# BOND ENERGIES AND BOND ANGLES

Emma Cooper

F226918

Table of Contents

[BOND ENERGIES AND BOND ANGLES 1](#_Toc166760941)

[PROJECT PROPOSAL 2](#_Toc166760942)

[Project Overview 2](#_Toc166760943)

[Project Aims 2](#_Toc166760944)

[PLANNING 4](#_Toc166760945)

[Initial Planning 4](#_Toc166760946)

[Rendered Graphic of Molecule 5](#_Toc166760947)

[Creating Orca Input file 8](#_Toc166760948)

[Orca Output file Handling 9](#_Toc166760949)

[Create Graphical User Interface (GUI) – Input and Output 11](#_Toc166760950)

[Graphing the results 15](#_Toc166760951)

[CODE IMPLEMENTATION 16](#_Toc166760952)

[TESTING 16](#_Toc166760953)

[GUI *using Tkinter* 17](#_Toc166760954)

[Molecule Rendering *using Turtle* and Creating Orca Input File 19](#_Toc166760955)

[Searching Output File 20](#_Toc166760956)

[Graphing Results *using MatPlotLib* 21](#_Toc166760957)

[Overall Testing 21](#_Toc166760958)

[IMPROVEMENTS 22](#_Toc166760959)

[CONCLUSION 22](#_Toc166760960)

## PROJECT PROPOSAL

### Project Overview

For this project, I will create a program that takes inputs from a user through a clear, user-friendly GUI, completes tailored calculations to the user inputs, and produces a mathematical and graphic output for the user. The program will focus on the relationship between the bond angle and the overall molecule energy, by taking the inputs from the user of the molecule type and angle of the specified bond and producing a numerical value for the single point energy as well as a simple rendered graphic of the molecule, and plot the energy value on the graphic against any previous values calculated..

### Project Aims

The following project aims are the goals, processes, and outputs that should be achieved by the final program. These will function as guides for the testing of each developed version and the final version of the code.

|  |  |  |
| --- | --- | --- |
| Code Functions | Overview | Aims |
| Inputs   * Molecule type * Bond angle | The user should be able to give both a molecule type (from the options given) and a bond angle between three atoms within the molecule. | * Selection panel for molecule type (discrete options) * Entry box for the bond angle * Enter button that should only collect inputs when all fields are filled, otherwise prompt with an error message |
| GUI | The GUI should be aesthetically appealing, clear and simple to understand. Should also have separate input and output menus within the GUI | * Labels and prompts should be included to provide clear and easy understanding for a user. * Aspects of the GUI should not merge due to colour and should have no overlayed/intercepting elements. |
| Calculations   * Final bond energy | Using inputs provided by the user through the GUI, the program should create and run an orca file, then search the output file for the final energy. Then return this in sensible units and format to the user. | * Should create correctly format an Orca input file. * Should run Orca input file. * Should search the Orca output file for the final bond energy. * Should be given in \_\_ units. * Should be given to sensible significant figures dependent on the users’ input – one less significant figure. |
| Outputs:   * Simple rendered molecule * Final bond energy * Histrotic Graph of Results | Should give a simple rendered model of a molecule with users’ specified bond angle, onto the output menu. Should also give the user the final bond energy of the molecule to the correct format and units.  Display a plot of the single-plot energy and agree with any previously calculated values. | * Should produce a simple render molecule where which element is different colours, and the users’ bond angle is shown. * Should output the final bond energy to the output menu in the correct format and with the specified units. * Should provide the user with a plot with current and previous single-point energies calculated. |

Here is an example of what the GUI (graphic user interface) could look like to achieve the goals stated.

A computer screen shot of a test

Description automatically generated

Entry box for the bond angle of the molecule.

The colour of the button could changes colour to indicate selection.

Figure 1. Example of User Input GUI

Simple rendered molecule of H20 molecule:

* Oxygen is shown in *RED.*
* Hydrogen is shown in *WHITE.*

A screenshot of a computer screen

Description automatically generated

Botton to allow users to return to the input menu and recalculate.

Figure 2. Example of User Output GUI

## PLANNING

### Initial Planning

Before any coding begins, plotting out the steps taken to produce the required goals and output is necessary, to do this a flowchart is used to show the basic processes and any key points within the process, which can then be broken down further into proposed pseudocode for each step, and exampled in greater detail the method and techniques used to.

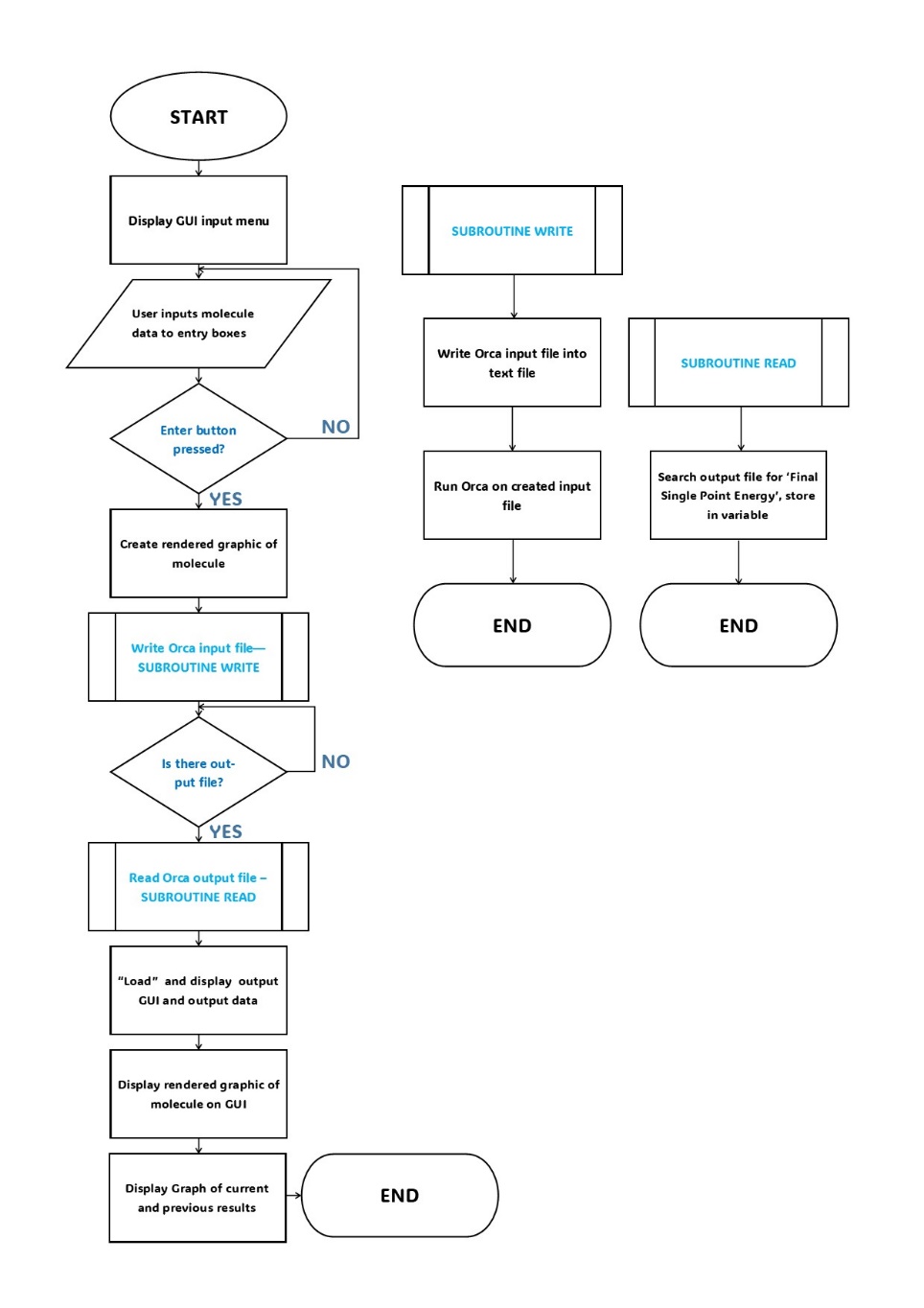
The following flowchart shows the basic steps that will be followed, as well as the separate subroutines for dealing with Orca files. In addition, the setting of elements and building of GUIs should be completed before ever called, therefore can be called, and displayed when required - display called as per the following flowchart.

Figure 3. A flowchart to show the basic and key steps for the final code to roughly follow.

This flowchart shows the structure that the final code could follow for the key processes. Now that the overall process is planned, the key points can be planned in further detail in pseudocode. These key points and functions include:

* Rendering the simple graphic of the molecule.
* Generating the input file for Orca with calculated coordinates and in the correct format.
* Searching the Orca output file once the calculation has finished.
* Graph and record the current and previous results.
* Creating GUI

One point that sticks out in the flowchart is the use of separate input and output GUIs to allow the user to see the previous result while entering the new inputs.

### Rendered Graphic of Molecule

The first key process in the flowchart in Figure 3 is rendering the molecule selected in the user input GUI. This should calculate and store the coordinates relative to each of the atoms to be used later when creating the Orca input file, but also be able to render the molecule once a function is called.

Below is the proposed pseudocode for this process:

import draw *//’draw’ represents a graphic library*

import maths *//’maths’ represents a mathematical library*

class atom: *//Creates a class where objects have attributes of all information that may be required for any elements that may be in molecule*

function init(self, element, symbol, size, colour):

self.element = element

self.symbol = symbol

self.size = size

self.colour = colour

*//Molecule type, mol\_type, updated when button registers being pressed by the user*

*//Bond angle, bond\_angle, is retrieved from entry box and updated when enter button is pressed by the user*

*//Bond length, bond\_length, is set went the subroutine is called.*

class molecule: *//Creates class where objects have attributes of molecules geometry and composition*

function init(self, composition, co\_ords):

self.composition = composition

self.co\_ords = co\_ords

self.symbol\_comp = []

for atom in composition:

atom = atom.split('\_')

self.symbol\_comp.append(atom[0])

function calculate\_co\_ords(self, bond\_angle, bond\_length): *//Calculate the position of atoms to the origin for the 3 atom molecule*

self.co\_ords[1] = [0,0]

i = bond\_length\*maths.cos(bond\_angle/2)

j = bond\_length\*maths.sin(bond\_angle/2)

self.co\_ords[0] = [-i,j]

self.co\_ords[2] = [i,j]

return

function type\_2\_render(self, index): *//Return rendering information of atom specified in composition by index – colour and size*

if self.symbol\_comp[index] == hydro.symbol:

return hydro.size, hydro.colour

elif self.symbol\_comp[index] == oxy.symbol:

return oxy.size, oxy.colour

return

function render(self, bond\_angle): //Draws/renders molecule

draw.hidepen()

for index in range(0,length(self.co\_ords),2): *//Draws lines representing bonds between the molecule.*

One way this could be further improved is by giving the users control of the composition of the molecule completely, by allowing them to select each atom individually. This aspect could be tested and investigated further, once the full program is complete.

draw.origin()

draw.pendown()

draw.goto(self.co\_ords[index][0],self.co\_ords[index][1])

draw.penup()

draw.origin() *//Draws central atom - central atom always at the origin*

radius, colour = self.type\_2\_render(1)

draw.dot(radius, colour)

draw.goto(self.co\_ords[0][0],self.co\_ords[0][1]) #Draws left atom

radius, colour = self.type\_2\_render(0)

draw.dot(radius, colour)

draw.goto(self.co\_ords[2][0],self.co\_ords[2][1]) #Draws right atom

radius, colour = self.type\_2\_render(2)

draw.dot(radius, colour)

return

*//Creates different elemental atoms*

hydro = atom("Hydrogen", "H", 24, 'black')

oxy = atom("Oxygen", "O", 36, 'red')

if mol\_type == "H2O":

work\_mol = molecule(['H\_1', 'O\_1', 'H\_2'], [-1,-1,-1])

elif mol\_type == "H3":

work\_mol = molecule(['H\_1', 'H\_2', 'H\_3'], [-1,-1,-1])

work\_mol.render(bond\_angle)

### Creating Orca Input file

To create an orca file, there is some key information that is required:

* The algorithm that should be used for the calculations.
* The type and number of atoms in the molecule structure.
* The location of each atom in the relative space.

The algorithm that is used is not dependent on the user’s inputs and therefore can be easily defined in a variable. However, the number, type, and location of the atoms are dependent on the user’s input. This is solved by modeling the molecule first as the required information can be found as attributes of an object under the ‘molecules’ class.

To build this input file, a new text (.txt) file should be created, and then the input information for Orca should be written into the new file in the correct format. Then the complete built input file should be run through Orca for an output file by using the command prompt.

To be able to accomplish this, a library should be imported to allow the code to interact with the operating system and command prompt, so that Orcca may be run. The name of the output file should be stored in the code so the file can be searched through for the final single-point energy.

Below is the proposed code for this process:

import os *//os represents a library that allows interactions with operation system*

import date\_time *//date\_time represents a library that retrieves current dare and time information from the operating system*

… *//assuming all previous pseudocode is impliented*

function file\_name\_gen(mol\_type):

name = (mol\_type + ‘\_’ + str(date\_time.today()) + ‘\_’ + str(date\_time.now())) *//Creates file name including the molecule type, current date and time*

return name

### Orca Output file Handling

function build\_orca\_file (mol\_type, mol\_config, mol\_co\_ords): *//Function generates the Orca input file*

orca\_function = ‘!B3LYP def2-SVP SP’ *//variable for the orca function for calculation*

work\_file\_name = file\_name\_gen(mol\_type)

os.mkdir(file\_name) *//Makes new directory with the name of the input file*

os.chdir(‘/’+ str(file\_name)) *//Move to newly made directory*

file\_script = open(work\_file\_name+’.txt,’w’) *//Creates a new text file that can be written in by the program*

file\_script.write(“#Input generated by program\n” + str(orca\_function) + “\n\n\* xyz 0 1 \n”) *//Creates the default beginning of every orca file.*

for atom\_index in range(length(mol\_config)):

file\_script.write(" " + str(mol\_config[atom\_index]) +" "+ str(mol\_co\_ords[atom\_index][0]) +" "+ “0.0” +" "+ str(mol\_co\_ords[atom\_index][1]) + ”\n") *//Writes the element and co-ordinates into text file*

file\_script.write(“\*\n”)

file\_script.close() *//Closes text file*

return work\_file\_name

function run\_orca(file\_name):

orca\_command\_line = (‘orca’, str(file\_name), ‘>’, str(‘output\_’+file\_name))

os.system(orca\_command\_line) *//Runs orca command line in CMD*

return str(‘output\_’+file\_name) *//Function return the output file name*

input\_file\_name = build\_orca\_file(mol\_type, work\_mol.symbol\_comp, work\_mol.co\_ords)

output\_file\_name = run\_orca(input\_file\_name)

Once an Orca file has been created and run through the command line, Orca saves the output file in the same directory as the original Orca file is run from. This Orca file should be checked for in the directory, once Orca has finished all the required calculations, and opened with the program (as read-only as no changes will be made to this text file).

The file should be searched through to find the final single-point energy of the molecule, and this value stored within an appropriately named variable, so it may be displayed on the output GUI. This can be completed by looping through each line in the text file and the key phrase ‘FINAL SINGLE POINT ENERGY’ checked for.

There should also be a way to deal with an Orca error file, where there is no final single-point energy calculated. Through the GUI the user should be notified and asked if the user inputs should be rerun, to do this there should be a variable to indicate whether the final single-point energy has been found or not.

Below is the proposed code for this process:

function search\_output\_file(file\_name): *//Functions that opens, reads, and searches the Orca output file*

file = open(file\_name, ‘r’) *//Opens output file in read-only format and stores in variable file*

found = False *//Variable indicating whether the final single-point energy has been found or not*

for line in file.readlines(): *//.readlines() breaks the file into an array with each line separated, then for loop move through the array of individual lines*

line = line.split() *//Splits line into individual words and creates an array*

if length(line) == 5:

if line[0] == ‘FINAL’ and line[1] == ‘SINGLE’ and line[2] == ‘POINT’ and line[3] == ‘ENERGY’: *//Checks contents of array correct line (as line format is known, this can then be easily be checked for)*

final\_energy = float(line[4]) *//Stores energy in variable as float instead of string*

found = True //Updates variable to show final single-point energy is found

break

file.close()

*//To handle an Orca error file, when the variable found is never updated from False*

if (not found):

print(“Error: Final Single-Point Energy not found”) *//Prints error message into console*

return found, final\_energy *//Returns the found status and the final single-point energy of the molecule*

found\_status, final\_energy = search\_output\_file(output\_file\_name) *//Runs function to search output file*

### Create Graphical User Interface (GUI) – Input and Output

For this program, a GUI should be used to collect the user’s inputs and show the user the output of the orca file and the simply rendered molecule in a visually pleasant way. Both input and output GUIs should be in separate windows to allow the user to see both the input values and calculated output and rendering of the molecule at the same time. The output window should either open or update whenever any button in the input window is pressed.

A GUI uses the same elements with varying parameters, because of this a class can be used to simplify the creation, building, and packing of each element. The class should collect the parameters consistent across all the elements used (labels, buttons, and entry boxes), if any additional information is needed for an element a daughter class should be created, which inherits the parent attributes class. The class should also contain the functions to build and pack the elements onto the GUI windows.

Additionally, a graphical interface library could be imported and used to create both GUIs.

Below is the proposed code to create the input GUI:

import graphic *//’graphic’ represents a GUI library that should be used*

global mol\_type

global bond\_angle

mol\_type = “H2O” *//Sets a default molecule type*

bond\_angle = 180 *//Sets a default bond angle*

*//Button functions*

function change\_mol\_type(mew\_mol): *//Function changes the variable mol\_type to the option selected by the user*

mol\_type = new\_mol

return

function collect\_angle(entry\_box): *//Function collects the value from the entry box in GUI*

angle = entry\_box.get()

try: *//Checks to see if the value in ‘angle’ is a numerical value*

angle = float(angle)

bond\_angle = angle

except:

print(“Error: No number entered”)

return

class elements\_: *//General class with all attributes and functions for widgets in a GUI*

function \_\_init\_\_(self, window, geo, colour, co\_ords):

self.height = geo[1]

self.width = geo[0]

self.colour = colour

self.window = window

self.co\_ords = co\_ords

self.font = “Trebuchet MS”

self.fontsize = 14

function build\_entrybox(self): *//Function that creates and entry box*

\_entry = Entry(self.window, width = self.width, font = (self.font, str(self.fontsize)))

\_entry.pack()

\_entry.place(y = self.co\_ords[1], x = self.co\_ords[0]) *//Places entry box in specific place on the input GUI*

return \_entry

class label\_(elements\_) *//Subclass for label widgets – attributes and functions specific to labels*

function \_\_init\_\_(self, window, geo, colour, co\_ords, text):

elements\_.\_\_init\_\_(self, window, geo, colour, co\_ords)

self.text = text

function build(self): *//Builds label*

\_label = Label(self.window, text = self.text, width = self.width, height = self.height, bg = self.colour, font = (self.font, str(self.fontsize)))

\_label.pack()

\_label.place(y = self.co\_ords[1], x = self.co\_ords[0])

class button\_(elements\_): *//Subclass for buttons widgets – attributes and functions specific to buttons*

function \_\_init\_\_(self, window, geo, colour, co\_ords, text, function):

elements\_.\_\_init\_\_(self, window, geo, colour, co\_ords)

self.text = text

self.function = function

function build(self): *//Builds buttons*

\_button = Button(self.window, text = self.text, width = self.width, height = self.height, bg = self.colour, font = (self.font, str(self.fontsize)), command = self.function)

\_button.pack()

\_button.place((y = self.co\_ords[1], x = self.co\_ords[0]))

*//Creates input window*

Input\_window = Graphic()

Input\_window.geometry(“600x500”)

Input\_winow.title(“Input Window”)

*//Creates all widgets for the Input GUI Window*

mol\_type\_label = label\_(input\_window, [25,2], ‘light grey’, [10,75], “Pick a molecule to model:”)

angle\_label = label\_(input\_window, [25,2], ‘light grey’, [10,75], “Enter the angle between atoms:”)

H2O\_button = button\_( input\_window, [25,2], ‘light grey’, [10,75], “Water”, lambda: change\_mol\_type(“H2O”))

H3\_button = button\_( input\_window, [25,2], ‘light grey’, [10,75], “Trihydrogen”, lambda: change\_mol\_type(“H3”))

Enter\_button = button\_( input\_window, [25,2], ‘light grey’, [10,75], “Water”, lambda: collect\_inputs(\_entry))

Angle\_entry = elements\_(input\_window, [30,5], ‘light grey’, [10,215])

*//Build all the widgets*

mol\_tye\_label.build()

angle\_label.build()

\_entry = angle\_entry.build\_entrybox()

H2O\_button.build()

H3\_button.build()

enter\_button.build()

*//Opens window and loops to check for user interacts*

input\_window.mainloop()

Below is the proposed code to create the output GUI:

*//Under ‘elements\_’ class*

function build\_canvas(self): *//Creates canvas from general attributes*

\_canvas = Canvas(output\_window, bg = self.colour, width = self.width, height = self.height)

\_canvas.pack()

\_canvas.place(y = self.co\_ords[1], x = self.co\_ords[0])

…

output\_window = graphic()

output\_window.geometry(“600x500”)

output\_window.title(“Output Window”)

*//Creates all the widgets for the Output GUI*

mol\_model = elements\_(output\_window, [550,300], ‘light grey’, [25,10])

final\_point\_label = label\_(output\_window, [25,2], ‘light grey’, [10, 350], “Final Single Point Energy: “)

energy \_label = label\_(output\_window, [25,2], ‘light grey’, [250,350], f”{final\_energy:.2f}”)

mol\_model.build()

final\_point\_label.build()

energy\_label.build()

output\_window.mainloop()

### Graphing the results

This graph should plot the current single-point energy calculated as well as any previous energies calculated since the program began. This is so that results are comparable so the impact of the change in molecule and angle can be easily observed by the user.

This graph should be easily readable and formatted smartly, and therefore should include the following:

* Graph and axis titles (with units included).
* The x-axis should be between 0 and 360.
* Different colors for each different molecule option.

To be able to create a graph a library could be imported.

Below is the proposed pseudocode for this section:

import plot *//’Plot’ represents a library that could be used to produce a graph*

import maths *//’maths’ represents a mathematics library*

results = [] *//An array that contians all newly generated results will be added*

colours = [] *//An array that contains colours that each result should be plotted in, at the corresponding index*

function cont\_plot(result, colour, filename): *//Function that can be called to plot the previous and current results*

results.appened(result) *//Adds result to array*

colours.append(colour) *//Adds corresponding colour to array*

formatted = maths.transpose(results) *//Changes format of result so all x and y coordinates are grouped together*

for i in range(length(results)): *//Loop plots all result in array in the correct colour*

plot.scatter(formatter[0][i], formatted[1][i], colour = colours[i])

plot.xlimit(0,360) *//Formats graph correctly*

plot.title(“Historic Single-Point Energies)

plot.xlabel(“Angle (Degree)”)

plot.ylabel(“Final Single-Point Energy”)

plot.grid()

plot.show() *//Shows graph created*

return plot *//Returns plot to where function was called*

## CODE IMPLEMENTATION

## TESTING

When testing code it can be useful to test individual sections separately, this allows easier and precise tracking of any issues or shortfalls within the code.

The sections that will be tested individually are as follows:

* GUI *using Tkinter*
* Molecule Rendering *using Turtle*
* Creating an Orca input file
* Search contents of an Orca output file
* Graphing results *using MatPlotLib*

To test this code, extra code may be added to simulate the running of the full code.

Once individual sections are tested, the full code is tested to see how these sections interact with each other.

### GUI *using Tkinter*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Function/  Section | Line Number/s | Target/Aim | Action | Expected Outcome | Actual Outcome | Improvements Required |
| INPUT GUI | 131-134 | Create a window for the input GUI. | >Run | >Window open to desktop | >Matches. | None |
| INPUT GUI | 137, 140 | Label showing the type of molecule that is currently selected by the user, and stored in variable ‘mol\_type’. | >Run | >Label: “MOLECULE TYPE: H2O”. | >Matches | None |
| INPUT GUI,  CHANGE MOLECULE TYPE | 143-146, 157-160, 23-32 | Button for each molecule option, with a label indicating the molecule. | >Run  >Pressed one of the buttons | >4 buttons (in a row) with the molecule name in the center.  >Molecule label is updated. | >Matches  >Matches + Message in the shell. | Could use a dropdown menu instead, so inputs are more flexible to the user. |
| INPUT GUI | 138, 141, 34-48 | Label showing the bond angle that has been entered by the user and currently stored in variable ‘bond \_angle’. | >Run | >Label: “BOND ANGLE: 120”. | >Matches | None |
| INPUT GUI | 151, 155 | Entry box for the user to type the desired bond angle in. | >Run  >Type in number | >Blank entry box of decent size, that number may be typed in. | >Matches  >Matches | None |
| INPUT GUI, COLLECT BOND ANGLE | 147, 161 | Button with ‘ENTER’ label, that instructs collection of value in entry box. Updates the angle label, if a number is not entered the angle label and variable ‘bond\_anlge’ are not updated. | >Run  >Enter an integer, and press the button.  >Enter a decimal, and press the button.  >Enter a word or letter, and press the button. | >Button with ‘ENTER’ in the center appears.  >Bond angle label updated.  >Bond angle label updated.  >Bond angle label NOT updated. | >Matches  >Matches  >Matches  >Matches + Message in the shell. | None |
| INPUT GUI | 148, 162 | Button with ‘GENERATE’ label, that being following processes: generation, calculations, reading of Orca file, and rendering of model, with a graph of results. | >Run  >Press button | >Button with ‘GENERATE’ in the center appears.  >When pressed, message “Running…” into the shell. |  | None |
| OUTPUT GUI | 112-115 | Create a separate window for output GUI | >Run | >Window open to desktop. | >Matches | None |
| OUTPUT GUI | 119,124 | A canvas is created in the top center of the window that the rendered model can be drawn into. | >Run | >Canvas in the top center of the window. | >Matches | None |
| OUTPUT GUI | 120, 126 | Label showing the final single-point energy calculated and stored in variable ‘final energy’. | >Run | Label: ‘FINAL SINGLE-PINT ENERGY: 0.00’. | >Matches | None |
| OUTPUT GUI | 118,125 | A canvas is created at the bottom center of the window that the graph will be ‘printed’ into. | >Run | >Canvas in the bottom center of the window. | >Matches | None |
| INPUT GUI | 5-6 | Default values for molecule type and bond angle. | >Run | >Default values appear in labels in the window. | >Matches  >Defaults are:  ‘H2O’ and ‘120’. | None |
| INPUT AND OUTPUT GUI | N/A | Can exit out of the window. | >Press the exit button (top right-hand corner of the window) | >Close windows | >Matches | None |
| INPUT AND OUTPUT GUI | N/A | Clear design, without merging of widgets due to overlap or colour. | >Run | >Window clear to understand. No widgets merge due to overlap or colour | >Matches | None |

### Molecule Rendering *using Turtle* and Creating Orca Input File

To test this code, an assumed molecule has been created to work with. Within the larger code, this molecule would be created based on the user’s input from the GUIs. A default bond angle is also assumed as 120° when calculating the coordinate positions of the atoms.

The assumed molecule used is water, this has two differing types of atoms, with different colours and sizes under the element class.

The molecule will be drawn in a turtle window, where within the complete code it will be drawn within a canvas in the GUI output window.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Function/  Section | Line Number/s | Target/Aim | Action | Expected Outcome | Actual Outcome | Improvements Required |
| DEFINE OBJECT IN ATOM CLASS | 103-106 | Define an object for each atom type (used in molecules) created under the ‘atom’ class. | >Run + print all attributes of the ‘hydro’ object under the ‘atom’ class. | >Printed into the shell: “**Hydrogen H 30 grey**” | >Matches | None |
| CALCULATE COORDINATES OF MOLECULE ATOMS | 36-47 | Use the given molecule information and calculate where the atoms are. The central atom should be at the origin. | >Run + print the calculated coordinates. | >Print into shell: “**[[-86.602…,49.999….], [0,0],** **[86.602…,49.999…]]**” | >Matches | None |
| RENDER | 88-90 | An oxygen atom is drawn at the center of the Turtle window. The dot should be red and larger than “hydrogen” atoms. | >Run | >Red dot at the center of the Turtle window, dot larger than others. | >Matches | None |
| RENDER | 84-86, 92-94 | Two hydrogen atoms are drawn at the same distance and angle respective to the carbon. The dot should be grey and smaller than the “oxygen” atom. | >Run | >Two gray dots at the same distance and angle from the white dot. Dots are smaller than the red dot. | >Matches | None |
| RENDER | 78-82 | Lines beneath the dots and connects the all “atoms”, like bonds. | >Run | >Black lines between atoms, that do not visibly overlap on top of the the dots. | >Matches | None |
| FILE NAME GENERATION | 109-112 | Should create a file name with the molecule type, current date, and time. | >Run + Print out filename generated | >Prints in shell: “H2O\_2024-05-15\_14-08-42.txt” | > Matches | None |
| BUILD ORCA FILE | 119-121 | Should create a new text file with the filename generated. | >Run + Check documents folder | >New text file called “H2O\_2024-05-15\_14-08-42.txt” | >Matches | None |
| BUILD ORCA FILE | 122-130 | The text file should contain all the information for the orca calculation: the orca function, the charge of the molecule, and the atom symbol with corresponding coordinates. | >Run + Open text file | >Text file should contain: “! B3LYP def2-SVP SP *(as orca function)*, \* xyz 0 1, and atom symbol with corresponding coordinates. | >Matches | Add the ability for the user to select the Orca command line that will be used to calculate the single-point energy. |
| BUILD ORCA FILE | 120 | Creates a directory (with the same name and orca input file), for all orca files to be saved in. | >Run + Check documents folder | >Directory created with name: “H2O\_2024-05-15\_14-08-42.txt” | >Matches | None |

### Searching Output File

To be able to accurately test this function, an example Orca output file has been added to the code directory and is used within the function as well as a blank text file has been added.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Function/  Section | Line Number/s | Target/Aim | Action | Expected Outcome | Actual Outcome | Improvements Required |
| SEARCH FILE | 2-20 | Use the function to search through a completed Orca output text file for final single-point energy. | >Run COMPLETE Orca file + Print if found or not and the single-point energy. | >Print into the shell: “**True -76.321273083168**” | >Matches | None |
| SEARCH FILE | 2-20 | Use the function to search through an empty text file for the final single-point energy. | >Run EMPTY text file + Print if found or not and the single-point energy. | >Print into the shell: “**False 0.00**” | >Mathes + Error message: “**Error: Final Single-Point Energy not found**” | None |

### Graphing Results *using MatPlotLib*

To test this function a loop will be implemented to run the function multiple times in a row, each time a randomly generated result will be passed through the function and a set colour (red) will be used.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Function/  Section | Line Number/s | Target/Aim | Action | Expected Outcome | Actual Outcome | Improvements Required |
| PLOT | 8-23 | When the function is called 20 times with 20 different results, 20 graphs are produced with an incrementally increasing number of plotted points. | >Run | >20 different graphs with an increasing number of plotted points – in red. | >Matches | None |
| PLOT | 8-23 | When the function is called 5 times with 5 different results and 5 different colours, 5 graphs are produced with an incrementally increasing number of plotted points, all in different colours. | >Run | >5 different graphs with an increasing number of plotted points all in different colours | >Matches | None |
| PLOT | 15-19 | All graphs produced should be clearly labeled. | >Run | All Grpahs including title, axis titles, grid, and the x-axis is limited to 0-360. | >Matches | None |

### Overall Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Function/  Section | Line Number/s | Target/Aim | Action | Expected Outcome | Actual Outcome | Improvements Required |
| ASSIGNING MOLECULE | 11-25 | Uses ‘mol\_type’ and creates an object in the molecule class. | >Run + Press the ‘GENERATE’ button. | >Molecule created shown by a message printed into the shell. | >Matches | None |
| RUNNING | 26 | Calls function to calculate coordinates under the ‘molecule’ class for molecule created as an object under the class. | >Run + Press the ‘GENERATE’ button. | >Molecules in the correct position in the simplified rendered molecule. | >Matches | None |
| RUNNING | 27 | Calls function to build input Orca file, storing the returned filename in a variable. | >Run + Press the ‘GENERATE’ button. | >New directory and text file created in documents. | >Matches | None |
| RUNNING | 28 | Calls function to run the Orca file, storing the output file name returned in a variable. | >Run + Press the ‘GENERATE’ button. | >All output files in documents, including the output file with single-point energy in. | >Matches | None |
| RUNNING | 31 | Calls function to search the output Orca file for the final single-point energy, stores returned value in a variable called ‘final\_energy’. | >Run + Press the ‘GENERATE’ button. | >Updated final single-point energy on output window and prints value to the shell. | >Matches | None |
| RUNNING | 33 | >Calls to plot the current and previous results on a graph. Stored returned save plot’s file name in a variable. | >Run + Press the ‘GENERATE’ button. | >Plot .png file saved in documents + plot shown in the output window of GUI. | >Matches | None |
| OUTPUT | 35 | >Calls the function to re-configure the output window with all the results. | >Run + Press the ‘GENERATE’ button. | >All results appear on the output window GUI. | >Matches | None |
| RUNNING | 36 | >Calls the function to render molecule into the canvas in the output window. | >Run + Press the ‘GENERATE’ button. | >Simle Rendered molecule directly drawn into the canvas in the output window of GUI. | >Matches | None |

## IMPROVEMENTS

While all functions and sections of code work well, and final code works fluently together, there were a two improvements that were raised as int the testing process that would overall increase the funactionality of the code. This improvements are:

* Instead of set molecules, the user selected each of the 3 moelcules from 3 different drop down menus. This allows more flexiablity and increase fnctionality of the program.
* In addition allow the colour the final single-poit energy is plotted in on the graph displaying the current and previous results.
* Another imporvemnt is allowing the user to pick Orca function used to calculate the single-point energy to the molecule, this can be do through a dropdown menu on the input window of the GUI.

Once these improvements are made to the code, these sections will be retest to specficall test point related to these functions.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Function/  Section | Line Number/s | Target/Aim | Action | Expected Outcome | Actual Outcome | Improvements Required |
| INPUT  BUILD ORCA FILE | 169. 180  121, 133, 135 | Dropdown box with Orca function options. Option selected is ued in Orca calulction, appears in input file. | >Run + Press the ‘GENERATE’ button. | >Dropdown box with Orca function options + Orca function selected in text file. | >Matches | None |
| INPUT | 158- 161 | Labels indictaing title of program and explain input widgets | >Run | >Label: “CALCULATE FINAL SINGLE-POINT ENERGY”  >Label: ”ORCA FUNCTION”  >Label: “BUILD MOLECULE: “ | >Matches | None |
| INPUT | 165- 167, 183- 185 | Dropdwon menus with atom options (element symbols) that the user can pick from. Working molecule and ‘mol\_type’ build from these user inputs. | >Run + Press the ‘GENERATE’ button. | >3 dropdown menus, in a row, contain atom options for molecule. Molecule type at being of directory and input file name matches options chosen. | >Matches | None |
| INPUT | 168, 186 | Dropdwon menu with colour optios of plot point next drawn onto graph (calculation). | >Run + Press the ‘GENERATE’ button. | >Colour selected in used in plot. | >Matches | None |
| OUTPUT | 161, 171, 15 | Label shoing the final single-point energy shows result in 2 different units. | >Run + Press the ‘GENERATE’ button. | >Label shows Eh and eV values for the single-point energy. | >Matches | None |

## CONCLUSION