Supporting Information

Figure SI-1. Bond lengths (Å) for lowest energy spin state for each possible hydrogen position.

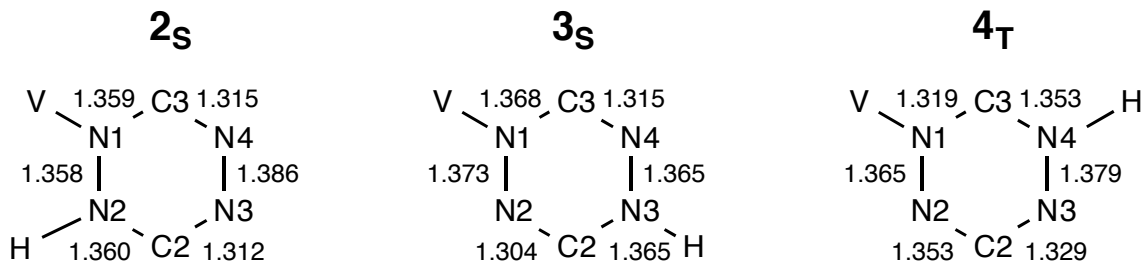


Figure SI-2. Optimized structure of singlet $[(btzp\text{-}H)VCl_2O]^0$ with corresponding bond lengths that can be viewed Table SI-1.

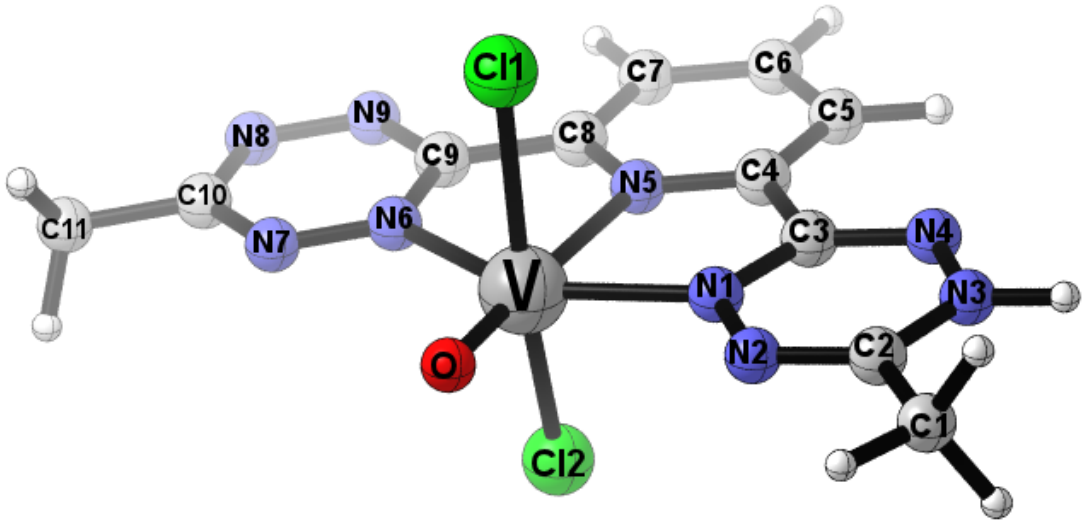


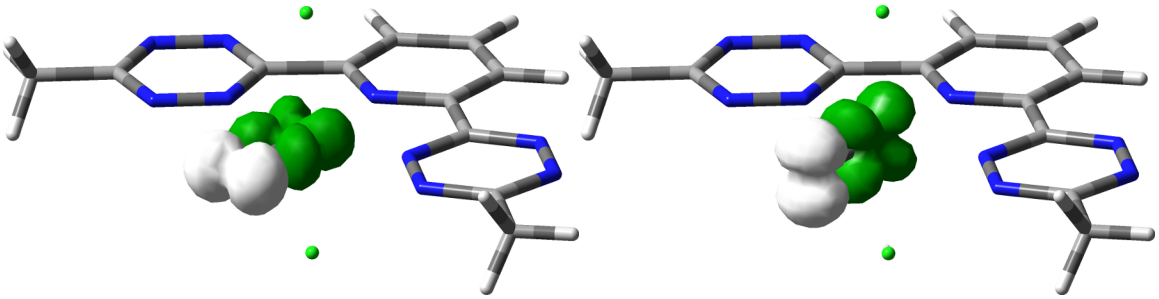
Table SI-1. Bond lengths (Å) for Optimized Structure and Experimental Results

	3_T	3_S	X-ray
C1–C2	1.495	1.496	1.4844(15)
C2–N2	1.305	1.304	1.3094(12)
C2–N3	1.367	1.365	1.3494(13)
C3–C4	1.480	1.479	1.4693(13)
C3–N4	1.322	1.315	1.3004(12)
C4-N5	1.330	1.328	1.3278(12)
C4–C5	1.398	1.398	1.3943(12)

C5–C6	1.397	1.397	1.3894(15)
C6–C7	1.397	1.398	1.3937(16)
C7–C8	1.397	1.396	1.3892(13)
C8–C9	1.473	1.474	1.4706(14)
C9–N9	1.332	1.331	1.3386(13)
C10–C11	1.495	1.495	1.4853(16)
N1–C3	1.355	1.368	1.3822(11)
N1–N2	1.370	1.373	1.3784(11)
N3–N4	1.367	1.365	1.3548(13)
N5–C8	1.331	1.331	1.3350(12)
N6–C9	1.361	1.361	1.3455(12)
N6–N7	1.308	1.305	1.3166(12)
N7–C10	1.349	1.350	1.3416(13)
N8–C10	1.345	1.344	1.3493(15)
N8–N9	1.322	1.324	1.3180(15)
V1–Cl1	2.375	2.384	2.3823(3)
V1–Cl2	2.375	2.384	2.3672(3)
V1–N1	2.156	2.070	1.9743(8)
V1–N5	2.276	2.259	2.1869(8)
V1–N6	2.159	2.185	2.1946(8)
V1–O1	1.579	1.578	1.6000(7)
N3-H*	1.011	1.010	--

CalcID	**Spin State**	**V1–N1** Bond Length (Å)
$\mathbf{2}_S$	Singlet	2.048
$\mathbf{2}_T$	Triplet	2.159
$\mathbf{3}_S$	Singlet	2.070
$\mathbf{3}_T$	Triplet	2.156
$\mathbf{4}_S$	Singlet	2.139
$\mathbf{4}_T$	Triplet	2.155
X-ray	--	1.9743(8)

Figure SI-3. Orbital density difference isosurface plots (0.01 au) between the α and β orbitals for both V–O π bonds. Green corresponds to the excess in the α orbital, white corresponds to the excess in the β orbital.



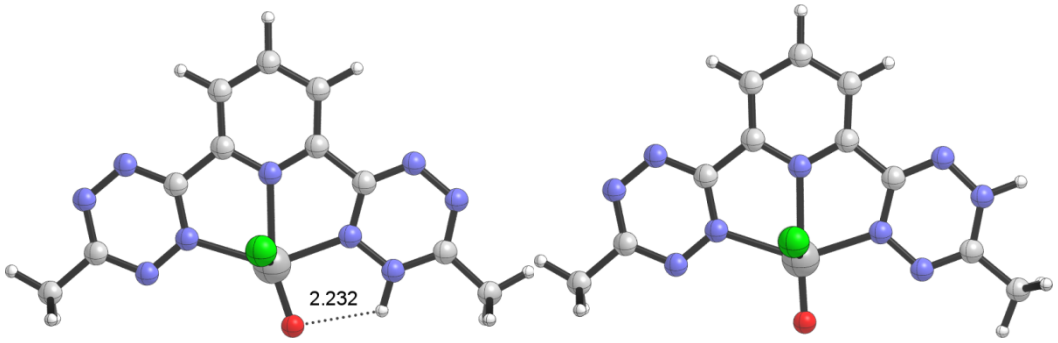
Spin polarization can arise from different amount of electron transfer in the α and β orbital subspaces. If we think about the polarization in an extreme limit as electron transfer from the oxo to the vanadium center, we can think about whether the oxygen would want to transfer an α or β electron to vanadium. If a β electron is transferred it will pair up with the α electron already in the V(IV) center. For a first-row transition metal, this is not energetically preferred. Doubly occupying one of these small d orbitals carries a large electrostatic penalty. Because we have electron rich π-donor ligands like Cl and O, we expect the other d-orbitals to be energetically accessible. Therefore it is most likely that an α electron will be transferred from oxygen to vanadium. In the less extreme limit of polarization, this results in orbitals with V–O bonding character that have more vanadium character in the α orbital and more oxygen character in the β orbital.

Appendices

Appendix 1. Explanation of spin polarization on oxo ligand.

We first sought to address the spin polarization of these species to look at the Mulliken spin density values for the main atoms involved. We focus our analysis on species 001 but the polarization is similar in both 001 and 005a. The Mulliken analysis shows that four atoms have appreciable (> 0.01) spin: V with 1.10, O with –0.18, and both Cl with 0.04. This quantitative analysis matches well to the qualitative spin density plot in Figure 2. Because only a fraction of an electron is being transferred between V and O; the spin polarization due to this transfer is subtle. We returned to the corresponding orbital analysis to look for orbitals that had overlaps not equal to one. Interestingly, the two V–oxo bonds fit this description with overlaps of 0.996 each. This is still a large amount of overlap but the spin polarization is slight (as the Mulliken analysis shows). To better visualize the difference between the α and β orbitals, we calculated the orbital density for each orbital by squaring it and then taking the difference between these two orbital densities. This will accentuate the slight differences between the α and β orbitals. These plots are shown in Figure SI-3 where green corresponds to an excess of α and white corresponds to an excess of β.

Appendix 2. Top-down view of the optimized structures showing H-bonding in $\mathbf{2_S}$ (left) vs. $\mathbf{3_S}$ (right).



This interaction biases the energy landscape for most favored nitrogen for the added hydrogen. The crystallographic work shows *inter*molecular N3H···Cl hydrogen bonding, which is *not* modeled in our single molecule calculation.

Appendix 3: thermo.py

User Tutorial:

1. Create the .log file by running a calculation with GaussView
2. Open Linux or a command line editor
3. Enter this line of code
    a. python thermo.py XXXXXXX
        i. where XXXXXXX is the name of the .log file
        ii. multiple files can be listed with only blank spaces between each file name
            ex. python thermo.py XXXXXXX YYYYYYY ZZZZZZZ
    b. ./thermo.py XXXXXXX
        i. This command works all the same
4. Two additional features for the user to interpret
    a. If any .log files are seen printed as *XXXXXXX, the calculation has been run with imaginary frequencies and should be reran
    b. If any .log files are seen printed as +YYYYYYY, the calculation has been run with a temperature that is not room temperature (298.15 K)

Future improvements to this program:

- If the user doesn't specify a file, prompt him or her.
- If the user listed a file that does not end in .log, notify the user and run the program for all files that do start in .log

- Ask the user if he or she wants to run all files in a certain directory or folder
- Use more robust expressions like those of the regular expression module to enhance the usability and clarity of the program to replace the lackluster line.startswith() commands.
- Ask the user if he or she wants the energies to be exported a .txt file or an excel spreadsheet
- Add a GUI component that adds aesthetic appeal to the program

Appendix 4: geom.py

User Tutorial

5. Create the .log file by running a calculation with Gaussian09
6. Open Linux or a command line editor
7. Enter this line of code
    a. python geom.py XXXXXXX
        i. where XXXXXXX is the name of the .log file
        ii. multiple files can be listed with only blank spaces between each file name
            ex. python thermo.py XXXXXXX YYYYYYY ZZZZZZZ
    b. ./geom.py XXXXXXX
        i. This command works all the same
8. Two inputs the user must specify
    a. "Pick an optimization between 1 and X"
        i. Last or All are options
        ii. This allows for the user to see his or her preferred optimization
    b. "Atomic Symbol, Atomic Name, or Atomic Number?"
        i. What format do you want each atom to be printed in?

Future improvements to this program:

- If the user doesn't specify a file, prompt him or her.
- If the user listed a file that does not end in .log, notify the user and run the program for all files that do start in .log
- Ask the user if he or she wants to run all files in a certain directory or folder
- Export to an excel spreadsheet
- Export to a new input file that can be ran as a new calculation
    - Line formatting would be critical
- Add debug statements for any user input statements that are not provided with the correct information

Appendix 5: imag.py

User Tutorial:

9. Create the .log file by running a calculation with Gaussian09
10. Open Linux or a command line editor
11. Enter this line of code
    a. python geom.py XXXXXXX
        i. where XXXXXXX is the name of the .log file
        ii. multiple files can be listed with only blank spaces between each file name
            ex. python thermo.py XXXXXXX YYYYYYY ZZZZZZZ
    b. ./geom.py XXXXXXX
        i. This command works all the same
12. Two inputs the user must specify
    a. "Pick an optimization between 1 and X"
        i. This allows for the user to see his or her preferred optimization
    b. "Pick an imaginary frequency displacement between 1 and X"
        i. This allows for the user to decide which imaginary frequency he or she wants to displace along.

Future improvements to this program:

- Export to a new input file that can be ran as a new calculation
    - Line formatting would be critical
- Add debug statements for any user input statements that are not provided with the correct information

Interpretation of Flow Chart:

Read file: Both a geom_grab and geom_print definition call a filename which is inputted on the command line by the user. Example code:

```
def example(filename):

    f = open(filename, 'r')

example(sys.argv[1])
```

Find geom matches: Regular expressions are utilized to compile a pattern which the program parses each line of the input file to find matches.

```
for line in f:

    pattern = re.compile("hello")

    iterator = pattern.finditer(line)
```

Append array: the module numpy which allows for multidimensional arrays is called.

Array = numpy.append(Array, (line[1], line[2]))

User input: two questions are asked from the user. The user must answer which optimization he or she wants (or ask for all of them). Then the user must pick between how he or she wishes for the output to be printed; with atomic symbols, atomic names, or atomic numbers.

User_input = raw_input("What Optimization would you like? ")

Atomic_convert: A program outside of geom.py is called as a module to help assist with the converting of the atomic number which is given in the geometry to the preferred notation desired by the user. This program utilizes many of the previous commands such as opening a file and appending an array. The program finally uses conditional statements to find the place in the array that the input file specifies.

Print Optimizations: many commands are integrated within the definition of geom_print to print the optimizations in a meaningful way. The main aspect of this definition is the calculation of the offset. This offset serves to shift a certain amount within the array in order to print the correct parts of the array.

Why did we use this formula…?

Print "{0:>15} {1:>15}".format("Element", "Coordinate")

This command shifts each string over a certain amount of characters (15, in this case).

Offset = 4*num_atoms*(int(i)-1)+(4*k)

Where num_atoms = j/(num_geoms*4)

j = size of array

num_geoms = amount of matches compiler finds (amount of optimizations that the input file gives)

i = desired optimization inputted by user

k = counter variable that helps find the place in array

Interpretation of Flow Chart:

Read file: Both a geom_grab and geom_print definition call a filename which is inputted on the command line by the user. Example code:

def example(filename):

f = open(filename, ‘r’)

example(sys.argv[1])

Find geom matches: Regular expressions are utilized to compile a pattern which the program parses each line of the input file to find matches.

for line in f:

pattern = re.compile(“hello”)

iterator = pattern.finditer(line)

Append newarray1: the module numpy which allows for multidimensional arrays is called.

Offset = 4*num_atoms*(int(i)-1)+(4*k)

Where num_atoms = j/(num_geoms*4)

j = size of array

num_geoms = amount of matches compiler finds (amount of optimizations that the input file gives)

i = desired optimization inputted by user

k = counter variable that helps find the place in array

This offset calculation accounts for a multidimensional array being stored as a single dimension array. We look to print all of the atoms within a certain optimization.

4*num_atoms*(int(i)-1) accounts for the optimization preferred, as we have four pieces to each optimization (AN, X, Y, and Z).

4*k adjusts for each atom within the desired optimization

newarray1 = numpy.append(newarray1,

Atomic_Convert(n, option), geom_array[offset +1], …)

Find Imag Matches:

The very same method used for “Find Geom Matches” is used.

This time we are looking for the imaginary frequency displacements.

User input: two questions are asked from the user. The user must answer which geometry optimization he or she wants (last is default). Then the user must specify which imaginary frequency displacement he or she wants.

User_input = raw_input("What Optimization would you like? ")

Atomic_convert: A program outside of geom.py is called as a module to help assist with the converting of the atomic number which is given in the geometry to the preferred notation desired by the user. This program utilizes many of the previous commands such as opening a file and appending an array. The program finally uses conditional statements to find the place in the array that the input file specifies. In this program, atomic symbol has been defaulted and atomic name and/or number can be added to the program if necessary.

Append newarray2: A similar method to "Append newarray1" is utilized. However, the inner calculations have differences that are specified below.

AN_offset = (11*(n - 1)) + (11*l*k)

q = imag_array[AN_offset + 1]

offset = (11*(n - 1)) + (3*(m - 1)) + (11*l*k)

newarray2 = numpy.append(newarray2,
(Atomic_Convert.convert(q,option),

imag_array[offset + 2], …)

~see notes for description of variables~

Consolidate newarray 1 + 2

size = len(newarray1)

We calculate the size by taking the length of newarray1 (should be same size as newarray2). We then divide this number by four, as there are four variables (AN, X, Y, Z) that repeat themselves.

while (k<(size/4)):

We loop over this calculation as it prints each respective atom within the desired geometry and imaginary frequency displacement.

a = newarray1[4*k]

We need to calculate the Atomic Symbol only once.

b = float(newarray1[1+4*k]) + alpha * float(newarray2[1+4*k])

Converting each coordinate of the geometry and displacement, then adding them with a user defined alpha (default at 1).

k += 1

Print new geometry: This serves to add labels and print formatting for each of the new coordinates in the geometry.

Print “{0:>15} {1:>15}”.format(“Element”, “Coordinate”)

This command shifts each string over a certain amount of characters (15, in this case).

print "{0:^10} {1:>15} {2:>15} {3:>15}".format(a, b, …)

Appendix 6. Source code for thermo.py

```
#!/usr/bin/python
# May 7th, 2013
# Adam Terwilliger
# Extraction of Energies from Gassian Output File
global temperature
temperature = 0
global Ucorr, U, Ezpe, H, G
Ucorr = 0.0             # We define our Global variables and initialize them to 0.0
U = 0.0                 # in turn, we can return floats
Ezpe = 0.0
H = 0.0
G = 0.0

import sys
import re

def temp_check(line):

        if line.startswith(" Temperature"):
                if "298.150" in line:
                        ""
                else:
                        global temperature
                        temperature=1

def energy_finder(filename):

 f = open(filename, 'r')

 for line in f:

        temp_check(line)
```

```
        if line.startswith(" Thermal correction to Energy"):
                temp = re.split('\=', line)             # Pulls Correction to convert E(SCF) to U
                Ucorr = float(temp[1])
        elif line.startswith(" Sum of electronic and zero-point"):
                temp = re.split('\=', line)
                Ezpe = float(temp[1])                   # Pulls Sum of E(SCF) and ZPE
        elif line.startswith( " Sum of electronic and thermal Energies"):
                temp = re.split('\=', line)             # Pulls Internal Energy(U)
                U = float(temp[1])
        elif line.startswith(" Sum of electronic and thermal Enth"):
                temp = re.split('\=', line)
                H = float(temp[1])                      # Pulls Enthalpy (H)
        elif line.startswith(" Sum of electronic and thermal Free"):
                temp = re.split('\=', line)
                G = float(temp[1])                      # Pulls Gibbs Free Energy (G)

        if line.startswith(" ****** "):
                filename = "*" + filename

E = U - Ucorr                   # Calculation of E(SCF)
ZPE = Ezpe - E                  # Calculation of Zero-Point Energy
TS = H - G                      # Calculation of Entropy using the difference between Free Energy
                                        # and Enthalpy

if (temperature == 1):
        filename = "+" + filename

print '{0:>15} {1:>14.6f} {2:>14.6f} {3:>14.6f} {4:>14.6f} {5:>14.6f} {6:>14.6f}'.format(filename,E, ZPE, U, H, TS, G)
f.close()

if (len(sys.argv) == 1):
        print "Please enter a valid filename."
else:
                print "\n{0:>11} {1:>13} {2:>16} {3:>12} {4:>14} {5:>15} {6:>13}".format("filename", "E","ZPE", "U", "H", "TS",
"G")
                print "   ----------------------------------------------------------------------------------------------------"
                i = 0
                while i < (len(sys.argv)-1):
                        i += 1
                        energy_finder(sys.argv[i])
                print "   ----------------------------------------------------------------------------------------------------"
                print "       *indicates molecule has imaginary frequencies"
                print "         +indicates temperature is not 298.15 K"
```

Appendix 7. Source code for geom.py

```
#!/usr/bin/python
# Adam Terwilliger

import sys, re, numpy, string, Atomic_Convert

# initialize globals
p_check = False
num_geoms = 0
geom_array = []
option = 0

def geom_grab(filename):
        global p_check, geom_array, num_geoms
        f = open(filename, 'r')
        for line in f:

                # compiles a pattern and finds iterations of that pattern with each line
                p1 = re.compile("Standard orientation:")
                iterator1 = p1.finditer(line)
                p2 = re.compile("Rotational constants")
                iterator2 = p2.finditer(line)
                for match in iterator1:
                        p_check = True
```

```
                        num_geoms += 1
                for match in iterator2:
                        p_check = False

                # p_check allows us to keep only desired lines
                if (p_check == True):
                        line = line.split()

                        # we append an array with the desired aspects found from each line
                        if (len(line) == 6) and (line[1] != "Number"):
                                geom_array = numpy.append(geom_array, ([line[1]], [line[3]], [line[4]], [line[5]]))
        f.close()

def geom_print(filename):
        global geom_array, num_geoms, option

        # user input that allows for an optimization to be chosen
        print "\nPick an optimization between 1 and " + str(num_geoms) + "!"
        print " *If you would like all of them, type 'ALL'\n  *If you would like the last one, type 'LAST'"
        input = (raw_input("Here we go: "))

        #converts any input to lowercase to avoid case-sensitive
        i = string.lower(input)

        # user_input looks to pick up what the user desires (sym, name, num)
        user_input = raw_input('\nAtomic Symbol, Atomic Name, or Atomic Number? \n')
    ui = string.lower(user_input)
    if ui in ('atomic symbol', 'symbol', 'as', 'sym'):
        option = 1
    elif ui in ('atomic name', 'name', 'aname'):
        option = 2
    elif ui in ('atomic number', 'anum,' 'number', 'num'):
                 option = 3

        # j is the total size of the geom_array
        j = geom_array.size
        # number of atoms is calculated by taking the total size and dividing by number of matches times number of variables kept(4)
        num_atoms = j/(num_geoms*4)
        #Atomic_Convert.convert(n)
        #conditionals that allow for last or all geometries to be chosen
        if (i == "last"):
                i = num_geoms
        if (i == "all"):
                i = 0
        if (i>0):
                # conditional flag if true print, if false don't
                # print formatting
                print "\n{0:^10} {1:>12} {2:>15} {3:>15}".format("Element", "X", "Y", "Z")
                print "-----------------------------------------------------------------"
                print "Optimization #" + str(i)
                print "----------------"
                k=0
                while (k<num_atoms):
                        offset = 4*num_atoms*(int(i)-1)+(4*k)   # each geom block = 4 times the number of atoms
               n = geom_array[offset+0]
                        print "{0:^10} {1:>15} {2:>15} {3:>15}".format(Atomic_Convert.convert(n,option),
geom_array[offset+1],geom_array[offset+2], geom_array[offset+3])
                        #return Atomic_Convert.convert(n,option), geom_array[offset+1], geom_array[offset+2], geom_array[offset+3]
                        k +=1
        else:
                print "\n{0:^10} {1:>12} {2:>15} {3:>15}".format("Element", "X", "Y", "Z")
                print "-----------------------------------------------------------------"
                i = 1
                while (i<=num_geoms):
                        print ""
                        print "Optimization #" + str(i)
                        print "----------------"
                        k = 0
                        while (k<num_atoms):
                offset = 4*num_atoms*(int(i)-1)+(4*k)   # each geom block = 4 times the number of atoms
```

```
                        n = geom_array[offset+0]
                                                print "{0:^10} {1:>15} {2:>15} {3:>15}".format(Atomic_Convert.convert(n,option),
geom_array[offset+1],geom_array[offset+2], geom_array[offset+3])
                        k +=1
                                i +=1


geom_grab(sys.argv[1])
geom_print(sys.argv[1])

#geom_print(..., True) ---> picks up print statements
```

Appendix 8. Source code for disp.py

```
# Adam Terwilliger
# May 20, 2013
# Program combines geom_fix.py and imag_fix.py
# Adjusts the final geometry of a system with imaginary frequencies
# Displaces along that imaginary frequency to create a new geometry

import sys, re, numpy, math, Atomic_Convert

# initialize globals
p1_check = p2_check = False
num_geoms = num_atoms = num_freq = offset = AN_offset = size = num_imag = 0
new_line = geom_array = imag_array = newarray1 = newarray2 = []
i = counter1 = counter2 = counter3 = 0
n = option = 1
#---------------------------------------------------------------------------
def geom_grab(filename):
        """Creates two patterns. Finds number of geometries. Creates an array
        with only the desired pieces of the geometry (AN, X, Y, Z)"""
    global p1_check, num_geoms, geom_array
    f = open(filename, 'r')
    for line in f:
        p1 = re.compile("Standard orientation:")
        iterator1 = p1.finditer(line)
        p2 = re.compile("Rotational constants")
        iterator2 = p2.finditer(line)
        for match in iterator1:
            p1_check = True
            num_geoms += 1
        for match in iterator2:
            p1_check = False
        if (p1_check == True):
            line = line.split()
            if (len(line) == 6) and (line[1] != "Number"):
                geom_array = numpy.append(geom_array, (line[1], line[3], line[4], line[5]))
    f.close()
#---------------------------------------------------------------------------
def geom_print(filename):
        """Appends a new array with all of the atoms in the final geometry"""
    global geom_array, num_atoms, num_geoms, counter1, newarray1, option
    geom_size = len(geom_array)
    num_atoms = geom_size/(num_geoms*4)
    while (counter1<num_atoms):
        offset = 4*num_atoms*(int(i)-1)+(4*counter1)
        newarray1 = numpy.append(newarray1, (Atomic_Convert.convert((geom_array[offset+0]),option), geom_array[offset+1],
geom_array[offset+2], geom_array[offset+3]))
        counter1 +=1
#---------------------------------------------------------------------------
def imag_fix(filename):
        """Creates four patterns.  Finds number of imaginary frequencies. Compiles an array that keeps
        eleven pieces (AN, Atom Number, (X1, Y1, Z1), (X2, Y2, Z2), (X3, Y3, Z3))"""
        global p2_check, counter2, num_imag, num_freq, new_line, imag_array
    f = open(filename, 'r')
    for line in f:
        p1 = re.compile("normal coordinates:")
        iterator1 = p1.finditer(line)
        p2 = re.compile("Frequencies")
```

```
        iterator2 = p2.finditer(line)
        p3 = re.compile("Thermochemistry")
        iterator3 = p3.finditer(line)
        p4 = re.compile("imaginary frequencies \(nega")
        iterator4 = p4.finditer(line)
                for match in iterator1:
            p2_check = True
        for match in iterator2:
            if (p2_check == True):
                counter2 += 1
        for match in iterator3:
            p2_check = False
        for match in iterator4:
            line = line.split()
            num_imag = float(line[1])/3
                                num_freq = 3 * num_imag
        if (p2_check == True):
            line = line.split()
            if (counter2 <= math.ceil(num_imag)):
                new_line = line
        if (len(new_line) > 10) and (new_line[0] != "Atom"):
            imag_array = numpy.append(imag_array, (new_line[0:]))
         f.close()
#---------------------------------------------------------------------------
def imag_print(filename):
         """Prints the desired imaginary frequency."""
    global n, imag_array, rowreset, AN_offset, newarray2, num_imag
         if (int(3*num_imag) == 1):
        freq_num = 1
         else:
                freq_num = float(raw_input("Here we go: "))
         size = len(imag_array)
    rowreset = math.ceil(num_imag)
         l = size/(11*rowreset)
    k = int(math.ceil(freq_num/3) - 1)
    if ((freq_num % 3) == 0):
        m = 3
    else:
        m = int(freq_num % 3)
    while (n <= l):
        AN_offset = (11*(n - 1)) + (11*l*k)
        imag_offset = (11*(n - 1)) + (3*(m - 1)) + (11*l*k)
        newarray2 = numpy.append(newarray2, (Atomic_Convert.convert((imag_array[AN_offset + 1]),option), imag_array[imag_offset + 2],
imag_array[imag_offset + 3], imag_array[imag_offset + 4]))
        n += 1
#---------------------------------------------------------------------------
def imag():
         """Consolidates the four main definitions. Manipulates the two arrays to form a single array.
         Prints the final displaced geometry (without imaginary frequencies) """
         global num_imag, k, newarray1, newarray2, counter3
         geom_grab(sys.argv[1])
         geom_print(sys.argv[1])
         imag_fix(sys.argv[1])
         if (num_imag == 0):
         print "\nCongratulations, this calculation has no imaginary frequencies!\n"
                quit()
         else:
                if ((3*num_imag) > 1):
                print "\nPlease pick a frequency between 1 and " + str(int(3*num_imag)) + "!\n"
                else:
                print "\nYou have one imaginary frequency!\n"
         imag_print(sys.argv[1])
         alpha = 1
         size = len(newarray2)
         print "\n{0:^10} {1:>8} {2:>15} {3:>15}".format("Element", "X", "Y", "Z")
    print "------------------------------------------------------------"
         while (counter3<(size/4)):
                AN = newarray1[4*counter3]
                X = float(newarray1[1+4*counter3]) + alpha * float(newarray2[1+4*counter3])
                Y = float(newarray1[2+4*counter3]) + alpha * float(newarray2[2+4*counter3])
```

```
                        Z = float(newarray1[3+4*counter3]) + alpha * float(newarray2[3+4*counter3])
                        print "{0:^10} {1:>13.6f} {2:>15.6f} {3:>15.6f}".format(AN, X, Y, Z)
                        counter3 += 1
imag()
#-------------------------------------------------------------------------------
```

Appendix 9. Source code for imag.py

```
# Adam Terwilliger
# 05/15/13
# Program that finds imaginary frequencies and records the displacements
# Will serve to create a new geometry that is without imag freq

p1_check = False
p2_check = False
i = 0
j = 0
freq = 0.0
new_line = []
imag_array = []
offset = 0
k = 0
l = 0
m = 0
n = 1
rowreset = 1
AN_offset = 0
import sys, re, numpy, math, string, Atomic_Convert

def imag_fix(filename):

            global p1_check, p2_check, i, j, new_line, imag_array, freq

      f = open(filename, 'r')

      for line in f:

# compiles patterns for which the frequencies can be extracted
                        p1 = re.compile("normal coordinates:")
            iterator1 = p1.finditer(line)

            p2 = re.compile("Frequencies")
            iterator2 = p2.finditer(line)

                        p3 = re.compile("Thermochemistry")
                        iterator3 = p3.finditer(line)

# this pattern gives the number of imaginary frequencies in each system, i
                        p4 = re.compile("imaginary frequencies \(nega")
                        iterator4 = p4.finditer(line)

            for match in iterator1:
                  p1_check = True
                                    k = 1
                        for match in iterator2:
                                    if (k == 1):
                                                j += 1
                        for match in iterator3:
                                    p1_check = False
                        for match in iterator4:
                                    line = line.split()
                                    i = float(line[1])/3
                        if (p1_check == True):
                                    line = line.split()
                                    if (j <= math.ceil(i)):
                                                new_line = line                  # i is the number of imaginary frequencies
                                                                                 # since there are three frequencies on each line and our frequencies
don't start til the third line,
                                                                                 # we use (j < ((i/3)+2)
                        if (len(new_line) > 10) and (new_line[0] != "Atom"):
```

```
                                imag_array = numpy.append(imag_array, (new_line[0], new_line[1], new_line[2], new_line[3], new_line[4],
new_line[5], new_line[6], new_line[7], new_line[8], new_line[9], new_line[10]))
        f.close()


def imag_print(filename):

        global imag_array, offset, m, n, l, k, rowreset, AN_offset

        if (int(3*i) == 1):
                freq_num = 1
        else:
                freq_num = float(raw_input("Here we go: "))

# user_input looks to pick up what the user desires (sym, name, num)
    user_input = raw_input('\nAtomic Symbol, Atomic Name, or Atomic Number? \n')
    ui = string.lower(user_input)
    if ui in ('atomic symbol', 'symbol', 'as', 'sym'):
        option = 1
    elif ui in ('atomic name', 'name', 'aname'):
        option = 2
    elif ui in ('atomic number', 'anum,' 'number', 'num', '#'):
        option = 3

        print "\n{0:^10} {1:>8} {2:>15} {3:>15}".format("Element", "X", "Y", "Z")
    print "-------------------------------------------------------"

        size = len(imag_array)
        if ((freq_num % 3) == 0):
                m = 3
        else:
                m = int(freq_num % 3)
        l = size/(11*2)
        k = int(math.ceil(freq_num/3) - 1)
        while (n <= l):
                if (freq_num > 1):
                        AN_offset = (11*(n - 1)) + (11*l*k)
                        q = imag_array[AN_offset + 1]
                        offset = (11*(n - 1)) + (3*(m - 1)) + (11*l*k)
                        #print "{0:^10} {1:>10} {2:>15} {3:>15}".format(imag_array[AN_offset + 1], imag_array[offset + 2],
                        imag_array[offset + 3], imag_array[offset + 4])
                        print "{0:^10} {1:>10} {2:>15} {3:>15}".format(Atomic_Convert.convert(q,option), imag_array[offset + 2],
                        imag_array[offset + 3], imag_array[offset + 4])
                        #return imag_array[AN_offset + 1], imag_array[offset + 2], imag_array[offset + 3], imag_array[offset + 4]
                        n += 1
                else:
                        AN_offset = (11*(n - 1)) + (11*l*k)
                        q = imag_array[AN_offset + 1]
                        offset = (11*(n - 1)) + (3*(m - 1)) + (11*l*k)
                        #print "{0:^10} {1:>10} {2:>15} {3:>15}".format(imag_array[AN_offset + 1], imag_array[offset + 2],
                        imag_array[offset + 3], imag_array[offset + 4])
                        print "{0:^10} {1:>10} {2:>15} {3:>15}".format(Atomic_Convert.convert(q,option), imag_array[offset + 2],
                        imag_array[offset + 3], imag_array[offset + 4])
                        #return imag_array[AN_offset + 1], imag_array[offset + 2], imag_array[offset + 3], imag_array[offset + 4]
                        n += 1

imag_fix(sys.argv[1])

if (i == 0):
        print "\nCongratulations, this calculation has no imaginary frequencies!\n"

elif (int(3*i) == 1):
        print "\nYou have one imaginary frequency!"
        imag_print(sys.argv[1])

else:
        print "\nPlease pick a frequency between 1 and " + str(int(3*i)) + "!"
        imag_print(sys.argv[1])
```

Appendix 10. Source code for UVVis.nb

```
SetDirectory["/Users/terwilla/00_Data/old"];
file = "test2.txt";
(*file1 = Input["Please choose .txt file"]*)
data = Flatten[Import[file, "Table"]];
NumOsc = Length[data]/2;
Osc=Table[data[[i]],{i,NumOsc}];
Energy = Table[data[[j+NumOsc]], {j,NumOsc}];
sigma = 1;

f[i_] = (Osc[[i]]/(3.483*(10^-5)*Sqrt[Pi]*sigma))*Exp[-((x-Energy[[i]])/sigma)^2];
g = Sum[(Osc[[i]]/(3.483*(10^-5)*Sqrt[Pi]*sigma))*Exp[-((x-Energy[[i]])/sigma)^2],{i,NumOsc}];

MaxOsc = 1/(3.483*(10^-5)*Sqrt[Pi]*sigma);

(*Sort[Osc]

Table[If[Osc[[i]] ¹ "0.", Osc[[i]]], {i,NumOsc}]*)

(*UserOsc = Input["Please specify the smallest Oscillator Strength you would like."]*)

(*NewOsc=If[ Osc[[i]] ¹ 0******,Table[data[[i]],{i,NumOsc}]


(*-h[j_] = x+ Energy[j]

Show[Table[Plot[h[j],{x, 0, Max[Energy]+2}, PlotRange®Full],{j,1,NumOsc}]]*)

(*Plot[h, {x,0, Max[Energy]+2},PlotRange ® Full]*)*)


xmin = 0;
xmax = Max[Energy]*1.1;
ymin = 0;
ymax = NMaxValue[g,{x,0,Max[Energy]}]*1.1;
(*InputField[Dynamic[xmin], FieldHint®"Enter X Min", FieldHintStyle®{Blue}]*)
(*Row[Table[InputField[x,FieldSize®5,BaselinePosition®p],{p,{Center,Center,Center}}],"xxxx"]
(rid[Table[InputField[{xmin,xmax,ymin,ymax},ImageSize®{w,h}],{h,{20,40,70}},{w,{30,50,70}}]])*)
(*Grid[Table[InputField[Dynamic[xmin, ImageSize®{30,30}]],InputField[ Dynamic[xmax]], InputField[Dynamic[ymin]],
InputField[Dynamic[ymax]]]]*)
Print["    ","Minimum","    ","Maximum"]
  Print["X","  ",InputField[Dynamic[xmin],ImageSize®{75,20}], InputField[Dynamic[xmax],ImageSize®{75,20}]]
Print["Y", "  ",InputField[Dynamic[ymin],ImageSize®{75,20}],InputField[Dynamic[ymax],ImageSize®{75,20}]]
(*Working Dynamic*)
(*InputField[Dynamic[xmin]]
InputField[Dynamic[xmax]]
InputField[Dynamic[ymin]]
InputField[Dynamic[ymax]]*)

(*Dynamic[Show[Table[Plot[{f[i],g},{x,xmin,xmax},PlotRange®{ymin,ymax}],{i,1,NumOsc}]]]*)

(*Working Total*)
(*Plot[{f[4],f[8],f[16],f[17],f[18],f[19],g}, {x,0,Max[Energy]+2},PlotRange ® Full]*)

(*BEST
Working Functions + Total*)
(*Show[Table[Plot[{f[i],g},{x,2,10},PlotRange®{0,10000}],{i,1,NumOsc}]]*)

(*Show[Table[Plot[{f[i],g},{x,2,10},PlotRange®{0,10000},Epilog®Line[{{Energy[[i]],0},{Energy[[i]],
NMaxValue[f[i],{x,0,Max[Energy]}]}}]],{i,1,NumOsc}]]
NMaxValue[f[4],{x,0,Max[Energy]}]*)

(*MaxOsc1 = NMaxValue[f[i],{x,0,Max[Energy]}]*)

Dynamic[Show[Table[Plot[{f[i]},{x,xmin,xmax},PlotRange®{ymin,ymax}],{i,1,NumOsc}],Table[ParametricPlot[{Energy[[i]],t},{t,(1*10^-
9),Osc[[i]]*MaxOsc},PlotStyle®RGBColor[0,1,0.5]],{i,1,NumOsc}],Plot[g,{x,xmin,xmax},PlotRange-
>{ymin,ymax},PlotStyle®RGBColor[1,0.5,0]]]]
```