000      6- 1    C2-H5
   2    8          1.093000000      6- 1    C2-H6
Type:command   (no of atoms:8)  (input charge:2)                      ?xyz?
Evc>> dge; 1 <1.6
Enter:(-i) (m (n))   (< max-dist.)   (% nf nl)     (h=help, q=skip)      *dge*
   i    m    j        tht(i,m,j)      charges    symbols
   2    1    3        109.839938      6- 6- 1    C2-C1-H1
   2    1    4        109.839938      6- 6- 1    C2-C1-H2
   2    1    5        109.839938      6- 6- 1    C2-C1-H3
   3    1    4        109.100000      1- 6- 1    H1-C1-H2
   3    1    5        109.100000      1- 6- 1    H1-C1-H3
   4    1    5        109.100000      1- 6- 1    H2-C1-H3
Type:command   (no of atoms:8)  (input charge:2)                      ?xyz?
Evc>> dge; 1 2 <1.6
Enter:(-i) (m (n))   (< max-dist.)   (% nf nl)     (h=help, q=skip)      *dge*
   i    m    n    j    phi(i,m,n,j)     charges        symbols
   3    1    2    4    -120.000000     1- 6- 6- 1    H1-C1-C2-H2
   3    1    2    5     120.000000     1- 6- 6- 1    H1-C1-C2-H3
   3    1    2    6     180.000000     1- 6- 6- 1    H1-C1-C2-H4
   3    1    2    7      60.000000     1- 6- 6- 1    H1-C1-C2-H5
   3    1    2    8     -60.000000     1- 6- 6- 1    H1-C1-C2-H6
   4    1    2    5    -120.000000     1- 6- 6- 1    H2-C1-C2-H3
   4    1    2    6     -60.000000     1- 6- 6- 1    H2-C1-C2-H4
   4    1    2    7    -180.000000     1- 6- 6- 1    H2-C1-C2-H5
   4    1    2    8      60.000000     1- 6- 6- 1    H2-C1-C2-H6
   5    1    2    6      60.000000     1- 6- 6- 1    H3-C1-C2-H4
   5    1    2    7     -60.000000     1- 6- 6- 1    H3-C1-C2-H5
   5    1    2    8     180.000000     1- 6- 6- 1    H3-C1-C2-H6
   6    1    2    7    -120.000000     1- 6- 6- 1    H4-C1-C2-H5
   6    1    2    8     120.000000     1- 6- 6- 1    H4-C1-C2-H6
   7    1    2    8    -120.000000     1- 6- 6- 1    H5-C1-C2-H6
```

Here is finally an example of how one can use a command file. Assume that you wish to input one or more tetrahedra somewhere in a molecule using `ige`. It may then be advantageous to have a file `tetrah.ang` that contains the single line:

```
$0= 109.47122063
```

At the beginning of the EVCLID-session one then reads this file and after that one can type `$0` instead of the full numerical expression for the tetrahedral angle.

```
Type:command      (h = help,  hh = basic help)                        ?xyz?
Evc>> << tetrah.ang
$0 assigned: 109.47122063
Type:command      (h = help,  hh = basic help)                        ?xyz?
Evc>> ige
Enter: ge i j ( k ( l))    1 line                      (h = help)     *ige*
ige1> 1.0  2 1
  1        0.000000000      0.000000000      0.000000000
  2        1.000000000      0.000000000      0.000000000
Enter: ge i j ( k ( l))    2 lines                     (h = help)     *ige*
ige1> $0  3 1 2; 1.0  3 1
  3       -0.333333333      0.942809042      0.000000000
```

This concludes our examples of some of the ways in which you can use EVCLID. There are, of course, many more.

Chapter 4

Platonic and Truncated Polyhedra

4.1 Regular Polygons and Platonic Polyhedra

To input a regular polygon you just have to give the command `ipo` and enter the desired number of vertices. If you wish you can also give the length of the radius (distance to midpoint) or the length of an edge (between two vertices). The radius is assumed if this number is given as negative, otherwise the edge is assumed.

To get atoms at the vertices of a regular (platonic) polyhedron you just type the first letter of its name (`c` for cube etc.). Then the edge length or radius length can be given as above.

4.2 Truncated Polyhedra

More or less regular polyhedra figure prominently in the world of molecular geometries. The five regular (Platonic) polyhedra all exist as cage-structure molecules. The tetrahedron, the cube, and the dodecahedron in the form of the hydrocarbons tetrahedrane, cubane, and dodecahedrane. Boron hydrides can take octahedral and icosahedral shapes. The most recent addition to this collection is the famous C_{60} graphite ball, footballene or Buckminsterfullerene (Kroto, 1992), which has the shape of the truncated icosahedron, one of the thirteen Archimedean semi-regular solids (Williams, 1979, Holden, 1991).

Considering these facts EVCLID naturally allows the input of atoms by specifying these to be at the vertices (corners) of such solids. The five Platonic, regular, polyhedra essentially have the coordinates of their vertices stored, according to formulae that can be found in Coxeter (1973). Apart from these five EVCLID can also generate those that arise from these by truncation. Truncation here means that the corners of the polyhedra are cut off (or 'sandpapered' down). Two different ways of doing this exist. For both methods each old corner gives rise to a polygon with the same number of vertices as the number of edges (polygons) that previously met at the corner. If the truncation is so deep that these new corner polygons meet at the middle of what was an edge, each old edge gives rise to a new vertex, and the old polygonal faces become new polygonal faces with the same number of corners as before. One might call this 'deep' truncation. Only two new polyhedra arise in this manner. These are the, so called, quasi-regular polyhedra, the cuboctahedron and the icosidodecahedron.

A less severe truncation that leaves part of the old edges intact, leads to new polyhedra with the old polygonal faces replaced by new ones that have twice the number of corners. This gives rise to five new polyhedra which simply are called the truncated tetrahedron, cube, octahedron, dodecahedron, and icosahedron respectively. They are called semi-regular. Quasi-regular is a special case of semi-regular, characterized by the fact that all dihedral angles between polygonal faces are the same in them. EVCLID generates these, seven, truncated polyhedra by running through the edges of the regular ones, inserting vertices at the appropriate places. See table 4.1 for a list; there are six more Archimedean, semi-regular polyhedra (Williams, 1979, Holden, 1991) that are not available in EVCLID.

Type	Name of Polyhedron	Faces, with no of edges:						Ed-	Ver-
		3	4	5	6	8	10	ges	tices
Platonic									
Regular	Tetrahedron	4						6	4
	Cube		6					12	8
	Octahedron	8						12	6
	Dodecahedron			12				30	20
	Icosahedron	20						30	12
Archimedean									
Quasi-regular	Cuboctahedron	8	6					24	12
	Icosidodecahedron	20		12				60	30
Semi-regular	Truncated tetrahedron	4			4			18	12
	Truncated cube	8				6		36	24
	Truncated octahedron		6		8			36	24
	Truncated dodecahedron	20					12	90	60
	Truncated icosahedron			12	20			90	60

Table 4.1: In EVCLID atoms can be specified as the vertices of the above polyhedra.

Bibliography

[1] M. J. Frisch, M. Head-Gordon, G. W. Trucks, J. B. Foresman, H. B. Schlegel, K. Raghavachari, M. A. Robb, J. S. Binkley, C. Gonzalez, D. J. Defrees, D. J. Fox, R. A Whiteside, R. Seeger, C. F. Melius, J. Baker, R. L. Martin, L. R. Kahn, J. J. P. Stewart, S. Topiol, and J. A. Pople, *Gaussian 90*, (Gaussian, Inc., Pittsburgh PA), 1990.

[2] M. Frisch, *Gaussian 90 User's Guide and Programmer's Reference*, (Gaussian, Inc., Pittsburgh PA), 1990.

[3] H. Essén, *On the General Transformation from Molecular Geometric Parameters to Cartesian Coordinates*, Journal of Computational Chemistry, Vol. **4**, p.136-141, 1983.

[4] H. Essén, *EUCLID: An Interactive System for Calculations Relating to Molecular Geometries*, QCPE-Program 452, Quantum Chemistry Program Exchange, QCPE Bulletin **3**, No 1, p.13, 1983.

[5] H. Essén and M. Svensson, *Calculation of Coordinates from Molecular Geometric Parameters and the Concept of a Geometric Calculator*, Computers & Chemistry, Vol. **20**, p.389-395, 1996.

[6] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vettering, *Numerical Recipes, The Art of Scientific Computing*, (Cambridge University Press, Cambridge, England), 1986.

[7] D. Pedoe, *Geometry, A Comprehensive Course*, (Dover Publications, Inc., New York), 1988.

[8] W. Klyne and V. Prelog, *Description of Steric Relationships Across Single Bonds*, Experientia, Vol.**XVI**. p.521, 1960.

[9] O. Kennard et al., editors, *Molecular Structure and Dimensions, Volume A1*, (A. Oosthoek, Utrecht, Nederlands), 1972.

[10] G. Dahlquist and Å. Björck, *Numerical Methods*, (Prentice-Hall, Inc., Englewood Cliffs, N.J.) 1974.

[11] H. Eyring, *The Resultant Electric Moment of Complex Molecules*, Physical Review **39**, p.746, 1932.

[12] J. Baker and W. J. Hehre, *Geometry Optimization in Cartesian Coordinates: The End of the Z-Matrix?*, Journal of Computational Chemistry, Vol.**12**, p.606, 1991.

[13] D. W. Noid, B. G. Sumpter, B. Wunderlich and G. A. Pfeffer, *Molecular Dynamics Simulation of Polymers: Methods for Optimal Fortran Programming*, Journal of Computational Chemistry, Vol.**11**, p.236, 1990.

[14] R. J. Gillespie and I. Hargittai, *The VSEPR Model of Molecular Geometry*, (Allyn and Bacon, Boston, Massachusetts) 1991.

[15] W. A. Sokalski, *Programmable Calculators: Evaluation of Molecular Cartesian Coordinates by a Programmable Calculator*, Computers & Chemistry, Vol. **4**, p.165-177, 1980.

[16] H. Bradford Thomson, *Calculation of Cartesian Coordinates and their Derivatives from Internal Molecular Coordinates*, Journal of Chemical Physics, Vol. **47**, p.3407-3410, 1967.

[17] R. L. Hildebrandt, *Cartesian Coordinates of Molecular Models*, Journal of Chemical Physics, Vol. **51**, p.1654-1659, 1969.

[18] Kroto H. W. (1992), C_{60}*: Buckminsterfullerene, the celestial sphere that fell to Earth*, Angewandte Chemie (english ed.), **31,** 111.

[19] Coxeter H. S. M. (1973) *Regular Polytopes*, § 3.7. Dover Publications, Inc., New York.

[20] Holden, A. (1991) *Shapes, Space and Symmetry*, p. 46. Dover Publications, Inc., New York, N.Y.

[21] Williams R. (1979) *The Geometrical Foundation of Natural Structure*, Dover Publications, Inc., New York, N. Y.