Tutor/s

Dra. Anna Maria Costa Arnau
Departament de Química Inorgànica i Orgànica

Dr. Jaume Vilarrasa i Llorens
Departament de Química Inorgànica i Orgànica

Grau de Química

Treball Final de Grau

Theoretical study of the relative stability of pyrrolidine enamines and their nitro derivatives in *α*, *β*, *γ* and *δ* positions. Estudio teórico de la estabilidad relativa de enaminas de la pirrolidina y de sus derivados nitrados en *α*, *β*, *γ* y *δ*.

Pol Sanz Berman

July 2020

Universitat de Barcelona

B:KC Barcelona Knowledge Campus
Campus d'Excel·lència Internacional

“A computer once beat me at chess, but it was no match for me at kick boxing.”

Emo Phillips

Me gustaría dedicar este apartado a las personas que me han prestado apoyo durante esta etapa de mi vida. En primer lugar, a mis amigos David y Fran, por su consejo y guía. A Laura, por su calor y soporte incondicional, en especial durante estos últimos meses. A mi familia, por confiar en mí, hasta cuando yo no lo hacía. Y finalmente a mis tutores Anna M y Jaume, por su enorme paciencia conmigo, el gran trabajo que realizan, y por hacerme sentir afortunado de que mi primer contacto con el mundo académico sea tan agradable. A todos os deseo lo mejor.

REPORT

CONTENTS

1. SUMMARY 3

2. RESUMEN 5

3. INTRODUCTION

3.1. Computational chemistry and model chemistries 7

3.2 Nitro-Michael reactions 12

4. OBJECTIVES 14

5. EXPERIMENTAL SECTION 16

6. RESULTS AND DISCUSSION

6.1 Initial Geometries 21

6.2 Calculation of ΔE, ΔG and ΔH for the equilibria studied 22

6.3 Results and discussion 28

7.CONCLUSIONS 30

8. REFERENCES AND NOTES 31

9. ACRONYMS 33

APPENDICES 35

Appendix 1: Additional in vacuo results 37

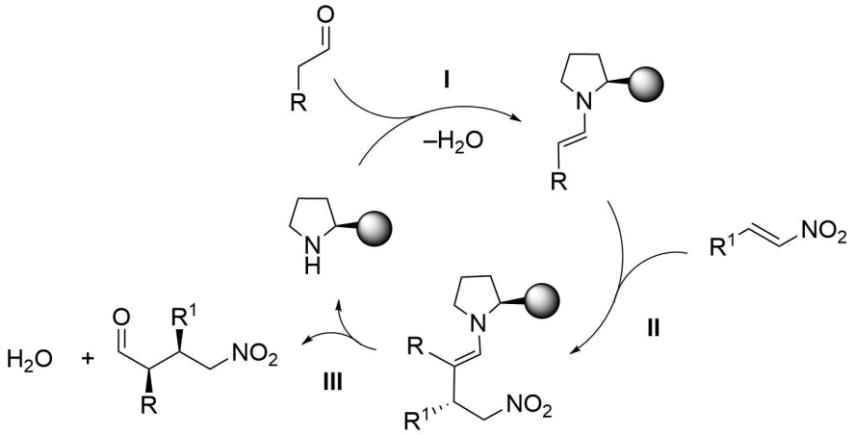
Appendix 2: Additional in water results 38

Appendix 3: Raw code from the scripts used 39

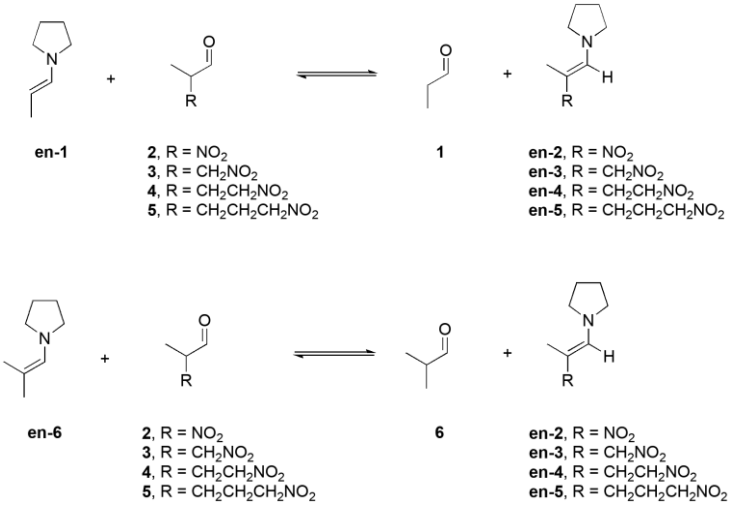
Appendix 4: Script output 51

1. SUMMARY

The chemistry of enamines has gained importance in recent years due to the increased popularity of asymmetric organocatalysis using chiral secondary amines. The main goal of this dissertation is to gain insight, from a theoretical point of view, of the position of the equilibrium involved in the catalytic cycle of nitro-Michael reactions between the product enamine, containing a nitro group, and the starting aldehyde enamine.



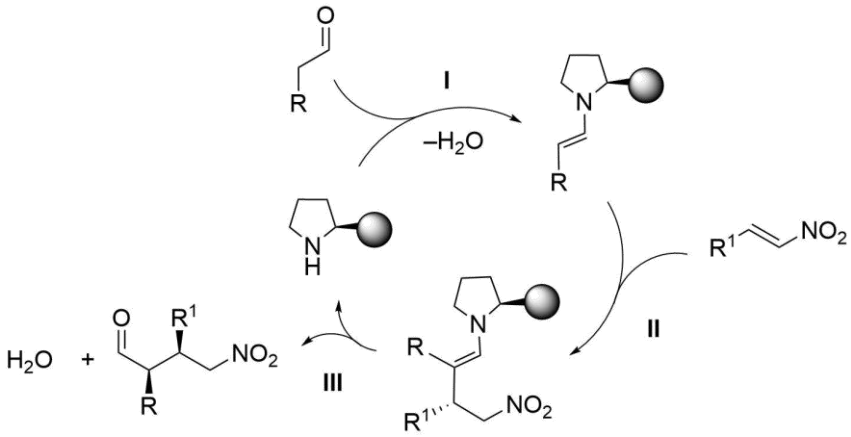
To do so, the energies of the species involved in the equilibria shown below were calculated, at different levels of theory. The calculations were performed with the Gaussian computational chemistry package and some scripts in Bash command language and in Python programming language were written to assist in the automatization of the process.



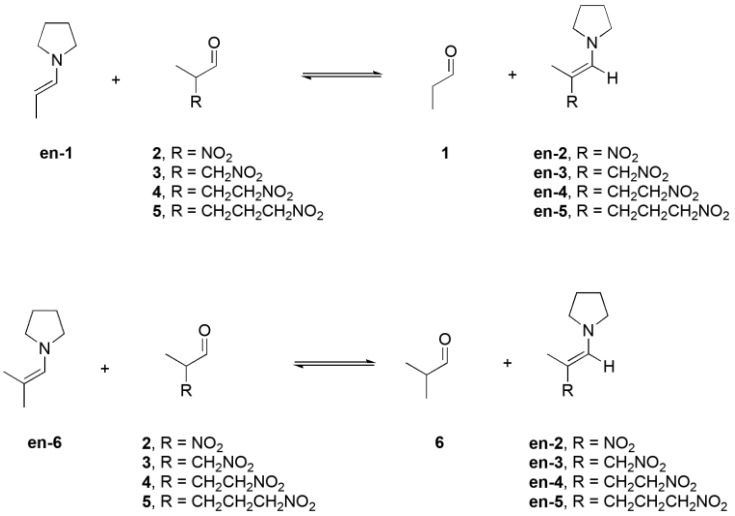
Keywords: organocatalysis, ab initio calculations, enamines, nitroalkenes, Python, Bash, scripting, cheminformatics.

2. RESUMEN

La química de las enaminas ha ganado importancia los últimos años debido al aumento de la popularidad de la organocatálisis asimétrica con aminas secundarias quirales. El objetivo principal de este TFG es comprender mejor, desde un punto de vista teórico, la posición del equilibrio involucrado en el ciclo catalítico de las reacciones nitro-Michael entre la enamina producto, que contiene un grupo nitro y la enamina del aldehído inicial.



Para logar esto se calcularon a diferentes niveles de teoría las energías de las especies involucradas en los equilibrios que se muestran a continuación. Los cálculos se llevaron a cabo con el paquete de cálculos químicos Gaussian. Además, se escribieron algunos scripts en el lenguaje de comandos Bash, y el lenguaje de programación Python que nos ayudaron a automatizar el proceso.



Palabras clave: organocatálisis, cálculos *ab initio*, enaminas, nitroalquenos, Python, Bash, scripting, cheminformatics.

3. INTRODUCTION

The chemistry of enamines has gained importance in recent years due to the increased popularity of organocatalysis.[1,2] For a long time, catalysis in chemistry focused almost completely in two groups of catalysts: metal-containing inorganic products and enzymes. Lately this approach has started to change, as completely organic catalysts, known as organocatalysts, are appearing as a third category with lots of potential for various applications. Organocatalysts tend to be less toxic and damaging to the environment. Some organocatalysts interact with the substrate via non-covalent interactions, such as hydrogen bonding, while others form a covalent adduct between catalyst and substrate, such as the formation, in several steps, of enamines from aldehydes and secondary amines, something that we will discuss in more detail later.[3] From all of the types of organocatalysis, enamine catalysis is, probably, one of the most important methods in this field. It usually involves an enamine intermediate that reacts with an electrophile or undergoes a pericyclic reaction. Several works from our group have studied the formation of enamines from secondary amines and carbonyl compounds.[4,5] In some cases, a theoretical study with calculations based on different density functional theory (DFT) methods was carried out to predict the $\Delta G°$ for exchange equilibria between these compounds. The goal of this project is to apply a similar methodology to improve our understanding of the reactions and intermediates involved in nitro-Michael reactions. To do so, an in-silico approach will be used, using the Gaussian software package to determine the relative stabilities for the above-mentioned compounds at different levels of theory.

3.1. COMPUTATIONAL CHEMISTRY AND MODEL CHEMISTRIES

During this project, one of our interests was to predict the relative energies of molecular systems. For this reason, computational chemistry methods were the perfect match, as reliable results can be obtained and there are many approaches available within a wide range of robust options with different computational cost.

Two types of methods can be used for the tasks of computing energy, optimizing geometry or calculating vibrational frequencies: molecular mechanics or electronic structure methods. The former uses the laws of classical mechanics to calculate properties of molecules, using force fields, each one modeling a certain type of system and therefore yielding good results in a non-expensive manner, but limited to the molecules for which the force field was parameterized. The latter, discussed in more depth in the following paragraphs, uses the laws of quantum mechanics as the basis for calculations. For our calculations, we used electronic structure methods such as DFT or Moller-Plesset perturbation theory (MP2). Unlike molecular mechanics, these methods don't make use of experimental parameters, as their calculations are based only on the laws of quantum mechanics and a few physical constants. So, they aren't limited to any specific system, but they have a much higher computational cost.[6]

According to quantum mechanics, particles can behave both as particles and as waves. The state of a system (and therefore its energy and many other properties), comprised by these particles can be defined by a wave function. This wave function can be described by the Schrödinger Equation:[6]

$$\left\{\frac{-h^2}{8m\pi^2}\nabla^2 + V\right\}\Psi(\vec{r},t) = \frac{ih}{2\pi}\frac{\delta\Psi(\vec{r},t)}{\delta t} \qquad (1)$$

On equation 1, **Ψ** is the wave function, and in a molecule, it would be a function of the coordinates of the particles in the system and time. **V** is the potential field in which the particle can move. If our system is independent of time, the Schrödinger equation can be simplified using separation of variables:[6]

$$H\Psi(\vec{r}) = E\Psi(\vec{r}) \qquad (2)$$

Where E is the energy of the particle and **H** the Hamiltonian operator. This operator is made by kinetic and potential energy terms:

$$H = T + V \qquad (3)$$

The kinetic energy term (**T**) is a summation of the del-squared operator (it is a Laplacian operator, i.e., partial differentiation with respect to x, y and z) applied to all molecules of the system and the potential energy term (**V**) is given by the Coulomb repulsion between each pair of charged entities:[6]

$$V = \frac{1}{4\pi\epsilon_0}\left(-\sum_{i}^{electrons}\sum_{I}^{nuclei}\left(\frac{Z_I e^2}{\Delta r_{iI}}\right) + \sum_{j}^{elec.}\sum_{j<i}^{elec.}\left(\frac{e^2}{\Delta r_{ij}}\right) + \sum_{I}^{nucl.}\sum_{J<I}^{nucl.}\left(\frac{Z_I Z_J e^2}{\Delta R_{ij}}\right)\right) \qquad (4)$$

Where *R* and *r* refer to the positions of nuclei and electrons, respectively. Z_k is the atomic number for the atom indicated by the sub-index and e is the charge for each particle, the first term corresponds to electron-nucleus attraction, the second to repulsion between electrons and the last to nucleus-nucleus repulsion.

For the solution of the Schrödinger equation to be practical to calculate, even in computers, several approximations must be made. The Born-Oppenheimer approximation simplifies the general problem by separating the motions of nuclei and electrons, due to the latter being many times smaller than the former. Electronic motion can be therefore described as taking place in a field of fixed nuclei. This allows to rewrite the Hamiltonian operator as an electronic Hamiltonian ($\mathbf{H^{elec}}$) which excludes the nuclei kinetic energy. Solving the Schrödinger equation using this operator instead will yield the effective nuclear potential function ($\mathbf{E^{eff}}$), which describes the potential energy surface for the system. The nuclear Hamiltonian ($\mathbf{H^{nuc}}$) can be deduced from E^{eff} which will describe the states of the nuclei.[6]

An additional approximation which can be done to ease the calculations cost is decomposing the wave function into a combination of orthonormal molecular orbitals (φ_n), forming the *Hartree product*. This function, however, is not anti-symmetric because swapping orbitals does not produce a sign change, which is a requirement of an electronic wave function. Due to the Pauli exclusion principle, any spin orbital wave function of any number of electrons must change sign when two electrons are swapped (as electrons are fermions, particles which exhibit anti-symmetry and a half-integral quantum number). Hence, we need to account for electron spin in our function, resulting in the inclusion of spin orbitals, a function which encompasses the electron's position and spin. Using spin orbitals, a determinant including all the possible orbitals of the electrons can be defined (known as Slater determinant), showing that the electron can be anywhere and therefore defining the wave function. A closed shell system can be built by defining n/2 molecular orbitals for a system with n electrons and then assigning them in pairs of opposite spin (α and β):[6,7]

$$\Psi(\vec{r}) = \frac{1}{\sqrt{n!}} \begin{vmatrix} \phi_1(\vec{r}_1)\alpha(1)\phi_1(\vec{r}_1)\beta(1) & \phi_2(\vec{r}_1)\alpha(1)\phi_2(\vec{r}_1)\beta(1) & \cdots & \phi_{\frac{n}{2}}(\vec{r}_1)\alpha(1)\phi_{\frac{n}{2}}(\vec{r}_1)\beta(1) \\ \phi_1(\vec{r}_2)\alpha(2)\phi_2(\vec{r}_2)\beta(2) & \phi_2(\vec{r}_2)\alpha(2)\phi_2(\vec{r}_2)\beta(2) & \cdots & \phi_{\frac{n}{2}}(\vec{r}_2)\alpha(2)\phi_{\frac{n}{2}}(\vec{r}_2)\beta(2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\vec{r}_n)\alpha(n)\phi_1(\vec{r}_n)\beta(n) & \phi_2(\vec{r}_n)\alpha(n)\phi_2(\vec{r}_n)\beta(n) & \cdots & \phi_{\frac{n}{2}}(\vec{r}_n)\alpha(n)\phi_{\frac{n}{2}}(\vec{r}_n)\beta(n) \end{vmatrix} \quad (5)$$

Another useful approximation consists of representing the molecular orbitals as linear combinations of a set of functions for one electron (basis functions). A molecular orbital is defined as:

$$\phi_i = \sum_{\mu=1}^{N} c_{\mu i}\chi_\mu \quad (6)$$

Where χ_n are the normalized basis functions and $c_{\mu i}$ the molecular orbital expansion coefficients. For an open shell system, the alpha and beta electrons should be in different orbitals. This is used in the unrestricted Hartree-Fock (HF) methods. The use of primitive Gaussian orbitals and its combination leads *to* contracted Gaussian orbitals, which can be used in the molecular orbital expression. After this, only the orbital expansion coefficients are left to be calculated, and using the variational principle, which states that the ground state of any anti-symmetric normalized electronic function will always have a greater energy than the exact wave function, the problem changes into finding the coefficients that minimize the function, getting as close as possible to the exact wave function. The orbital expansion coefficients can be described by the Roothaan-Hall equations:

$$\sum_{v=1}^{N} \left(F_{\mu v} - \varepsilon_i S_{\mu v}\right) c_{vi} \quad (7)$$

$$FC = SC\varepsilon \quad (8)$$

Equation 8 shows the matrix form of the Roothaan-Hall equation, where each element is a matrix. **ε** is a matrix comprised of orbital energies, **F** is the *Fock* matrix*,* which represents an average of all the electrons field on each orbital and contains the two-electron repulsion integrals*.* Under Hartree-Fock theory each electron sees all the other electrons as an average electron distribution, which is a limiting factor for this level of theory. **S** is the overlap matrix, showing the overlap between orbitals. Both sides of the equation depend on the molecular orbital expansion coefficient, therefore the equation must be solved iteratively. The procedure for doing so is known as the Self-Consistent Field (SCF) method. When the solution converges, the energy is at a minimum. The solution provides a set of orbitals, both occupied and unoccupied (virtual orbitals).

Higher theory levels, which go beyond SCF to include electronic correlation effects for electrons of opposite spin are known as Electron Correlation Methods. The most exhaustive method is the Full Configuration Interaction method. Configuration interaction methods (CI) add additional determinants instead of only using one as in HF theory. These determinants are formed

by substituting occupied orbitals with virtual ones, equivalent to exciting an electron to a higher energy orbital. When the wave function is a linear combination of all the possible determinant substitutions the method is Full CI. This method allows for the most complete treatment of the system, only limited by the basis set chosen. However, it is very expensive and impractical for complex systems, so Limited Configuration methods are commonly used instead. If we wanted to solve the Schrödinger equation for a multi-body system, say, a diamond crystal with a unit cell volume of 50 Å, formed by points spaced by 0.2 Å, the resulting point grid will be comprised by 6250 points. If there were two atoms per cell, with four valence electrons each, the combination would give $6250^{10} = 9 \cdot 10^{36}$ possible complex numbers. With arrays of this size, performing calculations is impossibly costly. That is why approximations are needed for all but the most trivial of systems.[7]

Other electron correlation methods employ the Møller-Plesset Perturbation Theory. Here, the Hamiltonian is divided in two parts:[6]

$$H = H_0 + \lambda V \quad (9)$$

Here λV is a perturbation added to the Hamiltonian, small in comparison to it, being V the perturbation and λ a product which multiplies the perturbation. Expanding this expression and substituting into the Schrödinger equation, a solution for $E^{(0)}$, $E^{(1)}$ and $E^{(2)}$ can be found. $E^{(0)}$ is the lowest energy eigenvalue of the unperturbed system. Adding $E^{(0)}$ and $E^{(1)}$ will yield the Hartree-Fock Energy (E^{HF}). $E^{(2)}$ will be the first perturbation to the E^{HF}, which will always be negative.

The last method for Electron Correlation we will discuss are Density Functional Theory (DFT) methods, that model electron correlation by employing functionals of the spatially dependent electron density. By applying these functionals the electronic energy can be now described by:[6]

$$E_{tot} = E^T + E^V + E^J + E^{XC} \quad (10)$$

Where E^T is the electronic kinetic energy, E^V the potential energy of nucleus-electron attraction and nucleus-nucleus repulsion, E^J is the electron-electron repulsion and E^{XC} the exchange-correlation term, which amounts for the rest of the electron-electron interactions (exchange energy from the anti-symmetry of the wave function and dynamic correlation in the motions of electrons). E^{XC} is usually approximated as an integral which treats spin density and optionally its gradients, and can be split in two terms, the exchange and the correlation terms. Exchange and correlation functionals can be defined for describing the two terms of the E^{XC}

expression and they can be of two types, which can be local functionals (only depend of the electron density ρ) and gradient-corrected functionals (which depend on both ρ and its gradient).

Various functionals have been formulated, some including a mixture of the exchange included in HF and DFT theory and adding DFT correlation, being classified as hybrid functionals. In our work most of the functionals we will use will be hybrid functionals, for example the Becke-style three-parameter functional (B3LYP).[8]

For calculations, a model chemistry should be chosen according to its practicality and what is needed. A model chemistry includes the theoretical procedure (or method) to be used and the basis set. How the models are used will be discussed in the experimental section. The models employed during this project are shown in Table 1.

Internal name	Method name	Basis set
DFTmin	B3LYP	6-31G(d)
MP2min[a]	MP2	6-31G(d)
M06min	M06-2X	6-31G(d)
M06max	M06-2X	6-311+G(d,p)

[a] using geometries optimized at the DFTmin level

Table 1. Model chemistries employed during our project. The internal name will be used for quick reference.

3.2 NITRO-MICHAEL REACTIONS

The equilibria that will be studied in the following sections are derived from two steps from the catalytic cycle of a nitro-Michael reaction. A Michael reaction consists of the addition of a nucleophile (donor) to an acceptor, normally an α, β-unsaturated carbonyl compound (enone), or an enoate.[9]

During the reaction, two asymmetric centers are generated. Using chiral catalysts, such as certain secondary amine organocatalysts, it is possible to control the stereochemistry of the newly formed stereocenters and carry out enantioselective Michael additions. This is very useful in a wide range of fields, for instance, in specialized pharmaceutical compound synthesis. Usually, these catalysts are derived from proline.[10]

In a nitro-Michael reaction (see Figure 1) a carbon nucleophile adds to the β position of an α, β-unsaturated nitro compound.

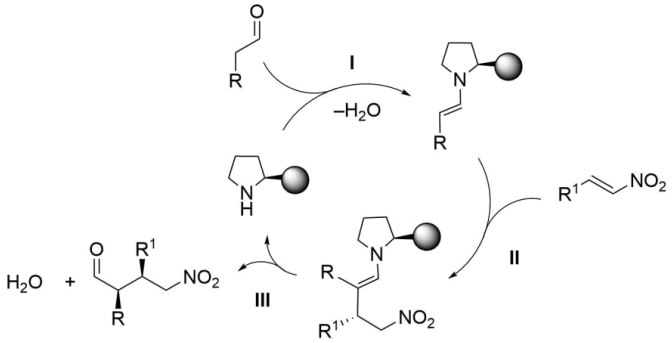


Figure 1. Schematic catalytic cycle of a nitro-Michael reaction.

In nitro-Michael reactions catalyzed by proline derivatives, the donor is the in-situ generated enamine, prepared from the aldehyde or ketone and the catalyst. This enamine then reacts with the nitro compound, yielding (in several steps) the desired product enamine.

Depending on the starting materials and reactions conditions, some nitro-Michael reactions don't work well under catalysis with secondary amines. To gain insight into why it is sometimes necessary to use a stoichiometric amount of the catalyst for the reaction to complete, we decided to study, computationally, the equilibrium shown in Figure 2, corresponding to steps I+III of the catalytic cycle. Water is released during the formation of the starting enamines (step I), while water is required during the last step(s), see eq III, to hydrolyze the final enamines (product enamines); for a successful catalytic cycle, the following equilibria must be shifted to the right.

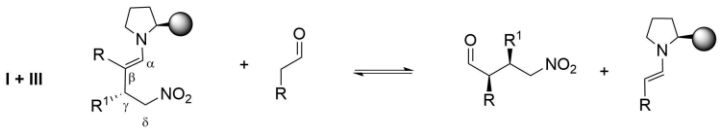


Figure 2. Equilibria to be studied.

Although only enamines with nitro groups in the δ position are relevant for the outcome of nitro-Michael reactions, we decided to extend our study to enamines and aldehydes containing nitro groups also in positions α, β, and γ.

4. OBJECTIVES

The main goal of this project is to assess the ease of formation of several pyrrolidine enamines derived from carbonyl compounds containing a nitro group in different positions, as well as the reverse reactions (the hydrolyses of the resulting enamines) to improve our understanding of the underlying reaction mechanisms in nitro-Michael reactions. To do this, several equilibrium reactions were defined (see Figure 8), and for each compound the goal was to:

- Obtain geometry coordinates for its global minimum. In molecules with several possible conformations, a conformational search was sometimes needed.
- Calculate the energy for each compound at different levels of theory:
    - B3LYP/6-31G(d)
    - MP2/6-31G(d)//B3LYP/6-31G(d)
    - M06-2X/6-31+G(d)
    - M06-2X/6-311+G(d)
- Calculate ΔG^{o} for each reaction
- Repeat the calculations using water as implicit solvent.

To fulfill these objectives, the following sub-goals appeared:

- To compare the results of conformational searches obtained using Avogadro and MacroModel programs to check if the results obtained with Avogadro are as reliable as with MacroModel.
- To compare the results obtained using different levels of theory for our systems, as the accuracy of the results may vary between methods.
- To write several scripts for automatically obtaining key parameters of our molecules in bulk, such as the planarity of an amine nitrogen, dihedral angles, etc.

- To design a fast and reliable result gathering script with the capability of automatically sending computed results in formatted tables including software-drawn molecule depictions.
- To create a software repository for helping with version control of the different scripts.

5. EXPERIMENTAL SECTION

The starting geometries of the different conformers of each molecule were built using several software packages. It is important to locate the lowest energy conformation of every molecule, in order to then obtain reliable energies of the equilibria shown in Figure 2. For the simplest compounds the starting geometries were entered manually using GaussView 6.0.[11] Gaussian job files (GJF) were used as inputs for the calculations, by specifying the main options and other parameters. A GJF file includes three main parts: the route line near the top of the document where model chemistries and job types are stated, a title and finally the molecule specifications, where the molecular system is defined. With larger molecules a conformational search using Maestro[12] was performed. The settings for the search were as follows: the OPLS_2005 force field was used, with water as solvent in some cases. The PRCG minimization method was used, with 2500 maximum iterations. Because of the COVID-19 epidemic, consistent access to our regular software tools was not possible, so open-source software alternatives were studied. The new packages used were Avogadro,[13] as the molecule editor, and Open Babel 3.0.0[14] for conformational searches. Both software utilities were useful for building the structures of the compounds, but we found the Open Babel conformational search to be somewhat limited in comparison to Maestro's. The 3D coordinates of the different molecules were stored as ".pdb" (protein data base files) files for compatibility, which later are converted to ".gjf" (Gaussian job files) using Open Babel and scripts written by our team. The scripts will be discussed later in this section. For the energy calculations, the Gaussian 16 software was used.[15]

Henceforth, the standard notation to specify the level of theory and basis set will be: "*level*$_1$ */basis*$_1$*//level*$_2$*/basis*$_2$*"*, when two different levels are used. This notation comprises two parts. The one to the left of the "//", which gives information of the level of theory and basis set used to calculate the energy and the one to the right, which shows the level of theory and basis set used in the geometry optimization. This separation is due to the smaller sensitivity of geometry to lower-level theory levels, allowing to use less expensive methods for this part of the calculation.[16]

Calculations were initially performed at the B3LYP/6-31G(d) level (DFTmin_opt), as it allows for much more accurate results than HF methods, for a fraction of the cost of an MP2 method. It uses B3LYP, a hybrid functional which yields good geometry results in most cases. However, in previous works of our group[17] it was found that the energies of enamines weren't described accurately using this method. Better energies are obtained at the MP2/6-31G(d)//B3LYP/6-31G(d) (MP2min) level of theory.

Next, calculations were also performed at the M06-2X/6-31G(d) (M06min) and M06-2X/6-311+G(d,p) (M06max) levels of theory. M06-2X is a more modern hybrid functional than B3LYP, it has been extensively used with good results in previous works by our group.[4,17]

For the free energy calculations, frequency jobs were run at the M06-2X/6-311+G(d,p) (M06max) level.

All the previous calculations were carried out without solvent and were repeated using water as the implicit solvent. This was done by employing Gaussian's self-consistent reaction field (SCRF) implementation, using the PCM solvation model, which creates a cavity where the solute goes by placing spheres in a simulated area filled with solvent and has a set of parameters for water, which can be called by adding "scrf=(pcm, water)" to the route line.

The whole set of calculations amount for the generation of many GJF files, so a script was utilized for easing the preparation of the calculation. This script, written in bash, helps converting the molecule depiction format to the GJF format, and then writes the route line and other specific options:

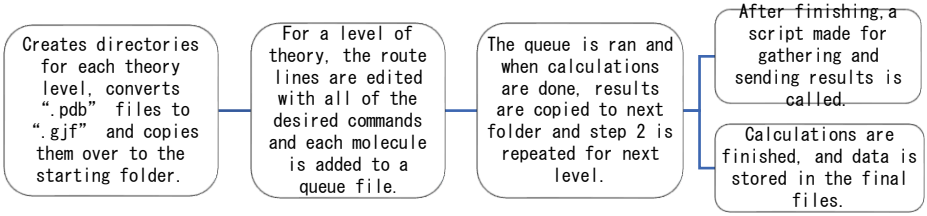


Figure 3. Diagram depicting the functioning of the various calculation scripts.

An additional script was written to calculate the distance of the enamine nitrogen to the plane formed by the three surrounding carbon atoms, as shown in Figure 4. This allowed us to further

characterize the nitrogen hybridization, as its distance to said plane depends on the atoms bonded to it.

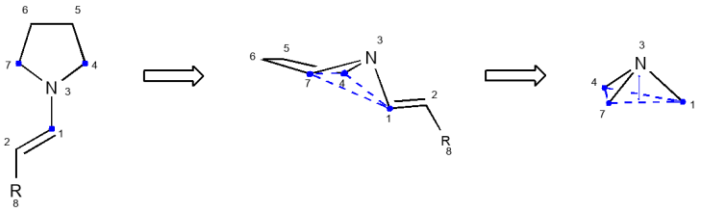


Figure 4. Schematic representation of the plane (in blue) formed by the enamine nitrogen and the three surrounding atoms. The distance to the plane is shown in the third step as a thin blue line.

The code for this program was written in Python 3.7 and is available in Appendix 3 near the end of the document. Its operation, in a simplified way, can be summarized by the flow diagram in Figure 5.

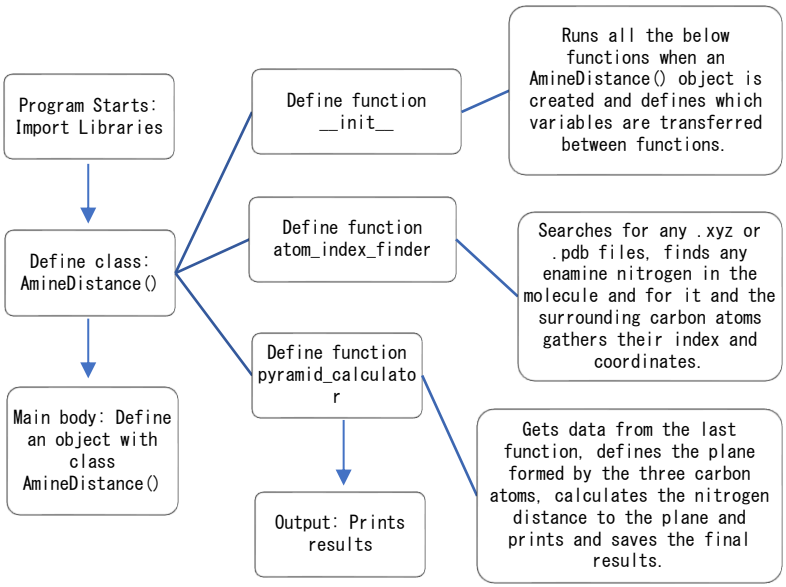


Figure 5. Diagram depicting the functioning of the script "amine_planar_distance.py".

In Python, classes are a means of grouping functions and data together that allows creating different objects. The objects can then have their attributes modified by using methods that are

built inside of the classes when created.[18] The script was chosen to be written in this way, as classes can be expanded upon and modified when needed, without altering significantly how the entire program works. Additional functions could be defined inside of the "*AmineDistance()"* class to gather results from additional file formats or improve detection of enamines using the *SMILES* identifiers, for example.

As calculations progress "**.log*" files are generated, which contain all information regarding our molecules, thus results must be manually extracted. This takes time, so work was put into creating a small script to automatically extract the data, classify and organize it. This script was once again written in Python 3.7 and its operation can be seen in the scheme of Figure 6.

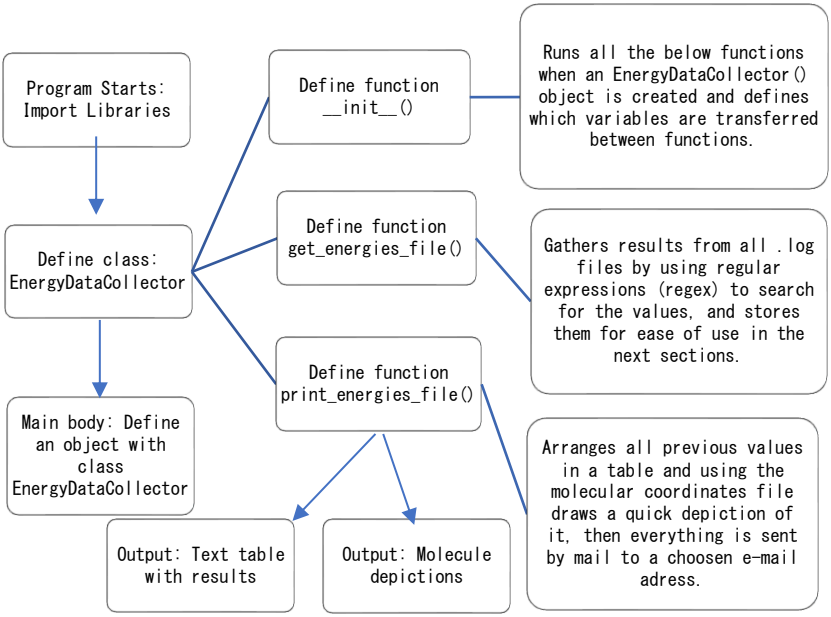


Figure 6. Diagram depicting the functioning of the script "energy_data_collector.py".

An example of a file generated by the script can be found in Appendix 4, showing gathered results and a few depictions.

Another script was written to automatically calculate the dihedral angle between two planes defined by 4 atoms in a set of molecules. Although not directly related to this project, the script will be useful to the group in other studies.

Also, an additional simple script was written to find the minimal energy molecule from a given set of results and calculates their proportions in an equilibrium following a Boltzmann distribution.

All the scripts developed for this project have been compiled in a *GitHub repository*, so all the files can be downloaded and used by everyone. The repository is found at the following URL (*https://github.com/tetsuo420/cognitive-episode/releases*) or by scanning the QR Code shown in Figure 7.

Figure 7. QR code for the repository containing all the scripts.

6. RESULTS AND DISCUSSION

6.1 INITIAL GEOMETRIES

Figure 8 shows the aldehydes and enamines used in this work. The geometry of the reagents was obtained using both GaussView and Avogadro interchangeably, as we found no differences between results.

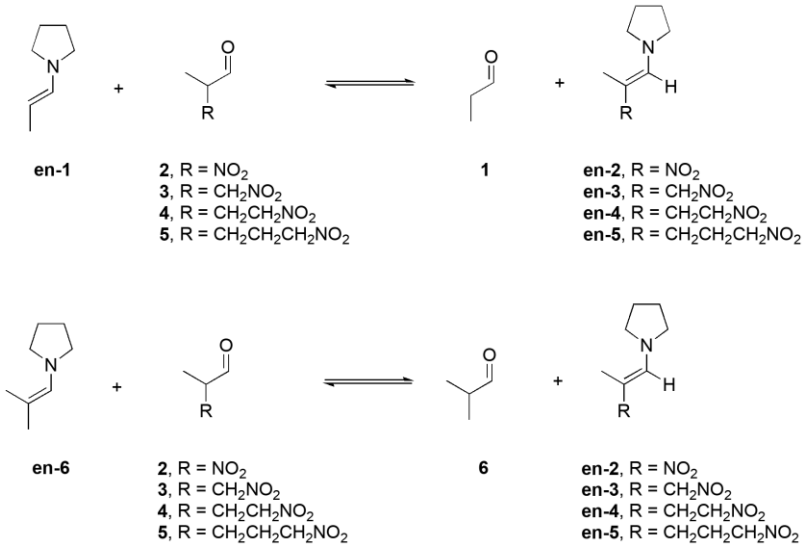


Figure 8. Summary of the reactions and compounds studied in this work.

For the simplest compounds, the lowest energy minimum (global minimum) was obtained manually. However, aldehydes and enamines with longer chains required a conformational search. This search was initially done with Maestro, but use of Avogadro's conformational search tool, which is a part of the Open Babel chemical toolbox was also examined.

Eventually it was found that, when optimized at the DFTmin level, minimum energy values obtained with Maestro were better than the ones obtained with Avogadro, so the rest of the work was done making use of Maestro. While more intricate, the command-line tool that Avogadro

makes use of to calculate the energy minima has a wide range of options that might prove useful, so further exploration of the software is encouraged.

For the enamines, the planarity of the nitrogen was evaluated using our script *amine_planar_distance.py* (described in the experimental section) with the geometries of the enamines obtained after optimization at the M06-2X/6-311+G(d,p) level of theory. The results obtained are shown in the next table and a clear tendency is evident: the less substituted the enamine, the more planar is the N, indicating an interaction between the enamine double bond and the lone electron pair from the nitrogen that has a hybridization close to sp^2.

Enamine	Distance between enamine nitrogen and plane [A] (a)
en-1	0,00205
en-6	0,03910
en-2	0,16928
en-3	0,24453
en-5	0,26382
en-4	0,28944

(a) Calculations obtained from our script *amine_planar_distance.py*. The script first searches for the nitrogen position, then finds the surrounding C atoms and finds their coordinates, which then uses to calculate the plane equation, from which the d(Nitrogen-Plane) is obtained by trigonometry calculations. See the experimental section for a more in depth look at the code.

Table 2. Results from planar distance calculations made by the script amine_planar_distance.py.

6.2 CALCULATION OF Δ*E*, Δ*G* AND Δ*H* FOR THE EQUILIBRIA STUDIED

Once the global energy minimum for each compound in Figure 8 was located, its geometry was optimized, and its energy calculated at different levels of theory (see Experimental Section).

Then ΔE for the reactions shown in Figure 8 could be computed. Table 3 shows the results obtained at different theory levels. The rest of the results can be found in Appendices 1 and 2.

We know, from previous works, that B3LYP/6-31G(d) energies are not reliable to study these systems. Better results are usually obtained with energies computed at the MP2/6-31G(d)//B3LYP/6-31G(d) level. We also used the more modern M06-2X functional. As can be seen in Table 3, the results vary depending on the level of theory used. Because the calculation time involved in the M06max calculations (M06-2X/6-311+G(d)) was reasonable we decided to use this more accurate, higher level of theory. For this reason, from now on only these results will be shown here. Complete results can be found in Appendices 1 and 2.

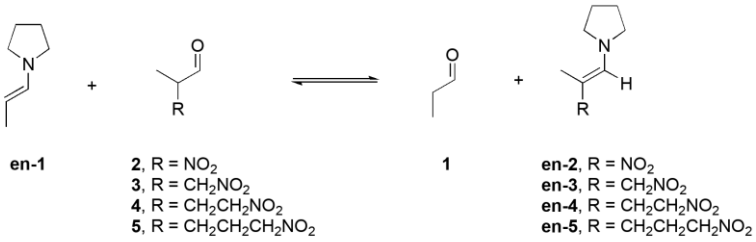


Figure 9. Reactions shown in Table 3.

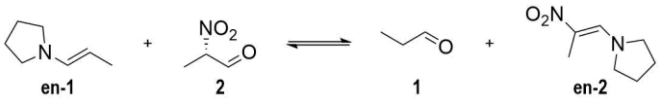
Level of theory	ΔE_{vacuum} (kcal/mol)		
	R = NO_2	**R = CH_2NO_2**	**R = $CH_2CH_2NO_2$**
DFTmin	–11,4	–1,4	–0,4
MP2min	–8,5	–0.8	–1,4
M06min	–10,2	–1,2	–4,6
M06max	–9,2	–0,9	–0,7

Table 3. ΔE for the reactions, without solvent.

Two different sets of equilibria have been studied. They differ in the enamine that equilibrates with the nitro-substituted aldehydes. The first set of reactions involve the equilibrium of the pyrrolidine enamine of propanal ((*E*)-1-(prop-1-en-1-yl)pyrrolidine, **en-1**) with nitro compounds **2** to **5** to yield propionaldehyde **(**propanal**, 1)** and a nitro-containing enamine (**en-2** to **en-5).** The latter set of reactions employ 1-(2-methylprop-1-en-1-yl)pyrrolidine **(en-6)** as a starting reagent.

The results for all the equilibria studied, at the M06-2X/6-311+G(d) level of theory are summarized in the following tables.

Reaction 1



M06max	**en-1** (Ha)	**2** (Ha)	**1** (Ha)	**en-2** (Ha)	**Total** (kcal/mol)
E_{vacuum}	−329,22733	−397,59514	−193,10903	−533,72807	**−9,2**
H^{o}_{gas}	−329,02614	−397,49834	−193,01805	−533,52050	**−8,8**
G^{o}_{gas}	−329,06932	−397,53839	−193,05116	−533,56969	**−8,2**
E_{water}	−329,23075	−397,60586	−193,11545	−533,74439	**−14,6**
H^{o}_{water}	−329,02993	−397,50924	−193,02455	−533,53721	**−14,2**
G^{o}_{water}	−329,07309	−397,54919	−193,05756	−533,58614	**−13,4**
(a) **1** and **en-1** were drawn and optimized with Avogadro software. **2** and **en-2** were drawn by GaussView and their lowest energy conformer was found by Maestro's software conformational search.					

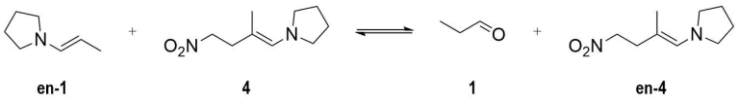
Table 4. Summary of results for reaction 1.

Reaction 2

M06max	**en-1** (Ha)	**3** (Ha)	**1** (Ha)	**en-3** (Ha)	**Total** (kcal/mol)
E_{vacuum}	−329,22733	−436,90376	−193,10903	−573,02356	**−0,9**
H^{o}_{gas}	−329,02614	−436,77660	−193,01805	−572,78627	**−1,0**
G^{o}_{gas}	−329,06932	−436,82050	−193,05116	−572,84061	**−1,2**
E_{water}	−329,23075	−436,91586	−193,11545	−573,03540	**−2,7**
H^{o}_{water}	−329,02993	−436,78879	−193,02455	−572,79844	**−2,7**
G^{o}_{water}	−329,07309	−436,83297	−193,05756	−572,85218	**−2,3**
(a) **1** and **en-1** were drawn and optimized with Avogadro software. **3** and **en-3** were drawn by GaussView and their lowest energy conformer was found by Maestro's software conformational search. All calculations done with Gaussian 16 software.					

Table 5. Summary of results for reaction 2.

Reaction 3



en-1 **4** **1** **en-4**

M06max	**en-1** (Ha)	**4** (Ha)	**1** (Ha)	**en-4** (Ha)	**Total** (kcal/mol)
E_{vacuum}	−329,22733	−476,20512	−193,10903	−612,33053	**−4,5**
H^{o}_{gas}	−329,02614	−476,04793	−193,01805	−612,06317	**−4,5**
G^{o}_{gas}	−329,06932	−476,09645	−193,05116	−612,11873	**−2,6**
E_{water}	−329,23075	−476,22008	−193,11545	−612,34024	**−3,0**
H^{o}_{water}	−329,02993	−476,06309	−193,02455	−612,07331	**−3,0**
G^{o}_{water}	−329,07309	−476,11123	−193,05756	−612,12945	**−1,7**

(a) **1** and **en-1** were drawn and optimized with Avogadro software. **4** and **en-4** were drawn by GaussView and their lowest energy conformer was found by Maestro's software conformational search. All calculations done with Gaussian 16 software.

Table 6. Summary of results for reaction 3.

Reaction 4

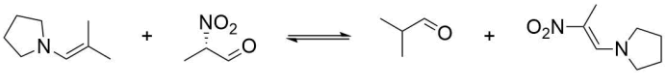
en-1 **5** **1** **en-5**

M06max	**en-1** (Ha)	**5** (Ha)	**1** (Ha)	**en-5** (Ha)	**Total** (kcal/mol)
E_{vacuum}	−329,22733	−515,51198	−193,10903	−651,63742	**−4,5**
H^{o}_{gas}	−329,02614	−515,32479	−193,01805	−651,33999	**−4,5**
G^{o}_{gas}	−329,06932	−515,37504	−193,05116	−651,39616	**−1,9**
E_{water}	−329,23075	−515,52614	−193,11545	−651,64578	**−2,7**
H^{o}_{water}	−329,02993	−515,33915	−193,02455	−651,34883	**−2,7**
G^{o}_{water}	−329,07309	−515,38952	−193,05756	−651,40505	**0,0**

(a) **1** and **en-1** were drawn and optimized with Avogadro software. **5** and **en-5** were drawn by GaussView and their lowest energy conformer was found by Maestro's software conformational search. All calculations done with Gaussian 16 software.

Table 7. Summary of results for reaction 4.

Reaction 5

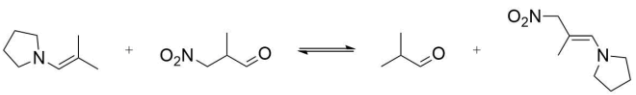


en-6 **2** **6** **en-2**

M06max	**en-6** (Ha)	**2** (Ha)	**6** (Ha)	**en-2** (Ha)	**Total** (kcal/mol)
E_{vacuum}	−368,53008	−397,59514	−232,41391	−533,72807	**−10,5**
H^{o}_{gas}	−368,29953	−397,49834	−232,29328	−533,52050	**−10,0**
G^{o}_{gas}	−368,34563	−397,53839	−232,32979	−533,56969	**−9,7**
E_{water}	−368,53305	−397,60586	−232,42007	−533,74439	**−16,0**
H^{o}_{water}	−368,30293	−397,50924	−232,29954	−533,53721	**−15,4**
G^{o}_{water}	−368,34910	−397,54919	−232,33594	−533,58614	**−14,9**
(a) **6** and **en-6** were drawn and optimized with Avogadro software. **2** and **en-2** were drawn by GaussView and their lowest energy conformer was found by Maestro's software conformational search.					

Table 8. Summary of results for reaction 5.

Reaction 6

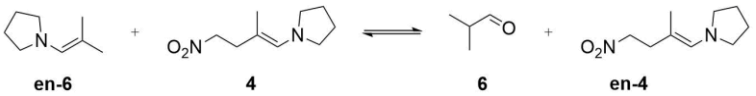


en-6 **3** **6** **en-3**

M06max	**en-6** (Ha)	**3** (Ha)	**6** (Ha)	**en-3** (Ha)	**Total** (kcal/mol)
E_{vacuum}	−368,53008	−436,90376	−232,41391	−573,02356	**−2,3**
H^{o}_{gas}	−368,29953	−436,77660	−232,29328	−572,78627	**−2,1**
G^{o}_{gas}	−368,34563	−436,82050	−232,32979	−572,84061	**−2,7**
E_{water}	−368,53305	−436,91586	−232,42007	−573,03540	**−4,1**
H^{o}_{water}	−368,30293	−436,78879	−232,29954	−572,79844	**−3,9**
G^{o}_{water}	−368,34910	−436,83297	−232,33594	−572,85218	**−3,8**
(a) **6** and **en-6** were drawn and optimized with Avogadro software. **3** and **en-3** were drawn by GaussView and their lowest energy conformer was found by Maestro's software conformational search.					

Table 9. Summary of results for reaction 6.

Reaction 7



en-6 **4** **6** **en-4**

M06max	**en-6** (Ha)	**4** (Ha)	**6** (Ha)	**en-4** (Ha)	**Total** (kcal/mol)
E_{vacuum}	−368,53008	−476,20512	−232,41391	−612,33053	**−5,8**
H^{o}_{gas}	−368,29953	−476,04793	−232,29328	−612,06317	**−5,6**
G^{o}_{gas}	−368,34563	−476,09645	−232,32979	−612,11873	**−4,0**
E_{water}	−368,53305	−476,22008	−232,42007	−612,34024	**−4,5**
H^{o}_{water}	−368,30293	−476,06309	−232,29954	−612,07331	**−4,3**
G^{o}_{water}	−368,34910	−476,11123	−232,33594	−612,12945	**−3,2**

(a) **6** and **en-6** were drawn and optimized with Avogadro software. **4** and **en-4** were drawn by GaussView and their lowest energy conformer was found by Maestro's software conformational search. All calculations done with Gaussian 16 software.

Table 10. Summary of results for reaction 7.

Reaction 8

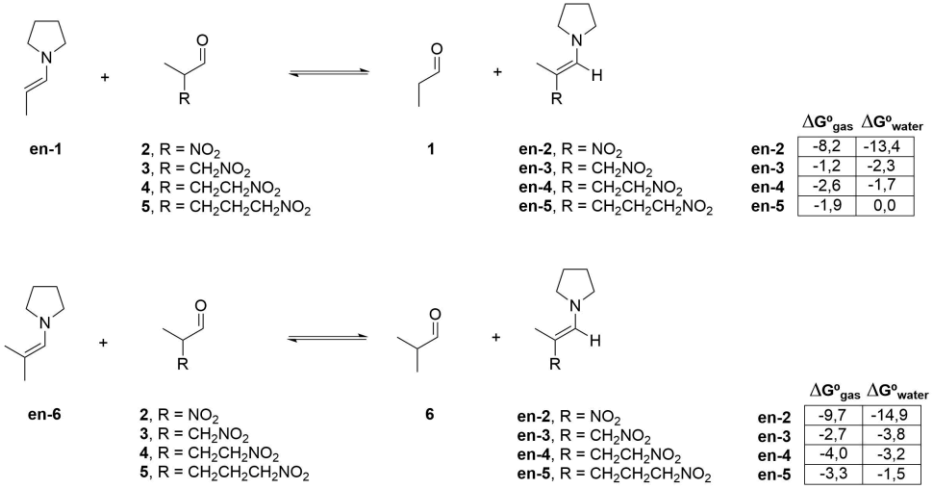
en-6 **5** **6** **en-5**

M06max	**en-6** (Ha)	**5** (Ha)	**6** (Ha)	**en-5** (Ha)	**Total** (kcal/mol)
E_{vacuum}	−368,53008	−515,51198	−232,41391	−651,63742	**−5,8**
H^{o}_{gas}	−368,29953	−515,32479	−232,29328	−651,33999	**−5,6**
G^{o}_{gas}	−368,34563	−515,37504	−232,32979	−651,39616	**−3,3**
E_{water}	−368,53305	−515,52614	−232,42007	−651,64578	**−4,2**
H^{o}_{water}	−368,30293	−515,33915	−232,29954	−651,34883	**−3,9**
G^{o}_{water}	−368,34910	−515,38952	−232,33594	−651,40505	**−1,5**

(a) **6** and **en-6** were drawn and optimized with Avogadro software. **5** and **en-5** were drawn by GaussView and its lowest energy conformer was found by Maestro's software conformational search.

Table 11. Summary of results for reaction 8.

6.3 RESULTS AND DISCUSSION



	ΔG°gas	ΔG°water
en-2	-8,2	-13,4
en-3	-1,2	-2,3
en-4	-2,6	-1,7
en-5	-1,9	0,0

	ΔG°gas	ΔG°water
en-2	-9,7	-14,9
en-3	-2,7	-3,8
en-4	-4,0	-3,2
en-5	-3,3	-1,5

Figure 10. Summary of the ΔG° results (in kcal/mol) obtained to date.

Figure 10 summarizes the ΔG° results obtained for the set of equilibria studied to date. Inspecting the results obtained for all the calculations it can be observed that most of the free energy values are negative, indicating that for our compounds the formation of the nitro-containing enamine is favored over the enamines with no nitro groups. This might be unfavorable because, in a nitro-Michael reaction, to complete the cycle, the product enamine, containing a nitro group, must equilibrate with the starting aldehyde to yield the product aldehyde, freeing the catalyst, so that the starting aldehyde enamine can be formed again in step I of the catalytic cycle (Figure 1). This could explain why some of these reactions need stoichiometric quantities of catalyst to reach completion or an excess of the starting aldehyde, to drive the equilibrium to the formation of the product aldehyde.

Upon further inspection of the results, the equilibria using **en-6** as a reagent always are more exergonic than the ones with **en-1**, possibly indicating that, the more substituted the starting aldehyde, the less favored the initial enamines and the worse the overall process will work.

For the α-nitroaldehydes the equilibrium is more shifted to the substituted enamine containing the nitro group. This agrees with what is known about the destabilizing interaction between a nitro group and a carbonyl group and the stabilizing effect of the conjugation between a nitro group and an enamine.

For the remaining nitro derivatives (β, γ, and δ), apart from the inductive or field effect (which is expected to decrease from α- to δ-NO_2 derivatives), the expected favorable through-space electrostatic interaction between the nitroalkyl moiety (a strong electron-withdrawing group) and the enamine substructure (with a strong electron-donating character) relatively stabilize the enamines of the products (the final enamines) in folded conformations. In short, the β-, γ-, and δ-nitroenamines examined in this work are relatively more stable and less prone to hydrolysis than it could be expected at first sight.

7. CONCLUSIONS

The study of the viability of the Michael and nitro-Michael reactions is an interesting area of research not only from the perspective of organocatalysis, but also in the context of the enantioselective synthesis of new compounds, or of the search for alternative synthetic routes, thanks to the low cost of the catalysts used. The use of computational techniques to predict the outcome of a reaction, to compute valuable properties, and to observe the trends that arise by easily modifying the structure of the species involved in a less expensive and laborious way is a very important technique that we have been able to take advantage of in this work.

Thus, a set of equilibria were chosen to gain insight into the mechanism of the nitro-Michael reactions. For each species its minimum energy conformation was located and its thermodynamic parameters, G^{o} and H^{o} were computed at different levels of theory in vacuum and in water using Gaussian 16. From these values, ΔG^{o} was estimated or obtained for each equilibrium. These values predict that most of the reactions are exergonic, indicating that the formation of the enamine that contains the nitro group is favored over the enamine without a nitro group. As stated in the discussion, the β-, γ-, and δ-nitroenamines examined in this work are relatively more stable and less prone to hydrolysis than it could be expected at first sight.

This fact can be counterproductive for the catalytic cycles of these nitro-Michael reactions to turn over. This could explain the requirement, in some cases, of stoichiometric amounts of catalyst or an excess of initial aldehyde, to favor the formation of the product and to assure that the cycle proceeds forward.

The degree of substitution of the starting aldehyde seems to have a notable effect on the position of the equilibria studied, as the reactions become more exergonic when a more substituted initial aldehyde is used, possibly indicating that the starting enamines are less favored.

Also, as a secondary objective, attempts have been made to automate as far as possible parts of the computational protocols. With the use of Python programming languages and the preparation of Bash scripts, we have been able to streamline processes in a remarkable way. Therefore, I highly recommend implementing any of these tools (or similar ones) for projects like this one. A software repository of the different scripts written was created.

8. REFERENCES AND NOTES

1. MacMillan, D. W. The Advent and Development of Organocatalysis. *Nature* **2008**, *455*, 304.
2. Reetz, M; List,B.; Jaroch, S. H. *Organocatalysis*, 2nd Ed; G. Stock and M. Lessl, Eds.; Springer: Berlin, 2008.
3. Mukherjee, S.; Yang, J. W.; Hoffmann, S.; List, B. Asymmetric Enamine Catalysis. *Chem. Rev.* **2007**, *107*, 5471.
4. Castro-Alvarez, A.; Carneros, H.; Sánchez, D.; Vilarrasa, J. Importance of the Electron Correlation and Dispersion Corrections in Calculations Involving Enamines, Hemiaminals, and Aminals. Comparison of B3LYP, M06-2X, MP2, and CCSD Results with Experimental Data. *J. Org. Chem.* **2015**, *80*, 11977.
5. Castro-Alvarez, A.; Carneros, H.; Costa, A. M.; Vilarrasa, J. Computer-Aided Insight into the Relative Stability of Enamines. *Synthesis* **2017***, 49*, 5285.
6. Foresman, J., Frisch, Æ. *Exploring Chemistry with Electronic Structure Methods*, 2nd. Ed; Gaussian Inc: Pittsburgh, 1996.
7. Giustino, F. *Materials Modelling using Density Functional Theory;* Oxford University Press: Oxford, 2014.
8. Becke, A. A new mixing of Hartree-Fock and local density-functional theories. *J. Chem. Phys.* **1993***, 98*, 1372.
9. Reyes-Rodríguez, G. J.; Rezayee, N. M.; Vidal-Albalat, A.; Jørgensen, K. A.; Prevalence of Diarylprolinol Silyl Ethers as Catalysts in Total Synthesis and Patents. *Chem. Rev.* **2019***, 119,* 4221.
10. Halland, N.; Hansen, T.; Jørgensen, K. A. Organocatalytic Asymmetric Michael Reaction of Cyclic 1,3-Dicarbonyl Compounds and α,β-Unsaturated Ketones - A Highly Atom-Economic Catalytic One-Step Formation of Optically Active Warfarin Anticoagulant. *Angew. Chem. Int. Ed.* **2003***, 42*, 4955.
11. Dennington, R.; Keith, T.; Millam, J. (2019). GaussView Version 6.
12. Schrödinger, LLC. Maestro. New York, NY.
13. Hanwell D. M.; Curtis E. D.; Lonie C. D; Avogadro: An advanced semantic chemical editor, visualization, and analysis platform. *J. Cheminformatics* **2012,** *4*, 17. Retrieved from http://avogadro.cc/.
14. O'Boyle, N. M.; Bank, M.; James, C. A.; Morley, C.; Vandermeersch, T.; Hutchison, G. R. Open Babel: An open chemical toolbox. *J. Cheminformatics* **2011***, 3*, 33.
15. Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Scuseria, G. E.; Robb, M. A.; Cheeseman, J. R.; Scalmani, G. V.; Barone, G. A; Petersson, H; et al. (2016). Gaussian 16, Revision C.01.

16. Jensen, F. *Introduction to Computational Chemistry*; John Wiley & Sons: Hoboken, 2006.
17. Castro-Alvarez, A.; Carneros, H.; Calafat, J.; Costa, A. M.; Marco, C.; Vilarrasa, J. NMR and Computational Studies on the Reactions of Enamines with Nitroalkenes That May Pass through Cyclobutanes. *ACS Omega* **2019**, *4*, 18167.
18. Python Software Foundation. Python Language Reference, version 3.7. Available at http://www.python.org.

9. ACRONYMS

B3LYP: 3-Parameter hybrid Becke exchange/Lee-Yang-Parr correlation functional

CCSD(T): Coupled Cluster with Single and Double (and perturbative Triple) excitations

CI: Configuration Interaction

GJF: Gaussian Job File

DFT: Density Functional Theory

HF: Hartree–Fock

M06-2X: Hybrid functional of Truhlar and Zhao, Minnesota 2006

MP2: Moller-Plesset 2nd order perturbation

PCM: Polarizable Continuum Model

PDB: Protein Data Bank

PRCG: Polak-Ribiere Conjugate Gradient

SCF: Self-Consistent Field

SCRF: Self-Consistent Reaction Field

APPENDICES

APPENDIX 1: ADDITIONAL *IN VACUO* RESULTS

Molec. Name	***(a)*** *(Ha)*	***(b)*** *(Ha)*	***(c)*** *(Ha)*	***(d)*** *(Ha)*	***(e)*** *(Ha)*	***(f)*** *(Ha)*
1	−193,14534	−192,51365	−193,04740	−193,04740	−193,10903	*G*°: −193,05116
						H°: −193,01805
2	−397,63870	−396,51816	−397,46878	–397,46878	−397,59514	*G*°: −397,53839
						H°: −397,49834
3	−436,95532	−435,68727	−436,76540	–436,76540	−436,90376	*G*°: −436,82050
						H°: −436,77660
4	−476,26710	–474,85068	−476,05613	–476,05613	−476,20512	*G*°: −476,09645
						H°: −476,04793
5	−515,58051	−514,01878	−515,35076	−515,35076	−515,51198	*G*°: −515,37504
						H°: −515,32479
6	−232,45934	−231,68115	−232,34149	–232,34148	−232,41391	*G*°: −232,32979
						H°: −232,29328
en-1	−329,30152	−328,13200	−329,13508	−329,13508	−329,22733	*G*°: −329,06932
						H°: −329,02614
en-2	−533,81305	−532,15001	−533,57270	−533,57270	−533,72807	*G*°: −533,56969
						H°: −533,52050
en-3	−573,11379	−571,30692	−572,85493	−572,85493	−573,02356	*G*°: −572,84061
						H°: −572,78627
en-4	−612,42395	−610,47127	−612,15108	−612,15108	−612,33053	*G*°: −612,11873
						H°: −612,06330
en-5	−651,74089	−649,64594	−651,44159	−651,44159	−651,63742	*G*°: −651,39616
						H°: −651,33999
en-6	−368,61332	−367,29843	−368,42796	−368,42796	−368,53008	*G*°: −368,34563
						H°: −368,29953

(a) 1-DFTmin_opt
(b) 2-MP2min_sp
(c) 3-M062Xmin_opt
(d) 4-M062Xmax_sp
(e) 5-M062Xmax_opt
(f) 6-freq_calc_test

APPENDIX 2: ADDITIONAL IN WATER RESULTS

Molec. Name	***(a)*** *(Ha)*	***(b)*** *(Ha)*	***(c)*** *(Ha)*	***(d)*** *(Ha)*	***(e)*** *(Ha)*	***(f)*** *(Ha)*
1	−193,15041	−192,51842	−193,05255	−193,11536	−193,11545	*G*°: −193,05756
						H°: −193,02455
2	−397,64733	−396,52607	−397,47770	−397,60565	−397,60586	*G*°: −397,54919
						H°: −397,50924
3	−436,96514	−435.69631	−436,77554	−436,91541	−436,91586	*G*° −436,83297
						H°: −436,78879
4	−476,27915	−474,86179	−476,06855	−476,21987	−476,22008	*G*°: −476,11123
						H°: −476,06309
5	−515,59197	−514,02920	−515,36236	−515,52586	−515,52614	*G*°: −515,38952
						H°: −515,33915
6	−232,46423	−231,68575	−232,34644	−232,41999	−232,42007	*G*°: −232,33594
						H°: −232,29954
en-1	−329,30412	−328,13476	−329,13798	−329,23068	−329,23075	*G*°: −329,07309
						H°: −329,02993
en-2	−533,82736	−532,16269	−533,58622	−533,74421	−533,74439	*G*°: −533,58614
						H°: −533,53721
en-3	−573,12358	−571,31585	−572,86510	−573,03513	−573,03540	*G*°: −572,85218
						H°: −572,79844
en-4	−612,43406	−610,48072	−612,15551	−612,33617	−612,34024	*G*°: −612,12945
						H°: −612,07331
en-5	−651,74572	−649,64745	−651,44943	−651,64066	−651,64578	*G*°: −651,40505
						H°: −651,34883
en-6	−368,61548	−367,30070	−368,43045	−368,53296	−368,53305	*G*°: −368,34910
						H°: −368,30293

(a) 1-DFTmin_opt_water
(b) 2-MP2min_sp_water
(c) 3-M062Xmin_opt_water
(d) 4-M062Xmax_sp_water
(e) 5-M062Xmax_opt_water
(f) 6-freq_calc_test

APPENDIX 3: RAW CODE FROM THE SCRIPTS USED

```
import math as m
import numpy as np
import os
import pprint as pp
import subprocess
from zenlog import log


class amine_distance():
    def __init__(self):
        result_dict = self.atom_index_finder()
        dist = self.pyramid_calculator(result_dict)

    def atom_index_finder(self):
        temp_file_list = [file for file in os.listdir() if file.endswith('.xyz')]
        result_dict = {}
        for file in temp_file_list:
            filename = file[:-4]
            result_dict[filename] = {}
            subprocess.call([f'obabel -i xyz {file} -O {filename}.pdb'], shell=True,
stdout=subprocess.DEVNULL,stderr=subprocess.STDOUT)
            with open(f'{filename}.pdb', 'r') as f:
                for line in f:
                    if ' N ' in line and ' O ' not in next(f):
                        nitrog_temp_list = line.strip()
                        nitrog_temp_list = nitrog_temp_list.split()
                        nitrogen_coords = [float(coordinate) for coordinate in
nitrog_temp_list[5:8]]
                        nitrog_posc = nitrog_temp_list[1]

                f.seek(0)

                # Following section could be improved
                bonded_atoms = [line.split()[1:] for line in f if 'CONECT' in line]
                bonded_atoms = [elements for elements in bonded_atoms if elements[0]
== nitrog_posc]

                for element in bonded_atoms:
                    element.remove(nitrog_posc)
                bonded_atoms = bonded_atoms[0]
                f.seek(0)
                atom_coords = []
```

```
                for line in f:
                    if line.split()[0] not in ['END', 'CONECT'] and line.split()[1]
in bonded_atoms:
                        temp_coords = [float(num) for num in line.split()[5:8]]
                        atom_coords.append(temp_coords)

                result_dict[filename]['atom_coords'] = atom_coords
                result_dict[filename]['nitrogen_coords'] = nitrogen_coords

        return result_dict

    def pyramid_calculator(self, result_dict):
        for file in result_dict.items():

            n_coord = np.array(file[1]['nitrogen_coords'])
            at_coord = file[1]['atom_coords']

            v1 = np.array(at_coord[1]) - np.array(at_coord[0])
            v2 = np.array(at_coord[2]) - np.array(at_coord[1])
            v3 = np.array(at_coord[0]) - np.array(at_coord[2])

            # Gives a vector with 3 coord. which correspond to ...
            # ... (a,b,c) in plane equation ax+by+cz+d=0
            abc = np.cross(v1,v2)
            d = -
(abc[0]*np.array(at_coord[0][0])+abc[1]*np.array(at_coord[0][1])+abc[2]*np.array(at_
coord[0][2]))

            # Equation was too long to fit in one line
            distance_num =
(abs((abc[0]*n_coord[0])+((abc[1]*n_coord[1])+((abc[2]*n_coord[2])+d))))
            distance_deno = (m.sqrt((abc[0]**2)+(abc[1]**2)+(abc[2]**2)))
            dist = distance_num/distance_deno

            print()
            log.i(f'Results from file ¥'{file[0]}¥': ')
            log.d(f'Plane equation: {abc[0]}x + {abc[1]}y + {abc[2]}z + {d}')
            log.d(f'Coordinates from surrounding atoms: {at_coord}')
            log.d(f'Nitrogen coordinates: {n_coord}')
            log.i(f'Nitrogen planar distance: {dist} (A)¥n')
            log.w(f'File ¥'{file[0]}¥' done, proceeding with next file if
possible...')
            subprocess.call([f'rm {file[0]}.pdb'], shell=True,
stdout=subprocess.DEVNULL,stderr=subprocess.STDOUT)
        return dist

at = amine_distance()
```

Figure 11. Code for the script "amine_planar_distance.py".

```
import datetime
import os

import numpy as np


# Change the values below to the index of the 4 atoms desired for calculations.
ATOMS = [8, 10, 60, 76]

class dihedral_angle_calculator():

    def __init__(self):
        pos_list = self.get_positions()
        dihedral_list = self.calculate_dihedral(pos_list)
        self.results_to_file(dihedral_list)

    def get_positions(self):
        temp_file_list = [
            file for file in os.listdir() if file.endswith('.xyz')]
        coord_dict = {}
        for file in temp_file_list:
            with open(file, 'r') as f:
                f.readline()
                f.readline()
                text = f.readlines()
                posc_list = [line.split()
                             for line in text if len(line.split()) == 4]
                for atom_number, atom in enumerate(posc_list):
                    atom[:0] = [atom_number]
                atom_calc_list = [at for at in posc_list if at[0] in ATOMS]
                coord_dict[file] = atom_calc_list
        return coord_dict

    def calculate_dihedral(self, coord_dict):
        dihedral_list = []
        for molecule in coord_dict.items():
            molec_name = (molecule[0]).replace('.xyz', '')
            atom_list = molecule[1:]
            for atom_set in atom_list:
                coord_set = []
                for atom_coords in atom_set:
                    atom_coords = [float(num) for num in atom_coords[2:]]
                    coord_set.append(np.array(atom_coords))
                vec1 = coord_set[1] - coord_set[0]
                vec2 = coord_set[2] - coord_set[1]
                vec3 = coord_set[3] - coord_set[2]
                n1 = np.cross(vec1, vec2)
                n2 = np.cross(vec2, vec3)
                uvec2 = vec2/(np.sqrt(np.sum(np.square(vec2))))
                m1 = np.cross(n1, uvec2)
                x = np.dot(n1, n2)
```

```
                y = np.dot(m1, n2)
                dihedral_rads = np.arctan2(y, x)
                dihedral = np.degrees(dihedral_rads)
                dihedral_list.append([molec_name, dihedral])
                print(f'The dihedral angle between the atoms {ATOMS} of '
                      f'{molec_name} is {dihedral:5f}°.')
        return dihedral_list

    def results_to_file(self, dihedral_list):
        time = datetime.datetime.now()
        with open(f'dihedral_results_{time.strftime("%d-%m-%y")}','w+') as f:
            for molecule in dihedral_list:
                line = f'{molecule[0]:30} {round(molecule[1],5):>40}°¥n'
                f.write(line)


at = dihedral_angle_calculator()
```

Figure 12. Code for the script "dihedral_calculator.py".

```
import datetime
import math as m
import os
import pprint as pp
from re import sub

import numpy as np
from prettytable import PrettyTable
from zenlog import log

KB = 0.0019872041 # kcal/mol
TEMP = 293.15 # K

class find_molecule_percentage():

    def __init__(self):
        data_list  = self.get_data()
        min_value, min_names = self.lowest_energy_molecule(data_list)
        boltz_list, boltz_final = self.boltzmann_distr(data_list, min_value)
        self.save_values_in_file(data_list, min_value, min_names, boltz_list,
boltz_final)

    def get_data(self):
        # Gets energies stored in *.txt files

        file_list = [file for file in os.listdir() if file.endswith('.txt')]

        if len(file_list) > 0:

            for current_file in file_list:
                log.i('Reading ¥'{}¥'.'.format(current_file))
                with open(current_file, 'r') as f:
                    text = f.readlines()
                    data_list = []
                    for line in text:
                        line_dict = {}
                        line = line.split()
                        line_dict['name'] = line[0]
                        line_dict['energy'] = float(line[-1])
                        data_list.append(line_dict)

        else:
            log.e('ERROR - File error: Wrong file format (must be .txt) or
¥'boltzmann.py¥' not in same folder as chosen text file. Exiting program...')
            quit()

        print(f'¥nPlease select the appropiate units for the results:¥n¥t(1) Hartree
(Ha)¥n¥t(2) kJ/mol')
        self.user_input = input('Please input (1) or (2) and press ENTER: ')

        if self.user_input == '1':
```

```
            # Orders data list
            ordered_data_list = [None] * (len(data_list)+1)
            for molecule in data_list:
                numer = int(sub("[^0-9]", "", molecule['name']))
                ordered_data_list[numer] = molecule
            data_list = ordered_data_list
            for item in data_list:
                if item is None:
                    data_list.remove(item)

        return data_list

    def lowest_energy_molecule(self, data_list):

        if self.user_input == '1':
            # Converts each energy value from Hartree (Ha) to kcal/mol
            for molecule in data_list:
                molecule['energy'] = molecule['energy'] * 627.51
            min_value = min([molecule['energy'] for molecule in data_list])
            min_names = []
            for molecule in data_list:
                molecule_data = list(molecule.values())
                if min_value in molecule_data:
                    min_names.append(molecule_data[0])
                molecule['energy'] -= min_value

        if self.user_input == '2':
            # Converts each energy value from kJ/mol to kcal/mol
            for molecule in data_list:
                molecule['energy'] = molecule['energy'] / 4.184
            min_value = min([molecule['energy'] for molecule in data_list])
            min_names = []
            for molecule in data_list:
                molecule_data = list(molecule.values())
                if min_value in molecule_data:
                    min_names.append(molecule_data[0])
                molecule['energy'] -= min_value

        if min_names[0] != '':
            log.i('Minimum energy {} was found on molecule(s) {}.'.format(min_value,
min_names))
        else:
            log.i('Minimum energy {} was found.'.format(min_value))

        return min_value, min_names

    def boltzmann_distr(self, data_list, min_value):
        energy_list = [np.float128(molecule['energy']) for molecule in data_list]
        boltz_list = [np.exp((0-en)/(KB*TEMP)) for en in energy_list]

        if self.user_input == '1':
```

```
            # Only leaves one energy minimum in boltz_list
            no1_count = 0
            for count, number in enumerate(boltz_list):
                if number == 1 and no1_count > 0:
                    del boltz_list[count]
                    no1_count += 1
                elif number == 1:
                    no1_count += 1

        boltz_total = sum(boltz_list)
        boltz_final = [round((bol/boltz_total)*100, 4) for bol in boltz_list]
        return boltz_list, boltz_final

    def save_values_in_file(self, data_list, min_value, min_names, boltz_list,
boltz_final):

        time = datetime.datetime.now()
        file_name = ('min_energy_table-{}.out'.format(time.strftime("%d-%m-
%y_%H:%M")))
        x = PrettyTable()
        x.field_names = ['Molecule Name', 'Energy', '% in equilibrium']
        energy_list = [molecule['energy'] for molecule in data_list]
        name_list = [molecule['name'] for molecule in data_list]

        result_dict_list = []
        for name in name_list:
            molecule_dict = {}
            molecule_dict['name'] = name
            result_dict_list.append(molecule_dict)

        count_min = 0
        for molecule in result_dict_list:
            for energy in energy_list:
                molecule['energy'] = energy
                energy_list.remove(energy)
                count_min += 1
                break

        if self.user_input == '1':
            suma_fin = 0
            for molecule in result_dict_list:
                for percent in boltz_final:
                    molecule['percent'] = percent
                    suma_fin += percent
                    boltz_final.remove(percent)
                    break

        elif self.user_input == '2':
            suma_fin = 0
            for molecule in result_dict_list:
                for percent in boltz_final:
```

```
                    molecule['percent'] = percent
                    suma_fin += percent
                    boltz_final.remove(percent)
                    break

        for molecule in result_dict_list:
            x.add_row([molecule['name'], molecule['energy'], molecule['percent']])

        table_title = ('Relative energies in kcal/mol')
        print(x.get_string(title=table_title))
        with open(file_name, 'w+') as f:
            f.write('Minimum energy: {}¥n'.format(min_value))
            f.write(str(x.get_string(title=table_title)))
        print(f'Sum of % in equilibrium: {suma_fin}')
        log.info('Data correctly saved as ¥'{}¥' in ¥'{}¥''.format(file_name,
os.getcwd()))
        return str(x)

aa = find_molecule_percentage()
```

Figure 13. Code for the script "boltzmann2.py".

```
import argparse
import datetime
import getpass
import os
import pprint as pp
import re
import subprocess

import prettytable
from prettytable import PrettyTable
from zenlog import log


DFT_RE = re.compile(r'SCF Done:.*=¥s+([^¥n]+¥d+¥.¥d+)')
M06_RE = re.compile(r'RM062X.*=¥s+([^¥n]+¥d+¥.¥d+)')
FRQ_RE = re.compile(r'Free Energies=¥s+([^¥n]+¥d+¥.¥d+)')
ENT_RE = re.compile(r'Enthalpies=¥s+([^¥n]+¥d+¥.¥d+)')
USER = getpass.getuser()


class energy_data_collector():

    def __init__(self):
        args = self.command_parser()
        energies, folder_names = self.get_energies_from_files()
        self.print_energies_on_file(energies, folder_names, args)

    def command_parser(self):
        parser = argparse.ArgumentParser(description='Calculate energy results.')
        parser.add_argument('-mail', type=str, nargs='?', default=' ', metavar='m',
dest='mail',
                            help='Input mail adress to recieve text results.')
        args = parser.parse_args()
        return args

    def get_energies_from_files(self):
        log.d('Getting energies from files.')
        energies_dict = {}
        folder_names = [folder for folder in next(os.walk('.'))[1] if
folder[0].isnumeric()]
        folder_names.sort()
        for folder_name in folder_names:
            try:
                temp_file_list = [file for file in os.listdir(folder_name) if
file.endswith('.log')]
            except FileNotFoundError:
                continue
            for molecule_name in temp_file_list:
                text = None
                temp_dict = {}
                with open(folder_name+'/'+molecule_name, 'r') as f:
```

```
                    text = f.read()
                try:
                    if 'freq' in folder_name:
                        free_energ = round(float(FRQ_RE.findall(text)[-1]), 5)
                        enthalp = round(float(ENT_RE.findall(text)[-1]), 5)
                        energy_value = 'Free Energy: ' + str(free_energ)
+'¥nEnthalpy: ' + str(enthalp)
                        temp_dict[folder_name] = energy_value
                    elif 'DFT' in folder_name or 'MP2min' in folder_name:
                        energy_value = round(float(DFT_RE.findall(text)[-1]), 5)
                        temp_dict[folder_name] = energy_value
                    elif 'M062X' in folder_name:
                        energy_value = round(float(M06_RE.findall(text)[-1]), 5)
                        temp_dict[folder_name] = energy_value
                    if molecule_name[:-4] not in energies_dict:
                        energies_dict[molecule_name[:-4]] = [temp_dict]
                    else:
                        energies_dict[molecule_name[:-4]].append(temp_dict)
                except IndexError:
                    log.e(f'Error on ¥'{folder_name+"/"+molecule_name}¥'.')
                    temp_dict[folder_name] = 'missing value'

        log.i(f'Energies from {len(energies_dict.keys())} files recovered.')
        return energies_dict, folder_names

    def print_energies_on_file(self, energy_dict, folder_names, args):
        time = datetime.datetime.now()
        file_name = (f'{USER} - {time.strftime("%d-%m-%y")}.txt')
        x = PrettyTable()
        folder_names.insert(0, 'Molecule Name')
        x.field_names = folder_names
        temp_table = list(energy_dict.values())
        temp_table2 = list(energy_dict.keys())
        err_cont = 0
        for j in temp_table:
            for i in temp_table2:
                cont = 0
                lista1 = [i]
                while cont < len(j):
                    try:
                        valor = list(j[cont].values())[0]
                    except IndexError:
                        valor = '    '
                    lista1.append(valor)
                    cont += 1
                try:
                    x.add_row(lista1)
                except Exception:
                    if err_cont == 0:
                        log.w('There is not the same number of files in all the
directories. Some results might be missing from the table...')
```

```
                        err_cont += 1
                    temp_table2.remove(i)
                    break
        x.sortby = 'Molecule Name'
        table_title = (f'{USER} - {time.strftime("%d-%m-%y")} - Gaussian Energy
Calculation Results')
        x.hrules = prettytable.ALL
        print(x.get_string(title=table_title))
        with open(file_name, 'w+') as f:
            f.write(str(x.get_string(title=table_title)))
        image_name = f'{time.strftime("%d-%m-%y")}-molecules.svg'
        subprocess.call([f'obabel -i pdb *.pdb -O {image_name} -d -xC'], shell=True,
stdout=subprocess.DEVNULL,stderr=subprocess.STDOUT)
        log.info(f'Data correctly saved as ¥'{file_name}¥' in ¥'{os.getcwd()}¥'')
        log.info(f'Molecule structure drawn in ¥'{time.strftime("%d-%m-%y")}-
molecules.svg¥' in ¥'{os.getcwd()}¥'')
        if args.mail != '':
            full_path = os.getcwd()
            folder_name = os.path.basename(full_path)
            email = args.mail
            subprocess.call(['echo "Calculations from folder ¥'{}¥' done, see
results attached..." | mail -s "Results - {}" -A "{}" -A "{}"
{}'.format(folder_name, folder_name, file_name, image_name, email)], shell=True,
stdout=subprocess.DEVNULL,stderr=subprocess.STDOUT)
            log.info('Results delivery attempted in ¥'{}¥''.format(email))
        return str(x)

at = energy_data_collector()
```

Figure 14. Code for the script "energy_data_collector.py".

```
import os
from zenlog import log

file_names = [filename for filename in os.listdir() if filename.endswith('.txt')]

for file in file_names:
    filename = file[:-4]
    string_table = []
    with open(file, 'r') as f:
        log.d(f'Reading file: ¥'{file}¥'')
        lines = f.readlines()
        for line in lines:
            line = line.strip().split()
            try:
                difference = abs(float(line[1])-float(line[2]))*627.51
                for index1, value in enumerate(line):
                    if (value.lstrip('+-').replace('.','')).isdigit():
                        line[index1] = str(round(float(value), 5))
                line.append(str(round(difference, 2))+'¥n')
                log.i(f'Energy difference in {line[0]} is {difference:.5}')
                line = '¥t'.join(line)
                string_table.append(line)
            except IndexError:
                log.e(f'File ¥'{file}¥' has incorrect format. Please make sure you
are in the correct directory, or change the file format.')
                quit()

    with open(f'diff_{filename}', 'w') as diff_text:
        for item in string_table:
            diff_text.write(item)

    log.d(f'File ¥'{file}¥' done.')
```

Figure 15. Code for the script "energy_data_collector.py"

APPENDIX 4: SCRIPT OUTPUT

castor - 18-04-20 - Gaussian Energy Calculation Results

Molecule Name	1-DFTmin_opt_water	2-MP2min_sp_water	3-M062Xmin_opt_water	4-M062Xmax_sp_water	5-M062X_opt_water	6-freq_calc_test
2-metilpropanal	-232.46423	-230.99179	-232.34644	-232.41999	-232.42007	Free Energy: -232.33594 Enthalpy: -232.29954
cs_12	-436.96514	-434.46441	-436.77554	-436.91541	-436.91586	Free Energy: -436.83297 Enthalpy: -436.78879
enamina2	-368.61548	-366.09643	-368.43045	-368.53296	-368.53305	Free Energy: -368.3491 Enthalpy: -368.30293

Figure 16. Sample of output from energy_data_collector.py, showing the results in Ha from 3 different example molecules.