

Evaluation of a High-Throughput Workflow with Machine Learning Aided Analysis of Reorganisation Energies in Organic Molecules

Kye Bong Choo
CID: 02150264

Supervisors:
Professor Jenny Nelson,
Dr Mohammed Azzouzzi

BSc Report
Imperial College London

CONTENTS

I	Introduction	2
I-A	Reorganisation Energy	2
I-B	Nelsen's 4-Point Method	3
I-C	Curvilinear Implementation by DUSHIN	3
II	Methodology	3
II-A	Computerised Workflow	3
II-B	Code Walkthrough	4
II-B1	Molecule()	4
II-B2	HPC()	4
II-B3	MongoDB()	4
II-B4	Funnel()	4
II-B5	SMILES_database()	4
II-B6	Tracker()	4
II-B7	Results()	4
II-B8	ReorganisationEnergy()	4
II-C	Optimisation of Geometry: Density Functional Theory	5
II-D	Calculating Energies: Four-Point Calculation	5
II-E	Calculating Energies: DUSHIN	5
II-F	Machine Learning Techniques: Support Vector Regression and Random Forest Regression	5
II-G	Challenges and Complications Throughout the Project	6
II-G1	High levels of symmetry in benzene	6
II-G2	Breaking of bonds during Gaussian geometry optimisation	6
II-G3	Long computational times and queues in the HPC	6
III	Data Analysis and Discussion	6
III-A	Impact of Level of Theory on Results	6
III-B	Discrepancies in the results of DUSHIN and from the 4-Point Calculation	7
III-C	Affects of Conjugation	7
III-D	Implementation of Machine Learning Algorithms	8
III-E	Discussion and Points of Improvements	9
IV	Conclusion	9
References		10
Appendix I: Notes to Appendices		10
Appendix II: List of molecules used for machine learning		11
Appendix III: Top parameter combinations for support vector regression in linear kernel		16
Appendix IV: Top parameter combinations for support vector regression in radial basis function (RBF) kernel		18
Appendix V: Top parameter combinations for support vector regression in sigmoid kernel		20
Appendix VI: Top parameter combinations for random forest regression		22
Appendix VII: Parameters used in neural network		24
Appendix VIII: List of all computed transitions		26
Appendix IX: Complete Code		41

Abstract

An investigation on the feasibility of large-scale analysis of organic molecule properties was conducted. A workflow for the computerised analysis using density functional theory (DFT) methods of the reorganisation energies were developed and tested with a subset of oligoacenes and oligothiophenes. The test molecules were optimised at various level of theory using Gaussian16, along with performing a vibrational mode frequency calculation. The reorganisation energies were then computed using Nelson's four-point method and a curvilinear vibrational mode calculation was performed using DUSHIN. The results were compared to check for accuracy.

Dependence on the functionals used in the DFT calculation is shown to have a significant impact on the computed HOMO-LUMO gap (HL-gap). The use of different basis sets was also seen to have a minor impact on the outcomes of the calculations. However, basic trends were seen to hold between different levels of theory for linear oligoacenes at conjugation between anthracene and hexacene, supporting the idea that the trend analysis would be independent of the level of theory used. The HL-gap and the reorganisation energies were compiled along with the molecular Morgan fingerprint of the corresponding molecules. A strong correlation between the length of conjugation and the reorganisation energies were identified. Several machine learning models were also trained on a group of 126 molecules to predict the HL-gap of the molecules, with the highest r-squared value of 0.41 achieved by support vector machines when tested on a different group of 14 molecules.

Findings show that it is possible for a high-throughput computerised analysis of molecular properties to be conducted. However, whether the resources required and the marginally increasing cost of improving accuracy is a main consideration. This could possibly be mitigated by better methodologies and larger training datasets.

I. INTRODUCTION

WITH the growing importance of the transition to renewable energy for environmental and strategic reasons, the Sun offers a good alternative with the annual flux of energy incident on Earth exceeding annual consumption by an estimated 4 orders of magnitudes [1]. However, the high cost of inorganic photovoltaic cells poses a challenge for the plan to transition towards net-zero [2]. Organic photovoltaic cells have been proposed as an alternative to traditional photovoltaics. With organic molecules' structural flexibility, the task is then finding molecules where efficiency is the highest.

An issue with the *ab initio* calculation of the energy levels of organic molecules is that the computational resources and computational time required are immense. As a reference, the time required to perform a neutral state density functional theory (DFT) calculation for hexamer-level molecules ranges from 10^1 to 10^3 minutes [3]. Therefore, the main motivation of this investigation is to explore the possibility of designing an automated workflow to ease the evaluation and identification of candidate molecules for further investigation and testing.

A. Reorganisation Energy

The efficiency of organic semiconductors could be improved by increasing the transition rate. According to Marcus theory, the transition rate k could be given by Eq 1 [4] [5]

$$k = A \times \exp \left[\frac{-\Delta G^*}{k_B T} \right], \quad (1)$$

where ΔG^* could be expanded as

$$\Delta G^* = \frac{\lambda}{4} \left[1 + \frac{\Delta G^0}{\lambda} \right]^2. \quad (2)$$

In Eq 1 and Eq 2, A is a term dependent on the nature of the electron transfer reaction, ΔG^0 is the standard free energy of reaction (zero for a self-exchange reaction), and λ would be the sum of a solvation and internal component of reorganisation energy denoted by $\lambda = \lambda_{\text{sol}} + \lambda_{\text{int}}$ respectively¹. For

¹Intuitively, the energy required for the solvent and the charge-carrying molecules to change geometry within the reaction cycle.

the purpose of our investigation, we will be investigating the internal reorganisation energy λ_{int} . The internal reorganisation energy could be further broken down into contributions from individual vibrational modes:

$$\lambda_{\text{int}} = \frac{1}{2} \sum_j k_j (Q_j^r - Q_j^p)^2, \quad (3)$$

where Q_j^r and Q_j^p are the equilibrium values for the j -th normal mode coordinate Q , and k_j is a reduced force constant $2k_j k_j^p / k_j^r + k_j^p$, with k_j^r and k_j^p being the force constants for the reactants and products respectively. The reorganisation energy contribution from each vibrational mode could be thought of as energy contained within a spring, where in this case the spring would be the electrostatic potential. This could further be simplified using the local harmonic approximation, which is valid for atomic separations near the optimal separation distances as shown in Fig 1.

From Eq 2, when $\lambda \geq G_0$ (normal regime), the transition rate could be maximised when λ is at its minimum of G_0 .

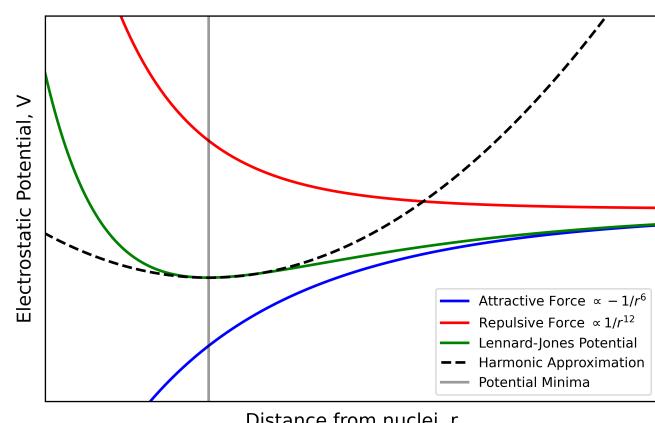


Fig. 1: Illustration of the harmonic approximation used in the derivation of the Marcus rate, valid when the displacement of the individual atoms about the points of equilibria are small where energy is at a minimum.

B. Nelsen's 4-Point Method

A simple way to compute the reorganisation energy is via Nelsen's 4-point method. This involves computing the self-consistent field (SCF) energy for a molecule at the four points shown in Fig 2, and is valid with the assumption that the Gibbs free energy surface is parabolic and have roughly equal curvature on both sides of the minima. [6]. The reorganisation energy λ is then simply given by:

$$\lambda_{\text{int}} = E_{\text{Pt.2}} - E_{\text{Pt.3}} + E_{\text{Pt.4}} - E_{\text{Pt.1}}. \quad (4)$$

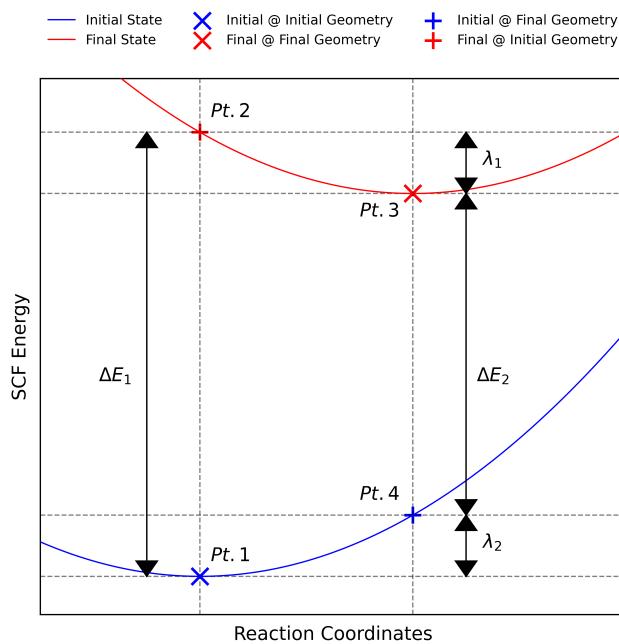


Fig. 2: Illustration of Nelsen's 4-Point method. The two lines 'initial state' and 'final state' would be indicative of the potential energy surface of the molecule at each state. When the molecule is given energy ΔE_1 , it moves from Pt 1 to Pt 2. However since Pt 2 would not be at the molecule's optimised state, it would release energy λ_1 to reach Pt 3. For the molecule to return to its initial state, it passes on energy ΔE_2 to fall to Pt 4, which then again releases energy λ_2 to go back to the initial optimised geometry at Pt 1.

C. Curvilinear Implementation by DUSHIN

An alternate method of calculating the reorganisation energy would be from Eq 3. The difficulty lies with the interpretation of ΔQ_j . For lattice-like inorganic semiconductors, a rectilinear approximation could be used where the vibrational modes and displacements are roughly parallel to the lattice vectors joining each atoms [7]. However, the approximation does not hold for large organic molecules where the carbon atoms are not joined in a lattice-like manner and are not bonded at a right angle. The actual vibrational displacements could be better represented using curvilinear coordinates as in the application DUSHIN [8]. Further discussion regarding the the implementation of DUSHIN could be found in section II-E.

II. METHODOLOGY

This section would serve as an overview of the entire investigation process. Throughout the section, note the large number of calculations, data transfers, and data extractions present which motivates the automation of the process.

A. Computerised Workflow

A computerised workflow could (1) reduce the need for human interaction to allow for an automated high throughput method of evaluating large numbers of molecular structures, and (2) reduce bias in the dataset by minimising the human intervention so that consistent methodology and steps are applied to all datasets.

The workflow starts off with declaring a molecule by generating an RDKit molecule object using the molecule's corresponding Simplified Molecular-Input Line-Entry System (SMILES) string, which reduces the molecular structure into a simple string that could be easily fed into code [9] [10]. By default, RDKit is only able to generate a 2D representation of the coordinates of the molecule, which causes problem when evaluating molecule that are non-planar in nature. By performing Merck Molecular Force Field (MMFF) optimisation, the 2D geometry was broken and a new set of coordinates corresponding to the MMFF optimised geometry was obtained [11] [12]. From our preliminary runs, this new geometry is closer to the actual optimised geometry and will save us considerable time in the upcoming Gaussian16 optimisation step, where we will be using this geometry as input.

Using Gaussian16's built in density functional theory (DFT) methods, the molecular structure was optimised through an iterative process. Further details regarding the DFT geometry optimisation would be included in the section II-C. After the ground state optimisation is completed, the workflow would extract the coordinates and resubmit it for optimisation at the charged state by changing the charge and multiplicity in the submission script. To obtain the excited state geometry, the same process was done but a time-dependent DFT (TD-DFT) was requested instead. The optimisation was staggered to be performed in steps of gradually expanding basis sets, each using the previous' geometry as a starting geometry. At each step, a frequency calculation was also done after the geometry optimisation. Due to the computationally intensive nature of DFT calculations, these tasks were ran in a high-performance-computer (HPC).

The workflow also reads and parse out important data from the Gaussian16 output. One of the primary functions is to check for normal termination and convergence, along with parsing out energy values for the 4-point calculation. If convergence was not achieved within the predetermined wall time set in the HPC, there is an option to resubmit and restart the calculation from the point where it dropped off previously. The parsed output data would then be compiled into a single document, which would then be used as training and testing data for machine learning algorithms.

Finally, the performance of different machine learning algorithms were compared.

B. Code Walkthrough

A 4000-lines long python code was written to facilitate the workflow described in section II-A. The code consists of 8 main classes: Molecule(), HPC(), MongoDB(), SMILES_database(), Funnel(), Tracker(), Results() and ReorganisationEnergy(). Within this section, snippets of code will be included after each description to provide a rough idea of how it could be used in practice. The full code could be found within Appendix IX.

1) *Molecule()*: Class containing methods to declare a molecule, generate information regarding the molecule's constituent atoms, coordinates, and connectivity of the atoms. It can also generate keyword lines and submission scripts for the HPC and Gaussian16.

```
from ReorganisationEnergy import Molecule

molecule_object = Molecule(
    molecule_name='benzene',
    SMILES='C1=CC=CC=C1')
molecule_object.create_molecule_documentation(*args)
molecule_object.generate_molecule_xyz(*args)
molecule_object.get_connectivity(*args)
molecule_object.keyword_line_generator(*args)
molecule_object.generate_gjf_sh_files(*args)
```

2) *HPC()*: Class containing methods to submit job and transfer files to and from the HPC. It can also query the HPC for the queue status and format the string output into a *pandas dataframe* which could be accessed much more easily by logical operations.

```
from ReorganisationEnergy import HPC

hpc = HPC(*args)
hpc.submit_job_gaussian(*args)
hpc.submit_job_dushin(*args)
hpc.transfer_chk_fchk_log_files_from_HPC(*args)
hpc.formatted_qstat()
```

3) *MongoDB()*: The workflow was designed to upload the data to a MongoDB database². This class contains all the methods relating to communicating, uploading, retrieving, and also updating records on the database. The MongoDB() object also write posts (instructions sent to the database) for all these processes. This class would be inherited (passed into) by the proceeding methods.

```
from ReorganisationEnergy import MongoDB

mongo = MongoDB(*args)
mongo.pull()
mongo.push_one_new_entry(*args)
mongo.replace_one_entry(*args)
mongo.update_one_entry(*args)
mongo.export_to_excel()
```

4) *Funnel()*: The funnel would be where the optimisation steps are defined along with the functional used in the DFT calculations. The purpose of declaring a separate method is to make it easier to loop through the entries within the funnel to get the different functional-basis set combinations.

²MongoDB's website could be found hosted here <https://www.mongodb.com/>

5) *SMILES_database()*: The SMILES database keeps track of the molecule names and their corresponding SMILES line (declared by the user) to prevent error during data entry. Methods were defined to pull the database from the online database, to download an excel version of the database, and also to check new entries for errors. This is introduced because SMILES lines are very similar to each other and mistakes could easily be made.

```
from ReorganisationEnergy import SMILES_database

SMILES = SMILES_database(*args)
SMILES.pull_SMILES_database()
SMILES.export_to_excel(*args)
SMILES.check_against_database(*args)
```

6) *Tracker()*: The tracker is a database that tracks the progress of each job queued or waiting to be ran on the HPC. It is hosted online on the MongoDB database but a separate Excel copy is kept locally for ease of reference. This class contains methods to read from, write to, and to download a copy of the tracker.

```
from ReorganisationEnergy import Tracker

tracker = Tracker(*args)
tracker.pull_tracker()
tracker.add_entry(*args)
tracker.update_entry(*args)
tracker.export_to_excel(*args)
```

7) *Results()*: Similar to the tracker, the results is also a MongoDB database but it contains the results of the calculations instead. The class also contains methods to read from, write to, and to download a copy of the results. No example code is given because the usage is similar to the class above.

8) *ReorganisationEnergy()*: Class that wraps up all the above functionalities and also enable automation of the workflow. Some key methods defined within this class is (1) the ability to automatically detect when a job is completed and have the data files transferred over to local storage, (2) the ability to detect if a Gaussian16 calculation is completed successfully, and if so (3) parse out the results and perform the required calculations for the reorganisation energies, and (4) the ability to automate the loop throughout the night³.

```
from ReorganisationEnergy import
    ReorganisationEnergy

ReorgE = ReorganisationEnergy(*args)
ReorgE.batch_add_molecule(*args)
ReorgE.submit_molecule_for_calculation(*args)
ReorgE.restart_molecule_calculation(*args)
ReorgE.transfer_chk_fchk_log_marcus_files_from_HPC()
ReorgE.check_qstat_and_transfer_files()
ReorgE.automate_submission_for_period_of_time(*args)
ReorgE.update_results_all()
```

³The Imperial HPC only allow 50 jobs to be queued at any one time. Therefore if the jobs were queued and ran overnight, the maximum possible jobs that can be completed overnight is 50 jobs (unless the user wakes up midnight to submit new jobs). The advantage of having this automation workflow is that the code could be ran on a laptop locally, and if it detects that a job has been completed it would submit new jobs. Without this system, it would be impractical to perform the over 4000 calculation jobs during the data collection phase for this project.

C. Optimisation of Geometry: Density Functional Theory

The electronic structure of a molecule is a many-bodies problem. Not only does the electrons interact with the nuclei, it also interacts with each other through Coulombic interactions and is also subject to the Pauli exclusion principle. Although it has been theoretically proven that there is an analytical solution for the wavefunction, it has not been done in practice to date. Rather than computing the electronic wavefunction directly, density functional theory (DFT) solves this many-bodies problem by replacing the individual electronic wavefunctions with an electron density ρ , similar to the idea of an electron cloud. DFT is derived from first principles and thus there exists an exact solution, but this exact solution has not been computed so far. The core idea of obtaining the optimal geometric state of a molecule lies with minimising the following equation derived with DFT:

$$E = T_0 + \int v(\mathbf{r})\rho(\mathbf{r})d\mathbf{r} + J[\rho] + E_{xc}[\rho], \quad (5)$$

where E is the total energy of the system, T_0 is the total energy of the fictitious system of non-interacting electrons, $\int v(\mathbf{r})\rho(\mathbf{r})d\mathbf{r}$ is the contribution from the electron nuclei interaction, $J[\rho]$ is the contribution from the electron cloud self-interaction, and finally $E_{xc}[\rho]$ is the correction term known as the exchange-correlation term that would be dependent on the DFT method we use⁴ [13] [14].

In the implementation of Gaussian16's DFT method using the Berny optimisation algorithm [15] [16], an initial Hessian was first predicted and a step was taken much like the Newton-Raphson method. Based on the change in the previous steps, the process was iterated until a local minima of the energies in Eq. 5 was found. This local minima was tested for stability and if it passes the test, the optimisation terminates and assumes this is the lowest energy geometric state of the molecule⁵.

After the geometry optimisation step, the force constants and vibrational frequencies were computed. The force constants were calculated analytically using the differential of the energy, and the vibrations were calculated using the second derivatives of the energy with respect to the Cartesian nuclear coordinates and then transforming to mass-weighted coordinates [15]. Both the force constant and the vibrational frequencies would be used in the calculation of the reorganisation energies using the DUSHIN method in Section II-E.

D. Calculating Energies: Four-Point Calculation

The four-point calculation is relatively simple to perform. A geometry optimisation job was first run for the ground and excited (or charged) states. The workflow would automatically parse out the self-consistent field (SCF) energies from the Gaussian16 output files. The geometry of the ground and excited states were also extracted and another calculation was performed at the excited and ground state respectively. The

⁴The reason why we do not have an exact solution to the equation is because no one has found an exact form of the exchange-correlation term yet.

⁵Gaussian16 could terminate at a local minima.

SCF energies were extracted and then the final reorganisation energy⁶ could be obtained using Eq. 4.

E_{SCF} in Fig 2	Geometry	Evaluation
Point 1	Ground	Ground
Point 2	Ground	Excited
Point 3	Excited	Excited
Point 4	Excited	Ground

TABLE I: Calculations done for the four-point method.

E. Calculating Energies: DUSHIN

For molecules which experience large-amplitude bending and deformation, an evaluation of the reorganisation energies via the vibrational modes under curvilinear coordinates should yield more accurate results [8]. Within DUSHIN, the application manipulates the vibrational modes from being only linear (front-and-back only) to bending motions. The partitioned reorganisation energies in Eq. 3 could then be rewritten as

$$\lambda_{int} = \frac{\hbar}{2} \sum_j v_j \delta_j^2, \quad (6)$$

where v_j are the vibrational frequencies of mode j and δ_j is the curvilinear displacements of the atoms.

To perform a DUSHIN calculation, the optimised geometry has to be fed into the HPC and the output would be a file containing the frequencies and energies of the vibrational modes. These were parsed out and documented for later analysis.

F. Machine Learning Techniques: Support Vector Regression and Random Forest Regression

Due to time limitation, only five machine learning algorithms were tested on a set of 140 molecules, where 90% of the molecules (126 molecules) were randomly chosen to be used as training data and the balance of 14 molecules used as test data. Ideally, far more data than 126 molecules would be required for the training of an algorithm.

The algorithms tried were support vector regressions (SVR) [17], random forest regression (RFR) [18], and neural network [19] [20]. For each molecule, a set of Morgan fingerprints consisting of 2048 bits was generated [21], where each bit would represent a certain feature within the molecule. These fingerprints would then be used as input features where the algorithms would try to extrapolate the HOMO-LUMO gap (predicted output variable)⁷.

As a quick introduction to the algorithms, an SVR tries to draw a decision boundary (hyperplane/surface) through the

⁶Intuitively the reorganisation energy here could be thought of as the energy loss as a result of geometry changes for each round trip in Fig. 2.

⁷There is no specific reason why HOMO-LUMO gap was chosen as the output. Given more time, other possible outputs (predicted variables) that could be investigated are molecular weight, reorganisation energies, and SCF energies.

dataset where the penalisation (variation from data) would be minimised, whereas a RFR generates an ensemble (collection) of decision trees where each tree would consist of nodes (decision criteria) leading to the predicted outcome [22] [23].

A key consideration for machine learning algorithms would be the tuning of its parameters. The parameters would control the complexity of the model (in laymen term: how much you force the model to fit to the training dataset) which would in turn affect the performance of the model. The relationship between the model complexity and performance is not linear and past a certain point there is a trade-off as shown in Fig. 3. In SVR, the key parameters would be the kernel, the regularisation c (penalty), and epsilon ε . The complexity of the RFR model on the other hand is controlled by the number of trees used, depth of trees, and other parameters controlling the splitting conditions of the individual decision trees.

A parameter scan was implemented where the code would try each combination of model parameters and take note of its performance (r^2 -value). The combination of parameters which returns the highest score would be noted down.

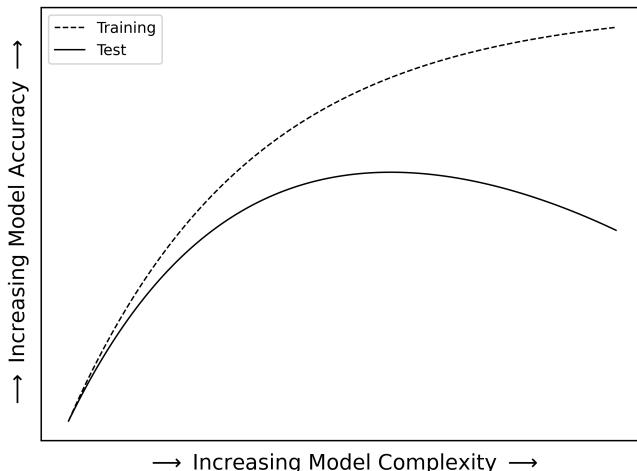


Fig. 3: Illustration of the relationship between the performance of a machine learning model and the complexity of the model. As complexity increases, the model tends to perform better at predicting outcomes. Past a certain point however, further increasing the complexity would lead to over-fitting which would decrease the performance of the model.

G. Challenges and Complications Throughout the Project

Throughout the project, numerous challenges were encountered and various measures were taken to overcome them. This section would outline some of the more notable issues and how it was resolved.

1) *High levels of symmetry in benzene:* Due to the high levels of symmetry in benzene, the DFT optimisation of the molecule does not terminate correctly under larger basis sets. The optimised charged state of benzene should not be symmetrical but instead slightly elongated. However, the optimisation would terminate prematurely due to the symmetrical starting coordinates. To resolve this issue, a small random

displacement was introduced to coordinates of the starting geometry. The optimisation was also tried at different basis sets to validate the results of each other.

2) *Breaking of bonds during Gaussian geometry optimisation:* During Gaussian optimisation, the bonds between atoms could sometimes be 'broken'⁸ due to the program taking too large of an optimisation step. To resolve this issue, the 'trust radius' in the Gaussian program has been reduced from the initial 0.3 Bohr to 0.1 Bohr. However, a new issue that might arise from this change is that by reducing the trust radius, the optimisation might not be able to shoot past a local minima. Therefore the atom connectivity was defined to solve this issue instead. Since it would be impractical and mistake-prone to manually define the connectivity of every atom within a molecule for all the evaluated molecules, a method was written instead to loop through all the molecules and to generate the connectivities.

3) *Long computational times and queues in the HPC:* Due to the fair usage policy implemented by Imperial College's Research Computing Services, after submitting many jobs within a short time frame there is a possibility that the jobs would be 'de-prioritised'. In other words, the jobs submitted would only run after the jobs of other users were completed, which is often in the middle of the night. Additionally with the limit of 50 jobs in the queue per user, this means that only 50 jobs can be ran overnight. To overcome this issue, the automation process was implemented so that jobs could be submitted continuously throughout the night. Additionally, a method was defined to allocate computing time and resources based on the molecular weight of the atoms so that over-allocation of resources could be prevented, which would extend the wait times.

III. DATA ANALYSIS AND DISCUSSION

The full results of the successful calculations could be found in Appendix VIII. The reorganisation energies computed through this method was obtained using DUSHIN. The functional and basis sets used for this set of calculations are the Becke-3 Lee-Young-Parr functional [24] [25], and the 6-311+G(d,p) Pople basis-set [26] [27] [28] [29] [30].

A. Impact of Level of Theory on Results

A main concern with the methodology of the usage of the density functional theory is that the choice of functional and basis sets would have an impact in the results obtained and thus leading to bias in the data analysis including the use of machine learning techniques to identify trends. Therefore an analysis using different combinations of functionals and basis sets was performed on 'linear oligoacenes'⁹ to investigate how the choice of functionals and basis-sets would affect the calculated values of the reorganisation energies¹⁰.

⁸Gaussian16 does not consider actual bonds within its code. The term broken bonds here simply refer to moving an atom too much so that it finds another inappropriate minima.

⁹Benzene, naphthalene, anthracene, tetracene, pentacene, hexacene *et cetera*.

¹⁰The analysis may be flawed in the sense that it might not be able to catch any affects on 'non-linear' oligoacenes or any other groups of organic molecules if there exists any.

Fig. 4 shows the values of the HL-gap computed against the computed DUSHIN reorganisation energies for the linear oligoacenes. It could be seen that there is a systematic displacement between the computed values of all the transitions when using the B3LYP and the CAM-B3LYP functionals, as evidenced by the near-parallel shift in the best-fit line through the data points.

To investigate the effects of different basis-sets on the calculated values, the data was further broken down by basis-set in Fig. 5 [31] [32] [33]. The main takeaway would be (1) that the use of CAM-B3LYP over B3LYP seems to contribute to an increase in the calculated reorganisation energies for the transition to the charged states, while the effects are less systematic for the transitions to the first excited states, and (2) the use of different basis-sets seems to have no impact on the computed reorganisation energies.

To conclude this section, although the use of different functionals would yield different calculated values for the reorganisation energies, it would have little impact on the trends (gradients in Fig. 4). Therefore, for the purpose of the following evaluation, the choice of functionals would be mostly irrelevant as long as the choice is consistent. The choice of basis-sets on the other hand seems to have no systematic effect on the calculated reorganisation energies, as could be seen in Fig 5. For consistency, the evaluation of the larger sets of molecules would be done at the B3LYP and 6-311+G(d,p) level.

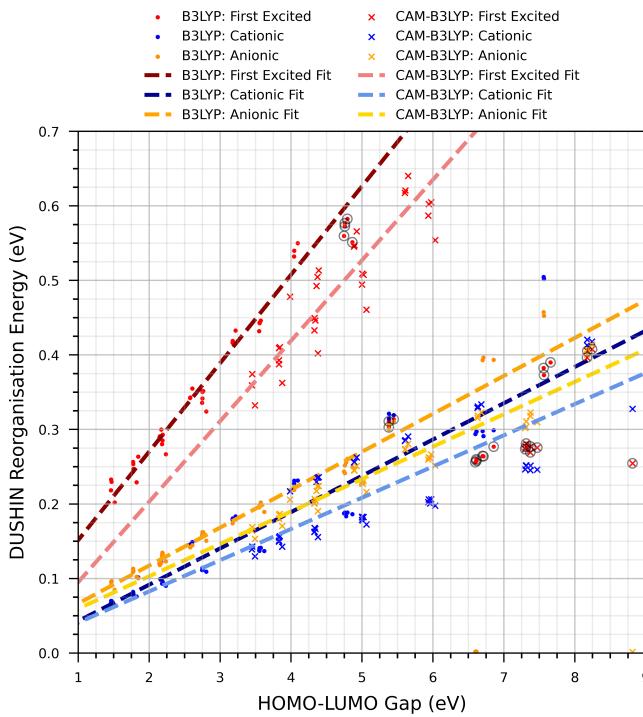


Fig. 4: The dashed lines are the best fit lines through each groups of molecules. The fit was perform with the exclusion of benzene and naphthalene, which could be seen circled in the figure. The use of different functionals seems to cause a parallel shift in the computed values.

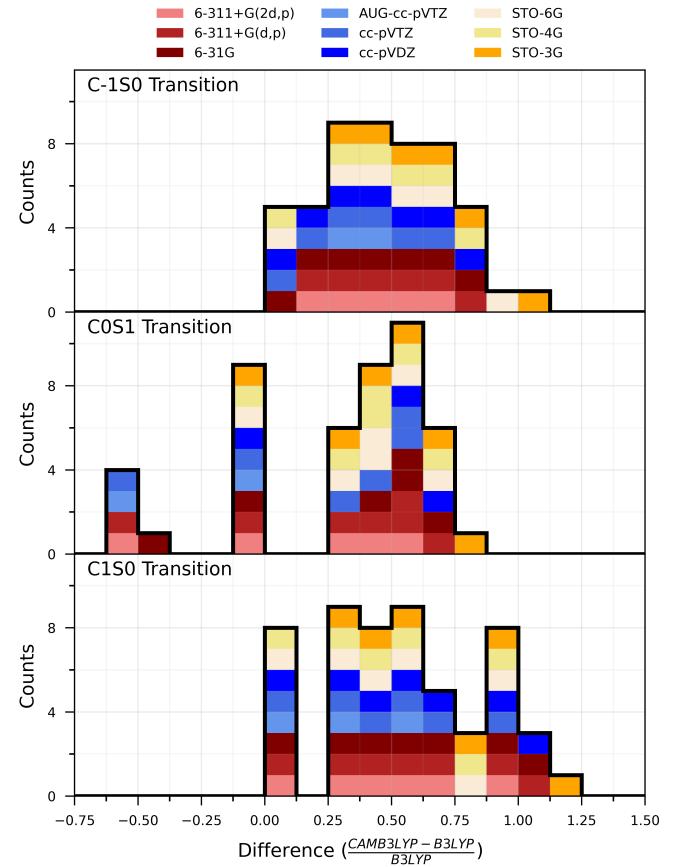


Fig. 5: As seen by the shape of the distribution for the transitions to the charged states (where the mean of the distribution is confidently above 0), the use of CAM-B3LYP would yield higher reorganisation energies compared to the use of the B3LYP functional. The analysis was also color-coded according to basis sets and based on the seemingly scattered distribution of values, there does not seem to be any systematic impact due to the choice of basis-sets.

B. Discrepancies in the results of DUSHIN and from the 4-Point Calculation

As a check on the methodology, the outputs of the values of the reorganisation energies from DUSHIN and from the 4-point calculations were compared in Fig.6. For the transition to first-excited state and cationic state, all results agree to within an accuracy of $\pm 10\%$ of each other. For the Anionic transition, there are 2 outliers where the disagreements is far larger. Both are the calculations for the benzene molecules.

The disagreements are also seemingly randomly distributed around the null and therefore it could be claimed that there is no significant impact (systemic shift of the reorganisation energies) between the choice of whether to use the DUSHIN value or the 4-point value. Thus for consistency we will use the DUSHIN values in the following sections.

C. Affects of Conjugation

For the linear oligoacenes, there seems to be a systematic relationship between the reorganisation energies against the

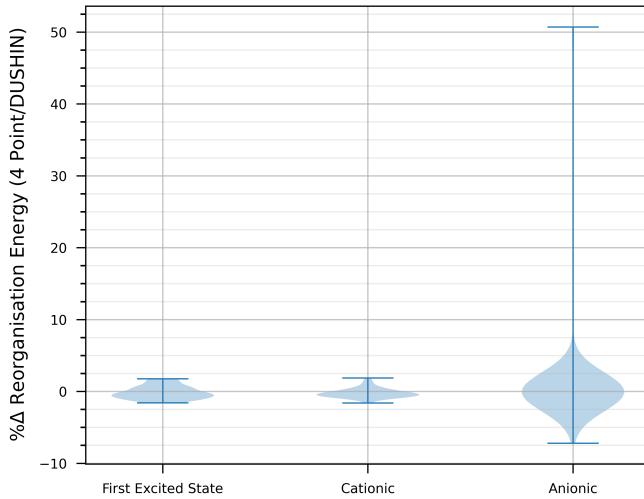


Fig. 6: Illustration of the fractional difference between the DUSHIN reorganisation energies and the 4-point reorganisation energies. Most results agree to within 10% of each other.

HL-gap, with the exception of benzene. In the top pane of Fig. 7, the line of best fit through the linear oligoacenes has a r^2 -value of more than 0.99 when fitting with the relation

$$\lambda_{\text{reorganisation}} = m \cdot E_{\text{HL-gap}} + c + \phi, \quad (7)$$

where m is the constant of proportionality between the HL-gap and the reorganisation energy, c would be a natural vertical displacement since the line does not cross the origin, and ϕ would be the vertical displacement due to the choice of functional (refer to Section III-A).

Another notable observation from Fig. 7 would be that for the measurements of the transitions to the first-excited state, most of the point lies below the best-fit line of the linear-oligoacenes. The reason for this is not clear and it is not known whether this is a coincidence or whether there exists theoretical backing to this observation. Given more time, this could be a significant point to further look into¹¹.

The same relationship was not observed for the transitions to the charged states and the points seems to be evenly scattered around the best-fit line for the linear-oligoacenes.

D. Implementation of Machine Learning Algorithms

A subset of the molecules as outlined in Appendix II was selected test the viability of using machine learning techniques to predict the properties of molecules¹². Several models were trained to predict the HL-gap of the molecules based on information from the molecular fingerprints of the molecules [21]. As mentioned in Section II-F, of the 140 molecules, 126 was chosen to be used as training data and the remaining 14

¹¹In my opinion this is significant because if we were to design organic semiconductors where there is an intermediate transition to the first-excited state before the transition to charged state, then the use of linear oligoacenes would be the most efficient.

¹²A subset was chosen because the complete dataset has gaps in the values where for example, the transition to the excited state was calculated but the transition to the charged states were not.

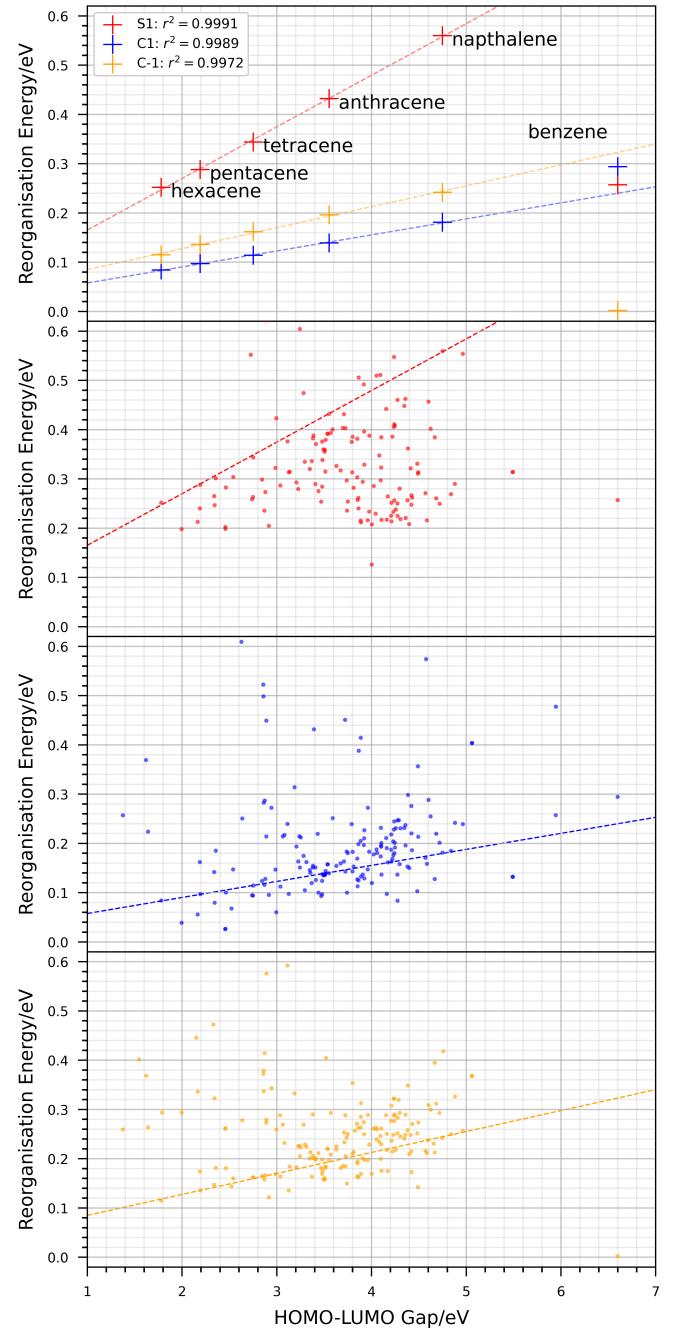


Fig. 7: Summary of the results in Appendix VIII. **Red:** Transitions to first excited-state, **blue:** transitions to cationic states, and **yellow:** transitions to anionic states. The top pane only shows the values for the linear oligoacenes with their best-fit lines along with the r^s -values. The same lines was plotted in the following 3 panes for each of the transitions.

was used as validation data to test for the goodness of the model. The results of each of the models are summarised in Table II, and the summarised results of the trials of each of the combination of parameters are summarised in Appendix III, IV V, VI, and VII.

Of the five different models tested, the support vector

Machine Learning Model Used	Best Test Score (r^2)	σ (Four-Fold)
Support Vector Regression - Linear Kernel	0.538	0.039
Support Vector Regression - RBF Kernel	0.542	0.052
Support Vector Regression - Sigmoid Kernel	0.539	0.039
Random Forest Regression	0.576	0.127
Neural Network	0.313	n/a

TABLE II: Summary table of the best results from each machine learning method.

regression models (SVM) seems to yield the best outcomes with the highest r^2 -test score of around 0.53 – 0.58, with only a four-fold variance of about 0.03 – 0.05. Although the random forest model yields a higher r^2 -score of 0.58, its four-fold variance is also significantly higher at 0.13, suggesting that the model is quite sensitive to the splitting of the dataset into training and validation sets and is prone to over-fit. The neural network has the lowest performance at $r^2 = 0.31$.

The low r^2 score across the models of less than 0.95 suggests that our models are not ready for use. However, it was successfully shown that machine learning models could be used to predict properties of molecules and provide results that are better-than-average (which would correspond to $r^2 = 0$). A comparison to another similar experiment done by Katubi [34] where the best $r^2 = 0.63$ was obtained using a random forest model suggest that the low significance of the results are probably due to the current scientific progress of machine learning models rather than an error in our experiment.

E. Discussion and Points of Improvements

One significant point of improvement would be to train the models on a far larger dataset. The dataset that was used largely consists of only basic variances of oligoacenes-like molecules and sulphur-containing molecules. This covers only quite a small subset of all the possible forms of molecules and the results could be limited.

Another point of improvement would be to incorporate more properties of the molecules in the training (for example, in the above methodology the model ‘sees’ only the molecular fingerprints of the molecules and their corresponding HL-gap). There could be some cross-relationship between different properties and this could help the model learn better, so it could be worth it to also reveal other properties during the training (continuing the above example, let the model ‘see’ the reorganisation energies too when predicting the HL-gap, or *vice versa*).

Another point of improvement would be to design features that better represent the physical properties of the molecules. Molecular fingerprints mostly encodes the connectivity of the atoms within the molecule, but just because some atoms are next to each other doesn’t necessarily mean that they will have similar properties. One feature that I could think of currently would be to represent the electron orbital in a numerical way and label each molecule with it, for example the number of π -

bonds, the bond-orders, and the directions of the bonds. More physical features might help the model learn and predict better.

One last thing that can be done to improve the results would be to run a finer parameter scan on the machine learning models. The scan that was performed was only very shallow due to the computational resources and the time available¹³. By performing a finer grid-search on the SVM or by using a wider/deeper neural network, better results might be obtained.

For the SVM models, we could also extract the feature importances to identify the main contributors to the decision making of the model.

IV. CONCLUSION

Over 4000 Gaussian16 calculations for the transitions to the first-excited, anionic, and cationic state for more than 700 molecules were performed¹⁴. Five different machine learning models was trained and tested on these molecules and the best r^2 -score achieved was 0.58 using the random forest model with a four-fold variance of 0.13. Alternatively, a SVM model using a sigmoid kernalisation would yield an accuracy of $r^2 = 0.54$ with a far lower four-fold variance of 0.04, which could be more resilient to over-fitting.

Although the score is not good enough to claim significance, it has been shown that such methodology could yield better-than-average/random results for which only an r^2 of 0 would be obtained. A key consideration in the pursuit of better models would be the decreasing marginal improvement in results compared to the increasing computational resources required to produce the training data and to train the machine learning models. Additionally, it is also evident that better methods (*i.e.* features) to represent molecules and their physical features are needed to push for better results in machine learning models.

ACKNOWLEDGMENT

I would like to give my gratitude to Professor Jenny Nelson and Doctor Mohammed Azzouzi for supervising my final year project, particularly for providing answers to my sometimes-weird questions, providing the original script from which I based the code that I wrote for this project on, and also for helping me debug my code.

¹³Due to the limitations, the parameter scan for the machine learning models was done on my (CUDA enabled) laptop and thus it was not done exhaustively.

¹⁴The numbers does not add up because the calculations for linear oligoacenes was also performed on 9 different basis sets and 2 functionals so there are many more combinations than 700×3 .

REFERENCES

- [1] M. Z. J. . M. A. Delucchi, "A plan to power 100% of the planet with renewables," pp. 58–65, 2009. [Online]. Available: <https://www.scientificamerican.com/article/a-path-to-sustainable-energy-by-2030/>
- [2] J. Hachmann, R. Olivares-Amaya, S. Atahan-Evrak, C. Amador-Bedolla, R. S. Sanchez-Carrera, A. Gold-Parker, L. Vogt, A. M. Brockway, and A. Aspuru-Guzik, "The harvard clean energy project: Large-scale computational screening and design of organic photovoltaics on the world community grid," *The journal of physical chemistry letters*, vol. 2, no. 17, pp. 2241–2251, 2011.
- [3] O. D. Abarbanel and G. R. Hutchison, "Machine learning to accelerate screening for marcus reorganization energies," *The Journal of chemical physics*, vol. 155, no. 5, pp. 054106–054106, 2021.
- [4] R. A. Marcus, "Electron transfer reactions in chemistry: Theory and experiment," <https://www.nobelprize.org/prizes/chemistry/1992/marcus/lecture/>, Dec 1992.
- [5] ———, "Electron transfer reactions in chemistry. theory and experiment," *Reviews of Modern Physics*, vol. 65, no. 3, p. 599–610, Jul 1993.
- [6] O. López-Estrada, H. G. Laguna, C. Barrueta-Flores, and C. Amador-Bedolla, "Reassessment of the four-point approach to the electron-transfer marcus-hush theory," *ACS Omega*, vol. 3, no. 2, pp. 2130–2140, 2018, pMID: 31458519. [Online]. Available: <https://doi.org/10.1021/acsomega.7b01425>
- [7] Y. Choi, H. Yun, and H. Jeong, "Development of a matlab algorithm for calculating reorganization energy utilizing rectilinear normal mode displacements: Investigation of the effect of substituents on electron and hole reorganization energies of styryl-capped silicon quantum dots," *Bulletin of the Korean Chemical Society (Print)*, vol. 42, no. 3, p. 435–445, Jan 2021.
- [8] J. Reimers, "A practical method for the use of curvilinear coordinates in calculations of normal-mode-projected displacements and duschinsky rotation matrices for large molecules," *The Journal of Chemical Physics*, vol. 115, pp. 9103–9109, 11 2001.
- [9] "RDKit: Open-source cheminformatics," <http://www.rdkit.org>, [Online; accessed 11-April-2013].
- [10] D. Weininger, "Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules," *Journal of Chemical Information and Computer Sciences*, vol. 28, no. 1, pp. 31–36, 1988. [Online]. Available: <https://doi.org/10.1021/ci00057a005>
- [11] T. A. Halgren, "Merck molecular force field. i. basis, form, scope, parameterization, and performance of mmff94," *Journal of Computational Chemistry*, vol. 17, no. 5–6, p. 490–519, Apr 1996.
- [12] ———, "Mmff vi. mmff94s option for energy minimization studies," *Journal of Computational Chemistry*, vol. 20, no. 7, p. 720–729, May 1999.
- [13] L. Pielia, *Ideas of quantum chemistry*. Elsevier, 2014.
- [14] N. M. Harrison, "An introduction to density functional theory."
- [15] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, G. A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. V. Marenich, J. Bloino, B. G. Janesko, R. Gomperts, B. Mennucci, H. P. Hratchian, J. V. Ortiz, A. F. Izmaylov, J. L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V. G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. J. Bearpark, J. J. Heyd, E. N. Brothers, K. N. Kudin, V. N. Staroverov, T. A. Keith, R. Kobayashi, J. Normand, K. Ragahavachari, A. P. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, J. M. Millam, M. Klene, C. Adamo, R. Cammi, J. W. Ochterski, R. L. Martin, K. Morokuma, O. Farkas, J. B. Foresman, and D. J. Fox, "Gaussian®16 Revision C.01," 2016, gaussian Inc. Wallingford CT.
- [16] H. B. Schlegel, "Optimization of equilibrium geometries and transition structures," *Journal of Computational Chemistry*, vol. 3, no. 2, pp. 214–218, 1982. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.540030212>
- [17] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [18] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.
- [19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser,
- M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [20] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
- [21] D. Rogers and M. Hahn, "Extended-connectivity fingerprints," *Journal of Chemical Information and Modeling*, vol. 50, no. 5, pp. 742–754, 2010, pMID: 20426451. [Online]. Available: <https://doi.org/10.1021/ci100050t>
- [22] A. Müller and S. Guido, *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 2016. [Online]. Available: <https://books.google.co.uk/books?id=vbQIDQAAQBAJ>
- [23] A. Burkov, *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019. [Online]. Available: <https://books.google.co.uk/books?id=0jbxwQEACAAJ>
- [24] A. D. Becke, "Density-functional thermochemistry. III. The role of exact exchange," , vol. 98, no. 7, pp. 5648–5652, Apr. 1993.
- [25] C. Lee, W. Yang, and R. G. Parr, "Development of the colle-salvetti correlation-energy formula into a functional of the electron density," *Phys. Rev. B*, vol. 37, pp. 785–789, Jan 1988. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevB.37.785>
- [26] T. Clark, J. Chandrasekhar, G. W. Spitznagel, and P. V. R. Schleyer, "Efficient diffuse function-augmented basis sets for anion calculations. iii. the 3-21+g basis set for first-row elements, li-f," *J. Comput. Chem.*, vol. 4, pp. 294–301, 1983.
- [27] M. M. Franci, W. J. Pietro, W. J. Hehre, J. S. Binkley, M. S. Gordon, D. J. DeFrees, and J. A. Pople, "Self-consistent molecular orbital methods. xxiii. a polarization-type basis set for second-row elements," *J. Chem. Phys.*, vol. 77, pp. 3654–3665, 1982.
- [28] R. Krishnan, J. S. Binkley, R. Seeger, and J. A. Pople, "Self-consistent molecular orbital methods. xx. a basis set for correlated wave functions," *J. Chem. Phys.*, vol. 72, pp. 650–654, 1980.
- [29] A. D. McLean and G. S. Chandler, "Contracted gaussian basis sets for molecular calculations. i. second row atoms, z=11-18," *J. Chem. Phys.*, vol. 72, pp. 5639–5648, 1980.
- [30] G. W. Spitznagel, T. Clark, P. von Ragué Schleyer, and W. J. Hehre, "An evaluation of the performance of diffuse function-augmented basis sets for second row elements, na-cl," *J. Comput. Chem.*, vol. 8, pp. 1109–1116, 1987.
- [31] J. Dunning, Thom H., "Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen," *The Journal of Chemical Physics*, vol. 90, no. 2, pp. 1007–1023, 01 1989. [Online]. Available: <https://doi.org/10.1063/1.456153>
- [32] R. A. Kendall, J. Dunning, Thom H., and R. J. Harrison, "Electron affinities of the first-row atoms revisited. Systematic basis sets and wave functions," *The Journal of Chemical Physics*, vol. 96, no. 9, pp. 6796–6806, 05 1992. [Online]. Available: <https://doi.org/10.1063/1.462569>
- [33] W. J. Hehre, R. F. Stewart, and J. A. Pople, "Self-consistent molecular-orbital methods. i. use of gaussian expansions of Slater-type atomic orbitals," *J. Chem. Phys.*, vol. 51, pp. 2657–2664, 1969.
- [34] K. M. Katubi, M. Saqib, M. Maryam, T. Mubashir, M. H. Tahir, M. Sulaman, Z. Alrowaili, and M. Al-Buriahi, "Machine learning assisted designing of organic semiconductors for organic solar cells: High-throughput screening and reorganization energy prediction," *Inorganic Chemistry Communications*, vol. 151, p. 110610, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1387700323002228>

APPENDIX I
NOTES TO APPENDICES

The following appendices are a only summary of the data that was used in the experiment, intended to give the reader a rough overview of the data. The full data could be found on https://imperiallondon-my.sharepoint.com/:f/g/personal/kbc121_ic_ac_uk/Er2sjJjYjHFCI9KRP_vS9doBCRprdFDQGgWJLrc2DtYEhg?e=xZEjwV and is over 174GB large. Please reach out if you have any difficulty accessing the link.

APPENDIX II
LIST OF MOLECULES USED FOR MACHINE LEARNING

Molecule SMILES	HOMO-LUMO gap (eV)
C1=CC=CC=C1	6.598216937
C1=CC=C2C=CC=CC2=C1	4.747842672
C1CCC2=CC=CC=C2C1	6.192767282
C1CC2=CC=CC=C2C1	4.958730916
C1CC2=C(C=C1)C3=CC=CC=C3C=C2	4.098578996
C1CCC2=C(C1)C=CC3=CC=CC=C23	4.646616315
C1C=CCC2=CC=CC=C21	5.944327325
C1CC2=CC3=CC=CC=C3C=C2C=C1	4.405523433
C1CCC2=CC3=CC=CC=C3C=C2C1	4.702943885
C1=CC=C2C=C3C=CC=CC3=CC2=C1	3.551630133
C1=CC=C2C(=C1)C=CC3=CC=CC=C32	4.68144689
C1=CC=C2C(=C1)C=CC3=C2C=CC4=CC=CC=C43	4.219125437
C1=CC=C2C(=C1)C=CC3=CC4=CC=CC=C4C=C32	3.723878208
C1=CC=C2C(=C1)C=CC3=CC4=C(C=CC5=CC=CC=C54)C=C32	3.849322698
C1=CC=C2C=C3C=C4C=CC=CC4=CC=CC=C2=C1	2.752159605
C1=CC=C2C=C3C4=CC=CC=C4C5=CC=CC=C5C3=CC2=C1	3.849594812
C1=CC=C2C(=C1)C=CC3=C2C4=CC=CC=C4C5=CC=CC=C35	4.051503298
C1=CC=C2C(=C1)C=CC3=C2C4=CC=CC=C4C=C3	4.171505512
C1=CC=C2C(=C1)C=CC3=C2C4=C3C5=CC=CC=C5C=C4	4.209057225
C1=CC=C2C(=C1)C=CC3=CC4=C(C=C32)C5=CC=CC=C5C=C4	3.887418639
C1CC2=CC=CC=C2C3=CC=CC=C31	4.88226692
C1=CC=C2C(=C1)C=CC3=C2C=CC4=CC5=CC=CC=C5C=C43	3.477615162
C1C2=CC=CC=C2C3=CC4=CC=CC=C4C=C31	4.666480627
C1CCC2=CC3=C(CCCC3)C=C2C1	5.681465334
C1C2=CC=CC=C2C3=CC=CC=C31	5.945415781
C1=CC=C2C=C3C(=CC2=C1)C=CC4=C3C=CC5=CC6=CC=CC=C6C=C54	3.126316166
C1CCC2=C(C1)C=CC3=C2CCCC3	5.938612934
C1=CC=C2C(=C1)C=CC3=C2C=CC4=C3C=CC5=CC=CC=C54	4.206880314
C1=CC=C2C=C3C=C4C=C5C=CC=CC5=CC4=CC3=CC2=C1	2.190244479
C1CC2=C(C3=CC=CC=C3C=C2)C4=C1C=CC5=CC=CC=C54	3.923881897
C1CC2=C(C=CC3=CC=CC=C23)C4=CC=CC=C41	4.247425279
C1=CC=C2C=C3C(=CC2=C1)C=CC4=CC5=CC=CC=C5C=C43	3.748640569
C1CCC2=C(C1)C=CC3=C2C=CC4=CC=CC=C43	4.562261018
C1CCC2=C(C1)C=CC3=CC4=C(C=CC5=CC=CC=C54)C=C23	3.710272515
C1=CC=C2C=C3C=C4C=C5C=C6C=CC=CC6=CC5=CC4=CC3=CC2=C1	1.77962466
C1=CC2=C3C(=C1)C=CC4=CC=CC(=C43)C=C2	3.810138302
C1=CC=C2C3=C4C(=CC2=C1)C=CC5=C4C(=CC=C5)C=C3	3.348361078
C1=CC=C2C(=C1)C3=CC=CC4=C3C5=C(C=CC=C25)C=C4	3.958984585
C1=CC=C2C(=C1)C=C3C=CC4=C5C3=C2C6=CC=CC=C6C5=CC=C4	3.381286855
C1=CC2=C3C(=C1)C4=CC=CC5=C4C6=C(C=C5)C=CC(=C36)C=C2	3.481968984
C1=CC=C2C3=C4C(=CC2=C1)C5=CC=CC=C5C6=CC=CC(=C64)C=C3	3.51271785
C1=CC=C2C3=C4C(=CC2=C1)C=CC5=C4C(=CC6=CC=CC=C56)C=C3	2.877604096
C1=CC=C2C3=C4C(=CC2=C1)C=CC5=CC6=CC=CC=C6C(=C54)C=C3	3.224821384
C1=CC2=C3C(=C1)C4=CC=CC5=C4C(=CC=C5)C3=CC=C2	2.985089071
C1=CC=C2C(=C1)C3=CC=CC=C3C4=CC=CC=C24	4.838728702
C1=CC2=C3C(=C1)C=C4C=CC5=C6C4=C3C(=CC6=CC=C5)C=C2	2.867535883
C1=CC2=C3C4=C1C=CC5=C4C6=C(C=C5)C=CC7=C6C3=C(C=C2)C=C7	4.00089012
C1=CC=C2C(=C1)C=C3C4=CC=CC=C4C5=C3C2=CC6=CC=CC=C65	3.241964557
C1=CC2=C3C4=C1C=CC5=CC6=C7C8=C(C=CC9=C8C1=C(C=C9)C=C(C3=C1C7=C54)C=C2)C=C6	2.916244264
C1=CC=C2C(=C1)C=CC3=C2C=C4C=CC=C5C4=C3C=C5	3.238154963
C1=CC=C2C=C3C4=C5C(=CC3=CC2=C1)C=CC6=C5C(=CC=C6)C=C4	2.843045635
C1=CC=C2C3=C4C(=CC=CC4=CC2=C1)C=C3	3.079512581
C1=CC=C2C3=C4C(=CC=C3)C5=CC=CC=C5C6=CC=CC(=C64)C2=C1	4.104837615
C1=CC=C2C(=C1)C3=CC=CC=C3C4=C2C5=CC=CC6=C5C4=CC=C6	3.252304884

Molecule SMILES	HOMO-LUMO gap (eV)
C1=CC=C2C(=C1)C=C3C=C4C=CC=CC4=C5C3=C2C6=CC=CC=C65	2.460997772
C1=CC=C2C(=C1)C=C3C=CC=C4C3=C2C5=CC=CC6=C5C4=CC=C6	2.539094451
C1=CC=C2C(=C1)C3=CC=CC=C3C4=C2C5=CC=CC=C5C6=CC=CC=C64	3.975583531
C1CC2=CC=CC3=C2C4=C(C=CC=C41)C=C3	4.611785741
C1=CC=C2C=C3C4=CC=CC5=C4C6=C(C=CC=C6C3=CC2=C1)C=C5	3.86211205
C1=CC=C2C3=C4C(=CC2=C1)C5=CC=CC=C5C6=CC7=CC=CC=C7C(=C64)C=C3	3.406865558
C1=CC=C3C4=C1C=CC5=C4C6=C(C=CC=C5)C=CC(=C36)C=C2	4.161437299
C1CC2=C3C(=CC=C4C3=C(CCC4)C=C2)C1	4.446068399
CC1=CC2=C3C(=C1)C=CC4=CC(=CC(=C43)C=C2)C	3.789457649
CC1=CC2=C3C(=C1)C=CC4=CC=CC(=C43)C=C2	3.800070089
CC1=C2C=CC3=CC=CC4=C3C2=C(C=C1)C=C4	3.724966663
CC1=CC2=CC=CC=C2C=C1	4.756278202
CC1=CC=CC2=CC=CC=C12	4.655596073
C1=CC2=C3C(=C1)C=CC3=CC=C2	3.886602298
CC1=C2C=CC3=CC=CC=C3C2=C(C4=CC=CC=C14)C	3.535575415
CC1=CC2=CC=CC=C2C3=C1C4=CC=CC=C4C=C3	4.099939566
CC1=CC2=C(C=C1)C=C(C=C2)C	4.746754217
CC1=CC2=CC=CC(=C2C=C1)C	4.665664286
CC(C)C1=CC2=C(C=C1)C=C(C=C2)C(C)C	4.662671033
CC1=C2C=CC3=CC=CC=C3C2=CC4=CC=CC=C14	3.629726811
CC1=C2C(=CC3=CC=CC=C13)C=CC4=CC=CC=C42	3.627277787
CC1=C2C=C3C=CC4=CC=CC=C4C3=CC2=CC=C1	3.696938935
CC(C)C1=CC2=CC=CC=C2C=C1	4.671378677
CC1=CC=C(C2=CC=CC=C12)C	4.555186058
CC1=CC=C(C=C1)S(=O)(=O)O	5.972082939
C1=CC=C2C(=C1)C3=C4C2=CC5=CC=CC6=C5C4=C(C=C6)C=C3	3.31189782
C1=CSC2=CC3=CC4=C(C=C3C=C21)C=C5C(=C4)C=C55	2.73773757
C1=CC2=C(C=CC3=C2C=CC4=C3SC=C4)C5=C1C=CS5	4.358447735
C1=CC=C2C(=C1)C3=CC4=C(C=C3S2)SC5=CC=CC=C54	4.273276096
C1=CC=C2C(=C1)C3=CC4=C(C=C3S2)C5=CC=CC=C5S4	4.002250689
C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C5=CC=CC=C5S4	4.273003982
C1=CSC(=C1)C#CC2=CC=C(C=C2)C#CC3=CC=CS3	4.406067661
C1=CC=C2C=C3C(=CC2=C1)C4=C(C5=C3SC=C5)SC=C4	3.515983217
C1=CC=C(C=C1)C#CC2=CC=C2)C#CC3=CC=CS3	3.606052905
C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)SC5=CC=CC=C54	4.39681579
C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)SC5=CC=CC=C5	4.103204932
C1=CC=C2C(=C1)C3=C(C4=C(C=C3)SC=C4)C5=C2SC=C5	4.23762918
C1=CSC2=CC3=CC4=C(C=C5C=CSC5=C4)C=C3C=C21	2.748077897
C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C5=CC=CC=C5S4	4.022659229
C1=CC2=C3C=CSC4=CC=CC(=C34)C5=C2C(=C1)SC=C5	2.354601252
C1=CC=C2C(=C1)C=C3C=C4C=CC=C4=C5C3=C2SS5	2.338002306
C1=CC2=C(C=CS2)C3=CC4=C(C=C31)C5=C(C=C4)SC=C5	3.535303301
C1=CC2=C(C=C3C=CSC3=C2)C4=C1C=C5C(=C4)C=CS5	3.8846975
C1=CC=C2C(=C1)C=C3=C2SC4=C3SC5=CC=CC=C54	4.081707937
C1=CC=C2C=C3C(=CC2=C1)C4=C(S3)C5=CC=CC=C5S4	3.62891047
C1=CC2=CC3=C(C=CC4=C3SC=C4)C=C2C5=C1C=CS5	3.502921751
C1=CC=C2C(=C1)C=C3=C2C4=C(S3)C5=CC=CC=C5S4	3.921432872
C1=CC2=C(C=CC3=C2C=CS3)C4=C1C5=C(C=C4)SC=C5	4.153545997
C#CC1=CC(=C(C=C1C2=CC=CS2)C#C)C3=CC=CS3	3.863472619
C1=CC=C2C(=C1)C3=C(S2)C=C4C=C5C(=CC4=C3)C=CS5	3.467819063
C1=CC2=C(C=CC3=C2C=CC4=C3C=CS4)C5=C1C=CS5	4.245792596
C1=CC=C2C(=C1)C=C(S2)C#CC3=CC4=CC=CC=C4S3	3.588909732
C1=CC2=CC3=C(C=C2C4=C1C=CS4)C5=C(C=C3)C=CS5	3.506187118
C1=CC=C2C(=C1)SC3=C(S2)C4=CC=CC5=C4C3=CC=C5	3.034885908

Molecule SMILES	HOMO-LUMO gap (eV)
C#CC1=CC(=C(C=C1C2=CSC=C2)C#C)C3=CSC=C3	4.195451531
[2H]C1=CC2=C(C=CS2)C3=CC4=C(C=C13)C5=C(C=C4[2H])SC=C5	3.535303301
[2H]C#CC1=CC(=C(C=C1C2=CSC=C2)C#C[2H])C3=CSC=C3	4.236812839
C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C=CC5=C4SC=C5	4.101844363
C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C=CC5=C4C=CS5	4.092592491
C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C5=C(C=C4)C=CS5	4.209601452
C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C5=C(C=C4)SC=C5	4.240078205
C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C5=C(C=C4)C=CS5	4.058850372
C1=CC=C2C(=C1)C3=C(S2)C=C4C(=C3)C=CC5=C4SC=C5	3.852043837
C1=CC=C2C(=C1)C3=C(S2)C=C4C(=C3)C=CC5=C4C=CS5	3.906738723
C1=CC=C2C(=C1)C3=C4C(=CC=C3)SC5=CC=CC(=C54)S2	4.237357066
[2H]C1=C2C(=CC=C1)C3=C4C(=CC=C3)SC5=CC=CC(=C54)S2	4.236812839
C1=CC=C2C=C3C(=CC2=C1)C4=C(C=CS4)C5=C3SC=C5	3.492037197
C1=CC2=C(C3=CSC=C3C=C2)C4=C1C=CC5=CSC=C54	3.737211787
C1=CC2=C(C3=C(C=C2)SC=C3)C4=C1C=CC5=C4C=CS5	4.239533977
C1=CC2=C3C=CC=C4C3=C(C=CS4)C5=C2C(=C1)SC=C5	3.111622017
C1=CC(=CC(=C1)C#CC2=CC=CS2)C#CC3=CC=CS3	3.906194496
C1=CC2=CC3=C(C=C2C4=CC5=C(C=CS5)C=C41)SC=C3	3.974222961
C1=CC=C2C(=C1)C=CC3=C2C=CC4=C3SC5=C4SC=C5	4.255316581
C1=CC=C2C3=C4C(=CC=C3)SSC5=CC=CC(=C54)C2=C1	3.91354157
C#CC1=C(C=CS1)C2=CC=C(C=C2)C3=C(SC=C3)C#C	4.085789645
C1=CC(=CC=C1C#CC2=CSC=C2)C#CC3=CSC=C3	3.712993653
C1=CC=C2C(=C1)C=CC3=C2C4=C(C=CS4)C5=C3SC=C5	4.096402085
C1=CC=C2C=C3C=C4C(=CC3=CC2=C1)C5=C(S4)C=CS5	3.038151274
C1=CC=C(C=C1)C#CC2=CC3=C(S2)C=CC4=C3C=CS4	4.143477784
C1=CC2=C(C=C3C=CSC3=C2)C4=CC5=C(C=CS5)C=C41	4.041707199
C1=CC2=C(C=CS2)C3=CC4=C(C=C31)C=C5C=CSC5=C4	3.136112265
C#CC1=CC(=C(C=C1C2=CSC=C2)C3=CSC=C3)C#C	4.253411784
C1=CC2=C(C=CS2)C3=CC4=C(C=CC5=C4C=CS5)C=C31	3.569861762
C1=CC2=CSC=C2C3=CC4=C(C=CC5=CSC=C54)C=C31	3.740749267
C1=CC2=CC3=CSC=C3C=C2C4=CC5=CSC=C5C=C41	3.207133983
C1=CC2=CSC=C2C3=CC4=C(C=C31)C5=CSC=C5C=C4	3.672992915
C1=CC2=C(C=CC3=C2C=CC4=CSC=C43)C5=CSC=C51	3.861567822
C1=CC=C(C=C1)C#CC2=CC3=C(S2)C=C4C=CSC4=C3	3.569589648
C1=CC=C2C=C3C(=CC2=C1)C=CC4=C3C5=C(S4)C=CS5	3.38727336
C1=CC2=C3C4C1C5=CSC=C5C4=CC=C3C6=CSC=C26	2.724403991
C1=CC=C2C(=C1)C3=C(S2)C4=CC=CC5=C4C(=CC=C5)S3	3.28087684
C1=CC=C2C(=C1)C3=C(S2)C4=CC=CC=C4S3	4.234363814
C1=CC2=C(C=CC3=C2SC=C3)C4=C1C=CS4	4.420489696
C1=CSC2=CC3=C(C=C21)C=C4C(=C3)C=CS4	3.438430766
C1=CC=C2C3=C(C4=C(C2=C1)C=CS4)SC=C3	4.229737878
C1=CC2=C3C(=CC=C4C3=C1C=CS4)C=CS2	3.062913636
C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)SC=C4	4.569608092
C1=CC=C2C(=C1)C3=C(C4=C2SC=C4)SC=C3	4.345658383
C1=CC=C2C(=C1)C(=S)C3=CC=CC=C3C2=S	2.456916064
C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3SC=C4	4.492871983
C1=CC=C2C(=C1)C=C(S2)C#CC3=CSC=C3	3.961161496
C1=CC=C2C(=C1)C3=C(S2)SC4=CC=CC=C43	4.666752741
C1=CC=C2C(=C1)C=C(S2)C#CC3=CC=CS3	3.720612841
C1=CC=C(C=CS2)C3=C1C4=C(C=C3)SC=C4	4.274092438
C1=CSC2=CC3=C(C=C21)C=C4C=CSC4=C3	3.478703618
C1=CC2=C3C(=CC=C4C3=C1C=CS4)SC=C2	2.889032878
C1=CC2=C(C3=C1C=CC4=C3SC=C4)SC=C2	4.584302241
C1=CC=C2C(=C1)C=CC3=C2C4=C(S3)SC=C4	4.35735928

Molecule SMILES	HOMO-LUMO gap (eV)
C1=C/C(=C\2/C=CC3=C2SC=C3)/C4=C1C=CS4	2.861549378
C1=CSC(=C1)C#C/C=C\C#CC2=CC=CS2	3.231624231
C1=CC=C2C(=C1)C3=C(S2)C=C4C=CSC4=C3	4.159804615
C#CC1=C(SC2=CC=CC=C21)C3=CC=CS3	3.870003352
C1=CC=C2C(=C1)C(=C3C=CC=CC3=C2[S-])[S-]	2.456371836
C1=CC2=C3C(=C1)C4=C(C3=CC=C2)SC=CS4	2.887944422
C1=C/C(=C\2/C=CC3=C2SC=C3)/C4=C1C=CS4	2.942367195
C1=CC=C2C=C3C(=CC2=C1)C4=C(S3)C=CS4	3.797076837
C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C=CS4	4.427564656
C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C=CS4	4.378856275
C1=CC2=C(C=CS2)C3=C1C=CC4=C3C=CS4	4.41232628
C#CC1=C2C(=CC=C1)SC3=CC=CC=C3S2	4.507566132
C\1=CC2=C/C1=C\3\C=CC4=C3C=CS4)C=CS2	2.871073363
C1=CC=C2C(=C1)C3=CSC=C3C4=CSC=C24	4.489334503
C1=CC2=C(C3=C1C=CC4=C3C=CS4)SC=C2	4.305113418
C1=C/C(=C\2/C=CC3=C2C=CS3)/C4=C1C=CS4	2.856923442
C1=CC=C2C(=C1)C3=CC=CC=C3C2=C([S-])[S-]	2.186706999
C\1=CC2=C/C1=C\3/C=CC4=C3C=CS4)C=CS2	2.890121333
C1=C/C(=C\2/C=CC3=C2C=CS3)/C4=C1C=CS4	2.858011897
C1=CC=C2C(=C1)C3=C(S2)C=CC4=CSC=C43	3.70782349
C1=CC=C2C(=C1)C3=C(C=CS3)C4=C2SC=C4	4.421850265
C1=CC=C2C(=C1)C3=C(S2)C4=CSC=CC4=C3	3.187269671
C1=CC=C2C(=C1)C3=CC4=CC=CSC4=C3S2	2.63487853
C#CC1=CC2=C(C=C1)SC3=CC=CC=C3S2	4.458041409
C1=CC2=C(C=CC3=CSC=C32)C4=CSC=C41	3.412307835
C1=CC2=CSC=C2C3=C1C=CC4=CSC=C43	3.367681162
C1=CC2=C(C=C3C=CSC3=C2)C4=C1C=CS4	3.924154011
C1=CSC2=CC3=C(C=C21)SC=C3	4.296405774
C1=CSC2=CC3=C(C=CS3)C=C21	4.48280377
C1=CC2=CSC=C2C3=CSC=C31	4.38348221
C1=CC=C2C(=C1)C3=C(S2)SC=C3	4.963084737
C1=CC2=C(C3=C1C=CS3)SC=C2	4.718182261
C1=CC=C2C(=C1)C3=C(S2)C=CS3	4.623758751
C1=CC2=C(C=CS2)C3=C1C=CS3	4.879545782
C1=CC2=C(C=CS2)C3=C1SC=C3	4.601717528
C1=CC(=S)C=C2C1=CC(=S)C=C2	1.993506156
C1=CC=C2C(=S)C=CC(=S)C2=C1	2.342900356
C=C1C2=C(C3=C1C=CS3)SC=C2	3.518976469
C1=CSC2=CC3=CSC=C3C=C21	3.294482533
C1=CC2=C(C=CS2)C3=CSC=C31	3.918983847
C1=CSC2=C1SC=C2	5.059957272
C1=CSC2=C1C=CS2	5.489625061
C1=CC(=S)C=CC1=S	2.16357732
[2H]C1=C(SC2=C1C(=C(S2)[2H])[2H])[2H]	5.489352947
[2H]C1=C(SC2=C1SC(=C2[2H])[2H])[2H]	5.059957272
C1=C2C=S=CC2=CS1	2.993252487

APPENDIX III

TOP PARAMETER COMBINATIONS FOR SUPPORT VECTOR REGRESSION IN LINEAR KERNEL

C	degree	epsilon	gamma	kernel	mean_test_score	std_test_score	rank_test_score
4	0.001	1	0.001	10 poly	0.538102986	0.038803544	1
100	10	1	0.001	0.001 poly	0.538101962	0.038806698	2
28	0.01	1	0.001	1 poly	0.538092528	0.038826061	3
52	0.1	1	0.001	0.1 poly	0.538088297	0.038811997	4
76	1	1	0.001	0.01 poly	0.538065969	0.038818418	5
33	0.01	1	0.01	1 poly	0.537730712	0.038669018	6
9	0.001	1	0.01	10 poly	0.537727886	0.038657562	7
105	10	1	0.01	0.001 poly	0.537727029	0.038641823	8
57	0.1	1	0.01	0.1 poly	0.537724921	0.038646558	9
81	1	1	0.01	0.01 poly	0.537724372	0.038649399	10
107	10	1	0.01	0.1 poly	0.523993053	0.035573734	11
84	1	1	0.01	10 poly	0.523992037	0.035621555	12
59	0.1	1	0.01	10 poly	0.523983628	0.035628756	13
109	10	1	0.01	10 poly	0.523981791	0.035594149	14
82	1	1	0.01	0.1 poly	0.523977533	0.035577206	15
58	0.1	1	0.01	1 poly	0.523974739	0.035610728	16
34	0.01	1	0.01	10 poly	0.523970603	0.035578494	17
106	10	1	0.01	0.01 poly	0.52397037	0.035578132	18
83	1	1	0.01	1 poly	0.523951663	0.03560998	19
108	10	1	0.01	1 poly	0.523937335	0.035621333	20
104	10	1	0.001	10 poly	0.523406787	0.036621704	21
53	0.1	1	0.001	1 poly	0.523404365	0.036624729	22
102	10	1	0.001	0.1 poly	0.523389467	0.036630605	23
103	10	1	0.001	1 poly	0.523387173	0.036633192	24
78	1	1	0.001	1 poly	0.52338624	0.036632126	25
77	1	1	0.001	0.1 poly	0.523385569	0.03660741	26
101	10	1	0.001	0.01 poly	0.52337311	0.036607209	27
79	1	1	0.001	10 poly	0.52336956	0.036647891	28
29	0.01	1	0.001	10 poly	0.523368849	0.036617751	29
54	0.1	1	0.001	10 poly	0.523363213	0.036636752	30
62	0.1	1	0.1	0.1 poly	0.521979481	0.042126516	31
38	0.01	1	0.1	1 poly	0.521971863	0.042118999	32
86	1	1	0.1	0.01 poly	0.521968049	0.042125874	33
110	10	1	0.1	0.001 poly	0.521954252	0.042153909	34
14	0.001	1	0.1	10 poly	0.521945923	0.042139375	35
114	10	1	0.1	10 poly	0.520029344	0.036738206	36
87	1	1	0.1	0.1 poly	0.520019075	0.036764671	37
64	0.1	1	0.1	10 poly	0.520017118	0.036763379	38

APPENDIX IV

TOP PARAMETER COMBINATIONS FOR SUPPORT VECTOR REGRESSION IN RADIAL BASIS FUNCTION (RBF) KERNEL

C	degree	epsilon	gamma	kernel	mean_test_score	std_test_score	rank_test_score
650	100	3	0.000001	0.0001 rbf	0.542296488	0.052223686	1
659	100	3	0.00001	0.0001 rbf	0.542273108	0.052210344	2
668	100	3	0.0001	0.0001 rbf	0.542251433	0.052194587	3
677	100	3	0.001	0.0001 rbf	0.542170289	0.052116939	4
686	100	3	0.01	0.0001 rbf	0.541184394	0.051463867	5
695	100	3	0.1	0.0001 rbf	0.52930615	0.05009908	6
578	10	3	0.00001	0.0001 rbf	0.503296041	0.044705586	7
569	10	3	0.000001	0.0001 rbf	0.50328497	0.044716367	8
587	10	3	0.0001	0.0001 rbf	0.503276279	0.04470192	9
596	10	3	0.001	0.0001 rbf	0.503221098	0.044749015	10
605	10	3	0.01	0.0001 rbf	0.50242286	0.044998521	11
649	100	3	0.000001	0.00001 rbf	0.501478032	0.03740398	12
658	100	3	0.00001	0.00001 rbf	0.501452294	0.037361334	13
667	100	3	0.0001	0.00001 rbf	0.501439178	0.037371138	14
676	100	3	0.001	0.00001 rbf	0.501353695	0.037282587	15
685	100	3	0.01	0.00001 rbf	0.500302539	0.036765626	16
694	100	3	0.1	0.00001 rbf	0.486821438	0.036309768	17
614	10	3	0.1	0.0001 rbf	0.486570014	0.043870057	18
570	10	3	0.000001	0.001 rbf	0.471309223	0.089111026	19
579	10	3	0.00001	0.001 rbf	0.471308463	0.089113931	20
588	10	3	0.0001	0.001 rbf	0.471278986	0.089131536	21
597	10	3	0.001	0.001 rbf	0.471125753	0.089136976	22
606	10	3	0.01	0.001 rbf	0.469512512	0.089185172	23
651	100	3	0.000001	0.001 rbf	0.469305506	0.089127481	24
660	100	3	0.00001	0.001 rbf	0.4693016	0.089116438	25
669	100	3	0.0001	0.001 rbf	0.469277536	0.089151404	26
678	100	3	0.001	0.001 rbf	0.469144607	0.089144194	27
687	100	3	0.01	0.001 rbf	0.467690351	0.089185882	28
615	10	3	0.1	0.001 rbf	0.448930347	0.086795056	29
696	100	3	0.1	0.001 rbf	0.448547807	0.086784264	30
489	1	3	0.000001	0.001 rbf	0.369632405	0.073851165	31
498	1	3	0.00001	0.001 rbf	0.369611211	0.073844377	32
507	1	3	0.0001	0.001 rbf	0.369594527	0.073851576	33
516	1	3	0.001	0.001 rbf	0.369493936	0.073862818	34
525	1	3	0.01	0.001 rbf	0.36842515	0.073793995	35
534	1	3	0.1	0.001 rbf	0.349958566	0.070071045	36
684	100	3	0.01	0.000001 rbf	0.190083555	0.019917548	37
693	100	3	0.1	0.000001 rbf	0.189515076	0.020542028	38

APPENDIX V

TOP PARAMETER COMBINATIONS FOR SUPPORT VECTOR REGRESSION IN SIGMOID KERNEL

	C	degree	epsilon	gamma	kernel	mean_test_score	std_test_score	rank_test_score
650	100	3	0.000001	0.0001	sigmoid	0.538600208	0.03872462	1
659	100	3	0.00001	0.0001	sigmoid	0.538591516	0.038707963	2
668	100	3	0.0001	0.0001	sigmoid	0.538575909	0.038729899	3
677	100	3	0.001	0.0001	sigmoid	0.53855306	0.038707056	4
686	100	3	0.01	0.0001	sigmoid	0.538196048	0.038543862	5
615	10	3	0.1	0.001	sigmoid	0.528770939	0.04699636	6
695	100	3	0.1	0.0001	sigmoid	0.522513008	0.042012686	7
606	10	3	0.01	0.001	sigmoid	0.517984497	0.067860244	8
597	10	3	0.001	0.001	sigmoid	0.511739739	0.074781991	9
588	10	3	0.0001	0.001	sigmoid	0.51098916	0.075450493	10
570	10	3	0.000001	0.001	sigmoid	0.510948126	0.075511859	11
579	10	3	0.00001	0.001	sigmoid	0.510946066	0.075561381	12
534	1	3	0.1	0.001	sigmoid	0.440638119	0.033929306	13
525	1	3	0.01	0.001	sigmoid	0.436644666	0.042374123	14
516	1	3	0.001	0.001	sigmoid	0.435245867	0.044640298	15
489	1	3	0.000001	0.001	sigmoid	0.435150313	0.044894259	16
498	1	3	0.00001	0.001	sigmoid	0.43513919	0.044874334	17
507	1	3	0.0001	0.001	sigmoid	0.435136984	0.044862614	18
569	10	3	0.000001	0.0001	sigmoid	0.430104348	0.025661095	19
587	10	3	0.0001	0.0001	sigmoid	0.430103248	0.025685008	20
578	10	3	0.00001	0.0001	sigmoid	0.430080235	0.025677133	21
596	10	3	0.001	0.0001	sigmoid	0.429983959	0.025563225	22
649	100	3	0.000001	0.00001	sigmoid	0.42977251	0.025666895	23
658	100	3	0.00001	0.00001	sigmoid	0.429753758	0.025671867	24
667	100	3	0.0001	0.00001	sigmoid	0.429750998	0.025685196	25
676	100	3	0.001	0.00001	sigmoid	0.429655636	0.025580529	26
605	10	3	0.01	0.0001	sigmoid	0.429125804	0.024964509	27
685	100	3	0.01	0.00001	sigmoid	0.428747967	0.024997423	28
614	10	3	0.1	0.0001	sigmoid	0.420453841	0.023970735	29
694	100	3	0.1	0.00001	sigmoid	0.420028758	0.024036584	30
454	0.1	3	0.1	0.01	sigmoid	0.314094062	0.03177342	31
445	0.1	3	0.01	0.01	sigmoid	0.306760701	0.033820887	32
436	0.1	3	0.001	0.01	sigmoid	0.305503874	0.035651853	33
427	0.1	3	0.0001	0.01	sigmoid	0.305126372	0.036171187	34
409	0.1	3	0.000001	0.01	sigmoid	0.305073885	0.036163043	35
418	0.1	3	0.00001	0.01	sigmoid	0.305067346	0.036171978	36
624	10	3	1	0.001	sigmoid	0.221731534	0.046871136	37

APPENDIX VI
TOP PARAMETER COMBINATIONS FOR RANDOM FOREST REGRESSION

	max_depth	max_leaf_nodes	min_samples_leaf	min_samples_split	n_estimators	mean_test_score	std_test_score	rank_test_score
559	19	32	2	2	69	0.576153294	0.12653342	1
625	21	32	2	2	69	0.576153294	0.12653342	1
691	23	32	2	2	69	0.576153294	0.12653342	1
757	25	32	2	2	69	0.576153294	0.12653342	1
823	27	32	2	2	69	0.576153294	0.12653342	1
889	29	32	2	2	69	0.576153294	0.12653342	1
570	19	33	2	2	69	0.576121405	0.126642185	7
636	21	33	2	2	69	0.576121405	0.126642185	7
702	23	33	2	2	69	0.576121405	0.126642185	7
768	25	33	2	2	69	0.576121405	0.126642185	7
834	27	33	2	2	69	0.576121405	0.126642185	7
900	29	33	2	2	69	0.576121405	0.126642185	7
504	17	33	2	2	69	0.576082448	0.126654882	13
493	17	32	2	2	69	0.576050081	0.126472899	14
526	17	35	2	2	69	0.575882761	0.126609757	15
515	17	34	2	2	69	0.575802662	0.12645441	16
581	19	34	2	2	69	0.575788946	0.126451497	17
647	21	34	2	2	69	0.575788946	0.126451497	17
713	23	34	2	2	69	0.575788946	0.126451497	17
779	25	34	2	2	69	0.575788946	0.126451497	17
845	27	34	2	2	69	0.575788946	0.126451497	17
911	29	34	2	2	69	0.575788946	0.126451497	17
592	19	35	2	2	69	0.575768129	0.126564318	23
658	21	35	2	2	69	0.57576661	0.126562421	24
724	23	35	2	2	69	0.57576661	0.126562421	24
790	25	35	2	2	69	0.57576661	0.126562421	24
856	27	35	2	2	69	0.57576661	0.126562421	24
922	29	35	2	2	69	0.57576661	0.126562421	24
558	19	32	2	2	68	0.575747027	0.129084383	29
624	21	32	2	2	68	0.575747027	0.129084383	29
690	23	32	2	2	68	0.575747027	0.129084383	29
756	25	32	2	2	68	0.575747027	0.129084383	29
822	27	32	2	2	68	0.575747027	0.129084383	29
888	29	32	2	2	68	0.575747027	0.129084383	29
427	15	32	2	2	69	0.575725418	0.126103404	35
569	19	33	2	2	68	0.57569573	0.129164972	36
635	21	33	2	2	68	0.57569573	0.129164972	36
701	23	33	2	2	68	0.57569573	0.129164972	36
767	25	33	2	2	68	0.57569573	0.129164972	36
833	27	33	2	2	68	0.57569573	0.129164972	36
899	29	33	2	2	68	0.57569573	0.129164972	36
503	17	33	2	2	68	0.575664487	0.129189771	42
492	17	32	2	2	68	0.57563015	0.129009223	43
560	19	32	2	2	70	0.575617338	0.12452524	44

APPENDIX VII
PARAMETERS USED IN NEURAL NETWORK

```
In [ ]: [TorchObject]: TorchObject initialised with random seed: 5.
[TorchObject]: To change random seed, redeclare TorchObject with TorchObject(random_seed_ = <seed>).
[TorchObject]: CUDA detected, will be using CUDA by default.
[TorchObject]: To change this, redeclare TorchObject with TorchObject(test_for_cuda_ = False).
[TorchObject]: Dataset loaded with:
[TorchObject]: Train data      : 61.54pc (128 counts)
[TorchObject]: Validation data : 23.08pc (48 counts)
[TorchObject]: Test data       : 15.38pc (32 counts)
[TorchObject]: Batch size     : 16
[TorchObject]: Shuffle        : True
[TorchObject]: Loss function initialised: MSELoss()
[TorchObject]: Optimiser initialised.
[TorchObject]: Display Model
Sequential(
  (0): Linear(in_features=2048, out_features=2048, bias=True)
  (1): LeakyReLU(negative_slope=0.01)
  (2): Linear(in_features=2048, out_features=512, bias=True)
  (3): LeakyReLU(negative_slope=0.01)
  (4): Linear(in_features=512, out_features=128, bias=True)
  (5): LeakyReLU(negative_slope=0.01)
  (6): Linear(in_features=128, out_features=32, bias=True)
  (7): LeakyReLU(negative_slope=0.01)
  (8): Linear(in_features=32, out_features=1, bias=True)
)
[TorchObject]: Loss Object
MSELoss()
[TorchObject]: Optimiser Object
SGD (
Parameter Group 0
  dampening: 0
  differentiable: False
  foreach: None
  lr: 0.001
  maximize: False
  momentum: 0.005
  nesterov: False
  weight_decay: 0
)
[TorchObject]: Model Summary by torchinfo.summary
=====
Layer (type:depth-idx)           Input Shape    Output Shape   Param #  Mult-Adds  Trainable
=====
Sequential                      [16, 2048]     [16, 1]        --        --        True
├─Linear: 1-1                   [16, 2048]     [16, 2048]    4,196,352  67,141,632  True
├─LeakyReLU: 1-2                [16, 2048]     [16, 2048]    --        --        --
├─Linear: 1-3                   [16, 2048]     [16, 512]     1,049,088  16,785,408  True
├─LeakyReLU: 1-4                [16, 512]      [16, 512]     --        --        --
├─Linear: 1-5                   [16, 512]      [16, 128]     65,664    1,050,624  True
├─LeakyReLU: 1-6                [16, 128]      [16, 128]     --        --        --
├─Linear: 1-7                   [16, 128]      [16, 32]      4,128    66,048    True
├─LeakyReLU: 1-8                [16, 32]       [16, 32]      --        --        --
└─Linear: 1-9                   [16, 32]       [16, 1]       33        528     True
=====
Total params: 5,315,265
Trainable params: 5,315,265
Non-trainable params: 0
Total mult-adds (M): 85.04
=====
Input size (MB): 0.13
Forward/backward pass size (MB): 0.35
Params size (MB): 21.26
Estimated Total Size (MB): 21.74
=====
```

In []:

APPENDIX VIII
LIST OF ALL COMPUTED TRANSITIONS

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
0	C1=CC=CC=C1	b3lyp/6-311+G(d,p)	First excited	0.2572752	6.598216937
1	C1=CC=C2C=CC=CC2=C1	b3lyp/6-311+G(d,p)	First excited	0.5595376	4.747842672
2	C1CCC2=CC=CC=C2C1	b3lyp/6-311+G(d,p)	First excited	5.741634	6.192767282
3	C1CC2=CC=CC=C2C=C1	b3lyp/6-311+G(d,p)	First excited	37.2958892	4.958730916
4	C1CC2=C(C=C1)C3=CC=CC=C3C=C2	b3lyp/6-311+G(d,p)	First excited	37.533064	4.098578996
5	C1CCC2=C(C1)C=CC3=CC=CC=C23	b3lyp/6-311+G(d,p)	First excited	111.2251852	4.646616315
6	C1C=CCC2=CC=CC=C21	b3lyp/6-311+G(d,p)	First excited	27.9542996	5.944327325
7	C1CC2=CC3=CC=CC=C3C=C2C=C1	b3lyp/6-311+G(d,p)	First excited	4.5231728	4.405523433
8	C1CCC2=CC3=CC=CC=C3C=C2C1	b3lyp/6-311+G(d,p)	First excited	73.11939	4.702943885
9	C1=CC=C2C=C3C=CC=CC3=CC2=C1	b3lyp/6-311+G(d,p)	First excited	0.4320904	3.551630133
10	C1=CC=C2C(=C1)C=CC3=CC=CC=C32	b3lyp/6-311+G(d,p)	First excited	0.2606232	4.68144689
11	C1=CC=C2C(=C1)C=CC3=C2C=CC4=CC=CC=C43	b3lyp/6-311+G(d,p)	First excited	0.3859996	4.219125437
12	C1=CC=C2C(=C1)C=CC3=CC4=CC=CC=C4C=C32	b3lyp/6-311+G(d,p)	First excited	0.4032356	3.723878208
13	C1=CC=C2C(=C1)C=CC3=CC4=C(C=CC5=CC=CC=C54)C=C32	b3lyp/6-311+G(d,p)	First excited	0.3034404	3.849322698
14	C1=CC=C2C=C3C=C4C=CC=CC4=CC3=CC2=C1	b3lyp/6-311+G(d,p)	First excited	0.343666	2.752159605
15	C1=CC=C2C=C3C4=CC=CC=C4C5=CC=CC=C5C3=CC2=C1	b3lyp/6-311+G(d,p)	First excited	0.3816596	3.849594812
16	C1=CC=C2C(=C1)C=CC3=C2C4=CC=CC=C4C5=CC=CC=C35	b3lyp/6-311+G(d,p)	First excited	0.5098136	4.051503298
17	C1=CC=C2C(=C1)C=CC3=C2C4=CC=CC=C4C=C3	b3lyp/6-311+G(d,p)	First excited	0.2646532	4.171505512
18	C1=CC=C2C(=C1)C=CC3=C2C=CC4=C3C5=CC=CC=C5C=C4	b3lyp/6-311+G(d,p)	First excited	0.2517448	4.209057225
19	C1=CC=C2C(=C1)C=CC3=CC4=C(C=C32)C5=CC=CC=C5C=C4	b3lyp/6-311+G(d,p)	First excited	0.2118664	3.887418639
20	C1CC2=CC=CC=C2C3=CC=CC=C31	b3lyp/6-311+G(d,p)	First excited	37.2856592	4.88226692
21	C1=CC=C2C(=C1)C=CC3=C2C=CC4=CC5=CC=CC=C5C=C43	b3lyp/6-311+G(d,p)	First excited	0.3758192	3.477615162
22	C1C2=CC=CC=C2C3=CC4=CC=CC=C4C=C31	b3lyp/6-311+G(d,p)	First excited	0.926714	4.666480627
23	C1CCC2=CC3=C(CCCC3)C=C2C1	b3lyp/6-311+G(d,p)	First excited	74.2574372	5.681465334
24	C1C2=CC=CC=C2C3=CC=CC=C31	b3lyp/6-311+G(d,p)	First excited	27.8148988	5.945415781
25	C1=CC=C2C=C3C(=CC2=C1)C=CC4=C3C=CC5=CC6=CC=CC=C6C=C54	b3lyp/6-311+G(d,p)	First excited	0.313906	3.126316166
26	C1CCC2=C(C1)C=CC3=C2CCCC3	b3lyp/6-311+G(d,p)	First excited	116.747984	5.938612934
27	C1=CC=C2C(=C1)C=CC3=C2C=CC4=C3C=CC5=CC=CC=C54	b3lyp/6-311+G(d,p)	First excited	0.2253328	4.206880314
28	C1=CC=C2C=C3C=C4C=C5C=CC=CC5=CC4=CC3=CC2=C1	b3lyp/6-311+G(d,p)	First excited	0.2881016	2.190244479
29	C1CC2=C(C3=CC=CC=C3C=C2)C4=C1C=CC5=CC=CC=C54	b3lyp/6-311+G(d,p)	First excited	37.3873764	3.923881897
30	C1CC2=C(C=CC3=CC=CC=C23)C4=CC=CC=C41	b3lyp/6-311+G(d,p)	First excited	32.9120676	4.247425279
31	C1=CC=C2C=C3C(=CC2=C1)C=CC4=CC5=CC=CC=C5C=C43	b3lyp/6-311+G(d,p)	First excited	0.2543612	3.748640569
32	C1CCC2=C(C1)C=CC3=C2C=CC4=CC=CC=C43	b3lyp/6-311+G(d,p)	First excited	5.3047076	4.562261018
33	C1CCC2=C(C1)C=CC3=CC4=C(C=CC5=CC=CC=C54)C=C23	b3lyp/6-311+G(d,p)	First excited	41.6004004	3.710272515
34	C1=CC=C2C=C3C=C4C=C5C=C6C=CC=CC6=CC5=CC4=CC3=CC2=C1	b3lyp/6-311+G(d,p)	First excited	0.2518316	1.77962466
35	C1=CC2=C3C(=C1)C=CC4=CC=CC=C(C=C43)C=C2	b3lyp/6-311+G(d,p)	First excited	0.2822488	3.810138302
36	C1=CC=C2C3=C4C(=CC2=C1)C=CC5=C4C(=CC=C5)C=C3	b3lyp/6-311+G(d,p)	First excited	0.3203416	3.348361078
37	C1=CC=C2C(=C1)C3=CC=CC4=C3C5=C(C=CC=C25)C=C4	b3lyp/6-311+G(d,p)	First excited	0.2163552	3.958984585
38	C1=CC=C2C(=C1)C=CC3=CC4=C5C3=C2C6=CC=CC=C6C5=CC=C4	b3lyp/6-311+G(d,p)	First excited	0.382788	3.381286855
39	C1=CC2=C3C(=C1)C4=CC=CC5=C4C6=C(C=C5)C=CC=C36)C=C2	b3lyp/6-311+G(d,p)	First excited	0.3386068	3.481968984
40	C1=CC=C2C3=C4C(=CC2=C1)C5=CC=CC=C5C6=CC=CC=C64)C=C3	b3lyp/6-311+G(d,p)	First excited	0.3151832	3.51271785
41	C1=CC=C2C3=C4C(=CC2=C1)C=CC5=C4C(=CC6=CC=CC=C56)C=C3	b3lyp/6-311+G(d,p)	First excited	0.2733084	2.877604096
42	C1=CC=C2C3=C4C(=CC2=C1)C=CC5=C6C=CC=CC=C6C(=C54)C=C3	b3lyp/6-311+G(d,p)	First excited	0.2797068	3.224821384
43	C1=CC2=C3C(=C1)C4=CC=CC5=C4C(=CC=C5)C3=CC=C2	b3lyp/6-311+G(d,p)	First excited	0.3224372	2.985089071
44	C1=CC=C2C(=C1)C3=CC=CC=C3C4=CC=CC=C24	b3lyp/6-311+G(d,p)	First excited	0.2690924	4.838728702
45	C1=CC2=C3C(=C1)C=C4C=CC5=C6C4=C3C(=CC6=CC=C5)C=C2	b3lyp/6-311+G(d,p)	First excited	0.2358356	2.867535883

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
46	C1=CC2=C3C4=C1C=CC5=C4C6=C(C=C5)C=CC7=C6C3=C(C=C2)C=C7	b3lyp/6-311+G(d,p)	First excited	0.1260088	4.00089012
47	C1=CC=C2C(=C1)C=C3C4=CC=CC=C4C5=C3C2=CC6=CC=CC=C65	b3lyp/6-311+G(d,p)	First excited	0.6044752	3.241964557
48	C1=CC2=C3C4=C1C=CC5=CC6=C7C8=C(C=CC9=C8C1=C(C=C9)C=C(C3=C1C7=C54)C=C2)C=C6	b3lyp/6-311+G(d,p)	First excited	0.2047364	2.916244264
49	C1=CC=C2C(=C1)C=CC3=C2C=C4C=CC=C5C4=C3C=C5	b3lyp/6-311+G(d,p)	First excited	0.921692	3.238154963
50	C1=CC=C2C=C3C4=C5C(=CC3=CC2=C1)C=CC6=C5C(=CC=C6)C=C4	b3lyp/6-311+G(d,p)	First excited	0.2987532	2.843045635
51	C1=CC=C2C3=C4C(=CC=CC4=CC2=C1)C=C3	b3lyp/6-311+G(d,p)	First excited	0.9005748	3.079512581
52	C1=CC=C2C3=C4C(=CC=C3)C5=CC=CC=C5C6=CC=CC(=C64)C2=C1	b3lyp/6-311+G(d,p)	First excited	0.2169256	4.104837615
53	C1=CC=C2C(=C1)C3=CC=CC=C3C4=C2C5=CC=CC6=C5C4=CC=C6	b3lyp/6-311+G(d,p)	First excited	0.6884604	3.252304884
54	C1=CC=C2C(=C1)C=C3C=C4C=CC=C4C=C5C3=C2C6=CC=CC=C65	b3lyp/6-311+G(d,p)	First excited	0.2828688	2.460997772
55	C1=CC=C2C(=C1)C=C3C=CC=C4C3=C2C5=CC=CC6=C5C4=CC=C6	b3lyp/6-311+G(d,p)	First excited	0.3040356	2.539094451
56	C1=CC=C2C(=C1)C3=CC=CC=C3C4=C2C5=CC=CC=C5C6=CC=CC=C64	b3lyp/6-311+G(d,p)	First excited	0.3879588	3.975583531
57	C1CC2=CC=CC3=C2C4=C(C=CC=C41)C=C3	b3lyp/6-311+G(d,p)	First excited	36.0939448	4.611785741
58	C1=CC=C2C=C3C4=CC=CC5=C4C6=C(C=CC=C6C3=CC2=C1)C=C5	b3lyp/6-311+G(d,p)	First excited	0.2402128	3.86211205
59	C1=CC=C2C3=C4C(=CC2=C1)C5=CC=CC=C5C6=CC7=CC=CC=C7C(=C64)C=C3	b3lyp/6-311+G(d,p)	First excited	0.2897012	3.406865558
60	C1=CC2=C3C4=C1C=CC5=C4C6=C(C=C5)C=CC(=C36)C=C2	b3lyp/6-311+G(d,p)	First excited	0.6490036	4.161437299
61	C1CC2=C3C(=CC=C4C3=C(CC4)C=C2)C1	b3lyp/6-311+G(d,p)	First excited	79.3815016	4.446068399
62	CC1=CC2=C3C(=C1)C=CC4=CC(=CC(=C43)C=C2)C	b3lyp/6-311+G(d,p)	First excited	0.3751248	3.789457649
63	CC1=CC2=C3C(=C1)C=CC4=CC=CC(=C43)C=C2	b3lyp/6-311+G(d,p)	First excited	0.385578	3.800070089
64	CC1=C2C=CC3=CC=CC4=C3C2=C(C=C1)C=C4	b3lyp/6-311+G(d,p)	First excited	2.0037904	3.724966663
65	CC1=CC2=CC=CC=C2C=C1	b3lyp/6-311+G(d,p)	First excited	0.7756076	4.756278202
66	CC1=CC=CC2=CC=CC=C12	b3lyp/6-311+G(d,p)	First excited	16.6651288	4.655596073
67	C1=CC2=C3C(=C1)C=CC3=CC=C2	b3lyp/6-311+G(d,p)	First excited	1.0726	3.886602298
68	CC1=C2C=CC3=CC=CC=C3C2=C(C=CC=CC=C14)C	b3lyp/6-311+G(d,p)	First excited	5.5073112	3.535575415
69	CC1=CC2=CC=CC=C2C3=C1C4=CC=CC=C4C=C3	b3lyp/6-311+G(d,p)	First excited	16.8443336	4.099939566
70	CC1=CC2=C(C=C1)C=C(C=C2)C	b3lyp/6-311+G(d,p)	First excited	16.6800708	4.746754217
71	CC1=CC2=CC=CC(=C2C=C1)C	b3lyp/6-311+G(d,p)	First excited	16.9045604	4.665664286
72	CC(C)C1=CC2=C(C=C1)C=C(C=C2)C(C)C	b3lyp/6-311+G(d,p)	First excited	70.5567936	4.662671033
73	CC1=C2C=CC3=CC=CC=C3C2=CC4=CC=CC=C14	b3lyp/6-311+G(d,p)	First excited	1.2319648	3.629726811
74	CC1=C2C(=CC3=CC=C13)C=CC4=CC=CC=C42	b3lyp/6-311+G(d,p)	First excited	17.2317592	3.627277787
75	CC1=C2C=C3C=CC4=CC=CC=C4C3=C2C=CC=C1	b3lyp/6-311+G(d,p)	First excited	0.403496	3.696938935
76	CC(C)C1=CC2=CC=CC=C2C=C1	b3lyp/6-311+G(d,p)	First excited	22.9467208	4.671378677
77	CC1=CC=C(C=CC=C12)C	b3lyp/6-311+G(d,p)	First excited	18.2250612	4.555186058
78	CC1=CC=C(C=C1)S(=O)(=O)O	b3lyp/6-311+G(d,p)	First excited	15.9317184	5.972082939
159	C1=CC=C2C(=C1)C3=C4C2=CC5=CC=CC6=C5C4=C(C=C6)C=C3	b3lyp/6-311+G(d,p)	First excited	0.6843808	3.31189782
241	C1=CSC2=CC3=CC4=C(C=C3C=C21)C=C5C(=C4)C=C55	b3lyp/6-311+G(d,p)	First excited	0.2588624	2.73773757
242	C1=CC2=C(C=CC3=C2C=CC4=C3SC=C4)C5=C1C=CC55	b3lyp/6-311+G(d,p)	First excited	0.2206456	4.358447735
243	C1=CC=C2C(=C1)C3=CC4=C(C=C3S2)SC5=CC=CC=C54	b3lyp/6-311+G(d,p)	First excited	0.224874	4.273276096
244	C1=CC=C2C(=C1)C3=CC4=C(C=C3S2)C5=CC=CC=C554	b3lyp/6-311+G(d,p)	First excited	0.2078984	4.002250689
245	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C5=CC=CC=C554	b3lyp/6-311+G(d,p)	First excited	0.2501948	4.273003982
246	C1=CSC(=C1)C#CC2=CC=C(C=C2)C#CC3=CC=CS3	b3lyp/6-311+G(d,p)	First excited	1.199762	4.406067661
247	C1=CC=C2C=C3C(=CC2=C1)C4=C(C=C5C3SC=C5)SC=C4	b3lyp/6-311+G(d,p)	First excited	0.3795392	3.515983217
248	C1=CC=C(C=C1)C#CC2=CC=C52)C#CC3=CC=CS3	b3lyp/6-311+G(d,p)	First excited	29.1408804	3.606052905
249	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)SC5=CC=CC=C54	b3lyp/6-311+G(d,p)	First excited	0.2084936	4.39681579
250	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C5=C(C=C4)SC=C5	b3lyp/6-311+G(d,p)	First excited	0.3844992	4.103204932
251	C1=CC=C2C(=C1)C3=C(C4=C(C=C3)SC=C4)C5=C2SC=C5	b3lyp/6-311+G(d,p)	First excited	0.5476584	4.23762918
252	C1=CSC2=CC3=CC4=C(C=C5C=CSC5=C4)C=C3=C21	b3lyp/6-311+G(d,p)	First excited	0.2634876	2.748077897

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
253	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C5=CC=CC=C5S4	b3lyp/6-311+G(d,p)	First excited	0.2856464	4.022659229
254	C1=CC2=C3C=CSC4=CC=CC(=C34)C5=C2C(=C1)SC=C5	b3lyp/6-311+G(d,p)	First excited	0.301382	2.354601252
255	C1=CC=C2C(=C1)C=C3C=C4C=CC=CC4=C5C3=C2SS5	b3lyp/6-311+G(d,p)	First excited	0.2653352	2.338002306
256	C1=CC2=C(C=CS2)C3=CC4=C(C=C31)C5=C(C=C4)SC=C5	b3lyp/6-311+G(d,p)	First excited	0.39184	3.535303301
257	C1=CC2=C(C=C3C=CSC3=C2)C4=C1C=C5C(=C4)C=CC5	b3lyp/6-311+G(d,p)	First excited	0.2155244	3.8846975
258	C1=CC=C2C(=C1)C=CC3=C2SC4=C3SC5=CC=CC=C54	b3lyp/6-311+G(d,p)	First excited	0.3477456	4.081707937
259	C1=CC=C2C=C3C(=CC2=C1)C4=C(S3)C5=CC=CC=C5S4	b3lyp/6-311+G(d,p)	First excited	0.3264424	3.62891047
260	C1=CC2=CC3=C(C=CC4=C3SC=C4)C=C2C5=C1C=CC5	b3lyp/6-311+G(d,p)	First excited	0.355508	3.502921751
261	C1=CC=C2C(=C1)C=CC3=C2C4=C(S3)C5=CC=CC=C5S4	b3lyp/6-311+G(d,p)	First excited	0.3962172	3.921432872
262	C1=CC2=C(C=CC3=C2C=CS3)C4=C1C5=C(C=C4)SC=C5	b3lyp/6-311+G(d,p)	First excited	0.44206	4.153545997
263	C#CC1=CC(=C(C=C1C2=CC=CS2)C#C)C3=CC=CS3	b3lyp/6-311+G(d,p)	First excited	0.5060688	3.863472619
264	C1=CC=C2C(=C1)C3=C(S2)C=C4C=C5C(=CC4=C3)C=CC5	b3lyp/6-311+G(d,p)	First excited	0.2540636	3.467819063
265	C1=CC2=C(C=CC3=C2C=C4=C3C=CS4)C5=C1C=CC5	b3lyp/6-311+G(d,p)	First excited	0.4085056	4.245792596
266	C1=CC=C2C(=C1)C=C(S2)C#CC3=CC4=CC=CC=C4S3	b3lyp/6-311+G(d,p)	First excited	0.4005324	3.588909732
267	C1=CC2=CC3=C(C=C2C4=C1C=CC5)C5=C(C=C3)C=CC5	b3lyp/6-311+G(d,p)	First excited	0.35898	3.506187118
268	C1=CC=C2C(=C1)SC3=C(S2)C4=CC=CC5=C4C3=CC=CC5	b3lyp/6-311+G(d,p)	First excited	1.4241896	3.034885908
269	C#CC1=CC(=C(C=C1C2=CSC=C2)C#C)C3=CSC=C3	b3lyp/6-311+G(d,p)	First excited	4.307264	4.195451531
270	[2H]C1=CC2=C(C=CS2)C3=CC4=C(C=C13)C5=C(C=C4[2H])SC=C5	b3lyp/6-311+G(d,p)	First excited	0.3918276	3.535303301
271	[2H]C#CC1=CC(=C(C=C1C2=CSC=C2)C#C[2H])C3=CSC=C3	b3lyp/6-311+G(d,p)	First excited	2.901972	4.236812839
272	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C=CC5=C4SC=C5	b3lyp/6-311+G(d,p)	First excited	0.3231688	4.101844363
273	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C=CC5=C4C=CC5	b3lyp/6-311+G(d,p)	First excited	0.5110908	4.092592491
274	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C5=C(C=C4)C=CC5	b3lyp/6-311+G(d,p)	First excited	0.2139248	4.209601452
275	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C5=C(C=C4)SC=C5	b3lyp/6-311+G(d,p)	First excited	0.4110972	4.240078205
276	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C5=C(C=C4)C=CC5	b3lyp/6-311+G(d,p)	First excited	0.6709888	4.058850372
277	C1=CC=C2C(=C1)C3=C(S2)C=C4C(=C3)C=CC5=C4SC=C5	b3lyp/6-311+G(d,p)	First excited	0.257548	3.852043837
278	C1=CC=C2C(=C1)C3=C(S2)C=C4C(=C3)C=CC5=C4C=CC5	b3lyp/6-311+G(d,p)	First excited	0.2943512	3.906738723
279	C1=CC=C2C(=C1)C3=C4C(=CC=C3)SC5=CC=CC(=C54)S2	b3lyp/6-311+G(d,p)	First excited	1.2734428	4.237357066
280	[2H]C1=C2C(=CC=C1)C3=C4C(=CC=C3)SC5=CC=CC(=C54)S2	b3lyp/6-311+G(d,p)	First excited	1.271496	4.236812839
281	C1=CC=C2C=C3C(=CC2=C1)C4=C(C=C54)C5=C3SC=C5	b3lyp/6-311+G(d,p)	First excited	0.3602696	3.492037197
282	C1=CC2=C(C3=CSC=C3C=C2)C4=C1C=CC5=C5C=C5	b3lyp/6-311+G(d,p)	First excited	0.2969552	3.73211787
283	C1=CC2=C(C3=C(C=C2)SC=C3)C4=C1C=CC5=C4C=CC5	b3lyp/6-311+G(d,p)	First excited	0.2566552	4.239533977
284	C1=CC2=C3C=CC=C4C3=C(C=C54)C5=C2C(=C1)SC=C5	b3lyp/6-311+G(d,p)	First excited	0.3761168	3.111622017
285	C1=CC(=CC(=C1)C#CC2=CC=CS2)C#CC3=CC=C3	b3lyp/6-311+G(d,p)	First excited	2.9436236	3.906194496
286	C1=CC2=CC3=C(C=C2C4=CC5=C(C=CS5)C=C41)SC=C3	b3lyp/6-311+G(d,p)	First excited	0.233678	3.974222961
287	C1=CC=C2C(=C1)C=CC3=C2C=C4=C3SC5=C4SC=C5	b3lyp/6-311+G(d,p)	First excited	0.2380924	4.255316581
288	C1=CC=C2C3=C4C(=CC=C3)SSC5=CC=CC(=C54)C2=C1	b3lyp/6-311+G(d,p)	First excited	2.3272816	3.91354157
289	C#CC1=C(C=CS1)C2=CC=C(C=C2)C3=C(SC=C3)C#C	b3lyp/6-311+G(d,p)	First excited	5.5001812	4.085789645
290	C1=CC(=CC=C1C#CC2=CSC=C2)C#CC3=CSC=C3	b3lyp/6-311+G(d,p)	First excited	5.5536004	3.712993653
291	C1=CC=C2C(=C1)C=CC3=C2C4=C(C=C54)C5=C3SC=C5	b3lyp/6-311+G(d,p)	First excited	0.2971536	4.096402085
292	C1=CC=C2C=C3C=C4C(=CC3=CC2=C1)C5=C(S4)C=CC5	b3lyp/6-311+G(d,p)	First excited	0.2867252	3.038151274
293	C1=CC=C(C=C1)C#CC2=CC3=C(S2)C=CC4=C3C=CS4	b3lyp/6-311+G(d,p)	First excited	1.9651148	4.143477784
294	C1=CC2=C(C=C3C=CSC3=C2)C4=CC5=C(C=CS5)C=C41	b3lyp/6-311+G(d,p)	First excited	0.2294992	4.041707199
295	C1=CC2=C(C=C52)C3=CC4=C(C=C31)C=C5C=CSC5=C4	b3lyp/6-311+G(d,p)	First excited	0.3143772	3.136112265
296	C#CC1=CC(=C(C=C1C2=CSC=C2)C3=CSC=C3)C#C	b3lyp/6-311+G(d,p)	First excited	4.2123172	4.253411784
297	C1=CC2=C(C=CS2)C3=CC4=C(C=C5=C4C=CS5)C=C31	b3lyp/6-311+G(d,p)	First excited	0.3933156	3.569861762
298	C1=CC2=CSC=C2C3=CC4=C(C=C5=CSC=C54)C=C31	b3lyp/6-311+G(d,p)	First excited	0.2357116	3.740749267

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
299	C1=CC2=CC3=CSC=C3C=C2C4=CC5=CSC=C5C=C41	b3lyp/6-311+G(d,p)	First excited	0.2930988	3.207133983
300	C1=CC2=CSC=C2C3=CC4=C(C=C31)C5=CSC=C5C=C4	b3lyp/6-311+G(d,p)	First excited	0.3172168	3.672992915
301	C1=CC2=C(C=CC3=C2C=CC4=CSC=C43)C5=CSC=C51	b3lyp/6-311+G(d,p)	First excited	0.2618012	3.861567822
302	C1=CC=C(C=C1)C#CC2=CC3=C(S2)C=C4C=CSC4=C3	b3lyp/6-311+G(d,p)	First excited	2.8269768	3.569589648
303	C1=CC=C2C=C3C(=CC2=C1)C=C4=C3C5=C(S4)C=CS5	b3lyp/6-311+G(d,p)	First excited	0.3882564	3.38727336
304	C1=CC2=C3C4C1C5=CSC=C5C4=CC=C3C6=CSC=C26	b3lyp/6-311+G(d,p)	First excited	0.551986	2.724403991
305	C1=CC=C2C(=C1)C3=C(S2)C4=CC=CC5=C4C(=CC=C5)S3	b3lyp/6-311+G(d,p)	First excited	0.4743744	3.28087684
422	C1=CC=C2C(=C1)C3=C(S2)C4=CC=CC=C4S3	b3lyp/6-311+G(d,p)	First excited	0.4053932	4.234363814
423	C1=CC2=C(C=CC3=C2SC=C3)C4=C1C=CS4	b3lyp/6-311+G(d,p)	First excited	0.2478636	4.420489696
424	C1=CSC2=CC3=C(C=C21)C=C4C(=C3)C=CS4	b3lyp/6-311+G(d,p)	First excited	0.2754412	3.438430766
425	C1=CC=C2C3=C(C4=C(C2=C1)C=CS4)SC=C3	b3lyp/6-311+G(d,p)	First excited	0.2330828	4.229737878
426	C1=CC2=C3C(=CC=C4C3=C1C=CS4)C=CS2	b3lyp/6-311+G(d,p)	First excited	0.6904072	3.062913636
427	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)SC=C4	b3lyp/6-311+G(d,p)	First excited	0.2710392	4.569608092
428	C1=CC=C2C(=C1)C3=C(C4=C2SC=C4)SC=C3	b3lyp/6-311+G(d,p)	First excited	0.4482848	4.345658383
429	C1=CC=C2C(=C1)(S)C3=CC=CC=C3C2=S	b3lyp/6-311+G(d,p)	First excited	0.1986232	2.456916064
430	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3SC=C4	b3lyp/6-311+G(d,p)	First excited	0.3138316	4.492871983
431	C1=CC=C2C(=C1)C=C(S2)C#CC3=CSC=C3	b3lyp/6-311+G(d,p)	First excited	1.6256648	3.961161496
432	C1=CC=C2C(=C1)C3=C(S2)SC4=CC=CC=C43	b3lyp/6-311+G(d,p)	First excited	0.3846852	4.666752741
433	C1=CC=C2C(=C1)C=C(S2)C#CC3=CC=CS3	b3lyp/6-311+G(d,p)	First excited	2.7037456	3.720612841
434	C1=CC2=C(C=CS2)C3=C1C4=C(C=C3)SC=C4	b3lyp/6-311+G(d,p)	First excited	0.460412	4.274092438
435	C1=CSC2=CC3=C(C=C21)C=C4C=CSC4=C3	b3lyp/6-311+G(d,p)	First excited	0.2845304	3.478703618
436	C1=CC2=C3C(=CC=C4C3=C1C=CS4)SC=C2	b3lyp/6-311+G(d,p)	First excited	0.6218724	2.889032878
437	C1=CC2=C(C3=C1C=CC4=C3SC=C4)SC=C2	b3lyp/6-311+G(d,p)	First excited	0.215822	4.584302241
438	C1=CC=C2C(=C1)C=CC3=C2C4=C(S3)SC=C4	b3lyp/6-311+G(d,p)	First excited	0.4627556	4.35735928
439	C1=C/C(=C\2/C=CC3=C2SC=C3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	First excited	1.088906	2.861549378
440	C1=CSC(=C1)C#C/C=C\CC2=CC=CS2	b3lyp/6-311+G(d,p)	First excited	5.5621688	3.231624231
441	C1=CC=C2C(=C1)C3=C(S2)C=C4=CSC4=C3	b3lyp/6-311+G(d,p)	First excited	0.2171116	4.159804615
442	C#CC1=C(SC2=CC=CC=C21)C3=CC=CS3	b3lyp/6-311+G(d,p)	First excited	5.9720384	3.870003352
443	C1=CC=C2C(=C1)C(=C3C=CC=CC3=C2[S-])[S-]	b3lyp/6-311+G(d,p)	First excited	0.2025912	2.456371836
444	C1=CC2=C3C(=C1)C4=C(C3=CC=C2)SC=CS4	b3lyp/6-311+G(d,p)	First excited	1.3719856	2.887944422
445	C1=C/C(=C\2/C=CC3=C2SC=C3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	First excited	0.8028752	2.942367195
446	C1=CC=C2C=C3C(=CC2=C1)C4=C(S3)C=CS4	b3lyp/6-311+G(d,p)	First excited	0.313472	3.797076837
447	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C=CS4	b3lyp/6-311+G(d,p)	First excited	0.2622724	4.427564656
448	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C=CS4	b3lyp/6-311+G(d,p)	First excited	0.2567668	4.378856275
449	C1=CC2=C(C=CS2)C3=C1C=CC4=C3C=CS4	b3lyp/6-311+G(d,p)	First excited	0.3214328	4.41232628
450	C#CC1=C2C(=CC=C1)SC3=CC=CC=C3S2	b3lyp/6-311+G(d,p)	First excited	2.4114404	4.507566132
451	C\1=CC2=C(/C=1\3/C=CC4=C3C=CS4)C=CS2	b3lyp/6-311+G(d,p)	First excited	0.8097448	2.871073363
452	C1=CC=C2C(=C1)C3=CSC=C3C4=CSC=C24	b3lyp/6-311+G(d,p)	First excited	0.3108804	4.489334503
453	C1=CC2=C(C3=C1C=CC4=C3C=CS4)SC=C2	b3lyp/6-311+G(d,p)	First excited	0.2185128	4.305113418
454	C1=C/C(=C\2/C=CC3=C2C=CS3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	First excited	1.0225784	2.856923442
455	C1=CC=C2C(=C1)C3=CC=CC=C3C2=C([S-])[S-]	b3lyp/6-311+G(d,p)	First excited	0.2403616	2.186706999
456	C\1=CC2=C(/C1=C\3/C=CC4=C3C=CS4)C=CS2	b3lyp/6-311+G(d,p)	First excited	1.2501432	2.890121333
457	C1=C/C(=C\2/C=CC3=C2C=CS3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	First excited	1.054806	2.858011897
458	C1=CC=C2C(=C1)C3=C(S2)C=CC4=CSC=C43	b3lyp/6-311+G(d,p)	First excited	0.4315944	3.70782349
459	C1=CC=C2C(=C1)C3=C(C=CS3)C4=C2SC=CS4	b3lyp/6-311+G(d,p)	First excited	0.2668728	4.421850265
460	C1=CC=C2C(=C1)C3=C(S2)C4=CSC=CC4=C3	b3lyp/6-311+G(d,p)	First excited	0.8989132	3.187269671

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
461	C1=CC=C2C(=C1)C3=CC4=CC=CSC4=C3S2	b3lyp/6-311+G(d,p)	First excited	0.7959808	2.63487853
462	C#CC1=CC2=C(C=C1)SC3=CC=CC=C3S2	b3lyp/6-311+G(d,p)	First excited	2.4903292	4.458041409
463	C1=CC2=C(C=CC3=CSC=C32)C4=CSC=C41	b3lyp/6-311+G(d,p)	First excited	0.370884	3.412307835
464	C1=CC2=CSC=C2C3=C1C=CC4=CSC=C43	b3lyp/6-311+G(d,p)	First excited	0.33604	3.367681162
465	C1=CC2=C(C=C3C=CSC3=C2)C4=C1C=CS4	b3lyp/6-311+G(d,p)	First excited	0.3290712	3.924154011
563	C1=CSC2=CC3=C(C=C21)SC=C3	b3lyp/6-311+G(d,p)	First excited	0.2801036	4.296405774
564	C1=CSC2=CC3=C(C=CS3)C=C21	b3lyp/6-311+G(d,p)	First excited	0.330832	4.48280377
565	C1=CC2=CSC=C2C3=CSC=C31	b3lyp/6-311+G(d,p)	First excited	0.3617204	4.38348221
566	C1=CC=C2C(=C1)C3=C(S2)SC=C3	b3lyp/6-311+G(d,p)	First excited	0.5541932	4.963084737
567	C1=CC2=C(C3=C1C=CS3)SC=C2	b3lyp/6-311+G(d,p)	First excited	0.2497236	4.718182261
568	C1=CC=C2C(=C1)C3=C(S2)C=CS3	b3lyp/6-311+G(d,p)	First excited	0.4020204	4.623758751
569	C1=CC2=C(C=CS2)C3=C1C=CS3	b3lyp/6-311+G(d,p)	First excited	0.2899492	4.879545782
570	C1=CC2=C(C=CS2)C3=C1SC=C3	b3lyp/6-311+G(d,p)	First excited	0.456816	4.601717528
571	C1=CC(=S)C=C2C1=CC(=S)C=C2	b3lyp/6-311+G(d,p)	First excited	0.1983876	1.993506156
572	C1=CC=C2C(=S)C=CC(=S)C2=C1	b3lyp/6-311+G(d,p)	First excited	0.2466484	2.342900356
573	C=C1C2=C(C3=C1C=CS3)SC=C2	b3lyp/6-311+G(d,p)	First excited	20.8524228	3.518976469
574	C1=CSC2=CC3=CSC=C3C=C21	b3lyp/6-311+G(d,p)	First excited	0.3349116	3.294482533
575	C1=CC2=C(C=CS2)C3=CSC=C31	b3lyp/6-311+G(d,p)	First excited	0.4920072	3.918983847
609	C1=CSC2=C1SC=C2	b3lyp/6-311+G(d,p)	First excited	0.623596	5.059957272
610	C1=CSC2=C1C=CS2	b3lyp/6-311+G(d,p)	First excited	0.3141044	5.489625061
611	C1=CC(=S)C=CC1=S	b3lyp/6-311+G(d,p)	First excited	0.2129452	2.16357732
612	[2H]C1=C(SC2=C1C(=C(S2)[2H])[2H])[2H]	b3lyp/6-311+G(d,p)	First excited	0.3141168	5.489352947
613	[2H]C1=C(SC2=C1SC(=C2[2H])[2H])[2H]	b3lyp/6-311+G(d,p)	First excited	0.6236208	5.059957272
614	C1=C2C=S=CC2=CS1	b3lyp/6-311+G(d,p)	First excited	0.4233112	2.993252487
79	C1=CC=CC=C1	b3lyp/6-311+G(d,p)	Cationic	0.2944628	6.598216937
80	C1=CC=C2C=CC=CC2=C1	b3lyp/6-311+G(d,p)	Cationic	0.1813996	4.747842672
81	C1CCC2=CC=CC=C2C1	b3lyp/6-311+G(d,p)	Cationic	36.96502	6.088003445
82	C1CC2=CC=CC=C2C=C1	b3lyp/6-311+G(d,p)	Cationic	35.3355732	4.958730916
83	C1CC2=C(C=C1)C3=CC=CC=C3C=C2	b3lyp/6-311+G(d,p)	Cationic	0.231942	4.098578996
84	C1CCC2=C(C1)C=CC3=CC=CC=C23	b3lyp/6-311+G(d,p)	Cationic	6.2023312	4.646616315
85	C1C=CCC2=CC=CC=C21	b3lyp/6-311+G(d,p)	Cationic	0.2573992	5.944327325
86	C1CC2=CC3=CC=CC=C3C=C2C=C1	b3lyp/6-311+G(d,p)	Cationic	31.5613852	4.405523433
87	C1CCC2=CC3=CC=CC=C3C=C2C1	b3lyp/6-311+G(d,p)	Cationic	78.3852236	4.593826226
88	C1=CC=C2C=C3C=CC=CC3=CC2=C1	b3lyp/6-311+G(d,p)	Cationic	0.1396364	3.551630133
89	C1=CC=C2C(=C1)C=CC3=CC=CC=C32	b3lyp/6-311+G(d,p)	Cationic	0.2199388	4.68144689
90	C1=CC=C2C(=C1)C=CC3=C2C=CC4=CC=C43	b3lyp/6-311+G(d,p)	Cationic	0.1686524	4.219125437
91	C1=CC=C2C(=C1)C=CC3=CC4=CC=CC=C4C=C32	b3lyp/6-311+G(d,p)	Cationic	0.1432324	3.723878208
92	C1=CC=C2C(=C1)C=CC3=CC4=C(C=CC5=CC=CC=C54)C=C32	b3lyp/6-311+G(d,p)	Cationic	0.1692848	3.849322698
93	C1=CC=C2C=C3C=C4C=CC=CC4=CC3=CC2=C1	b3lyp/6-311+G(d,p)	Cationic	0.1144892	2.751887491
94	C1=CC=C2C=C3C4=CC=CC=C4C5=CC=CC=C5C3=CC2=C1	b3lyp/6-311+G(d,p)	Cationic	0.1272364	3.849594812
95	C1=CC=C2C(=C1)C=CC3=C2C4=CC=CC=C4C5=CC=CC=C35	b3lyp/6-311+G(d,p)	Cationic	0.1880832	4.051503298
96	C1=CC=C2C(=C1)C=CC3=C2C4=CC=CC=C4C=C3	b3lyp/6-311+G(d,p)	Cationic	0.1616092	4.171505512
97	C1=CC=C2C(=C1)C=CC3=C2C=CC4=C3C5=CC=CC=C5C=C4	b3lyp/6-311+G(d,p)	Cationic	0.2183144	4.208785111
98	C1=CC=C2C(=C1)C=CC3=CC4=C(C=C32)C5=CC=CC=C5C=C4	b3lyp/6-311+G(d,p)	Cationic	0.1326552	3.887418639
99	C1CC2=CC=CC=C2C3=CC=CC=C31	b3lyp/6-311+G(d,p)	Cationic	31.9738588	4.88226692
100	C1=CC=C2C(=C1)C=CC3=C2C=CC4=CC5=CC=CC=C5C=C43	b3lyp/6-311+G(d,p)	Cationic	0.12431	3.477615162

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
101	C1C2=CC=CC=C2CC3=CC4=CC=CC=C4C=C31	b3lyp/6-311+G(d,p)	Cationic	29.5613644	4.666480627
102	C1CCC2=CC3=C(CCCC3)C=C2C1	b3lyp/6-311+G(d,p)	Cationic	134.0260448	5.761194696
103	C1C2=CC=CC=C2CC3=CC=CC=C31	b3lyp/6-311+G(d,p)	Cationic	0.477648	5.945687895
104	C1=CC=C2C=C3C(=CC2=C1)C=CC4=C3C=CC5=CC6=CC=CC=C6C=C54	b3lyp/6-311+G(d,p)	Cationic	0.0974392	3.126316166
105	C1CCC2=C(C1)C=CC3=C2CCCC3	b3lyp/6-311+G(d,p)	Cationic	94.177814	5.899972766
106	C1=CC=C2C(=C1)C=CC3=C2C=CC4=C3C=CC5=CC=CC=C54	b3lyp/6-311+G(d,p)	Cationic	0.187426	4.206880314
107	C1=CC=C2C=C3C=C4C=C5C=CC=CC5=CC4=CC3=CC2=C1	b3lyp/6-311+G(d,p)	Cationic	0.0973648	2.190516593
108	C1CC2=C(C3=CC=CC=C3C=C2)C4=C1C=CC5=CC=CC=C54	b3lyp/6-311+G(d,p)	Cationic	0.2271308	3.923337669
109	C1CC2=C(C=CC3=CC=CC=C23)C4=CC=CC=C41	b3lyp/6-311+G(d,p)	Cationic	0.191766	4.247425279
110	C1=CC=C2C=C3C(=CC2=C1)C=CC4=CC5=CC=CC=C5C=C43	b3lyp/6-311+G(d,p)	Cationic	0.1804572	3.748640569
111	C1CCC2=C(C1)C=CC3=C2C=CC4=CC=CC=C43	b3lyp/6-311+G(d,p)	Cationic	76.3040944	4.562261018
112	C1CCC2=C(C1)C=CC3=CC4=C(C=CC5=CC=CC=C54)C=C23	b3lyp/6-311+G(d,p)	Cationic	4.8524424	3.656938198
113	C1=CC=C2C=C3C=C4C=C5C=CC6=CC=CC5=CC4=CC3=CC2=C1	b3lyp/6-311+G(d,p)	Cationic	0.0842952	1.77962466
114	C1=CC2=C3C(=C1)C=CC4=CC=CC(=C43)C=C2	b3lyp/6-311+G(d,p)	Cationic	0.1531772	3.810410416
115	C1=CC=C2C3=C4C(=CC2=C1)C=CC5=C4C(=CC=C5)C=C3	b3lyp/6-311+G(d,p)	Cationic	0.146072	3.348088964
116	C1=CC=C2C(=C1)C3=CC=CC4=C3C5=C(C=CC=C25)C=C4	b3lyp/6-311+G(d,p)	Cationic	0.1472376	3.958984585
117	C1=CC=C2C(=C1)C=C3C=CC4=C5C3=C2C6=CC=CC=C6C5=CC=C4	b3lyp/6-311+G(d,p)	Cationic	0.15469	3.381286855
118	C1=CC2=C3C(=C1)C4=CC=CC5=C4C6=C(C=C5)=CC(=C36)C=C2	b3lyp/6-311+G(d,p)	Cationic	0.138012	3.481968984
119	C1=CC=C2C3=C4C(=CC2=C1)C5=CC=CC=C5C6=CC=C1(=C64)C=C3	b3lyp/6-311+G(d,p)	Cationic	0.1386816	3.51271785
120	C1=CC=C2C3=C4C(=CC2=C1)C=CC5=C4C(=CC6=CC=CC=C56)C=C3	b3lyp/6-311+G(d,p)	Cationic	0.128216	2.877876209
121	C1=CC=C2C3=C4C(=CC2=C1)C=CC5=CC6=CC=CC=C6C(=C54)C=C3	b3lyp/6-311+G(d,p)	Cationic	0.1634692	3.224821384
122	C1=CC2=C3C(=C1)C4=CC=CC5=C4C(=CC=C5)C3=CC=C2	b3lyp/6-311+G(d,p)	Cationic	0.1469276	2.985089071
123	C1=CC=C2C(=C1)C3=CC=CC=C3C4=CC=CC=C24	b3lyp/6-311+G(d,p)	Cationic	0.1850204	4.838728702
124	C1=CC2=C3C(=C1)C=CC=CC5=C6C4=C3C(=CC6=CC=C5)C=C2	b3lyp/6-311+G(d,p)	Cationic	0.1159772	2.867807996
125	C1=CC=C2C(=C1)C=C3C4=CC=CC=C4C5=C3C2=CC6=CC=CC=C65	b3lyp/6-311+G(d,p)	Cationic	0.1511932	3.241964557
126	C1=CC2=C3C4=C1C=CC5=CC6=C7C8=C(C=CC9=C8C1=C(C=C9)C=C(C3=C1C7=C54)C=C2)C=C6	b3lyp/6-311+G(d,p)	Cationic	0.0955296	2.91597215
127	C1=CC=C2C(=C1)C=CC3=C2C=C4C=CC=C5C4=C3C=C5	b3lyp/6-311+G(d,p)	Cationic	0.2145572	3.238154963
128	C1=CC=C2C=C3C4=C5C(=CC3=CC2=C1)C=CC6=C5C(=CC=C6)C=C4	b3lyp/6-311+G(d,p)	Cationic	0.1239008	2.843317749
129	C1=CC=C2C3=C4C(=CC=CC4=CC2=C1)C=C3	b3lyp/6-311+G(d,p)	Cationic	0.2170868	3.079784695
130	C1=CC=C2C3=C4C(=CC=C3)C5=CC=CC=C5C6=CC=CC(=C64)C2=C1	b3lyp/6-311+G(d,p)	Cationic	0.1369332	4.105109729
131	C1=CC=C2C(=C1)C3=CC=C3C4=C2C5=CC=CC6=C5C4=CC=C6	b3lyp/6-311+G(d,p)	Cationic	0.2124616	3.252304884
132	C1=CC=C2C(=C1)C=C3C=C4C=CC=C4C5=C3C2C6=CC=CC=C65	b3lyp/6-311+G(d,p)	Cationic	0.1005516	2.460997772
133	C1=CC=C2C(=C1)C=C3C=CC=C4C3=C2C5=CC=CC6=C5C4=CC=C6	b3lyp/6-311+G(d,p)	Cationic	0.1473988	2.538822337
134	C1=CC=C2C(=C1)C3=CC=CC=C3C4=C2C5=CC=CC=C5C6=CC=CC=C64	b3lyp/6-311+G(d,p)	Cationic	10.9663988	3.976127758
135	C1CC2=CC=CC3=C2C4=C(C=CC=C41)C=C3	b3lyp/6-311+G(d,p)	Cationic	36.040662	4.611785741
136	C1=CC=C2C=C3C4=CC=C5=C4C6=C(C=CC=C6C3=CC2=C1)C=C5	b3lyp/6-311+G(d,p)	Cationic	0.1250912	3.86211205
137	C1=CC=C2C3=C4C(=CC2=C1)C5=CC=CC=C5C6=CC=C7C=C7C(=C64)C=C3	b3lyp/6-311+G(d,p)	Cationic	0.15128	3.406865558
138	C1=CC2=C3C4=C1C=CC5=C4C6=C(C=C5)C=CC(=C36)C=C2	b3lyp/6-311+G(d,p)	Cationic	0.1904764	4.161437299
139	C1CC2=C3C(=CC=C4C3=C(CC4)C=C2)C1	b3lyp/6-311+G(d,p)	Cationic	71.727428	4.446068399
140	CC1=CC2=C3C(=C1)C=CC4=CC(=CC=C43)C=C2C	b3lyp/6-311+G(d,p)	Cationic	0.2394316	3.789457649
141	CC1=CC2=C3C(=C1)C=CC4=CC=CC(=C43)C=C2	b3lyp/6-311+G(d,p)	Cationic	0.18321	3.800070089
142	CC1=C2C=C3C=CC=C4C=C3C2=C(C=C1)C=C4	b3lyp/6-311+G(d,p)	Cationic	16.1239432	3.724694549
143	CC1=CC2=CC=CC=C2C=C1	b3lyp/6-311+G(d,p)	Cationic	16.7850492	4.756278202
144	CC1=CC=CC2=CC=C12	b3lyp/6-311+G(d,p)	Cationic	1.3446436	4.655323959
145	C1=CC2=C3C(=C1)C=CC3=CC=C2	b3lyp/6-311+G(d,p)	Cationic	0.4145196	3.886602298
146	CC1=C2C=CC3=CC=CC=C3C2=C(C4=CC=CC=C14)C	b3lyp/6-311+G(d,p)	Cationic	18.0141248	3.535575415

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
147	CC1=CC2=CC=CC=C2C3=C1C4=CC=CC=C4C=C3	b3lyp/6-311+G(d,p)	Cationic	0.1957588	4.099939566
148	CC1=CC2=C(C=C1)C=C(C=C2)C	b3lyp/6-311+G(d,p)	Cationic	15.9425808	4.746754217
149	CC1=CC2=CC=CC(=C2C=C1)C	b3lyp/6-311+G(d,p)	Cationic	33.1261164	4.665664286
150	CC(C)C1=CC2=C(C=C1)C=C(C=C2)C(C)C	b3lyp/6-311+G(d,p)	Cationic	24.9330148	4.662126806
151	CC1=C2C=CC3=CC=CC=C3C2=CC4=CC=CC=C14	b3lyp/6-311+G(d,p)	Cationic	16.3096456	3.629454697
152	CC1=C2C(=CC3=CC=CC=C13)C=CC4=CC=CC=C42	b3lyp/6-311+G(d,p)	Cationic	5.3257504	3.6275499
153	CC1=C2C=C3C=CC4=CC=CC=C4C3=CC2=CC=C1	b3lyp/6-311+G(d,p)	Cationic	1.4968164	3.696938935
154	CC(C)C1=CC2=CC=CC=C2C=C1	b3lyp/6-311+G(d,p)	Cationic	22.8050012	4.747842672
155	CC1=CC=C(C2=CC=CC=C12)C	b3lyp/6-311+G(d,p)	Cationic	17.3522376	4.555458171
156	CC1=CC=C(C=C1)S(=O)(=O)O	b3lyp/6-311+G(d,p)	Cationic	0.7946416	5.974531964
157	CC1=CC=C(C=C1)S(=O)(=O)Cl	b3lyp/6-311+G(d,p)	Cationic	62.5300504	5.124720372
158	C1=CC2=CC3=C4C5=C(C=C3)C=C6C=CC7=C8C6=C5C9=C3C4=C2C2=C1C=CC4=C(C=C12)C1=C(C=C4)C=C(C8=C19)C=C7	b3lyp/6-311+G(d,p)	Cationic	0.068014	2.520862822
160	C1=CC=C2C(=C1)C3=C4C2=CC5=CC=CC6=C5C4=C(C=C6)C=C3	b3lyp/6-311+G(d,p)	Cationic	0.16213	3.31189782
306	C1=CSC2=CC3=CC4=C(C=C3C=C21)C=C5C(=C4)C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.0945004	2.73773757
307	C1=CC2=C(C=C3=C2C=C4=C3SC=C4)C5=C1C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.1961184	4.358447735
308	C1=CC=C2C(=C1)C3=CC4=C(C=C3S2)SC5=CC=CC=C54	b3lyp/6-311+G(d,p)	Cationic	0.0839604	4.273276096
309	C1=CC=C2C(=C1)C3=CC4=C(C=C3S2)C5=CC=CC=C554	b3lyp/6-311+G(d,p)	Cationic	0.1198088	4.002250689
310	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C5=CC=CC=C554	b3lyp/6-311+G(d,p)	Cationic	0.136958	4.272731868
311	C1=CC=C2C=C3C(=CC2=C1)C4=C(C=C5=CSC=C5)SC=C4	b3lyp/6-311+G(d,p)	Cationic	0.1440012	3.515983217
312	C1=CC=C(C(=C1)C#CC2=CC=CS2)C#CC3=CC=CC3	b3lyp/6-311+G(d,p)	Cationic	2.7106896	3.604420222
313	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)SC5=CC=CC=C54	b3lyp/6-311+G(d,p)	Cationic	0.1503872	4.397632131
314	C1=CC=C2C(=C1)C3=C(S2)C=C4=C3C5=C(C=C4)SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.2009296	4.103204932
315	C1=CC=C2C(=C1)C3=C(C4=C(C=C3)SC=C4)C5=C2SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.2041908	4.228649423
316	C1=CSC2=CC3=CC4=C(C=C5C=CSC5=C4)C=C3C=C21	b3lyp/6-311+G(d,p)	Cationic	0.0941532	2.748077897
317	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C5=CC=CC=C5S4	b3lyp/6-311+G(d,p)	Cationic	0.1780268	4.022659229
318	C1=CC2=CC3=CSC4=CC=CC(=C34)C5=C2C(=C1)SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.1854792	2.354329138
319	C1=CC2=CC3=CC4=CC5=CSSC5=CC4=CC3=CC2=C1	b3lyp/6-311+G(d,p)	Cationic	0.2573496	1.373630778
320	C1=CC=C2C(=C1)C=C3C=C4C=CC=C5C=C2SS5	b3lyp/6-311+G(d,p)	Cationic	0.1422156	2.338002306
321	C1=CC2=C(C=CS2)C3=CC4=C(C=C31)C5=C(C=C4)SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.1578272	3.535575415
322	C1=CC2=C(C=C3C=CSC3=C2)C4=C1C5=C(C=C4)C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.20522	3.8846975
323	C1=CC=C2C=C3C(=CC2=C1)C4=C(C=S3)C5=CC=CC=C554	b3lyp/6-311+G(d,p)	Cationic	0.1546156	3.62891047
324	C1=CC2=CC3=C(C=CC4=C3SC=C4)C=C2C5=C1C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.1356436	3.502921751
325	C1=CC=C2C(=C1)C=C3C=C2C4=C(S3)C5=CC=CC=C5S4	b3lyp/6-311+G(d,p)	Cationic	0.2102916	3.921432872
326	C1=CC2=C(C=C3C=C2C=C3)C4=C1C5=C(C=C4)SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.210924	4.153001769
327	C#CC1=CC(=C(C=C1C2=CC=CS2)C#)C3=CC=CS3	b3lyp/6-311+G(d,p)	Cationic	0.3881448	3.864288961
328	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C5C(=CC4=C3)C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.093186	3.467819063
329	C1=CC2=C(C=C3C=C2C=CC4=C3C=CS4)C5=C1C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.2012892	4.245248368
330	C1=CC=C2C(=C1)C=C(S2)C#CC3=CC4=CC=CC=C4S3	b3lyp/6-311+G(d,p)	Cationic	0.2513728	3.589181846
331	C1=CC2=CC3=C(C=C2C4=C1C=CS4)C5=C(C=C3)C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.136648	3.506187118
332	C1=CC=C2C(=C1)SC3=C(S2)C4=CC=CC5=C4C3=CC=C5	b3lyp/6-311+G(d,p)	Cationic	1.0813048	3.034885908
333	C#CC1=CC(=C(C=C1C2=CSC=C2)C#)C3=CSC=C3	b3lyp/6-311+G(d,p)	Cationic	4.141166	4.155722907
334	[2H]C1=CC2=C(C=CS2)C3=CC4=C(C=C13)C5=C(C=C4[2H])SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.1578644	3.535575415
335	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C=CC5=C4SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.19344	4.101844363
336	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C5=C(C=C4)C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.1632088	4.209601452
337	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C5=C(C=C4)SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.180544	4.240078205
338	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C5=C(C=C4)C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.1807176	4.058850372

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
339	C1=CC=C2C(=C1)C3=C(S2)C=C4C(=C3)C=CC5=C4SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.1131128	3.852043837
340	C1=CC=C2C(=C1)C3=C(S2)C=C4C(=C3)C=CC5=C4C=C55	b3lyp/6-311+G(d,p)	Cationic	0.1368588	3.906738723
341	C1=CC=C2C(=C1)C3=C4C(=CC=C3)SC5=CC=CC(=C54)S2	b3lyp/6-311+G(d,p)	Cationic	0.6864516	4.237357066
342	[2H]C1=C2C(=CC=C1)C3=C4C(=CC=C3)SC5=CC=CC(=C54)S2	b3lyp/6-311+G(d,p)	Cationic	0.68727	4.237357066
343	C1=CC=C2C=C3C(=CC=C1)C4=C(C=CS4)C5=C3SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.1355692	3.492037197
344	C1=CC2=C(C3=CSC=C3C=C2)C4=C1C=CC5=CSC=C54	b3lyp/6-311+G(d,p)	Cationic	0.1836564	3.737483901
345	C1=CC2=C(C3=C(C=C2)SC=C3)C4=C1C=CC5=C4C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.2449372	4.239261863
346	C1=CC2=C3C=CC=C4C3=C(C=CS4)C5=C2C(=C1)SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.2395184	3.111894131
347	C1=CC(=CC(=C1)C#CC2=CC=CS2)C#CC3=CC=CS3	b3lyp/6-311+G(d,p)	Cationic	3.1869488	3.905378154
348	C1=CC2=CC3=C(C=C2C4=CC5=C(C=CS5)C=C41)SC=C3	b3lyp/6-311+G(d,p)	Cationic	0.18352	3.974222961
349	C1=CC=C2C(=C1)C=CC3=C2C=CC4=C3SC5=C4SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.2312228	4.255316581
350	C1=CC=C2C3=C4C(=CC=C3)SSC5=CC=CC(=C54)C2=C1	b3lyp/6-311+G(d,p)	Cationic	0.8475524	3.913813684
351	C#CC1=C(C=CS1)C2=CC=C(C=C2)C3=C(SC=C3)C#C	b3lyp/6-311+G(d,p)	Cationic	4.186674	4.092048264
352	C1=CC=C2C(=C1)C=CC3=C2C4=C(C=CS4)C5=C3SC=C5	b3lyp/6-311+G(d,p)	Cationic	0.173786	4.096402085
353	C1=CC=C2C=C3C=C4C(=CC3=CC2=C1)C5=C(S4)C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.1124184	3.038423388
354	C1=CC=C(C=C1)C#CC2=CC3=C(S2)C=CC4=C3C=CS4	b3lyp/6-311+G(d,p)	Cationic	2.923114	3.683333242
355	C#CC1=CC(=C(C=C1C2=CSC=C2)C3=CSC=C3)C#C	b3lyp/6-311+G(d,p)	Cationic	0.247256	4.286881789
356	C1=CC2=CSC=C2C3=C4C(=C(C=CC5=CSC=C54)C=C31	b3lyp/6-311+G(d,p)	Cationic	0.100254	3.740749267
357	C1=CC2=CC3=CSC=C3C=C2C4=CC5=CSC=C5C=C41	b3lyp/6-311+G(d,p)	Cationic	0.1939112	3.207133983
358	C1=CC2=CSC=C2C3=C4C(=C(C=C31)C5=CSC=C5C=C4	b3lyp/6-311+G(d,p)	Cationic	0.1596252	3.672992915
359	C1=CC2=C(C=CC3=C2C=CC4=CSC=C43)C5=CSC=C51	b3lyp/6-311+G(d,p)	Cationic	0.1988092	3.861567822
360	C1=CC2=C3C4C1C5=CSC=C5C4=CC=C3C6=CSC=C26	b3lyp/6-311+G(d,p)	Cationic	69.5200668	2.724403991
361	C1=CC=C2C(=C1)C3=C(S2)C4=CC=CC5=C4C(=CC=C5)S3	b3lyp/6-311+G(d,p)	Cationic	0.175088	3.28087684
466	C1=CC=C2C(=C1)C3=C(S2)C4=CC=CC=C4S3	b3lyp/6-311+G(d,p)	Cationic	0.2284204	4.234363814
467	C1=CC2=C(C=CC3=C2SC=C3)C4=C1C=CS4	b3lyp/6-311+G(d,p)	Cationic	0.2761604	4.420489696
468	C1=CC2=C3C(=C1)SSC4=CC=C1(=C43)C=C2	b3lyp/6-311+G(d,p)	Cationic	0.7743924	3.888234981
469	C1=CC=C2C3=C4C(=CC=C2)SSC4=CC2=C1	b3lyp/6-311+G(d,p)	Cationic	0.1512924	3.379109944
470	C1=CSC2=CC3=C(C=C21)C=C4C(=C3)C=CS4	b3lyp/6-311+G(d,p)	Cationic	0.1002788	3.438430766
471	C1=CC=C2C3=C(C4=C(C2=C1)C=CS4)SC=C3	b3lyp/6-311+G(d,p)	Cationic	0.1762164	4.229737878
472	C1=CC2=C3C(=CC=C4C3=C1C=CS4)C=C52	b3lyp/6-311+G(d,p)	Cationic	0.214458	3.026913636
473	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)SC=C4	b3lyp/6-311+G(d,p)	Cationic	0.17112	4.569608092
474	C1=CC2=CC3=C4C(=CC=CS4)SC3=C2C=C1	b3lyp/6-311+G(d,p)	Cationic	0.3695076	1.617716912
475	C1=CC2=C3C(=C1)C=CC4=C3C(=CSS4)C=C2	b3lyp/6-311+G(d,p)	Cationic	0.6092492	2.626715114
476	C1=CC=C2C(=C1)C3=C(C4=C2SC=C4)SC=C3	b3lyp/6-311+G(d,p)	Cationic	0.2313716	4.345658383
477	C1=CC=C2C(=C1)C(=S)C3=CC=C3C2=S	b3lyp/6-311+G(d,p)	Cationic	0.0263004	2.455827609
478	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3SC=C4	b3lyp/6-311+G(d,p)	Cationic	0.213714	4.492871983
479	C1=CC=C2C(=C1)C=C(S2)C#CC3=CSC=C3	b3lyp/6-311+G(d,p)	Cationic	0.2726512	3.961161496
480	C1=CC=C2C(=C1)C3=C(S2)SC4=CC=CC=C43	b3lyp/6-311+G(d,p)	Cationic	0.1278564	4.666752741
481	C1=CC=C2C(=C1)C=C(S2)C#CC3=CC=C53	b3lyp/6-311+G(d,p)	Cationic	0.450988	3.720884955
482	C1=CC2=C(C=CS2)C3=C1C4=C(C=C3)SC=C4	b3lyp/6-311+G(d,p)	Cationic	0.2472064	4.274092438
483	C1=CSC2=CC3=C(C=C21)C=C4C=CSC4=C3	b3lyp/6-311+G(d,p)	Cationic	0.096968	3.478703618
484	C1=CC2=C3C(=CC=C4C3=C1C=CS4)SC=C2	b3lyp/6-311+G(d,p)	Cationic	0.214334	2.889032878
485	C1=CC2=C(C3=C1C=C4=C3SC=C4)SC=C2	b3lyp/6-311+G(d,p)	Cationic	0.158782	4.584302241
486	C1=CC=C2C(=C1)C=CC3=C2C4=C(S3)SC=C4	b3lyp/6-311+G(d,p)	Cationic	0.237522	4.35735928
487	C1=C(C=C2/C=C3=C2SC=C3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	Cationic	0.2830176	2.861549378
488	C1=CC=C2C(=C1)C3=CSC(=C23)C4=CC=CS4	b3lyp/6-311+G(d,p)	Cationic	0.6806608	3.845513104

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
489	C1=CSC(=C1)C#C/C=C\C#CC2=CC=CS2	b3lyp/6-311+G(d,p)	Cationic	4.3437696	3.232440572
490	C1=CC=C2C(=C1)C3=C(S2)C=C4=CSC4=C3	b3lyp/6-311+G(d,p)	Cationic	0.0976004	4.159804615
491	C#CC1=C\{SC2=CC=CC=C21\}C3=CC=CS3	b3lyp/6-311+G(d,p)	Cationic	1.6564912	3.750273252
492	C1=CC=C2C(=C1)C(=C3C=CC=CC3=C2[S-])[S-]	b3lyp/6-311+G(d,p)	Cationic	0.0265236	2.456371836
493	C1=CC2=C3C(=C1)C4=C(C3=CC=C2)SC=CS4	b3lyp/6-311+G(d,p)	Cationic	0.8726252	2.887944422
494	C1=C/C(=C/2\ C=CC3=C2SC=CS3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	Cationic	0.2726016	2.942367195
495	C1=CC=C2C=C3C(=CC2=C1)C4=C(S3)C=CS4	b3lyp/6-311+G(d,p)	Cationic	0.1434432	3.797348951
496	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C=CS4	b3lyp/6-311+G(d,p)	Cationic	0.1861984	4.427292542
497	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C=CS4	b3lyp/6-311+G(d,p)	Cationic	0.1531524	4.378856275
498	C#CC1=C2C(=CC=C1)SC3=CC=CC=C3S2	b3lyp/6-311+G(d,p)	Cationic	1.3996624	4.507566132
499	C\1=CC2=C(/C1=C/3\ C=CC4=C3C=CS4)C=CS2	b3lyp/6-311+G(d,p)	Cationic	0.286998	2.871073363
500	C1=CC=C2C(=C1)C3=CSC=C3C4=CSC=C24	b3lyp/6-311+G(d,p)	Cationic	0.3566364	4.489334503
501	C1=CC2=C(C3=C1C=CC4=C3C=CS4)SC=C2	b3lyp/6-311+G(d,p)	Cationic	0.2310368	4.305113418
502	C1=C/C(=C/2\ C=CC3=C2C=CS3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	Cationic	0.5224492	2.856923442
503	C1=CC=C2C(=C1)C3=CC=CC=C3C2=C([S-])[S-]	b3lyp/6-311+G(d,p)	Cationic	0.1622292	2.187251226
504	C\1=CC2=C(/C1=C\3\ C=CC4=C3C=CS4)C=CS2	b3lyp/6-311+G(d,p)	Cationic	0.4494752	2.88848865
505	C1=C/C(=C/2\ C=CC3=C2C=CS3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	Cationic	0.4985544	2.858011897
506	C1=CC=C2C(=C1)C3=C(S2)C=CC4=CSC=C43	b3lyp/6-311+G(d,p)	Cationic	0.1636428	3.70782349
507	C1=CC=C2C(=C1)C3=C(C=CS3)C4=C2SC=C4	b3lyp/6-311+G(d,p)	Cationic	0.220348	4.421850265
508	C1=CC=C2C(=C1)C3=C(S2)C4=CSC=CC4=C3	b3lyp/6-311+G(d,p)	Cationic	0.3142036	3.187269671
509	C1=CC=C2C(=C1)C3=CC4=CC=CSC4=C3S2	b3lyp/6-311+G(d,p)	Cationic	0.2507652	2.63487853
510	C#CC1=CC2=C(C=C1)SC3=CC=CC=C3S2	b3lyp/6-311+G(d,p)	Cationic	1.3449164	4.458041409
511	C1=CC2=C(C=CC3=CSC=C32)C4=CSC=C41	b3lyp/6-311+G(d,p)	Cationic	0.125736	3.412307835
512	C1=CC2=CSC=C2C3=C1C=CC4=CSC=C43	b3lyp/6-311+G(d,p)	Cationic	0.1198212	3.367409048
513	C1=CC2=CC3=C4C(=CS4)C=CC3=C2C=C1	b3lyp/6-311+G(d,p)	Cationic	0.2238944	1.640846591
514	C1=CC2=C(C=C3C=CSC3=C2)C4=C1C=CS4	b3lyp/6-311+G(d,p)	Cationic	0.1290964	3.924154011
516	C1=CSC2=CC3=C(C=C21)SC=C3	b3lyp/6-311+G(d,p)	Cationic	0.1581	4.296405774
517	C1=CC2=C3C(=C1)SSC3=CC=C2	b3lyp/6-311+G(d,p)	Cationic	0.1405292	3.351898558
518	C1=CSC2=CC3=C(C=CS3)C=C21	b3lyp/6-311+G(d,p)	Cationic	0.1030936	4.482531656
519	C1=CC2=CSC=C2C3=CSC=C31	b3lyp/6-311+G(d,p)	Cationic	0.2985796	4.38348221
520	C1=CC=C2C(=C1)C3=C(S2)SC=C3	b3lyp/6-311+G(d,p)	Cationic	0.2392952	4.963084737
521	C1=CC2=C(C3=C1C=CS3)SC=C2	b3lyp/6-311+G(d,p)	Cationic	0.2017356	4.718182261
522	C1=CC=C2C(=C1)C3=C(S2)C=CS3	b3lyp/6-311+G(d,p)	Cationic	0.2550184	4.623758751
523	C1=CC2=C(C=CS2)C3=C1C=CS3	b3lyp/6-311+G(d,p)	Cationic	0.2419488	4.879817895
524	C1=CC2=C(C=CS2)C3=C1SC=C3	b3lyp/6-311+G(d,p)	Cationic	0.2885108	4.601717528
525	C1=CC(=S)C=C2C1=CC(=S)C=C2	b3lyp/6-311+G(d,p)	Cationic	0.03906	1.993506156
526	C1=CC=C2C(=C1)C=CC(=S)C2=S	b3lyp/6-311+G(d,p)	Cationic	0.9801828	2.148338944
527	C1=CC=C2C(=S)C=CC(=S)C2=C1	b3lyp/6-311+G(d,p)	Cationic	0.0795584	2.342900356
528	C=C1C2=C(C3=C1C=CS3)SC=C2	b3lyp/6-311+G(d,p)	Cationic	42.8313112	3.518976469
529	C1=CSC2=CC3=CSC=C3C=C21	b3lyp/6-311+G(d,p)	Cationic	0.1049908	3.294482533
530	C1=CC2=C(C=CS2)C3=CSC=C31	b3lyp/6-311+G(d,p)	Cationic	0.1967136	3.918983847
531	C1=CC2=C3C=CC=C3SSC2=C1	b3lyp/6-311+G(d,p)	Cationic	0.6642184	2.329566777
532	C1=CC=C2C(=C1)C=CC3=C2SS3	b3lyp/6-311+G(d,p)	Cationic	0.2196412	3.431083692
533	C1=CSC2=C1SC=C2	b3lyp/6-311+G(d,p)	Cationic	0.4038928	5.059957272
534	C1=CSC2=C1C=CS2	b3lyp/6-311+G(d,p)	Cationic	0.132494	5.489625061
535	C1=CC(=S)C=CC1=S	b3lyp/6-311+G(d,p)	Cationic	0.0560232	2.16357732

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
618	C1=CC(=S)C(=S)C=C1	b3lyp/6-311+G(d,p)	Cationic	0.7926204	1.789692873
619	[2H]C1=C(SC2=C1C(=C(S2)[2H])[2H])[2H]	b3lyp/6-311+G(d,p)	Cationic	0.1325064	5.489625061
620	[2H]C1=C(SC2=C1SC(=C2[2H])[2H])[2H]	b3lyp/6-311+G(d,p)	Cationic	0.4038308	5.059957272
621	C1=CC2=C(S2)C=C1S	b3lyp/6-311+G(d,p)	Cationic	0.5741696	4.576138825
622	C1=CC2=S=CC2=CS1	b3lyp/6-311+G(d,p)	Cationic	0.060326	2.994068829
623	C1=CC2=CSSC2=C1	b3lyp/6-311+G(d,p)	Cationic	0.4316936	3.39081084
161	C1CCC2=CC3=C(CCCC3)C=C2C1	b3lyp/6-311+G(d,p)	Anionic	184.8776884	5.76146681
162	C1CCC2=C(C1)C=CC3=C2CCCC3	b3lyp/6-311+G(d,p)	Anionic	207.4216572	5.93834082
163	C1CCC2=CC3=CC=CC=C3C=C2C1	b3lyp/6-311+G(d,p)	Anionic	0.2377204	4.593826226
164	C1CCC2=C(C1)C=CC3=C2C=CC4=CC=CC=C43	b3lyp/6-311+G(d,p)	Anionic	41.600326	4.583758013
165	C1CCC2=CC=CC=C2C1	b3lyp/6-311+G(d,p)	Anionic	75.01411	6.088003445
166	C1CCC2=C(C1)C=CC3=CC=CC=C23	b3lyp/6-311+G(d,p)	Anionic	106.4849008	4.646616315
167	C1CC2=C3C(=CC=C4C3=C(CCC4)=C2)C1	b3lyp/6-311+G(d,p)	Anionic	52.9133668	4.449061651
168	C1CC2=CC3=CC=CC=C3C=C2C=C1	b3lyp/6-311+G(d,p)	Anionic	37.0171992	4.405523433
169	C1CC2=CC=CC=C2C=C1	b3lyp/6-311+G(d,p)	Anionic	33.02244	4.958730916
170	C1CC2=C(C=C1)C3=CC=CC=C3C=C2	b3lyp/6-311+G(d,p)	Anionic	37.3657756	4.098578996
171	C1C=CCC2=CC=CC=C21	b3lyp/6-311+G(d,p)	Anionic	30.8223204	5.944327325
172	CC1=CC=C(C2=CC=CC=C12)C	b3lyp/6-311+G(d,p)	Anionic	18.6709652	4.555186058
173	CC1=CC2=CC=CC(=C2C=C1)C	b3lyp/6-311+G(d,p)	Anionic	0.3947044	4.665664286
174	CC1=C2C(=CC3=CC=CC=C13)C=CC4=CC=CC=C42	b3lyp/6-311+G(d,p)	Anionic	18.429314	3.6275499
175	CC1=CC=CC2=CC=CC=C12	b3lyp/6-311+G(d,p)	Anionic	0.2305904	4.655323959
176	CC1=C2C=CC3=CC=CC4=C3C2=C(C=C1)C=C4	b3lyp/6-311+G(d,p)	Anionic	0.2053936	3.724694549
177	CC(C)C1=CC2=C(C=C1)C=C(C=C2)C(C)C	b3lyp/6-311+G(d,p)	Anionic	90.2407768	4.662671033
178	CC1=CC2=C(C=C1)C=C(C=C2)C	b3lyp/6-311+G(d,p)	Anionic	33.4431348	4.746754217
179	CC1=CC2=C3C(=C1)C=CC4=CC(=CC(=C43)C=C2)C	b3lyp/6-311+G(d,p)	Anionic	16.5499328	3.789457649
180	CC1=CC2=CC=CC=C2C=C1	b3lyp/6-311+G(d,p)	Anionic	0.4182768	4.756278202
181	CC1=CC2=C3C(=C1)C=CC4=CC=CC(=C43)C=C2	b3lyp/6-311+G(d,p)	Anionic	0.3536232	3.800070089
182	CC(C)C1=CC2=CC=CC=C2C=C1	b3lyp/6-311+G(d,p)	Anionic	44.6110708	4.747842672
183	C1CC2=CC=CC3=C2C4=C(C=CC=C41)C=C3	b3lyp/6-311+G(d,p)	Anionic	31.7317984	4.611785741
184	C1C2=CC=CC=C2CC3=CC4=CC=CC=C4C=C31	b3lyp/6-311+G(d,p)	Anionic	30.8957532	4.666208513
185	C1CC2=C(C=CC3=CC=CC=C23)C4=CC=CC=C41	b3lyp/6-311+G(d,p)	Anionic	36.7601596	4.247425279
186	CC1=CC2=CC=CC=C2C3=C1C4=CC=CC=C4C=C3	b3lyp/6-311+G(d,p)	Anionic	16.8097872	4.099939566
187	CC1=C2C=CC3=CC=CC=C3C2=C(C4=CC=CC=C14)C	b3lyp/6-311+G(d,p)	Anionic	35.9539488	3.535575415
188	CC1=C2C=CC3=CC=CC=C3C2=C4=CC=CC=C14	b3lyp/6-311+G(d,p)	Anionic	0.2187608	3.629454697
189	C1CC2=C(C1)C=CC3=CC4=C(C=CC5=CC=CC=C54)C=C23	b3lyp/6-311+G(d,p)	Anionic	36.6384536	3.710272515
190	CC1=C2C=C3C=CC4=CC=CC=C4C3=CC2=CC=C1	b3lyp/6-311+G(d,p)	Anionic	0.1854668	3.696938935
191	C1C2=CC=CC=C2CC3=CC=CC=C31	b3lyp/6-311+G(d,p)	Anionic	30.7856536	5.945687895
192	C1CC2=CC=CC=C2C3=CC=CC=C31	b3lyp/6-311+G(d,p)	Anionic	37.0302812	4.88226692
193	C1=CC=C2C3=C4C(=CC=CC4=CC=C1)C=C3	b3lyp/6-311+G(d,p)	Anionic	0.2321032	3.079784695
194	C1=CC2=C3C(=C1)C=CC3=CC=C2	b3lyp/6-311+G(d,p)	Anionic	0.2856712	3.886874411
195	C1=CC=C2C=C3C=CC=CC3=CC2=C1	b3lyp/6-311+G(d,p)	Anionic	0.1962672	3.551630133
196	C1=CC=CC=C1	b3lyp/6-311+G(d,p)	Anionic	0.0022816	6.598216937
197	C1=CC=C2C(=C1)C=CC3=CC4=CC=CC=C4C=C32	b3lyp/6-311+G(d,p)	Anionic	0.1847848	3.723878208
198	C1=CC=C2C3=C4C(=CC2=C1)C=CC5=C4C(=CC=C5)C=C3	b3lyp/6-311+G(d,p)	Anionic	0.2005452	3.348088964
199	C1=CC=C2C=C3C4=CC=CC=C4C5=CC=CC=C5C3=CC2=C1	b3lyp/6-311+G(d,p)	Anionic	0.163804	3.849594812
200	C1=CC=C2C(=C1)C=CC3=C2C4=CC=CC=C4C=C3	b3lyp/6-311+G(d,p)	Anionic	0.174902	4.171233398

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
201	C1=CC=C2C(=C1)C3=CC=CC4=C3C5=C(C=CC=C25)C=C4	b3lyp/6-311+G(d,p)	Anionic	0.1972468	3.958984585
202	C1=CC=C2C(=C1)C=CC3=C2C=C4C=CC=C5C4=C3C=C5	b3lyp/6-311+G(d,p)	Anionic	0.2263868	3.238154963
203	C1=CC=C2C(=C1)C=CC3=C2C=CC4=CC=CC=C43	b3lyp/6-311+G(d,p)	Anionic	0.2145572	4.219125437
204	C1=CC2=C3C4=C1C=CC5=C4C6=C(C=C5)C=CC7=C6C3=C(C=C2)C=C7	b3lyp/6-311+G(d,p)	Anionic	0.16957	4.001162234
205	C1=CC=C2C3=C4C(=CC2=C1)C5=CC=CC=C5C6=CC7=CC=CC=C7C(=C64)C=C3	b3lyp/6-311+G(d,p)	Anionic	0.200818	3.406865558
206	C1=CC=C2C=C3C=C4C=C5C=C6C=CC=C6=CC5=CC4=CC3=CC2=C1	b3lyp/6-311+G(d,p)	Anionic	0.1147248	1.77962466
207	C1=CC=C2C=C3C4=CC=CC5=C4C6=C(C=CC=C6C3=CC2=C1)C=C5	b3lyp/6-311+G(d,p)	Anionic	0.162812	3.86211205
208	C1=CC=C2C3=C4C(=CC=C3)C5=CC=CC=C5C6=CC=CC(=C64)C2=C1	b3lyp/6-311+G(d,p)	Anionic	0.1752988	4.105109729
209	C1=CC=C2C3=C4C(=CC2=C1)C5=CC=CC=C5C6=CC=CC(=C64)C=C3	b3lyp/6-311+G(d,p)	Anionic	0.1818088	3.51271785
210	C1=CC=C2C(=C1)C=C3C=CC4=C5C3=C2C6=CC=CC=C6C5=CC=C4	b3lyp/6-311+G(d,p)	Anionic	0.2059268	3.381286855
211	C1=CC=C2C3=C4C(=CC2=C1)C=CC5=C4C(=CC6=CC=CC=C56)C=C3	b3lyp/6-311+G(d,p)	Anionic	0.1660732	2.877876209
212	C1=CC=C2C3=C4C(=CC2=C1)C=CC5=CC6=CC=CC=C6C(=C54)C=C3	b3lyp/6-311+G(d,p)	Anionic	0.225494	3.225093498
213	C1=CC2=C3C4=C1C=CC5=C4C6=C(C=C5)C=CC(=C36)C=C2	b3lyp/6-311+G(d,p)	Anionic	0.2176448	4.161437299
214	C1=CC2=C3C(=C1)C=C4C=CC5=C6C4=C3C(=CC6=CC=C5)C=C2	b3lyp/6-311+G(d,p)	Anionic	0.1576288	2.867807996
215	C1=CC=C2C(=C1)C=C3C=CC=C4C3=C2C5=CC=CC6=C5C4=CC=C6	b3lyp/6-311+G(d,p)	Anionic	0.1600592	2.539094451
216	C1=CC=C2C=C3C(=CC2=C1)C=C4=C3C=CC5=CC6=CC=CC=C6C=C54	b3lyp/6-311+G(d,p)	Anionic	0.1355816	3.126044052
217	C1=CC=C2C(=C1)C3=CC=CC=C3C4=C2C5=CC=CC=C5C6=CC=CC=C64	b3lyp/6-311+G(d,p)	Anionic	0.2570272	3.823199768
218	C1=CC2=C3C(=C1)C4=CC=CC5=C4C6=C(C=C5)C=CC(=C36)C=C2	b3lyp/6-311+G(d,p)	Anionic	0.175274	3.481968984
219	C1=CC=C2C=C3C4=C5C(=CC3=CC2=C1)C=CC6=C5C(=CC=C6)C=C4	b3lyp/6-311+G(d,p)	Anionic	0.165044	2.843045635
220	C1=CC=C2C(=C1)C3=CC=CC=C3C4=C2C5=CC=CC6=C5C4=CC=C6	b3lyp/6-311+G(d,p)	Anionic	0.2232744	3.252304884
221	C1=CC=C2C(=C1)C=C3C4=CC=CC=C4C5=C3C2=CC6=CC=CC=C65	b3lyp/6-311+G(d,p)	Anionic	0.2053688	3.241964557
222	C1=CC=C2C(=C1)C=C3C=C4C=CC=C4C3=C2C6=CC=CC=C65	b3lyp/6-311+G(d,p)	Anionic	0.1808292	2.460997772
223	C1=CC=C2C(=C1)C3=C4C2=CC5=CC=CC6=C5C4=C(C=C6)C=C3	b3lyp/6-311+G(d,p)	Anionic	0.2246136	3.31189782
224	C1=CC=C2C=CC=CC2=C1	b3lyp/6-311+G(d,p)	Anionic	0.2423952	4.747842672
225	C1=CC=C2C(=C1)C=CC3=CC4=C(C=CC5=CC=CC=C54)C=C32	b3lyp/6-311+G(d,p)	Anionic	0.1906872	3.849322698
226	C1=CC2=C3C4=C1C=CC5=CC6=C7C8=C(C=CC9=C8C1=C(C=C9)C=C(C3=C1C7=C54)C=C2)C=C6	b3lyp/6-311+G(d,p)	Anionic	0.1215572	2.91597215
227	C1=CC=C2C=C3C=C4C=C5C=CC=CC5=CC4=CC3=CC2=C1	b3lyp/6-311+G(d,p)	Anionic	0.1355568	2.190516593
228	C1=CC=C2C(=C1)C=CC3=C2C=CC4=C3C5=CC=CC=C5C=C4	b3lyp/6-311+G(d,p)	Anionic	0.30752	4.209057225
229	C1CC2=C(C3=CC=CC=C3C=C2)C4=C1C=CC5=CC=CC=C54	b3lyp/6-311+G(d,p)	Anionic	37.411792	3.923881897
230	C1=CC=C2C(=C1)C=CC3=C2C=CC=C4C5=CC=C5C=C43	b3lyp/6-311+G(d,p)	Anionic	0.1695948	3.477615162
231	C1=CC=C2C(=C1)C=CC3=C2C4=CC=CC=C4C5=CC=C35	b3lyp/6-311+G(d,p)	Anionic	0.2394192	4.051503298
232	C1=CC=C2C(=C1)C=CC3=CC4=C(C=C32)C5=CC=CC=C5C=C4	b3lyp/6-311+G(d,p)	Anionic	0.1694212	3.887418639
233	C1=CC=C2C=C3C(=CC2=C1)C=CC4=CC5=CC=CC=C5C=C43	b3lyp/6-311+G(d,p)	Anionic	0.2502568	3.748640569
234	C1=CC2=C3C(=C1)C4=CC=CC5=C4C(=CC=C5)C3=CC=C2	b3lyp/6-311+G(d,p)	Anionic	0.1690244	2.985361185
235	C1=CC=C2C(=C1)C=CC3=CC=CC=C32	b3lyp/6-311+G(d,p)	Anionic	0.3118476	4.68144689
236	C1=CC=C2C(=C1)C=CC3=C2C=CC4=C3C=CC5=CC=CC=C54	b3lyp/6-311+G(d,p)	Anionic	0.2766688	4.206880314
237	C1=CC2=C3C(=C1)C4=CC=CC=C4(=C43)C=C2	b3lyp/6-311+G(d,p)	Anionic	0.213962	3.810138302
238	C1=CC=C2C=C3C=C4C=CC=C4C=CC3=CC2=C1	b3lyp/6-311+G(d,p)	Anionic	0.1621176	2.751887491
239	C1=CC2=CC3=C4C5=C(C=C3)C=C6C=CC7=C8C6=C5C9=C3C4=C2C2=C1C=CC1=CC4=C(C=C12)C1=C(C=C4)C=C(C8=C19)C=C7	b3lyp/6-311+G(d,p)	Anionic	0.1439392	2.520318594
240	C1=CC=C2C(=C1)C3=CC=CC=C3C4=CC=CC=C24	b3lyp/6-311+G(d,p)	Anionic	0.2497732	4.839000816
362	C1=CC2=C(C=CC3=C2C=CC4=C3SC=C4)C5=C1C=CC55	b3lyp/6-311+G(d,p)	Anionic	0.2895152	4.358447735
363	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C5=CC=CC=C554	b3lyp/6-311+G(d,p)	Anionic	0.2516456	4.273003982
364	C1=CC=C2C=C3C(=CC2=C1)C4=C(C=C3SC=C5)C5=C4	b3lyp/6-311+G(d,p)	Anionic	0.1844252	3.515983217
365	C1=CC=C(C(=C1)C#CC2=CC=C5S2)C#CC3=CC=C53	b3lyp/6-311+G(d,p)	Anionic	1.96416	3.627277787
366	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)SC5=CC=CC=C54	b3lyp/6-311+G(d,p)	Anionic	0.2094236	4.397632131
367	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C5=C(C=C4)SC=C5	b3lyp/6-311+G(d,p)	Anionic	0.2613176	4.103204932

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
368	C1=CSC2=CC3=CC4=C(C=C5C=CSC5=C4)C=C3C=C21	b3lyp/6-311+G(d,p)	Anionic	0.1624524	2.748077897
369	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C5=CC=CC=C5S4	b3lyp/6-311+G(d,p)	Anionic	0.2608092	4.022659229
370	C1=CC2=C3C=CSC4=CC(=C34)C5=C2C(=C1)SC=C5	b3lyp/6-311+G(d,p)	Anionic	0.1811888	2.354601252
371	C1=CC2=CC3=CC4=CC5=CSSC5=CC4=CC3=CC2=C1	b3lyp/6-311+G(d,p)	Anionic	0.2593832	1.373358664
372	C1=CC=C2C(=C1)C=C3C=C4C=CC=CC4=C5C3=C2SS5	b3lyp/6-311+G(d,p)	Anionic	0.1471756	2.338002306
373	C1=CC2=C(C=CS2)C3=CC4=C(C=C31)C5=C(C=C4)SC=C5	b3lyp/6-311+G(d,p)	Anionic	0.220224	3.535303301
374	C1=CC2=C(C=C3C=CSC3=C2)C4=C1C=C5C(=C4)C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.2634628	3.8846975
375	C1=CC=C2C(=C1)C=CC3=C2SC4=C3SC5=CC=CC=C54	b3lyp/6-311+G(d,p)	Anionic	0.2340252	4.081707937
376	C1=CC=C2C=C3C(=CC2=C1)C4=C(S3)C5=CC=CC=C5S4	b3lyp/6-311+G(d,p)	Anionic	0.2249732	3.62891047
377	C1=CC2=CC3=C(C=CC4=C3SC=C4)C=C2C5=C1C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.1605552	3.502921751
378	C1=CC=C2C(=C1)C=CC3=C2C4=C(S3)C5=CC=CC=C5S4	b3lyp/6-311+G(d,p)	Anionic	0.2499096	3.921432872
379	C1=CC2=C(C=CC3=C2C=CS3)C4=C1C5=C(C=C4)SC=C5	b3lyp/6-311+G(d,p)	Anionic	0.2883248	4.153001769
380	C#CC1=CC=(C=C1C2=CC=CS2)C#C)C3=CC=CS3	b3lyp/6-311+G(d,p)	Anionic	1.0342716	3.863744733
381	C1=CC=C2C(=C1)C3=C(S2)C=C4C=C5C(=CC4=C3)C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.1813996	3.467819063
382	C1=CC2=C(C=CC3=C2C=CC4=C3C=CS4)C5=C1C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.2894532	4.245792596
383	C1=CC=C2C(=C1)C=C(S2)C#CC3=CC4=CC=CC=C4S3	b3lyp/6-311+G(d,p)	Anionic	10.0116236	3.589181846
384	C1=CC2=CC3=C(C=C2C4=C1C=CS4)C5=C(C=C3)C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.1576412	3.506187118
385	C1=CC=C2C(=C1)SC3=C(S2)C4=CC=CC5=C4C3=CC=C5	b3lyp/6-311+G(d,p)	Anionic	0.2880644	3.034885908
386	C#CC1=CC=(C=C1C2=CSC=C2)C#C)C3=CSC=C3	b3lyp/6-311+G(d,p)	Anionic	6.0705316	4.195179418
387	[2H]C1=CC2=C(C=CS2)C3=CC4=C(C=C13)C5=C(C=C4[2H])SC=C5	b3lyp/6-311+G(d,p)	Anionic	0.220224	3.535303301
388	[2H]C#CC1=CC=(C=C1C2=CSC=C2)C#C[2H])C3=CSC=C3	b3lyp/6-311+G(d,p)	Anionic	5.5174792	4.195179418
389	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C=CC5=C4SC=C5	b3lyp/6-311+G(d,p)	Anionic	0.2292636	4.101844363
390	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C=CC5=C4C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.2461896	4.092592491
391	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C5=C(C=C4)C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.2365052	4.209601452
392	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C5=C(C=C4)C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.2594824	4.058850372
393	C1=CC=C2C(=C1)C3=C(S2)C=C4C(=C3)C=CC5=C4SC=C5	b3lyp/6-311+G(d,p)	Anionic	0.1945808	3.851771723
394	C1=CC=C2C(=C1)C3=C(S2)C=C4C(=C3)C=CC5=C4C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.2008676	3.906738723
395	C1=CC=C2C(=C1)C3=C4C(=CC=C3)SC5=CC=CC(=C54)S2	b3lyp/6-311+G(d,p)	Anionic	0.3219536	4.237084952
396	[2H]C1=C2C(=CC=C1)C3=C4C(=CC=C3)SC5=CC=CC(=C54)S2	b3lyp/6-311+G(d,p)	Anionic	0.3220156	4.237084952
397	C1=CC=C2C=C3C(=CC2=C1)C4=C(C=CS4)C5=C3SC=C5	b3lyp/6-311+G(d,p)	Anionic	0.1580504	3.492037197
398	C1=CC2=C(C3=CSC=C3C=C2)C4=C1C=CC5=CSC=C54	b3lyp/6-311+G(d,p)	Anionic	0.2428788	3.737756015
399	C1=CC2=C(C=C2)SC=C3)C4=C1C=CC5=C4C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.2768548	4.239533977
400	C1=CC2=C3C=CC=C4C3=C(C=CS4)C5=C2C(=C1)SC=C5	b3lyp/6-311+G(d,p)	Anionic	0.592224	3.111894131
401	C1=CC=(CC(=C1)C#CC2=CC=CS2)C#CC3=CC=C3	b3lyp/6-311+G(d,p)	Anionic	5.83668	3.905378154
402	C1=CC2=CC3=C(C=C2C4=CC5=C(C=CS5)C=C41)SC=C3	b3lyp/6-311+G(d,p)	Anionic	0.2287428	3.974222961
403	C1=CC=C2C(=C1)C=CC3=C2C=CC4=C3SC5=C4SC=C5	b3lyp/6-311+G(d,p)	Anionic	0.2181532	4.255044467
404	C#CC1=C(C=CS1)C2=CC=C(C=C2)C3=C(SC=C3)C#C	b3lyp/6-311+G(d,p)	Anionic	10.7060608	4.084156962
405	C1=CC=(CC(=C1)C#CC2=CSC=C2)C#CC3=CSC=C3	b3lyp/6-311+G(d,p)	Anionic	13.6600012	4.190825596
406	C1=CC=(CC=C1C#CC2=CSC=C2)C#CC3=CSC=C3	b3lyp/6-311+G(d,p)	Anionic	0.2336656	3.712993653
407	C1=CC=C2C(=C1)C=CC3=C2C4=C(C=CS4)C5=C3SC=C5	b3lyp/6-311+G(d,p)	Anionic	0.1995904	4.096402085
408	C1=CC=C2C=C3C=C4C(=CC3=CC2=C1)C5=C(S4)C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.1639404	3.038423388
409	C1=CC=C(C=C1)C#CC2=CC3=C(S2)C=CC4=C3C=CS4	b3lyp/6-311+G(d,p)	Anionic	2.548262	3.683605356
410	C1=CC2=C(C=C3C=CSC3=C2)C4=CC5=C(C=CS5)C=C41	b3lyp/6-311+G(d,p)	Anionic	0.2631652	4.041707199
411	C1=CC2=C(C=CS2)C3=CC4=C(C=C31)C=C5C=CSC5=C4	b3lyp/6-311+G(d,p)	Anionic	0.1840656	3.136384379
412	C#CC1=CC=(C=C1C2=CSC=C2)C3=CSC=C3)C#C	b3lyp/6-311+G(d,p)	Anionic	0.3195356	4.286609675
413	C1=CC2=C(C=CS2)C3=CC4=C(C=CC5=C4C=CS5)C=C31	b3lyp/6-311+G(d,p)	Anionic	0.2083572	3.569861762

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
414	C1=CC2=CSC=C2C3=CC4=C(C=CC5=CSC=C54)C=C31	b3lyp/6-311+G(d,p)	Anionic	0.1496556	3.740749267
415	C1=CC2=CC3=CSC=C3C=C2C4=CC5=CSC=C5C=C41	b3lyp/6-311+G(d,p)	Anionic	0.182962	3.207133983
416	C1=CC2=CSC=C2C3=CC4=C(C=C31)C5=CSC=C5C=C4	b3lyp/6-311+G(d,p)	Anionic	0.1811516	3.672992915
417	C1=CC2=C(C=CC3=C2C=CC4=CSC=C43)C5=CSC=C51	b3lyp/6-311+G(d,p)	Anionic	0.2625204	3.861567822
418	C1=CC=C(C=C1)C#CC2=CC3=C(S2)C=C4C=CSC4=C3	b3lyp/6-311+G(d,p)	Anionic	0.2927888	3.569589648
419	C1=CC=C2C=C3C(=CC2=C1)C=C4=C3C5=C(S4)C=CS5	b3lyp/6-311+G(d,p)	Anionic	0.2006692	3.38727336
420	C1=CC2=C3C4C1C5=CSC=C5C4=CC=C3C6=CSC=C26	b3lyp/6-311+G(d,p)	Anionic	71.1481124	2.724403991
421	C1=CC=C2C(=C1)C3=C(S2)C4=CC=CC5=C4C(=CC=C5)S3	b3lyp/6-311+G(d,p)	Anionic	0.269018	3.28087684
515	C1=CC=C2C(=C1)C3=C(S2)C4=CC=CC=C4S3	b3lyp/6-311+G(d,p)	Anionic	0.2881264	4.234363814
516	C1=CC2=C(C=CC3=C2SC=C3)C4=C1C=CS4	b3lyp/6-311+G(d,p)	Anionic	0.2149168	4.420217582
517	C1=CC2=C3C(=C1)SSC4=CC=CC(=C43)C=C2	b3lyp/6-311+G(d,p)	Anionic	0.3128644	3.888234981
518	C1=CC=C2C3=C4C(=CC=C3)SSC4=CC2=C1	b3lyp/6-311+G(d,p)	Anionic	0.2900608	3.379382058
519	C1=CSC2=CC3=C(C=C21)C=C4C(=C3)C=CS4	b3lyp/6-311+G(d,p)	Anionic	0.2112836	3.438430766
520	C1=CC=C2C3=C(C4=C(C2=C1)C=CS4)SC=C3	b3lyp/6-311+G(d,p)	Anionic	0.2485952	4.230009992
521	C1=CC2=C3C(=CC=C4C3=C1C=CS4)C=CS2	b3lyp/6-311+G(d,p)	Anionic	0.2691544	3.062913636
522	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)SC=C4	b3lyp/6-311+G(d,p)	Anionic	0.216008	4.569608092
523	C1=CC2=CC3=C4C(=CC=CS4)SC3=C2C=C1	b3lyp/6-311+G(d,p)	Anionic	0.3686148	1.619349595
524	C1=CC=C2C(=C1)C3=C(C4=C2SC=C4)SC=C3	b3lyp/6-311+G(d,p)	Anionic	0.2411428	4.345658383
525	C1=CC=C2C(=C1)C(=S)C3=CC=CC=C3C2=S	b3lyp/6-311+G(d,p)	Anionic	0.2611936	2.456099723
526	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3SC=C4	b3lyp/6-311+G(d,p)	Anionic	0.273668	4.492871983
527	C1=CC=C2C(=C1)C=C(S2)C#CC3=CSC=C3	b3lyp/6-311+G(d,p)	Anionic	0.2887464	3.961161496
528	C1=CC=C2C(=C1)C3=C(S2)SC4=CC=CC=C43	b3lyp/6-311+G(d,p)	Anionic	0.2128708	4.666752741
529	C1=CC=C2C(=C1)C=C(S2)C#CC3=CC=CS3	b3lyp/6-311+G(d,p)	Anionic	0.2880892	3.721157069
530	C1=CC2=C(C=CS2)C3=C1C4=C(C=C3)SC=C4	b3lyp/6-311+G(d,p)	Anionic	0.283526	4.274092438
531	C1=CSC2=CC3=C(C=C21)C=C4C=CSC4=C3	b3lyp/6-311+G(d,p)	Anionic	0.1984124	3.478703618
532	C1=CC2=C3C(=CC=C4C3=C1C=CS4)SC=C2	b3lyp/6-311+G(d,p)	Anionic	0.273358	2.889032878
533	C1=CC2=C(C3=C1C=CC=C3SC=C4)SC=C2	b3lyp/6-311+G(d,p)	Anionic	0.2114076	4.584302241
534	C1=CC=C2C(=C1)C=CC3=C2C4=C(S3)SC=C4	b3lyp/6-311+G(d,p)	Anionic	0.2485456	4.35735928
535	C1=C/C(=C2/C=CC3=C2SC=C3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	Anionic	0.3368708	2.861549378
536	C1=CC=C2C(=C1)C3=CSC(=C23)C4=CC=CS4	b3lyp/6-311+G(d,p)	Anionic	0.2708656	3.845513104
537	C1=CSC(=C1)C#C/C=C\CC2=CC=CS2	b3lyp/6-311+G(d,p)	Anionic	3.1184636	3.231624231
538	C1=CC=C2C(=C1)C3=C(S2)C=C4C=CSC4=C3	b3lyp/6-311+G(d,p)	Anionic	0.2504924	4.159804615
539	C#CC1=C(SC2=CC=CC=C21)C3=CC=CS3	b3lyp/6-311+G(d,p)	Anionic	4.0064276	3.751633822
540	C1=CC=C2C(=C1)C(=C3C=CC=C3C=C2[S-])[S-]	b3lyp/6-311+G(d,p)	Anionic	0.2614044	2.456371836
541	C1=CC2=C3C(=C1)C4=C(C3=CC=C2)SC=CS4	b3lyp/6-311+G(d,p)	Anionic	0.2793224	2.88848865
542	C1=C/C(=C2/C=CC3=C2SC=C3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	Anionic	0.3430584	2.942367195
543	C1=CC=C2C=C3C(=CC2=C1)C4=C(S3)C=CS4	b3lyp/6-311+G(d,p)	Anionic	0.2100064	3.797348951
544	C1=CC=C2C(=C1)C3=C(S2)C4=C(C=C3)C=CS4	b3lyp/6-311+G(d,p)	Anionic	0.2705184	4.427564656
545	C1=CC=C2C(=C1)C3=C(S2)C=CC4=C3C=CS4	b3lyp/6-311+G(d,p)	Anionic	0.2579944	4.378584161
546	C1=CC2=C(C=CS2)C3=C1C=CC4=C3C=CS4	b3lyp/6-311+G(d,p)	Anionic	0.2483472	4.41232628
547	C\1=CC2=C(/C1=C/3/C=CC4=C3C=CS4)C=CS2	b3lyp/6-311+G(d,p)	Anionic	0.414408	2.871073363
548	C1=CC=C2C(=C1)C3=CSC=C3C4=CSC=C24	b3lyp/6-311+G(d,p)	Anionic	0.1420048	4.489334503
549	C1=CC2=C(C3=C1C=CC4=C3C=CS4)SC=C2	b3lyp/6-311+G(d,p)	Anionic	0.245644	4.305113418
550	C1=C/C(=C2/C=CC3=C2C=CS3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	Anionic	0.3724836	2.856923442
551	C1=CC=C2C(=C1)C3=CC=C3C2=C([S-])[S-]	b3lyp/6-311+G(d,p)	Anionic	0.174034	2.188067568
552	C\1=CC2=C(/C1=C/3/C=CC4=C3C=CS4)C=CS2	b3lyp/6-311+G(d,p)	Anionic	0.5758188	2.889849219

	SMILES	Functional/Basis Set	Transition	Reorganisation Energy/eV	HOMO-LUMO gap
553	C1=C/C(=C/2\C=CC3=C2C=CS3)/C4=C1C=CS4	b3lyp/6-311+G(d,p)	Anionic	0.3787332	2.858011897
554	C1=CC(=C2C=CC(=C3C=CC#S3)C=C2)S#C1	b3lyp/6-311+G(d,p)	Anionic	0.4013012	1.545062511
555	C1=CC=C2C(=C1)C3=C(S2)C=CC4=CSC=C43	b3lyp/6-311+G(d,p)	Anionic	0.25327	3.70782349
556	C1=CC=C2C(=C1)C3=C(C=CS3)C4=C2SC=C4	b3lyp/6-311+G(d,p)	Anionic	0.1731908	4.421850265
557	C1=CC=C2C(=C1)C3=C(S2)C4=CSC=CC4=C3	b3lyp/6-311+G(d,p)	Anionic	0.3328408	3.187269671
558	C1=CC=C2C(=C1)C3=CC4=CC=CSC4=C3S2	b3lyp/6-311+G(d,p)	Anionic	0.2778096	2.63487853
559	C1=CC2=C(C=CC3=CSC=C32)C4=CSC=C41	b3lyp/6-311+G(d,p)	Anionic	0.1983628	3.412035721
560	C1=CC2=CSC=C2C3=C1C=CC4=CSC=C43	b3lyp/6-311+G(d,p)	Anionic	0.1637792	3.367681162
561	C1=CC2=CC3=C4C(=CSS4)C=CC3=C2C=C1	b3lyp/6-311+G(d,p)	Anionic	0.2637108	1.640846591
562	C1=CC2=C(C=C3C=CSC3=C2)C4=C1C=CS4	b3lyp/6-311+G(d,p)	Anionic	0.1977552	3.924154011
593	C1=CSC2=CC3=C(C=C21)SC=C3	b3lyp/6-311+G(d,p)	Anionic	0.2929748	4.296405774
594	C1=CC2=C3C(=C1)SSC3=CC=C2	b3lyp/6-311+G(d,p)	Anionic	0.2134412	3.351898558
595	C1=CSC2=CC3=C(C=CS3)C=C21	b3lyp/6-311+G(d,p)	Anionic	0.251658	4.482531656
596	C1=CC2=CSC=C2C3=CSC=C31	b3lyp/6-311+G(d,p)	Anionic	0.3487624	4.38348221
597	C1=CC=C2C(=C1)C3=C(S2)SC=C3	b3lyp/6-311+G(d,p)	Anionic	0.2564692	4.963084737
598	C1=CC2=C(C3=C1C=CS3)SC=C2	b3lyp/6-311+G(d,p)	Anionic	0.2930368	4.715733236
599	C1=CC=C2C(=C1)C3=C(S2)C=CS3	b3lyp/6-311+G(d,p)	Anionic	0.2995344	4.623758751
600	C1=CC2=C(C=CS2)C3=C1C=CS3	b3lyp/6-311+G(d,p)	Anionic	0.3261324	4.879817895
601	C1=CC2=C(C=CS2)C3=C1SC=C3	b3lyp/6-311+G(d,p)	Anionic	0.3090824	4.601717528
602	C1=CC(=S)C=C2C1=CC(=S)C=C2	b3lyp/6-311+G(d,p)	Anionic	0.2937808	1.993506156
603	C1=CC=C2C(=C1)C=CC(=S)C2=S	b3lyp/6-311+G(d,p)	Anionic	0.4457304	2.148338944
604	C1=CC=C2C(=S)C=CC(=S)C2=C1	b3lyp/6-311+G(d,p)	Anionic	0.3226356	2.342900356
605	C=C1C2=C(C3=C1C=CS3)SC=C2	b3lyp/6-311+G(d,p)	Anionic	0.4044756	3.518976469
606	C1=CSC2=CC3=CSC=C3C=C21	b3lyp/6-311+G(d,p)	Anionic	0.2290404	3.294482533
607	C1=CC2=C(C=CS2)C3=CSC=C31	b3lyp/6-311+G(d,p)	Anionic	0.2742508	3.918983847
608	C1=CC2=C3C=CC=C3SSC2=C1	b3lyp/6-311+G(d,p)	Anionic	0.4723904	2.329294663
624	C1=CSC2=C1SC=C2	b3lyp/6-311+G(d,p)	Anionic	0.3677964	5.059957272
625	C1=CC(=S)C=CC1=S	b3lyp/6-311+G(d,p)	Anionic	0.3362508	2.16357732
626	C1=CC(=S)C(=S)C=C1	b3lyp/6-311+G(d,p)	Anionic	0.2937932	1.786971735
627	[2H]C1=C(SC2=C1SC(=C2[2H])[2H])[2H]	b3lyp/6-311+G(d,p)	Anionic	0.3682676	5.059957272
628	C1=C2C=S=CC2=CS1	b3lyp/6-311+G(d,p)	Anionic	0.2594824	2.993252487

APPENDIX IX
COMPLETE CODE

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Mar  4 16:27:52 2024
4
5 @author: kyebchoo
6 """
7 #%%
8 # Importing modules
9
10 # NUMPY
11 import numpy as np
12 from numpy import random
13
14 # SCIPY
15 import scipy as sci
16 import scipy.constants as constants
17 import scipy.stats as stats
18
19 # PANDAS
20 import pandas as pd
21 pd.set_option('display.max_colwidth', None)
22 pd.set_option('display.max_columns', None)
23 pd.set_option('display.max_rows', None)
24
25 # MATPLOTLIB-PYPLOT
26 import matplotlib.pyplot as plt
27 from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)
28
29 # SYSTEM CONTROLS
30 import os
31 import importlib
32 from copy import deepcopy
33 import paramiko
34
35 # RDKit (CHEMISTRY)
36 import rdkit
37 from rdkit import Chem
38 from rdkit.Chem import AllChem
39 from rdkit.Chem import Draw
40 from rdkit.Chem import Descriptors
41 from rdkit.Chem import PandasTools
42 from rdkit.Chem.Draw import IPythonConsole
43 from rdkit.Chem import rdDepictor
44 from rdkit.Chem.Draw import rdMolDraw2D
45
46 # import kora.install.rdkit
47
48 # DATE AND TIME CONTROLS
49 from datetime import datetime, timedelta
50 import time
51
52 # OTHERS
53 from pymongo import MongoClient
54 import urllib.parse
55 import pprint
56 import ast
57 import cclib
58 from pathlib import Path
59 from ase import Atoms
60 import warnings
61 from openbabel import pybel
62 from tqdm import tqdm
63 #%%
64
65 def timestamp() -> str:
66     """
67     Returns the standardised formatted timestamp for use in later methods, in the form (YYYYMMDDHHMMSS)
68
69     Returns
70     -----
71     str
72         Formatted timestamp.
73
74     """
75     return str(datetime.now().split('.')[0].replace('-', '') .replace(' ', '') .replace(':', '') )
76
77 def tag(molecule: str) -> str:
78     """
79     Returns the standardised formmated tag for use in later methods, in the form of {truncated molecule name}_{YYYYMMDDHHMMSS}.
80     The code would automatically remove symbols and truncate the name to 8 letters.
81
82     Parameters
83     -----
84     molecule : str
85         Name of molecule (preferably IUPAC).

```

```

86
87     Returns
88     -----
89     str
90         Standardised tag.
91
92     """
93     molecule = molecule.replace('[', '').replace(']', '').replace('-', '_').replace(',', '_').replace('.', '_').replace('(', '_')
94     tag_name = '%s_%s' % (molecule, timestamp())
95     return tag_name
96
97 def is_notebook() -> bool:
98     """
99     Method to test if the script is being ran in a Jupyter notebook environment.
100
101    Returns
102    -----
103    bool
104        TRUE if Jupyter notebook detected, and FALSE otherwise.
105
106    """
107    try:
108        shell = get_ipython().__class__.__name__
109        if shell == 'ZMQInteractiveShell':
110            return True # Jupyter notebook or qtconsole
111        elif shell == 'TerminalInteractiveShell':
112            return False # Terminal running IPython
113        else:
114            return False # Other type (?)
115    except NameError:
116        return False # Probably standard Python interpreter
117
118 def display_if_notebook(display_item, otherwise: bool = False):
119     """
120     Display or print out the display_item if script is being ran in a Jupyter notebook environment.
121
122     Parameters
123     -----
124     display_item : TYPE
125         Any item, whether it be string or table, to be displayed if in a Jupyter notebook environment.
126     otherwise : bool, optional
127         Whether to display or print out the display_item if not in a Jupyter notebook environment. The default is False.
128
129     Returns
130     -----
131     None.
132
133    """
134    if is_notebook():
135        try:
136            display(display_item) # ignore error if running in Spyder
137        except:
138            pass
139    else:
140        if otherwise:
141            display(display_item) # ignore error if running in Spyder
142
143 def check_directory(directory: str):
144     """
145     Method to check if a directory exists, and to create the directory if it does not exist.
146
147     Parameters
148     -----
149     directory : str
150         Path to directory.
151
152     Returns
153     -----
154     None.
155
156    """
157    if not os.path.exists(directory):
158        os.makedirs(directory)
159        pass
160
161 def convert_atomic_number_to_symbol(atomic_number: int) -> str:
162     """
163     Convert atomic number to atomic symbols representing the elements.
164
165     Parameters
166     -----
167     atomic_number : int
168         Atomic number to be converted.
169
170     Returns

```

```

171 -----
172     str
173         Symbol of the corresponding element.
174
175 """
176     return Chem.GetPeriodicTable().GetElementSymbol(atomic_number)
177
178 def format_basis_set(basis_set: str):
179 """
180     Method to format the basis set, removing invalid characters when being used as part of a filename.
181
182     Parameters
183     -----
184     basis_set : str
185         Basis set input.
186
187     Returns
188     -----
189     TYPE
190         Formatted basis set string.
191
192 """
193     return basis_set.replace('+', 'p').replace('*', 's').replace('(', '(').replace(')', ')').replace(',', ',')
194
195 #%%
196 class Molecule():
197
198     def __init__(self,
199                 molecule_name: str,
200                 molecule_id: str,
201                 SMILES: str,
202                 parent_folder: str = os.getcwd(),
203                 debug: bool = False):
204
205         """
206             Creates a 'Molecule' class for later operations. The molecule could be declared directly from 'Molecule()', or be loaded from a previous documentation with 'Molecule.load_molecule()' .
207
208             The path structure of the system would be:
209                 {parent_folder}/Data/{molecule_id}/{molecule_documents}
210
211         Parameters
212         -----
213         molecule_name : str
214             Name of the molecule, preferably IUPAC name.
215         molecule_id : str
216             A unique ID for this specific molecule. File and document naming would be dependent on this ID and thus it must be unique.
217         SMILES : str
218             SMILES string for the molecule.
219         parent_folder : str, optional
220             The main folder which you would be working in. All subfolders and files would be generated under the parent folder.
221             The default is os.getcwd().
222         debug : bool, optional
223             Toggle to print additional lines during operation for debugging. The default is False.
224
225         Returns
226         -----
227         None.
228
229         """
230         self._molecule_name = molecule_name
231         self._molecule_id = molecule_id
232         self._SMILES = SMILES
233         self._parent_folder = parent_folder
234         self._debug = debug
235
236         check_directory("%s/Data/%s/" % (self._parent_folder, self._molecule_id))
237
238     @classmethod
239     def load_molecule(cls,
240                     molecule_id: str,
241                     parent_folder: str = os.getcwd(),
242                     debug: bool = False):
243
244         """
245             Alternative initialisation method where the key information is extracted from a previously generated documentation
246
247             Only the molecule ID and the parent folder has to be declared, additional entries would be searched within the parent folder.
248
249             Parameters
250             -----
251             molecule_id : str
252                 Molecule ID specific to the molecule. This should be previously generated when the molecule was first initialised.
253                 If the molecule has not been generated before, the molecule has to be initialised the ordinary way.

```

```

parent_folder : str, optional
    The parent folder where all further files and folders are generated and located. The default is os.getcwd().
debug : bool, optional
    Toggle to print additional lines during operation for debugging. The default is False.

Raises
-----
Exception
    Fail to read documentation. Check if molecule ID is correct, or that the respective files are located in the
correct location.

Returns
-----
list
    Required entries to load molecule.

"""
if debug:
    print('\nLoading molecule from %s/%s/%s_documentation.txt' % (parent_folder, molecule_id, molecule_id))
try:
    with open("%s/Data/%s/%s_documentation.txt" % (parent_folder, molecule_id, molecule_id), "r") as documentation:
        content = documentation.read().split('\n')
        molecule_name = content[0]
        molecule_id = content[1]
        SMILES = content[2]
        parent_folder = content[3]
    return cls(molecule_name, molecule_id, SMILES, parent_folder, debug)
except:
    raise Exception('Failed to read documentation from %s/%s/%s_documentation.txt' % (parent_folder, molecule_id, molecule_id))

def print_if_debug(self, lines: str):
    """
    Prints out the lines of in debug mode.

    Parameters
    -----
    lines : str
        Lines to be printed out.

    Returns
    -----
    None.

    """
    if self._debug:
        print(lines)

def create_molecule_documentation(self, allow_aromatic_bond: bool = True) -> None:
    """
    Creates molecule documentation so that future initialisation can be done with just specifying the molecule ID.
    The following documents would be generated under parent_folder/Data/molecule_ID:
        (1) Text (.txt) document containing the initialisation lines.
        (2) Excel (.xlsx) file containing the initial geometry and also connectivity of the atoms within the molecule.
            The connectivity would include aromatic bonds of order 1.5. To disable this, set allow_aromatic_bond to
False.
        Single and double bonds would be assigned by RDKit Kekulisation.

    Parameters
    -----
    allow_aromatic_bond : bool, optional
        Allows aromatic bonds of order 1.5 to be generated in the connectivity document. The default is True.

    Returns
    -----
    None.

    """
    self.print_if_debug('\nIn function Molecule.create_molecule_documentation():')
    molecule = Chem.AddHs(Chem.MolFromSmiles(self._SMILES))
    kekulized_molecule = Chem.AddHs(Chem.MolFromSmiles(self._SMILES))
    Chem.Kekulize(kekulized_molecule)
    if self._debug:
        display_if_notebook(molecule, otherwise = True)
    dataframe = pd.DataFrame(columns = ['Index', 'Atomic Number', 'Element', 'Hybridisation', 'Neighbours[Index, Atomic
Number, Element, Bond, Bond Order]'])
    for atom in molecule.GetAtoms():
        index = atom.GetIdx()
        atomic_number = atom.GetAtomicNum()
        element = atom.GetSymbol()
        hybridisation = str(Chem.rdcchem.HybridizationType.values[atom.GetHybridization()]).split(',')[-1]
        self.print_if_debug('Atom with index: %s\n%s\n%s\nNeighbours:' % (index, atomic_number, element))
        neighbours = []
        for neighbor in atom.GetNeighbors():
            neighbor_index = neighbor.GetIdx()

```

```

333         neighbor_atomic_number = neighbor.GetAtomicNum()
334         neighbor_symbol = neighbor.GetSymbol()
335         neighbor_bond_type = str(molecule.GetBondBetweenAtoms(atom.GetIdx(), neighbor.GetIdx()).GetBondType())
336         if allow_aromatic_bond == True:
337             neighbor_bond_order = str(molecule.GetBondBetweenAtoms(atom.GetIdx(), neighbor.GetIdx()).GetBondTypeAsDouble())
338         else:
339             neighbor_bond_order = str(kekulized_molecule.GetBondBetweenAtoms(atom.GetIdx(), neighbor.GetIdx()).GetBondTypeAsDouble())
340         neighbors.append([neighbor_index, neighbor_atomic_number, neighbor_symbol, neighbor_bond_type,
341         neighbor_bond_order])
342         self.print_if_debug(' %s\n %s\n %s\n %s' % (str(neighbor_index), str(neighbor_atomic_number),
343         neighbor_symbol, str(neighbor_bond_type), str(neighbor_bond_order)))
344     dataframeloc[len(dataframe)] = [index, atomic_number, element, hybridisation, neighbors]
345     if self._debug:
346         display_if_notebook(dataframe, otherwise = True)
347     dataframe.to_excel(r'%s/Data/%s_%s_documentation.xlsx' % (self._parent_folder, self._molecule_id, self._molecule_id))
348
349     with open("%s/Data/%s_%s_documentation.txt" % (self._parent_folder, self._molecule_id, self._molecule_id), "w") as documentation:
350         documentation.write("%s\n%s\n%s\n%s\nCreated on %s" % (self._molecule_name, self._molecule_id, self._SMILES,
351         self._parent_folder, str(datetime.now())))
352         self.print_if_debug('Molecule documentation created sucessfully.')
353
354 def get_SMILES(self) -> str:
355     """
356     Returns the SMILES string of the molecule.
357
358     Returns
359     -----
360     str
361         SMILES string of the molecule.
362
363     """
364     self.print_if_debug('\nIn function Molecule.get_SMILES():')
365     return self._SMILES
366
367 def get_molecule_name(self) -> str:
368     """
369     Returns the name of the molecule.
370
371     Returns
372     -----
373     str
374         Name of the molecule.
375
376     """
377     self.print_if_debug('\nIn function Molecule.get_molecule_name():')
378     return self._molecule_name
379
380 def get_molecule_id(self) -> str:
381     """
382     Returns the ID of the molecule.
383
384     Returns
385     -----
386     str
387         ID of the molecule.
388
389     """
390     self.print_if_debug('\nIn function Molecule.get_molecule_id():')
391     return self._molecule_id
392
393 def generate_xyz_from_rdkit_mol(self,
394                                     random_low: float = 0.00,
395                                     random_high: float = 0.00) -> list[str, int]:
396     """
397     Generates the cartesian x,y,z coordinates using built-in RDKit functions.
398     The molecule is optimised using Merck molecular force field methods.
399
400     The returned xyz coordinate block DOES NOT terminate with two blank lines.
401
402     Returns
403     -----
404     list[str, int]
405         List with the [0] first entry being the xyz coordinates, and
406         the [1] second entry being the total number of atoms in the molecule.
407
408     """
409     self.print_if_debug('\nIn function Molecule.generate_xyz_from_rdkit_mol():')
410     Mol = Chem.AddHs(Chem.MolFromSmiles(self._SMILES))
411     AllChem.EmbedMolecule(Mol)
412     AllChem.MMFFOptimizeMolecule(Mol, maxIters = 1000)
413
414     for c in Mol.GetConformers():

```

```

412     # only single conformer is chosen currently, might alter in future
413     positions = c.GetPositions()
414     atoms_symbols = np.array([at.GetSymbol() for at in Mol.GetAtoms()])
415     atoms = Atoms(atoms_symbols, positions=positions)
416     num_atoms: int = len(atoms)
417     types = atoms.get_chemical_symbols()
418     all_atoms = zip(types, atoms.get_positions())
419     a_str = ""
420     for atom in all_atoms:
421         if random_low == 0.00 and random_high == 0.00:
422             a_str += atom[0] + " " + ".join([str(x) for x in atom[1]]) + "\n"
423         elif random_low != random_high and random_low < random_high:
424             a_str += atom[0] + " " + ".join([str(x + random.uniform(low = random_low, high = random_high)) for x in atom[1]]) + "\n"
425         else:
426             raise Exception('Invalid random_low and/or random_high input.')
427     # a_str = a_str + "\n\n"
428     self.print_if_debug(a_str)
429     return a_str, num_atoms
430
431 def generate_xyz_from_pymol_mol(self, pbmol) -> list[str, int]:
432     """
433     Generates the cartesian x,y,z coordinates from a pymol object.
434
435     Parameters
436     -----
437     pbmol : TYPE
438         pymol object.
439
440     Returns
441     -----
442     list[str, int]
443         List with the [0] first entry being the xyz coordinates, and
444         the [1] second entry being the total number of atoms in the molecule.
445
446     """
447     self.print_if_debug('\nIn function Molecule.generate_xyz_from_pymol_mol():')
448     warnings.warn('This method has not been tested. If the xyz block generated has a double-blank line termination, it
449     should be truncated as it would affect future processes.')
450     # read mol and outputs xyz with a small optimisation
451     pbmol.write("xyz", r"%s/Data/%s/%s_pybel_buffer.xyz" % (self._parent_folder, self._molecule_id, self._molecule_id),
452     , overwrite = True)
453     with open(r"%s/Data/%s/%s_pybel_buffer.xyz" % (self._parent_folder, self._molecule_id, self._molecule_id), "r") as
454     file:
455         split: list = file.read().split("\n\n")
456         pybabelxyz: str = split[1]
457         num_atoms: int = int(split[0])
458         self.print_if_debug(pybabelxyz)
459         return pybabelxyz, num_atoms
460
461 def generate_xyz_from_xyz_file(self, filename: str) -> list[str, int]:
462     """
463     Generates the cartesian x,y,z coordinates from a .xyz file. The filename should contain the filename with the .xyz
464     suffix.
465     The file should also be located in the molecule file under the folder {parent_folder}/Data/{molecule_id}/.
466
467     Parameters
468     -----
469     filename : str, optional
470         Filename of the xyz file with the suffix (.xyz).
471
472     Returns
473     -----
474     list[str, int]
475         List with the [0] first entry being the xyz coordinates, and
476         the [1] second entry being the total number of atoms in the molecule.
477
478     """
479     self.print_if_debug('\nIn function Molecule.generate_xyz_from_xyz_file():')
480     warnings.warn('This method has not been tested. If the xyz block generated has a double-blank line termination, it
481     should be truncated as it would affect future processes.')
482     if filename == '':
483         filename = r"%s/Data/%s/%s" % (self._parent_folder, self._molecule_id, self._molecule_id)
484     with open(filename, "r") as file:
485         lines: str = file.readlines()[2:]
486         mol_xyz: str = ''.join(lines)
487         num_atoms: int = len(lines)
488         # mol_xyz: str = mol_xyz + "\n\n"
489         mol_xyz: str = mol_xyz
490         self.print_if_debug(mol_xyz)
491         return mol_xyz, num_atoms
492
493 def generate_xyz_from_log_file(self,
494                               filename: str,
495                               random_low: float = 0.0000,
496                               random_high: float = 0.0001) -> list[str, int]:

```

```

492 """
493     Generates the cartesian x,y,z coordinates from a Gaussian16 .log output file. The filename should contain the
494     filename with the .log suffix.
495     The file should also be located in the molecule file under the folder {parent_folder}/Data/{molecule_id}/.
496
497     Parameters
498     -----
499     filename : str
500         Filename of the log file with the suffix (.log).
501
502     Returns
503     -----
504     list[str, int]
505         List with the [0] first entry being the xyz coordinates, and
506         the [1] second entry being the total number of atoms in the molecule.
507
508     """
509     self.print_if_debug('\nIn function Molecule.generate_xyz_from_log_file():')
510     # generate XYZ string from calculation output
511     gaussian_data = cclib.io.ccread(r'%s/Data/%s/%s' % (self._parent_folder, self._molecule_id, filename))
512     gaussian_data = cclib.io.ccread(r'%s' % (filename))
513     # generate xyz file from gaussian log output
514     num_atoms: int = gaussian_data.natom
515     coords = gaussian_data.atomcoords
516     coords = coords[-1, :, :]
517     atomnos = gaussian_data.atommno
518     xyz_str_gauss = ""
519     for i in range(0, len(coords)):
520         xyz_str_gauss += (str(convert_atomic_number_to_symbol(int(atommno[i]))))
521         + " "
522         + " ".join([str(float(x)) + random.uniform(low = random_low, high = random_high) for x in
523 coords[i]])
524         + "\n"
525     self.print_if_debug("There are %i atoms and %i MOs" % (gaussian_data.natom, gaussian_data.nmo))
526     self.print_if_debug("Obtained opt geometry from optimised log file for: \n%s" % (filename))
527     self.print_if_debug(xyz_str_gauss)
528     return xyz_str_gauss, num_atoms
529
530 def generate_molecule_xyz(self,
531     input_object = '',
532     name_extension: str = '',
533     save: bool = True,
534     random_low: float = 0.0000,
535     random_high: float = 0.0001) -> str:
536
537     """
538     Generates molecule xyz by detecting the input_object. The input_object can either be:
539     1) an rdkit mol class
540     2) a pymol class
541     3) an xyz file (located within the molecule file) with the structure already in it
542     4) a Gaussian16 output .log file with previous calculation of the molecule
543
544     If in the form of a filename, the file must be located within the folder:
545         {parent_folder}/Data/{molecule_id}/
546
547     The {name_extension} is the suffix to append at the end of the xyz filename: {molecule_id}_{name_extension}.xyz.
548
549     Parameters
550     -----
551     input_object : TYPE, optional
552         Input object of the form of either (1) an RDKit molecule, (2) pymol molecule, (3) xyz file, or (4) .log
553         Gaussian16 output file.
554         The default is '', where the geometry from the SMILES-generated RDKit molecule would be used.
555     name_extension : str, optional
556         Optional extension to be added to the end of the .xyz file. The default is ''.
557     save : bool, optional
558         Toggle to save the xyz geometry as a .xyz file. The default is True.
559
560     Returns
561     -----
562     str
563         Molecule xyz block.
564
565     """
566     self.print_if_debug('\nIn function Molecule.generate_molecule_xyz():')
567     # 1) an rdkit mol class
568     if type(input_object) is Chem.rdcchem.Mol or (input_object == ''):
569         a_str_gauss, num_atoms = self.generate_xyz_from_rdkit_mol(random_low = random_low,
570                                         random_high = random_high)
571         self.print_if_debug('RDKit molecule geometry used.')
572     # 2) a pymol class
573     try:
574         from openbabel import pybel as pb
575         if type(input_object) is pb.Molecule:
576             a_str_gauss, num_atoms = self.generate_xyz_from_pymol_mol(pbmol = input_object)
577             self.print_if_debug('Pymol molecule geometry used.')
578     except ImportError:
579

```

```

575     print("openbabel not found")
576     pass
577
578     if type(input_object) is str:
579         if Path(input_object).is_file():
580             # 3) path to an xyz file with the structure already in it
581             if Path(input_object).suffix == ".xyz":
582                 a_str_gauss, num_atoms = self.generate_xyz_from_xyz_file(filename = input_object)
583                 self.print_if_debug('Geometry extracted from %s.' % input_object)
584             # 4) path to a file with previous calculation of the molecule
585             elif Path(input_object).suffix == ".log":
586                 a_str_gauss, num_atoms = self.generate_xyz_from_log_file(filename = input_object,
587                                                               random_low = random_low,
588                                                               random_high = random_high)
589                 self.print_if_debug('Geometry extracted from %s.' % input_object)
590             else:
591                 raise Exception('Input in generate_molecule_xyz not recognised.\n%s' % (input_object))
592     if save:
593         if name_extension != '':
594             name_extension = '_%s' % (name_extension)
595             with open(r"%s/Data/%s/%s.sxyz" % (self._parent_folder, self._molecule_id, self._molecule_id, name_extension),
596                       "wt") as textfile:
597                 a_str = "%s\n" % (str(num_atoms)) + "%s\n" % (self._molecule_name) + a_str_gauss
598                 textfile.write(a_str)
599                 self.print_if_debug('xyz geometry saved as; %s_%s.sxyz.' % (self._molecule_id, name_extension))
600     return a_str_gauss
601
602 def keyword_line_generator(self,
603                             functional_basisset: str,
604                             calculation_type: str = 'OptFreq',
605                             state_TD: int = 0,
606                             restart: bool = False,
607                             higher_max_cycles: int = 0,
608                             set_trust_radius_bohr: float = 0,
609                             explicitly_specify_connectivity: bool = True,
610                             integral: str = 'SuperFine',
611                             overwrite_Freq: list[list, list] = [[], []],
612                             overwrite_Opt: list[list, list] = [[], []],
613                             overwrite_NoSymm: list[list, list] = [[], []],
614                             overwrite_state_TD: list[list, list] = [[], []],
615                             overwrite_Geom: list[list, list] = [[], []],
616                             overwrite_Integral: list[list, list] = [[], []]
617                             ) -> str:
618 """
619 Method to write the keyword line for the Gaussian16 script. Supports by default the methods:
620 (1) OptFreq:
621     Optimise molecule followed by a frequency calculation.
622 (2) Freq:
623     Frequency calculation only.
624 (3) Manual:
625     Manually input and specify keywords. For example to perform an optimisation job...
626     ...with increased MaxCycles of 200 and reduced MaxStep of 0.10Bohr:
627         overwrite_Opt = [[True, 'Opt'], ['MaxCycles=200', 'MaxStep=10']]
628     ...as a restart job:
629         overwrite_Opt = [[True, 'Opt'], ['Restart']]
630 Overwrite lines would overwrite any inputs generated beforehand.
631
632 Parameters
633 -----
634     functional_basisset : str
635         Functional and basisset in the form of <functional>/<basis_set>, e.g. B3LYP/6-31G(d,p).
636     calculation_type : str, optional
637         Calculation choice of 'OptFreq', 'Opt', or 'Manual'. The default is 'OptFreq'.
638     state_TD : int, optional
639         Excitation state of the molecule. The default is 0.
640     restart : bool, optional
641         Whether to include the Restart keyword under the Opt() line. The default is False.
642     higher_max_cycles : int, optional
643         Manually input an integer number of max cycles. The default is 0, where the Gaussian16 default would be used.
644     set_trust_radius_bohr : float, optional
645         Manually set the maximum trust radius (step size) in units of Bohr for the optimisation.
646         The default is 0, where the Gaussian16 default would be used.
647     explicitly_specify_connectivity : bool, optional
648         Explicitly specify bonds and connectivity. The default is True.
649     overwrite_Freq : list[list, list], optional
650         Overwrite line for Freq keyword. The default is [[], []] for no overwrite.
651     overwrite_Opt : list[list, list], optional
652         Overwrite line for Opt keyword. The default is [[], []] for no overwrite.
653     overwrite_NoSymm : list[list, list], optional
654         Overwrite line for NoSymm keyword. The default is [[], []] for no overwrite.
655     overwrite_state_TD : list[list, list], optional
656         Overwrite line for TD keyword. The default is [[], []] for no overwrite.
657     overwrite_Geom : list[list, list], optional
658         Overwrite line for Geom keyword. The default is [[], []] for no overwrite.
659
660 Raises

```

```

660 -----
661     ValueError
662         Calculation type not recognised, check 'calculation_type' input.
663
664     Returns
665 -----
666     str
667         Keyword line to be passed into the method/function generating the Gaussian runscripts..
668
669 """
670 self.print_if_debug('\nIn function Molecule.keyword_line_generator():')
671 # keyword_line: list[bool, list[]] = [bool, arguments]
672 opt_line = [[False, 'Opt'], []]
673 freq_line = [[False, 'Freq'], []]
674 nosymm_line = [[False, 'NoSymm'], []]
675 TD_line = [[False, 'TD'], []]
676 Geom_line = [[False, 'Geom'], []]
677 integral_line = [[False, 'Integral'], []]
678
679 keyword_line = '#P'
680
681 if calculation_type.lower() == 'optfreq' or calculation_type.lower() == 'freqopt':
682     opt_line[0][0] = True
683     freq_line[0][0] = True
684     nosymm_line[0][0] = True
685 elif calculation_type.lower() == 'opt':
686     opt_line[0][0] = True
687     nosymm_line[0][0] = True
688 elif 'marcus':
689     nosymm_line[0][0] = True
690     Geom_line = [[True, 'Geom'], ['check']]
691     explicitly_specify_connectivity = False
692     restart = False
693     higher_max_cycles = 0
694     set_trust_radius_bohr = 0
695     # state_TD = 0
696 elif calculation_type.lower() == 'manual':
697     print('Using manual input to generate keyword line for Gaussian16.')
698 else:
699     raise ValueError('\nCalculation type not recognised: %s' % (calculation_type))
700
701 if higher_max_cycles != 0:
702     opt_line[1].append('MaxCycles=%s' % (higher_max_cycles))
703
704 if set_trust_radius_bohr != 0:
705     set_trust_radius_bohr = round(set_trust_radius_bohr, 2)
706     max_step = int(set_trust_radius_bohr/0.01)
707     opt_line[1].append('MaxStep=%s' % (max_step))
708
709 if state_TD != 0:
710     TD_line[0][0] = True
711     TD_line[1] = ['NStates=10', 'Root=%s' % (int(state_TD))]
712
713 if restart == True:
714     opt_line[1].append('Restart')
715     Geom_line[0][0] = True
716     Geom_line[1].append('check')
717
718 if explicitly_specify_connectivity == True:
719     Geom_line = [[True, 'Geom'], ['Connectivity']]
720
721 if integral.lower() == 'superfine':
722     integral_line = [[True, 'Integral'], ['SuperFine']]
723 elif integral.lower() == 'ultrafine':
724     integral_line = [[True, 'Integral'], ['UltraFine']]
725 else:
726     integral_line = [[True, 'Integral'], ['UltraFine']]
727     warnings.warn('Integral not recognised, switching to default = UltraFine')
728
729 if overwrite_Freq != [[], []]:
730     freq_line = overwrite_Freq
731     warnings.warn('Overwriting Freq line.')
732 if overwrite_Opt != [[], []]:
733     opt_line = overwrite_Opt
734     warnings.warn('Overwriting Opt line.')
735 if overwrite_NoSymm != [[], []]:
736     nosymm_line = overwrite_NoSymm
737     warnings.warn('Overwriting NoSymm line.')
738 if overwrite_state_TD != [[], []]:
739     TD_line = overwrite_state_TD
740     warnings.warn('Overwriting TD line.')
741 if overwrite_Geom != [[], []]:
742     Geom_line = overwrite_Geom
743     warnings.warn('Overwriting Geom line.')
744 if overwrite_Integral != [[], []]:
745     integral_line = overwrite_Integral

```

```

746     warnings.warn('Overwriting Geom line.')
747
748     self.print_if_debug('freq_line: %s\nopt_line: %s\nnosymm_line: %s\nTD_line: %s\nGeom_line: %s' % (freq_line,
749     opt_line, nosymm_line, TD_line, Geom_line))
750
751     def keyword_line_argument_writer(keyword: list[[bool, str], list]) -> str:
752         """
753             Method to format the keyword array blocks into proper lines.
754
755             Parameters
756             -----
757             keyword : list[[bool, str], list]
758                 Keyword array block.
759
760             Raises
761             -----
762             Exception
763                 Keyword_line_argument_writer failed to write arguments, check input list.
764
765             Returns
766             -----
767             str
768                 Keyword line.
769
770             """
771     self.print_if_debug('\nIn function keyword_line_argument_writer():')
772     if keyword[0][0] == True:
773         keyword_line_argument = '%s' % (keyword[0][1])
774
775         arguments: str = ''
776         for i in range(0, len(keyword[1])):
777             if arguments == '':
778                 arguments = '%s' % (keyword[1][i])
779             else:
780                 arguments = arguments + ', ' + '%s' % (keyword[1][i])
781         if arguments != '':
782             keyword_line_argument = keyword_line_argument + '(' + arguments + ')'
783     return keyword_line_argument
784 elif keyword[0][0] == False:
785     return ''
786 else:
787     raise Exception('\nKeyword_line_argument_writer failed to write arguments for %s' % (keyword))
788
789 for keyword in [opt_line, freq_line, nosymm_line, TD_line, Geom_line, integral_line]:
790     if keyword_line_argument_writer(keyword = keyword) != '':
791         keyword_line = keyword_line + ' ' + keyword_line_argument_writer(keyword = keyword)
792     keyword_line = keyword_line + ' ' + functional_basisset
793     self.print_if_debug('Keyword line: %s' % (keyword_line))
794
795 def get_connectivity(self) -> str:
796     """
797         Returns the connectivity of the bonds in a molecule as a block to be passed into the Gaussian16 runscript.
798
799         Returns
800         -----
801         write_text : str
802             Connectivity text block.
803
804         """
805     self.print_if_debug('\nIn function Molecule.get_connectivity():')
806     dataframe = pd.read_excel(r'%s/Data/%s/%s_documentation.xlsx' % (self._parent_folder, self._molecule_id, self._molecule_id))
807     if self._debug:
808         display_if_notebook(dataframe, otherwise = True)
809     write_text: str = ''
810     for index in range(0, len(dataframe)):
811         parent_index = dataframe.at[index, 'Index'] + 1
812         line = '%s' % (parent_index)
813         neighbors = dataframe.at[index, 'Neighbors[Index, Atomic Number, Element, Bond, Bond Order]']
814         neighbors = ast.literal_eval(neighbors)
815         for i in range(0, len(neighbors)):
816             neighbor_index = neighbors[i][0] + 1
817             bond_type = neighbors[i][4]
818             line = line + ' ' + str(neighbor_index) + ' ' + str(bond_type)
819             # print(line)
820             if write_text == '':
821                 write_text = line
822             else:
823                 write_text = write_text + '\n' + line
824     write_text = write_text
825     self.print_if_debug(write_text)
826     with open(r'%s/Data/%s/%s_connectivity.txt' % (self._parent_folder, self._molecule_id, self._molecule_id), 'wt') as textfile:
827         textfile.write(write_text)
828         textfile.close()

```

```

829     return write_text
830
831     def generate_gjf_file(self,
832         input_object,
833         excited_state: int,
834         charge: int,
835         functional: str,
836         basisset: str,
837         number_of_proc: int,
838         memory: int,
839         calculation_type: str = 'OptFreq',
840         connectivity: bool = True,
841         keyword_line_arguments_input: list[[str, str]] = []) -> str:
842     """
843     Method to write the Gaussian runscript.
844
845     Parameters
846     -----
847     input_object : TYPE
848         Input object to be fed into Molecule.generate_molecule_xyz().
849     excited_state : int
850         Excitation state.
851     charge : int
852         Charge of molecule.
853     functional : str
854         Functional to be used in DFT calculation.
855     basisset : str
856         Basis set to be used in DFT calculation.
857     number_of_proc : int
858         Number of processors.
859     memory : int
860         Total amount of memory to be divided between the processors.
861     calculation_type : str, optional
862         Calculation type. The default is 'OptFreq'.
863     connectivity : bool, optional
864         Whether to use molecule connectivity. The default is True.
865     keyword_line_arguments_input : list[[str, str]], optional
866         Overwrite lines. The default is [].
867
868     Returns
869     -----
870     gjf_file : str
871         Gaussian16 job script.
872
873     """
874     self.print_if_debug('\nIn function Molecule.generate_gjf_file():')
875     functional_basisset = '%s/%s' % (functional, basisset)
876     keyword_line_arguments: dict = {'functional_basisset': functional_basisset,
877                                     'calculation_type': calculation_type,
878                                     'state_TD': excited_state,
879                                     'restart': False,
880                                     'higher_max_cycles': int(1024),
881                                     'set_trust_radius_bohr': float(0.30),
882                                     'explicitly_specify_connectivity': connectivity,
883                                     'overwrite_Freq': [[], []],
884                                     'overwrite_Opt': [[], []],
885                                     'overwrite_NoSymm': [[], []],
886                                     'overwrite_state_TD': [[], []]}
887
888     # keyword_line_arguemnts_input = [['calculation_type', 'Opt'], ['restart', True], ['higher_max_cycles', 1024]]
889     for i in range(0, len(keyword_line_arguments_input)):
890         keyword_line_arguments['%s' % (keyword_line_arguments_input[i][0])] = keyword_line_arguments_input[i][1]
891
892     keyword_line = self.keyword_line_generator(functional_basisset = keyword_line_arguments['functional_basisset'],
893                                              calculation_type = keyword_line_arguments['calculation_type'],
894                                              state_TD = keyword_line_arguments['state_TD'],
895                                              restart = keyword_line_arguments['restart'],
896                                              higher_max_cycles = keyword_line_arguments['higher_max_cycles'],
897                                              set_trust_radius_bohr = keyword_line_arguments['set_trust_radius_bohr'],
898                                              ),
899     explicitly_specify_connectivity = keyword_line_arguments[''
900     explicitly_specify_connectivity'],
901     overwrite_Freq = keyword_line_arguments['overwrite_Freq'],
902     overwrite_Opt = keyword_line_arguments['overwrite_Opt'],
903     overwrite_NoSymm = keyword_line_arguments['overwrite_NoSymm'],
904     overwrite_state_TD = keyword_line_arguments['overwrite_state_TD'])
905
906     name_extension = '%s_%s_C%sS%s' % (functional, format_basis_set(basisset), charge, excited_state)
907     Mol_xyz = self.generate_molecule_xyz(input_object = input_object,
908                                         name_extension = name_extension,
909                                         save = True)
910
911     gjf_file = "%s_%s" % (self._molecule_id, name_extension)
912     title_line = '%s %s' % (gjf_file, keyword_line_arguments['calculation_type'])

```

```

913     a_str_gaussian = (f"%nprocshared={number_of_proc} \n" +
914         "\n" +
915         f"%mem={memory}GB \n" +
916         f"%chk={gjf_file}.chk \n" +
917         keyword_line + "\n" +
918         "\n" +
919         title_line + "\n"
920         "\n" +
921         "%s %s\n" % (charge, abs(charge) + 1) +
922         Mol_xyz + '\n')
923
924     if connectivity == True:
925         a_str_gaussian = (a_str_gaussian +
926             self.get_connectivity() + '\n' +
927             '\n')
928
929     self.print_if_debug('\nGaussian file:\n=====\\n%s\\n=====\\n' % (a_str_gaussian))
930     with open('%s/Data/%s.%gjf' % (self._parent_folder, self._molecule_id, gjf_file), mode = "wt") as writefile:
931         writefile.write(a_str_gaussian)
932
933     return gjf_file
934
935 def generate_gjf_restart_file(self,
936     excited_state: int,
937     charge: int,
938     functional: str,
939     basisset: str,
940     number_of_proc: int,
941     memory: int,
942     calculation_type: str = 'OptFreq',
943     keyword_line_arguments_input: list[[str, str]] = []):
944     self.print_if_debug('\\nIn function Molecule.generate_gjf_file():')
945     functional_basisset = '%s/%s' % (functional, basisset)
946     keyword_line_arguments: dict = {'functional_basisset': functional_basisset,
947         'calculation_type': calculation_type,
948         'state_TD': excited_state,
949         'restart': True,
950         'higher_max_cycles': int(1024),
951         'set_trust_radius_bohr': float(0.30),
952         'explicitly_specify_connectivity': False,
953         'overwrite_Freq': [[], []],
954         'overwrite_Opt': [[], []],
955         'overwrite_NoSymm': [[], []],
956         'overwrite_state_TD': [[], []]}
957
958     # keyword_line_arguements_input = [['calculation_type', 'Opt'], ['restart', True], ['higher_max_cycles', 1024]]
959     for i in range(0, len(keyword_line_arguments_input)):
960         keyword_line_arguments['%s' % (keyword_line_arguments_input[i][0])] = keyword_line_arguments_input[i][1]
961
962     keyword_line = self.keyword_line_generator(functional_basisset = keyword_line_arguments['functional_basisset'],
963         calculation_type = keyword_line_arguments['calculation_type'],
964         state_TD = keyword_line_arguments['state_TD'],
965         restart = keyword_line_arguments['restart'],
966         higher_max_cycles = keyword_line_arguments['higher_max_cycles'],
967         set_trust_radius_bohr = keyword_line_arguments['set_trust_radius_bohr'],
968     ],
969     explicitly_specify_connectivity = keyword_line_arguments['explicitly_specify_connectivity'],
970     overwrite_Freq = keyword_line_arguments['overwrite_Freq'],
971     overwrite_Opt = keyword_line_arguments['overwrite_Opt'],
972     overwrite_NoSymm = keyword_line_arguments['overwrite_NoSymm'],
973     overwrite_state_TD = keyword_line_arguments['overwrite_state_TD'])
974
975     name_extension = '%s_%s_C%sS%s' % (functional, format_basis_set(basisset), charge, excited_state)
976
977     gjf_file = "%s_%s" % (self._molecule_id, name_extension)
978
979     title_line = '%s %s' % (gjf_file, keyword_line_arguments['calculation_type'])
980
981     a_str_gaussian = (f"%nprocshared={number_of_proc} \n" +
982         "\n" +
983         f"%mem={memory}GB \n" +
984         f"%chk={gjf_file}.chk \n" +
985         keyword_line + "\n" +
986         "\n" +
987         title_line + "\n"
988         "\n" +
989         "%s %s\n" % (charge, abs(charge) + 1))
990
991     self.print_if_debug('\nGaussian file:\n=====\\n%s\\n=====\\n' % (a_str_gaussian))
992     with open('%s/Data/%s.%gjf' % (self._parent_folder, self._molecule_id, gjf_file), mode = "wt") as writefile:
993         writefile.write(a_str_gaussian)
994
995     return gjf_file
996
997 def generate_gjf_marcus_file(self,

```

```

997         charge_geom: int,
998         charge_eval: int,
999         state_geom: int,
1000        state_eval: int,
1001        functional: str,
1002        basisset: str,
1003        number_of_proc: int,
1004        memory: int,
1005        keyword_line_arguments_input: list[[str, str]] = [] -> str:
1006    self.print_if_debug('\nIn function Molecule.generate_gjf_marcus_file():')
1007    functional_basisset = '%s/%s' % (functional, basisset)
1008    keyword_line_arguments: dict = {'functional_basisset': functional_basisset,
1009                                    'calculation_type': 'marcus',
1010                                    'state_TD': state_eval,
1011                                    'restart': False,
1012                                    'overwrite_Freq': [[], []],
1013                                    'overwrite_Opt': [[], []],
1014                                    'overwrite_NoSymm': [[], []],
1015                                    'overwrite_state_TD': [[], []]}
1016
1017 # keyword_line_arguements_input = [['calculation_type', 'Opt'], ['restart', True], ['higher_max_cycles', 1024]]
1018 for i in range(0, len(keyword_line_arguments_input)):
1019     keyword_line_arguments['%s' % (keyword_line_arguments_input[i][0])] = keyword_line_arguments_input[i][1]
1020
1021 keyword_line = self.keyword_line_generator(functional_basisset = keyword_line_arguments['functional_basisset'],
1022                                             calculation_type = keyword_line_arguments['calculation_type'],
1023                                             state_TD = keyword_line_arguments['state_TD'],
1024                                             restart = keyword_line_arguments['restart'],
1025                                             overwrite_Freq = keyword_line_arguments['overwrite_Freq'],
1026                                             overwrite_Opt = keyword_line_arguments['overwrite_Opt'],
1027                                             overwrite_NoSymm = keyword_line_arguments['overwrite_NoSymm'],
1028                                             overwrite_state_TD = keyword_line_arguments['overwrite_state_TD'])
1029
1030 oldchk_name_extension = '%s_%s_C%sS%' % (functional, format_basis_set(basisset), charge_geom, state_geom)
1031 newchk_name_extension = '%s_%s_C%sS%s_opt_geom_at_C%sS%' % (functional, format_basis_set(basisset), charge_geom,
1032 state_geom, charge_eval, state_eval)
1033
1034 oldchk_gjf_file = "%s_%s" % (self._molecule_id, oldchk_name_extension)
1035 newchk_gjf_file = "%s_%s" % (self._molecule_id, newchk_name_extension)
1036
1037 title_line = '%s %s' % (newchk_gjf_file, keyword_line_arguments['calculation_type'])
1038
1039 a_str_gaussian = (f"%nprocshared={number_of_proc} \n" +
1040                     "\n" +
1041                     f"%mem={memory}GB \n" +
1042                     f"%oldchk={oldchk_gjf_file}.chk" + '\n' +
1043                     f"%chk={newchk_gjf_file}.chk \n" +
1044                     keyword_line + "\n" +
1045                     "\n" +
1046                     title_line + "\n" +
1047                     "\n" +
1048                     "%s %s\n" % (charge_eval, abs(charge_eval) + 1) +
1049                     '\n')
1050
1051 self.print_if_debug('\nGaussian file:\n===== \n%s\n===== ' % (a_str_gaussian))
1052 with open('%s/Data/%s.%s.gjf' % (self._parent_folder, self._molecule_id, newchk_gjf_file), mode = "wt") as
1053     writefile:
1054         writefile.write(a_str_gaussian)
1055
1056 return newchk_gjf_file
1057
1058 def generate_sh_file(self,
1059                      timeout_hr: int,
1060                      number_of_proc: int,
1061                      memory_gjf_gb: int,
1062                      charge: int,
1063                      excited_state: int,
1064                      functional_basisset: str) -> str:
1065
1066     """
1067     Method to write the job script for a Gaussian16 job in the HPC.
1068
1069     Parameters
1070     -----
1071     timeout_hr : int
1072         Timeout settings in hours.
1073     number_of_proc : int
1074         Number of processors.
1075     memory_gjf_gb : int
1076         Total memory allocated for the Gaussian job.
1077     charge : int
1078         Charge of the molecule.
1079     excited_state : int
1080         Excitation state of the molecule.
1081     functional_basisset : str
1082         Functional and basis set in the form {functional}_{basis_set}.
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3
```

```

1081     Returns
1082     -----
1083     qsub_script : str
1084         Submission script.
1085
1086     """
1087     self.print_if_debug('\nIn function Molecule.generate_sh_file():')
1088     timeout = '%sh' % (timeout_hr)
1089     walltime = '%s:10:00' % (timeout_hr)
1090     gjf_file = "%s_%s_C%sS%s" % (self._molecule_id, functional_basisset, charge, excited_state)
1091
1092     qsub_script = "#!/bin/sh\n"
1093     qsub_script += "#PBS -l walltime=%s\n" % (walltime)
1094     qsub_script += ("#PBS -l select=1:ncpus=%s" % (int(number_of_proc)) +
1095                     ":mem=%sGB" % (int(memory_gjf_gb + 8)) +
1096                     ":avx=true\n\n")
1097     qsub_script += "module load gaussian/g16-c01-avx\n"
1098     qsub_script += "cp $PBS_O_WORKDIR/" + gjf_file + ".gjf ./\n"
1099     qsub_script += "cp $PBS_O_WORKDIR/" + gjf_file + ".chk ./\n"
1100     qsub_script += "timeout %s g16 %s.gjf \n" % (timeout, gjf_file)
1101     qsub_script += "formchk %s.chk %s.fchk\n" % (gjf_file, gjf_file)
1102     qsub_script += "cp *.log $PBS_O_WORKDIR\ncp *.chk $PBS_O_WORKDIR\ncp *.fchk $PBS_O_WORKDIR"
1103     with open('%s/Data/%s.%s.sh' % (self._parent_folder, self._molecule_id, gjf_file), mode = "wt", newline="\n") as
1104         text_file:
1105             text_file.write(qsub_script)
1106             text_file.close()
1107     self.print_if_debug('\nqsub_script:\n=====\\n%s\\n=====' % (qsub_script))
1108     return qsub_script
1109
1109 def generate_sh_marcus_file(self,
1110                             timeout_hr: int,
1111                             number_of_proc: int,
1112                             memory_gjf_gb: int,
1113                             charge_geom: int,
1114                             charge_eval: int,
1115                             state_geom: int,
1116                             state_eval: int,
1117                             functional_basisset: str) -> str:
1118     self.print_if_debug('\nIn function Molecule.generate_sh_marcus_file():')
1119     timeout = '%sh' % (timeout_hr)
1120     walltime = '%s:10:00' % (timeout_hr)
1121     old_gjf_file = "%s_%s_C%sS%s" % (self._molecule_id, functional_basisset, charge_geom, state_geom)
1122     newchk_name_extension = '%s_C%sS%_opt_geom_at_C%sS%' % (functional_basisset, charge_geom, state_geom,
1123     charge_eval, state_eval)
1124     gjf_file = "%s_%s" % (self._molecule_id, newchk_name_extension)
1125
1125     qsub_script = "#!/bin/sh\n"
1126     qsub_script += "#PBS -l walltime=%s\n" % (walltime)
1127     qsub_script += ("#PBS -l select=1:ncpus=%s" % (int(number_of_proc)) +
1128                     ":mem=%sGB" % (int(memory_gjf_gb + 4)) +
1129                     ":avx=true\n\n")
1130     qsub_script += "module load gaussian/g16-c01-avx\n"
1131     qsub_script += "cp $PBS_O_WORKDIR/" + old_gjf_file + ".chk ./\n"
1132     qsub_script += "cp $PBS_O_WORKDIR/" + gjf_file + ".gjf ./\n"
1133     qsub_script += "cp $PBS_O_WORKDIR/" + gjf_file + ".chk ./\n"
1134     qsub_script += "timeout %s g16 %s.gjf \n" % (timeout, gjf_file)
1135     qsub_script += "formchk %s.chk %s.fchk\n" % (gjf_file, gjf_file)
1136     qsub_script += "cp *.log $PBS_O_WORKDIR\ncp *.chk $PBS_O_WORKDIR\ncp *.fchk $PBS_O_WORKDIR"
1137     with open('%s/Data/%s.%s.sh' % (self._parent_folder, self._molecule_id, gjf_file), mode = "wt", newline="\n") as
1138         text_file:
1139             text_file.write(qsub_script)
1140             text_file.close()
1141     self.print_if_debug('\nqsub_script:\n=====\\n%s\\n=====' % (qsub_script))
1142     return qsub_script
1143
1143 def generate_gjf_sh_files(self,
1144                           input_object,
1145                           excited_state: int,
1146                           charge: int,
1147                           functional: str,
1148                           basisset: str,
1149                           number_of_proc: int,
1150                           memory: int,
1151                           timeout_hr: int,
1152                           calculation_type: str = 'OptFreq',
1153                           connectivity: bool = True,
1154                           keyword_line_arguments_input: list[[str, str]] = []):
1155     """
1156     Generates both Gaussian16 script and the HPC job script.
1157
1158     Parameters
1159     -----
1160     input_object : TYPE
1161         Input object to be fed into Molecule.generate_molecule_xyz().
1162     excited_state : int
1163         Excitation state of the molecule.

```

```

1164 charge : int
1165     Charge of molecule.
1166 functional : str
1167     Functional to be used in DFT calculation.
1168 basisset : str
1169     Basis set to be used in DFT calculation.
1170 number_of_proc : int
1171     Number of processors.
1172 memory : int
1173     Total amount of memory to be divided between the processors.
1174 timeout_hr : int
1175     Timeout settings in hours.
1176 calculation_type : str, optional
1177     Calculation type. The default is 'OptFreq'.
1178 connectivity : bool, optional
1179     Whether to use molecule connectivity. The default is True.
1180 keyword_line_arguments_input : list[[str, str]], optional
1181     Overwrite lines. The default is [].

1182
1183 Returns
1184 -----
1185 None.

1186 """
1187 self.print_if_debug('\nIn function Molecule.generate_gjf_sh_files():')
1188 gjf = self.generate_gjf_file(input_object = input_object,
1189                             excited_state = excited_state,
1190                             charge = charge,
1191                             functional = functional,
1192                             basisset = basisset,
1193                             number_of_proc = number_of_proc,
1194                             memory = memory,
1195                             calculation_type = calculation_type,
1196                             connectivity = connectivity,
1197                             keyword_line_arguments_input = keyword_line_arguments_input)
1198 sh = self.generate_sh_file(timeout_hr = timeout_hr,
1199                             number_of_proc = number_of_proc,
1200                             memory_gjf_gb = memory,
1201                             charge = charge,
1202                             excited_state = excited_state,
1203                             functional_basisset = '%s_%s' % (functional, format_basis_set(basisset)))
1204 self.print_if_debug(gjf)
1205 self.print_if_debug(sh)

1206 def generate_gjf_sh_marcus_files(self,
1207                                     charge_geom: int,
1208                                     charge_eval: int,
1209                                     state_geom: int,
1210                                     state_eval: int,
1211                                     functional: str,
1212                                     basisset: str,
1213                                     number_of_proc: int,
1214                                     memory: int,
1215                                     timeout_hr: int,
1216                                     calculation_type: str = 'marcus',
1217                                     keyword_line_arguments_input: list[[str, str]] = []):
1218 self.print_if_debug('\nIn function Molecule.generate_gjf_sh_marcus_files():')
1219 gjf = self.generate_gjf_marcus_file(charge_geom = charge_geom,
1220                                       charge_eval = charge_eval,
1221                                       state_geom = state_geom,
1222                                       state_eval = state_eval,
1223                                       functional = functional,
1224                                       basisset = basisset,
1225                                       number_of_proc = number_of_proc,
1226                                       memory = memory,
1227                                       keyword_line_arguments_input = keyword_line_arguments_input)
1228 sh = self.generate_sh_marcus_file(timeout_hr = timeout_hr,
1229                                   number_of_proc = number_of_proc,
1230                                   memory_gjf_gb = memory,
1231                                   charge_geom = charge_geom,
1232                                   charge_eval = charge_eval,
1233                                   state_geom = state_geom,
1234                                   state_eval = state_eval,
1235                                   functional_basisset = '%s_%s' % (functional, format_basis_set(basisset)))
1236 self.print_if_debug(gjf)
1237 self.print_if_debug(sh)

1238
1239 def generate_gjf_sh_restart_files(self,
1240                                     excited_state: int,
1241                                     charge: int,
1242                                     functional: str,
1243                                     basisset: str,
1244                                     number_of_proc: int,
1245                                     memory: int,
1246                                     timeout_hr: int,
1247                                     calculation_type: str = 'OptFreq',
1248                                     keyword_line_arguments_input: list[[str, str]] = [])
1249

```

```

1250                                     keyword_line_arguments_input: list[[str, str]] = []):
1251     self.print_if_debug('\nIn function Molecule.generate_gjf_sh_restart_files():')
1252     gjf = self.generate_gjf_restart_file(excited_state = excited_state,
1253                                         charge = charge,
1254                                         functional = functional,
1255                                         basisset = basisset,
1256                                         number_of_proc = number_of_proc,
1257                                         memory = memory,
1258                                         calculation_type = calculation_type,
1259                                         keyword_line_arguments_input = keyword_line_arguments_input)
1260     sh = self.generate_sh_file(timeout_hr = timeout_hr,
1261                               number_of_proc = number_of_proc,
1262                               memory_gjf_gb = memory,
1263                               charge = charge,
1264                               excited_state = excited_state,
1265                               functional_basisset = '%s%s' % (functional, format_basis_set(basisset)))
1266     self.print_if_debug(gjf)
1267     self.print_if_debug(sh)
1268
1269 #%%
1270
1271 class MongoDB():
1272
1273     def __init__(self,
1274                  username: str,
1275                  password: str,
1276                  collection: str,
1277                  database: str = 'New_Reorganisation_Energy_Calc',
1278                  connection_string: str = 'mongodb+srv://<user>:<password>@cluster0.wxpvcvv.mongodb.net/?retryWrites=true&
w=majority&appName=Cluster0',
1279                  debug: bool = False):
1280         """
1281             Establishing connection to a MongoDB database.
1282
1283             Parameters
1284             -----
1285             username : str
1286                 Username used to access that database.
1287             password : str
1288                 Password corresponding to username.
1289             collection : str
1290                 Name of the collection to be accessed.
1291             database : str, optional
1292                 Name of the database to be accessed.. The default is 'Reorganisation_Energy_Calculation'.
1293             connection_string : str, optional
1294                 Connection string to access the database. The default is 'mongodb+srv://<user>:<password>@cluster0.wxpvcvv.
mongodb.net/?retryWrites=true&w=majority&appName=Cluster0'.
1295             debug : bool, optional
1296                 Toggle debug mode. The default is False.
1297
1298             Returns
1299             -----
1300             None.
1301
1302             """
1303             self._username = username
1304             self._password = password
1305             self._collection = collection
1306             self._database = database
1307             self._connection_string = connection_string
1308             self._debug = debug
1309
1310     def __repr__(self):
1311         """
1312             Official representation
1313
1314             Returns
1315             -----
1316             string
1317                 Official representation of class.
1318
1319             return '\nMongoDB connection:\n Username = %s\n Password = %s\n Database = %s\n Collection = %s\n Connection
string = %s' % (self._username, self._password, self._database, self._collection, self._connection_string)
1320
1321     def __str__(self):
1322         """
1323             Simplified string representation.
1324
1325             Returns
1326             -----
1327             string
1328                 String representation of class.
1329
1330             """
1331             return '\n%s\n%s\n%s\n%s\n%s' % (self._username, self._password, self._database, self._collection, self._connection_string)

```

```

1332 def connect_mongo(self):
1333     """
1334     Connects to MongoDB and returns the database.
1335
1336     Returns
1337     -----
1338     database : MongoDB Database
1339         Full database.
1340
1341     """
1342     if self._debug:
1343         print('\nIn function MongoDB.connect_mongo():')
1344         print('Username: %s\nPassword: %s\nDatabase: %s\nConnection string: %s' % (self._username, self._password,
1345 self._collection, self._connection_string))
1346     client = MongoClient(self._connection_string.replace('<user>', self._username).replace('<password>', urllib.parse.
1347 quote_plus(self._password)))
1348     database = client[self._database]
1349     return database
1350
1351 def pull(self):
1352     """
1353     Method to pull the collection from the online MongoDB database.
1354
1355     Returns
1356     -----
1357     df : pandas.core.frame.DataFrame
1358         Dataframe of all the entries in the collection.
1359
1360     """
1361     if self._debug:
1362         print('\nIn function MongoDB.pull():')
1363     database = self.connect_mongo()['%s' % (self._collection)].find()
1364     df = pd.DataFrame(list(database)).iloc[:, :]
1365     return df
1366
1367 def create_post(self, keys: list, values: list) -> dict:
1368     """
1369     Takes in keys and values in list/array form and organise it into a post for pushing onto MongoDB database.
1370
1371     Parameters
1372     -----
1373     keys : list
1374         List of keys in entry.
1375     values : list
1376         List of values corresponding to the keys.
1377
1378     Raises
1379     -----
1380     ValueError
1381         Length of keys and values does not match.
1382
1383     Returns
1384     -----
1385     dict
1386         A MongoDB post that can be pushed onto the collection.
1387
1388     """
1389     if self._debug:
1390         print('\nIn function MongoDB.create_post():')
1391         print('keys: %s\nvalues: %s' % (keys, values))
1392         print('length (keys): %s\nlength (values): %s' % (len(keys), len(values)))
1393     if len(keys) != len(values):
1394         raise ValueError('Length of input is not the same:\n keys = %s\n values = %s' % ((len(keys)), len(values)))
1395     post: dict = {}
1396     for i in range(0, len(keys)):
1397         post[keys[i]] = values[i]
1398     return post
1399
1400 def push_one_new_entry(self, keys: list, values: list):
1401     """
1402     Method to create a new entry on the online collection.
1403
1404     Parameters
1405     -----
1406     keys : list
1407         List of keys.
1408     values : list
1409         List of values corresponding to above keys.
1410
1411     Raises
1412     -----
1413     Exception
1414         Push unsuccesfull.
1415
1416     Returns

```

```

1416 -----
1417     None.
1418
1419 """
1420 post: dict = self.create_post(keys = keys, values = values)
1421 if self._debug:
1422     print('\nIn function MongoDB.push_one_entry():')
1423     pprint.pprint(post)
1424 try:
1425     self.connect_mongo()['%s' % (self._collection)].insert_one(post)
1426 except:
1427     raise Exception('\nPush failed.')
1428
1429 def replace_one_entry(self, keys: list, values: list):
1430 """
1431     Method to replace one entry of the online collection.
1432
1433     Parameters
1434 -----
1435     keys : list
1436         List of keys.
1437     values : list
1438         List of values corresponding to above keys.
1439
1440     Raises
1441 -----
1442     Exception
1443         Push unsucessfull.
1444
1445     Returns
1446 -----
1447     None.
1448
1449 """
1450 post: dict = self.create_post(keys = keys, values = values)
1451 if self._debug:
1452     print('\nIn function MongoDB.replace_one_entry():')
1453     pprint.pprint(post)
1454 try:
1455     self.connect_mongo()['%s' % (self._collection)].replace_one({'_id':post['_id']}, post)
1456 except:
1457     raise Exception('\nReplace failed.')
1458
1459 def update_one_entry(self, _id: str, keys: list, values: list):
1460 """
1461     Method to only edit one entry on the MongoDB database.
1462
1463     Parameters
1464 -----
1465     _id : str
1466         ID of the entry.
1467     keys : list
1468         Keys to be changed.
1469     values : list
1470         The corresponding values to the keys.
1471
1472     Raises
1473 -----
1474     Exception
1475         Update failed.
1476
1477     Returns
1478 -----
1479     None.
1480
1481 """
1482 post: dict = self.create_post(keys = keys, values = values)
1483 if self._debug:
1484     print('\nIn function MongoDB.update_one_entry():')
1485 try:
1486     self.connect_mongo()['%s' % (self._collection)].update_one({'_id': _id}, {'$set': post})
1487 except:
1488     raise Exception('\nUpdate failed.')
1489
1490 def export_to_excel(self, path: str = ''):
1491 """
1492     Method to export the entire online collection to an Excel file saved locally.
1493
1494     Parameters
1495 -----
1496     path : str, optional
1497         Path to write the Excel file. The default is ''': the current working directory would be used.
1498
1499     Raises
1500 -----
1501     Exception

```



```

1588     stdin, stdout, stderr = client.exec_command("/opt/pbs/bin/qstat ")
1589
1590     q_status = stdout.read()
1591
1592     self.print_if_debug('Standard output:\n%s' % (stdout.read()))
1593     self.print_if_debug('Standard error:\n%s' % (stderr.read()))
1594
1595     if stdout.channel.recv_exit_status() != 0:
1596         print('\nError encountered:')
1597         print("Standard error:\n%s" % (stderr.read()))
1598         print("Exit status: {}".format(stdout.channel.recv_exit_status()))
1599     client.close()
1600     q_status = str(q_status)[2:-1].replace('\\n', '\n')
1601     return q_status
1602
1603 def formatted_qstat(self) -> pd.core.frame.DataFrame:
1604     """
1605         Method to return the queue status as a pandas dataframe.
1606
1607     Returns
1608     -----
1609     formatted_q_status : pd.core.frame.DataFrame
1610         Queue status.
1611
1612     """
1613     self.print_if_debug('\nIn function HPC.formatted_qstat():')
1614     q_status = self.qstat()
1615     self.print_if_debug('%s' % (q_status))
1616     q_status = q_status.split('\n')
1617     q_status = q_status[2:-1]
1618     formatted_q_status = pd.DataFrame(columns = ['Job ID', 'Class', 'Job Name', 'Status', 'Comment'])
1619     for i in range(0, len(q_status)):
1620         job_id = q_status[i][0:14].replace(' ', '')
1621         job_class = q_status[i][15:30].replace(' ', '')
1622         job_name = q_status[i][31:51].replace(' ', '')
1623         job_status = q_status[i][52:59].replace(' ', '')
1624         job_comment = q_status[i][60:]
1625         formatted_q_status.loc[len(formatted_q_status)] = [job_id, job_class, job_name, job_status, job_comment]
1626     return formatted_q_status
1627
1628 def transfer_from_HPC(self,
1629                         local_folder: str,
1630                         remote_folder: str,
1631                         filename: str):
1632     """
1633         Method to transfer files from the HPC to local device.
1634
1635     Parameters
1636     -----
1637     local_folder : str
1638         Path to local folder where the file would be saved.
1639     remote_folder : str
1640         Path to remote folder where the file is located.
1641     filename : str
1642         Filename of the file to be copied.
1643
1644     Returns
1645     -----
1646     None.
1647
1648     """
1649     self.print_if_debug('\nIn function HPC.transfer_from_HPC():')
1650     with paramiko.SSHClient() as client:
1651         client.load_system_host_keys()
1652         client.set_missing_host_key_policy(paramiko.WarningPolicy)
1653         client.connect(hostname = self._hostname,
1654                         username = self._username,
1655                         password = self._password)
1656         sftp_client = client.open_sftp()
1657         sftp_client.get(r'%s/%s' % (remote_folder, filename),
1658                         r'%s/%s' % (local_folder, filename))
1659         sftp_client.close()
1660         client.close()
1661     self.print_if_debug('\nTransfer sucessfull.')
1662
1663 def submit_job_gaussian(self,
1664                         local_folder: str,
1665                         remote_folder: str,
1666                         filename: str):
1667     """
1668         Method to submit a Gaussian16 job in the HPC.
1669         The required scripts (.gjf and .sh) should be located in the local folder.
1670
1671     Parameters
1672     -----
1673     local_folder : str

```

```

1674     Local folder where the .gjf and .sh job scripts are located.
1675     remote_folder : str
1676         Remote folder where scripts will be transferred to.
1677     filename : str
1678         Full filename of the scripts. Example {filename}.sh and {filename}.gjf.
1679
1680 Returns
1681 -----
1682 None.
1683
1684 """
1685 self.print_if_debug('\nIn function HPC.submit_job_gaussian():')
1686 with paramiko.SSHClient() as client:
1687     client.load_system_host_keys()
1688     client.set_missing_host_key_policy(paramiko.WarningPolicy)
1689
1690     # Establish SSH connection
1691     client.connect(hostname = self._hostname,
1692                     username = self._username,
1693                     password = self._password)
1694
1695     # Establish SFTP connection ftp_client=ssh.open_sftp()
1696     stdin, stdout, stderr = client.exec_command("mkdir -p " + remote_folder)
1697     with client.open_sftp() as sftp0:
1698         # Push job submission script to a particular path on the cluster
1699         sftp0.put(r'%s/%s.sh' % (local_folder, filename), r'%s/%s.sh' % (remote_folder, filename))
1700         sftp0.put(r'%s/%s.gjf' % (local_folder, filename), r'%s/%s.gjf' % (remote_folder, filename))
1701     # Submit our Grid Engine job by running a remote 'qsub' command over SSH
1702     stdin, stdout, stderr = client.exec_command(f"cd {remote_folder} \n" +
1703                                                 f"chmod +rwx {filename}.sh\n" +
1704                                                 f"/opt/pbs/bin/qsub {filename}.sh \n")
1705
1706     # Show the standard output and error of our job
1707     stdout_copy = deepcopy(stdout.read())
1708     self.print_if_debug('\nStandard output:\n%s' % (stdout_copy))
1709     self.print_if_debug('\nStandard error:\n%s' % (stderr.read()))
1710     print(stdout.channel.recv_exit_status())
1711     if stdout.channel.recv_exit_status() != 0:
1712         print('\nError encountered:')
1713         print("\nStandard error:\n%s" % (stderr.read()))
1714         print("Exit status: {}".format(stdout.channel.recv_exit_status()))
1715     client.close()
1716 job_id = int(''.join(filter(str.isdigit, str(stdout_copy))))
1717 return job_id
1718
1719 def restart_job_gaussian(self,
1720                         local_folder: str,
1721                         remote_folder: str,
1722                         filename: str):
1723     self.print_if_debug('\nIn function HPC.restart_job_gaussian():')
1724     with paramiko.SSHClient() as client:
1725         client.load_system_host_keys()
1726         client.set_missing_host_key_policy(paramiko.WarningPolicy)
1727
1728         # Establish SSH connection
1729         client.connect(hostname = self._hostname,
1730                         username = self._username,
1731                         password = self._password)
1732
1733         # Establish SFTP connection ftp_client=ssh.open_sftp()
1734         stdin, stdout, stderr = client.exec_command("mkdir -p " + remote_folder)
1735         with client.open_sftp() as sftp0:
1736             # Push job submission script to a particular path on the cluster
1737             sftp0.put(r'%s/%s.sh' % (local_folder, filename), r'%s/%s.sh' % (remote_folder, filename))
1738             sftp0.put(r'%s/%s.gjf' % (local_folder, filename), r'%s/%s.gjf' % (remote_folder, filename))
1739             sftp0.put(r'%s/%s.chk' % (local_folder, filename), r'%s/%s.chk' % (remote_folder, filename))
1740         # Submit our Grid Engine job by running a remote 'qsub' command over SSH
1741         stdin, stdout, stderr = client.exec_command(f"cd {remote_folder} \n" +
1742                                                 f"chmod +rwx {filename}.sh\n" +
1743                                                 f"/opt/pbs/bin/qsub {filename}.sh \n")
1744
1745         # Show the standard output and error of our job
1746         stdout_copy = deepcopy(stdout.read())
1747         self.print_if_debug('\nStandard output:\n%s' % (stdout_copy))
1748         self.print_if_debug('\nStandard error:\n%s' % (stderr.read()))
1749         print(stdout.channel.recv_exit_status())
1750         if stdout.channel.recv_exit_status() != 0:
1751             print('\nError encountered:')
1752             print("\nStandard error:\n%s" % (stderr.read()))
1753             print("Exit status: {}".format(stdout.channel.recv_exit_status()))
1754     client.close()
1755 job_id = int(''.join(filter(str.isdigit, str(stdout_copy))))
1756 return job_id
1757
1758 def transfer_chk_fchk_log_files_from_HPC(self,
1759                                         local_folder: str,
1760                                         remote_folder: str,
1761                                         filename: str):
1762
1763     # Establish SSH connection
1764     client.connect(hostname = self._hostname,
1765                     username = self._username,
1766                     password = self._password)
1767
1768     # Establish SFTP connection ftp_client=ssh.open_sftp()
1769     stdin, stdout, stderr = client.exec_command("mkdir -p " + remote_folder)
1770     with client.open_sftp() as sftp0:
1771         # Push job submission script to a particular path on the cluster
1772         sftp0.put(r'%s/%s.sh' % (local_folder, filename), r'%s/%s.sh' % (remote_folder, filename))
1773         sftp0.put(r'%s/%s.gjf' % (local_folder, filename), r'%s/%s.gjf' % (remote_folder, filename))
1774         sftp0.put(r'%s/%s.chk' % (local_folder, filename), r'%s/%s.chk' % (remote_folder, filename))
1775
1776         # Submit our Grid Engine job by running a remote 'qsub' command over SSH
1777         stdin, stdout, stderr = client.exec_command(f"cd {remote_folder} \n" +
1778                                                 f"chmod +rwx {filename}.sh\n" +
1779                                                 f"/opt/pbs/bin/qsub {filename}.sh \n")
1780
1781         # Show the standard output and error of our job
1782         stdout_copy = deepcopy(stdout.read())
1783         self.print_if_debug('\nStandard output:\n%s' % (stdout_copy))
1784         self.print_if_debug('\nStandard error:\n%s' % (stderr.read()))
1785         print(stdout.channel.recv_exit_status())
1786         if stdout.channel.recv_exit_status() != 0:
1787             print('\nError encountered:')
1788             print("\nStandard error:\n%s" % (stderr.read()))
1789             print("Exit status: {}".format(stdout.channel.recv_exit_status()))
1790     client.close()
1791
1792 job_id = int(''.join(filter(str.isdigit, str(stdout_copy))))
1793 return job_id

```

```

1760 """
1761 Transfer the calculation files from the HPC to local folder.
1762
1763 Parameters
1764 -----
1765 local_folder : str
1766     Local folder where the files will be saved.
1767 remote_folder : str
1768     Remote folder where the files will be transferred from.
1769 filename : str
1770     Full filename of the files. Example {filename}.fchk, {filename}.chk, and {filename}.log.
1771
1772 Returns
1773 -----
1774 None.
1775
1776 """
1777 self.print_if_debug('\nIn function HPC.transfer_chk_fchk_log_files_from_HPC():')
1778 try:
1779     self.print_if_debug('Transferring log files from HPC:\nlocal folder = %s\nremote folder = %s\nfile = %s.log' %
1780 (local_folder, remote_folder, filename))
1781     self.transfer_from_HPC(local_folder = local_folder,
1782                             remote_folder = remote_folder,
1783                             filename = r'%s.log' % (filename))
1784 except:
1785     print('\nFailed to transfer log files.')
1786 try:
1787     self.print_if_debug('Transferring chk files from HPC:\nlocal folder = %s\nremote folder = %s\nfile = %s.chk' %
1788 (local_folder, remote_folder, filename))
1789     self.transfer_from_HPC(local_folder = local_folder,
1790                             remote_folder = remote_folder,
1791                             filename = r'%s.chk' % (filename))
1792 except:
1793     print('\nFailed to transfer chk files.')
1794 try:
1795     self.print_if_debug('Transferring fchk files from HPC:\nlocal folder = %s\nremote folder = %s\nfile = %s.fchk' %
1796 (local_folder, remote_folder, filename))
1797     self.transfer_from_HPC(local_folder = local_folder,
1798                             remote_folder = remote_folder,
1799                             filename = r'%s.fchk' % (filename))
1800 except:
1801     print('\nFailed to transfer fchk files.')
1802
1803 def copy_fchk_log_files_for_dushin(self,
1804                                     remote_dushin_folder: str,
1805                                     remote_log_folder: str,
1806                                     filename_initial_state: str,
1807                                     filename_final_state: str):
1808 """
1809 Transfer .fchk files and .log files between two folders in the HPC.
1810
1811 Parameters
1812 -----
1813 remote_dushin_folder : str
1814     Folder where DUSHIN would be ran.
1815 remote_log_folder : str
1816     Folder where the fchk and log files are saved.
1817 filename_initial_state : str
1818     Filenamne of the initial state calculation, e.g. in the form {filename_initial_state}.log.
1819 filename_final_state : str
1820     Filename of the final state calculation, e.g. in the form {filename_final_state}.log.
1821
1822 Returns
1823 -----
1824 None.
1825
1826 """
1827 self.print_if_debug('\nIn function HPC.copy_fchk_log_files_for_dushin():')
1828
1829 self.print_if_debug('Copying fchk files for:\n %s.fchk and\n %s.fchk' % (filename_initial_state,
1830 filename_final_state))
1831 with paramiko.SSHClient() as client:
1832     client.load_system_host_keys()
1833     client.set_missing_host_key_policy(paramiko.WarningPolicy)
1834     client.connect(hostname = self._hostname,
1835                     username = self._username,
1836                     password = self._password)
1837     stdin, stdout, stderr = client.exec_command(f"cd {remote_log_folder} \n " +
1838                                                 f"cp {filename_initial_state}.fchk {remote_dushin_folder} \n" +
1839                                                 f"cp {filename_final_state}.fchk {remote_dushin_folder} \n")
1840     # Show the standard output and error of our job
1841     self.print_if_debug('Standard output:\n%s' % (stdout.read()))
1842     self.print_if_debug('Standard error:\n%s' % (stderr.read()))
1843     if stdout.channel.recv_exit_status() != 0:
1844         print('\nError encountered:')
1845         print("Standard error:\n%s" % (stderr.read()))

```

```

1842         print("Exit status: {}".format(stdout.channel.recv_exit_status()))
1843         client.close()
1844     self.print_if_debug('fchk files copied sucessfully')
1845
1846     self.print_if_debug('Copying log files for:\n %s.log and\n %s.log' % (filename_initial_state, filename_final_state
1847 ))
1848     with paramiko.SSHClient() as client:
1849         client.load_system_host_keys()
1850         client.set_missing_host_key_policy(paramiko.WarningPolicy)
1851         client.connect(hostname = self._hostname,
1852                         username = self._username,
1853                         password = self._password)
1854         stdin, stdout, stderr = client.exec_command(f"cd {remote_log_folder} \n " +
1855                                         f"cp {filename_initial_state}.log {remote_dushin_folder} \n" +
1856                                         f"cp {filename_final_state}.log {remote_dushin_folder} \n")
1857         # Show the standard output and error of our job
1858         self.print_if_debug('Standard output:\n%s' % (stdout.read()))
1859         self.print_if_debug('Standard error:\n%s' % (stderr.read()))
1860         if stdout.channel.recv_exit_status() != 0:
1861             print('\nError encountered:')
1862             print("Standard error:\n%s" % (stderr.read()))
1863             print("Exit status: {}".format(stdout.channel.recv_exit_status()))
1864         client.close()
1865     self.print_if_debug('log files copied sucessfully')
1866
1867     def submit_job_dushin(self,
1868                           remote_dushin_folder: str,
1869                           remote_bin_path: str,
1870                           initial_state: str,
1871                           final_state: str,
1872                           mol_id_functional_basisset: str):
1873         self.print_if_debug('\nIn function HPC.submit_job_dushin():')
1874         filename_initial_state = '%s_%s' % (mol_id_functional_basisset, initial_state)
1875         filename_final_state = '%s_%s' % (mol_id_functional_basisset, final_state)
1876         self.print_if_debug('\nDushin started for %s %s %s.' % (mol_id_functional_basisset, initial_state, final_state))
1877         with paramiko.SSHClient() as client:
1878             client.load_system_host_keys()
1879             client.set_missing_host_key_policy(paramiko.WarningPolicy)
1880             client.connect(hostname = self._hostname,
1881                           username = self._username,
1882                           password = self._password)
1883
1884         stdin, stdout, stderr = client.exec_command(
1885             f"cd {remote_dushin_folder} \n " +
1886             f"export PATH=$PATH:{remote_bin_path} \n" +
1887             f"find -type f -name '*.sh.*' -delete \n" +
1888             f"cd {remote_dushin_folder} \n" +
1889
1890             f"split_log.sh {filename_initial_state}.log \n" +
1891             f"mv {filename_initial_state}.log org_{filename_initial_state}.log\n" +
1892             f"mv freq_1_{filename_initial_state}.log {filename_initial_state}.log \n" +
1893
1894             f"split_log.sh {filename_final_state}.log \n" +
1895             f"mv {filename_final_state}.log org_{filename_final_state}.log\n" +
1896             f"mv freq_1_{filename_final_state}.log {filename_final_state}.log \n" +
1897
1898             f"find -type f -name '*freq_2*' -delete \n" +
1899             f"rundushin.sh {mol_id_functional_basisset} {initial_state} {final_state}")
1900
1901         if stdout.channel.recv_exit_status() != 0:
1902             print('\nError encountered:')
1903             print("\nStandard error:\n%s" % (stderr.read()))
1904             print("Exit status: {}".format(stdout.channel.recv_exit_status()))
1905         client.close()
1906     self.print_if_debug('DUSHIN submission sucessful')
1907 #%%
1908
1909 class SMILES_database():
1910
1911     def __init__(self,
1912                  username: str,
1913                  password: str,
1914                  collection: str = 'SMILES_Database',
1915                  database: str = 'New_Reorganisation_Energy_Calc',
1916                  connection_string: str = 'mongodb+srv://<user>:<password>@cluster0.wxpvcww.mongodb.net/?retryWrites=true&
1917 w=majority&appName=Cluster0',
1918                  debug: bool = False):
1919         """
1920             Database of SMILES stored through a MongoDB connection.
1921
1922             Parameters
1923             -----
1924             username : str
1925                 Username to connect to MongoDB databsse.
1926             password : str

```

```

1926     Password to connect to MongoDB database.
1927     collection : str, optional
1928         Name of the collection on the database. The default is 'SMILES_Database'.
1929     database : str, optional
1930         Name of the database. The default is 'New_Reorganisation_Energy_Calc'.
1931     connection_string : str, optional
1932         Connection string to connect to MongoDB. The default is 'mongodb+srv://<user>:<password>@cluster0.wxpvcwv.
1933     mongodb.net/?retryWrites=true&w=majority&appName=Cluster0'.
1934     debug : bool, optional
1935         Toggle for debugging. The default is False.
1936
1937     Returns
1938     -----
1939     None.
1940
1941     """
1942     self._username = username
1943     self._password = password
1944     self._collection = collection
1945     self._database = database
1946     self._connection_string = connection_string
1947     self._debug = debug
1948     self._MongoDB = MongoDB(username = self._username,
1949                             password = self._password,
1950                             collection = self._collection,
1951                             database = self._database,
1952                             connection_string = self._connection_string,
1953                             debug = self._debug)
1954
1955     def __repr__(self):
1956         """
1957             Official representation
1958
1959             Returns
1960             -----
1961             string
1962                 Official representation of class.
1963
1964             return '\nMongoDB connection:\n Username = %s\n Password = %s\n Database = %s\n Collection = %s\n Connection
1965             string = %s' % (self._username, self._password, self._database, self._collection, self._connection_string)
1966
1967     def __str__(self):
1968         """
1969             Simplified string representation.
1970
1971             Returns
1972             -----
1973             string
1974                 String representation of class.
1975
1976             return '\n%s\n%s\n%s\n%s\n%s' % (self._username, self._password, self._database, self._collection, self.
1977             _connection_string)
1978
1979     def pull_SMILES_database(self):
1980         """
1981             Get the SMILES database as a pandas dataframe.
1982
1983             Raises
1984             -----
1985             Exception
1986                 Failed to fetch SMILES database.
1987
1988             Returns
1989             -----
1990             dataframe : pandas.core.frame.DataFrame
1991                 SMILES database as a pandas dataframe.
1992
1993             if self._debug:
1994                 print('\nIn function SMILES_database.pull_SMILES_database():')
1995             try:
1996                 dataframe = self._MongoDB.pull()
1997                 return dataframe
1998             except:
1999                 raise Exception('Failed to fetch SMILES database.')
2000
2001     def export_to_excel(self, path: str = ''):
2002         """
2003             Downloads a copy of the SMILES database as an .xlsx file.
2004
2005             Parameters
2006             -----
2007             path : str, optional
2008                 Full path of the folder where the file would be saved. The default is ''.

```

```

2009     Raises
2010     -----
2011     Exception
2012         Export to excel not sucessfull..
2013
2014     Returns
2015     -----
2016     None.
2017
2018     """
2019     if path == '':
2020         path = '%s/SMILES_database.xlsx' % (os.getcwd())
2021     dataframe = self.pull_SMILES_database()
2022     if self._debug:
2023         print('\nIn function SMILES_database.export_to_excel():')
2024         print('Export to location:\n%s' % (path))
2025         pprint.pprint(dataframe)
2026     try:
2027         dataframe.to_excel(r'%s' % (path))
2028     except:
2029         raise Exception('\nExport to excel not sucessfull.')
2030     pass
2031
2032 def check_against_database(self, iupac_name: str, SMILES: str = '') -> list[str, str]:
2033     """
2034     Check the molecule name and SMILES against database. If both molecule name and SMILES are declared, checks for a
2035     match.
2036     If only the Name is declared, checks whether the molecule is in the database, and returns the corresponding SMILES
2037
2038     Parameters
2039     -----
2040     iupac_name : str
2041         Molecule name.
2042     SMILES : str, optional
2043         SMILES string. The default is ''.
2044
2045     Raises
2046     -----
2047     Exception
2048         SMILES string not declared and not found in database.
2049     Exception
2050         Input SMILES does not match database.
2051
2052     Returns
2053     -----
2054     list[str, str]
2055         The molecule name and SMILES string if no error is encountered.
2056
2057     """
2058     if self._debug:
2059         print('\nIn function SMILES_database.in_database():')
2060         print('SMILES: %s\niupac_name: %s' % (SMILES, iupac_name))
2061     dataframe = self.pull_SMILES_database()
2062     # if SMILES not declared, ...
2063     if SMILES == '':
2064         # ... try looking for SMILES in database, ...
2065         try:
2066             SMILES = dataframe.at[dataframe[dataframe['_id']] == iupac_name].index[0], 'SMILES'
2067             print('Molecule found in database.')
2068             return iupac_name, SMILES
2069         # ... if not found in database, raise exception
2070         except:
2071             raise Exception('SMILES string not declared and not found in database.')
2072     # if SMILES declared, ...
2073     else:
2074         try:
2075             # ... compare SMILES to database, ...
2076             SMILES_found_in_database = dataframe.at[dataframe[dataframe['_id']] == iupac_name].index[0], 'SMILES'
2077             print('Molecule found in database.')
2078         except:
2079             SMILES_found_in_database = 'not found'
2080             print('Molecule not found in database.')
2081         # ... if SMILES from input and database does not match, raise exception, ...
2082         if SMILES_found_in_database != SMILES and SMILES_found_in_database != 'not found':
2083             raise Exception('Input SMILES does not match database:\nInput: %s\nDatabase: %s' % (SMILES,
2084 SMILES_found_in_database))
2085         # otherwise if SMILES not found in database, append and write to database
2086         elif SMILES_found_in_database == 'not found':
2087             self._MongoDB.push_one_new_entry(keys = ['_id', 'SMILES'], values = [iupac_name, SMILES])
2088             print('Molecule added to database:\nMolecule Name: %s\nSMILES: %s' % (iupac_name, SMILES))
2089             return iupac_name, SMILES
2090
2091         else:
2092             print('Molecule matched with database')
2093             return iupac_name, SMILES

```

```

2092 #%%
2093
2094 class Funnel():
2095
2096     def __init__(self,
2097                  path: str = r'%s\Funnel.xlsx' % (os.getcwd()),
2098                  debug: bool = False):
2099         self._funnel = pd.DataFrame(columns = ['MMFFO', 'Functional', 'Basis Set 1', 'Basis Set 2', 'Basis Set 3'])
2100         self._path = path
2101         self._debug = debug
2102
2103     def retrieve(self):
2104         return self._funnel
2105
2106     def add_workflow(self,
2107                      MMFFO: str = 'Yes',
2108                      functional: str = 'B3LYP',
2109                      basis_set_1: str = '6-31G(d)',
2110                      basis_set_2: str = '6-311+G(d,p)',
2111                      basis_set_3: str = 'No'):
2112         if self._debug:
2113             print('\nIn function Funnel.add_workflow():')
2114             print('MMFFO = %s\nFunctional = %s\nBasis Set 1 = %s\nBasis Set 2 = %s\nBasis Set 3 = %s' % (MMFFO, functional,
2115 , basis_set_1, basis_set_2, basis_set_3))
2116         self._funnel.loc[len(self._funnel)] = [MMFFO, functional, basis_set_1, basis_set_2, basis_set_3]
2117
2118     def workflow_indices(self):
2119         return list(self._funnel.index)
2120
2121     def MMFFO(self, index):
2122         return self._funnel.at[index, 'MMFFO']
2123
2124     def functional(self, index):
2125         return self._funnel.at[index, 'Functional']
2126
2127     def basis_set_1(self, index):
2128         return self._funnel.at[index, 'Basis Set 1']
2129
2130     def basis_set_2(self, index):
2131         return self._funnel.at[index, 'Basis Set 2']
2132
2133     def basis_set_3(self, index):
2134         return self._funnel.at[index, 'Basis Set 3']
2135
2136     def export_to_excel(self, path: str = ''):
2137         if path == '':
2138             path = self._path
2139         dataframe = self.retrieve()
2140         if self._debug:
2141             print('\nIn function Funnel.export_to_excel():')
2142             print('Export to location:\n%s' % (path))
2143             pprint.pprint(dataframe)
2144         try:
2145             dataframe.to_excel(r'%s' % (path))
2146         except:
2147             raise Exception('\nExport to excel not sucessfull.')
2148
2149     def load_from_excel(self, path: str = ''):
2150         if path == '':
2151             path = self._path
2152         if self._debug:
2153             print('\nIn function Funnel.load_from_excel():')
2154             print('Load from location:\n%s' % (path))
2155         try:
2156             self._funnel = pd.read_excel(path)[['MMFFO', 'Functional', 'Basis Set 1', 'Basis Set 2', 'Basis Set 3']]
2157         except:
2158             raise Exception('Read unsucessfull, unable to load from folder %s' % (path))
2159 #%%
2160
2161 class Tracker():
2162
2163     def __init__(self,
2164                  username: str,
2165                  password: str,
2166                  collection: str = 'Tracker',
2167                  database: str = 'New_Reorganisation_Energy_Calc',
2168                  connection_string: str = 'mongodb+srv://<user>:<password>@cluster0.wxpvcwv.mongodb.net/?retryWrites=true&
2169 w=majority&appName=Cluster0',
2170                  debug: bool = False):
2171         self._username = username
2172         self._password = password
2173         self._collection = collection
2174         self._database = database
2175         self._connection_string = connection_string
2176         self._debug = debug

```



```

2256                                         username, 'No comment', 'Entry created on %s' % (str(datetime.now()))
2257                                         ).split('.')[0]), timestamp())
2258                                         )
2259                                         self.print_if_debug('\nEntry sucessfully created.')
2260                                         # self._MongoDB.push_one_new_entry(keys = ['_id', 'IUPAC Name', 'SMILES', 'MMFFO',
2261                                         #                                     'Functional', 'Basis Set 1', 'Basis Set 2', 'Basis Set 3',
2262                                         #                                     'COS0: BS1', 'COS0: BS2', 'COS0: BS3',
2263                                         #                                     'COS1: BS1', 'COS1: BS2', 'COS1: BS3', 'COS1-BS1: DUSHIN', 'COS1-
2264                                         BS2: DUSHIN', 'COS1-BS3: DUSHIN',
2265                                         #                                     'C1S0: BS1', 'C1S0: BS2', 'C1S0: BS3', 'C1S0-BS1: DUSHIN', 'C1S0-
2266                                         BS2: DUSHIN', 'C1S0-BS3: DUSHIN',
2267                                         #                                     'C1S0: BS1', 'C1S0: BS2', 'C1S0: BS3', 'C1S0-BS1: DUSHIN', 'C1S0-
2268                                         S0-BS2: DUSHIN', 'C-1S0-BS3: DUSHIN',
2269                                         #
2270                                         #                                     'User', 'Comment', 'Activity Log', 'Last Updated'],
2271                                         #                                     values = [molecule_id, iupac_name, SMILES, MMFFO,
2272                                         #                                               functional, basis_set_1, basis_set_2, basis_set_3,
2273                                         #                                               'Not Started', 'Not Started', 'Not Started',
2274                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2275                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2276                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2277                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2278                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2279                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2280                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2281                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2282                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2283                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2284                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2285                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2286                                         #                                               'Not Started', 'Not Started', 'Not Started', 'Not Started',
2287                                         if len(duplicates) == 0:
2288                                         create_empty_entry(molecule_id = molecule_id,
2289                                         iupac_name = iupac_name,
2290                                         SMILES = SMILES,
2291                                         MMFFO = MMFFO,
2292                                         functional = functional,
2293                                         basis_set_1 = basis_set_1,
2294                                         basis_set_2 = basis_set_2,
2295                                         basis_set_3 = basis_set_3,
2296                                         username = self._username)
2297                                         else:
2298                                         print('\nDuplicate found:')
2299                                         print(duplicates)
2300                                         print('\nDo you want to proceed with adding entry? (Y/N)')
2301                                         reply = input()
2302                                         if reply == 'Y' or reply == 'y':
2303                                         create_empty_entry(molecule_id = molecule_id,
2304                                         iupac_name = iupac_name,
2305                                         SMILES = SMILES,
2306                                         MMFFO = MMFFO,
2307                                         functional = functional,
2308                                         basis_set_1 = basis_set_1,
2309                                         basis_set_2 = basis_set_2,
2310                                         basis_set_3 = basis_set_3,
2311                                         username = self._username)
2312                                         del reply
2313                                         # self.export_to_excel()
2314                                         return molecule_id
2315
2316 def update_entry(self,
2317     _id: str,
2318     keys: list,
2319     values: list):
2320     self.print_if_debug('\nIn function Tracker.update_entry():')
2321     # dataframe = self.pull_tracker()
2322     keys.append('Last Updated')
2323     values.append('%s' % (timestamp()))
2324     # if _id not in datafarme['_id'].values:
2325     #     raise ValueError('_id not found in database: %s' % (_id))
2326     if len(keys) != len(values):
2327         raise ValueError('Length of keys and values does not match.')
2328     self._MongoDB.update_one_entry(_id = _id,
2329                                   keys = keys,
2330                                   values = values)
2331
2332     # self.export_to_excel()
2333
2334 #%%
2335
2336 class Results:
2337
2338     def __init__(self,
2339         username: str,
2340         password: str,
2341         collection: str = 'Results',
2342         database: str = 'New_Reorganisation_Energy_Calc',
2343         connection_string: str = 'mongodb+srv://<user>:<password>@cluster0.wxpvcwv.mongodb.net/?retryWrites=true&
2344         w=majority&appName=Cluster0',
2345         debug: bool = False):

```

```

2333     self._username = username
2334     self._password = password
2335     self._collection = collection
2336     self._database = database
2337     self._connection_string = connection_string
2338     self._debug = debug
2339
2340     self._MongoDB = MongoDB(username = self._username,
2341                             password = self._password,
2342                             collection = self._collection,
2343                             database = self._database,
2344                             connection_string = self._connection_string,
2345                             debug = self._debug)
2346
2347     def print_if_debug(self, line):
2348         if self._debug == True:
2349             print(line)
2350
2351     def pull_results(self):
2352         if self._debug:
2353             self.print_if_debug('\nIn function Results.pull_results():')
2354         try:
2355             dataframe = self._MongoDB.pull()
2356             return dataframe
2357         except:
2358             raise Exception('Failed to fetch results database.')
2359
2360     def export_to_excel(self, parent_folder: str = ''):
2361         if parent_folder == '':
2362             parent_folder = r'%s' % (os.getcwd())
2363
2364         dataframe = self.pull_results()
2365         if self._debug:
2366             self.print_if_debug('\nIn function Results.export_to_excel():')
2367             self.print_if_debug('Export to location:\n%s' % (parent_folder))
2368             pprint.pprint(dataframe)
2369         try:
2370             dataframe.to_excel(r'%s/Results.xlsx' % (parent_folder))
2371         except:
2372             raise Exception('\nExport to excel not sucessfull.')
2373
2374     def create_empty_entry(self,
2375                           molecule_id: str,
2376                           iupac_name: str,
2377                           SMILES: str,
2378                           MMFFO: str,
2379                           functional: str,
2380                           basis_set: str,
2381                           initial_charge_and_state: str,
2382                           final_charge_and_state: str):
2383
2384         # try:
2385         self._MongoDB.push_one_new_entry(keys = ['_id', 'IUPAC Name', 'SMILES', 'MMFFO', 'Functional', 'Basis Set',
2386                                         'Initial Charge and State', 'Final Charge and State', 'Alpha Occ Orbitals
2387                                         List', 'Alpha Virt Orbitals List', 'HOMO-LUMO Gap',
2388                                         'Initial CS at Initial Optimised Geometry', 'Initial CS at Final
2389                                         Optimised Geometry', 'Final CS at Initial Optimised Geometry', 'Final CS at Final Optimised Geometry',
2390                                         'Vibrational Frequencies', 'Displacements', 'Reorganisation Energies', ,
2391                                         'Parsed SCF Summary', 'Initial Atoms Summary',
2392                                         'Final Atoms Summary',
2393                                         '4-Point Reorganisation Energy', 'DUSHIN Reorganisation Energy',
2394                                         'Difference (4-point & DUSHIN)', 'Difference Percentage (4-point & DUSHIN
2395                                         )',
2396                                         'Comment', 'Last Updated'],
2397                                         values = ['%s_%s_%s_%s_%s' % (molecule_id, functional, format_basis_set(basis_set
2398                                         ), initial_charge_and_state, final_charge_and_state), iupac_name, SMILES, MMFFO, functional, basis_set,
2399                                         initial_charge_and_state, final_charge_and_state, [], [], 0,
2400                                         None, None, None, None,
2401                                         [], [], [], '', '',
2402                                         None, None,
2403                                         None, None,
2404                                         None, None])
2405
2406         self.print_if_debug('Entry sucessfully created.')
2407         # except:
2408         #     self.print_if_debug('Entry already exist.')
2409
2410     def update_entry(self,
2411                     molecule_id: str,
2412                     keys: str,
2413                     values: str):
2414
2415         self.print_if_debug('\nIn function Results.update_entry():')
2416         # dataframe = self.pull_results()
2417         keys.append('Last Updated')
2418         values.append('%s' % (timestamp()))
2419         # if _id not in datafram['_id'].values:
2420         #     raise ValueError('_id not found in database: %s' % (_id))
2421         if len(keys) != len(values):
2422             raise ValueError('Length of keys and values does not match.')

```

```

2414     self._MongoDB.update_one_entry(_id = molecule_id,
2415                                     keys = keys,
2416                                     values = values)
2417     # self.export_to_excel()
2418
2419 #%%
2420
2421 class ReorganisationEnergy():
2422
2423     def __init__(self,
2424                  HPC_username: str,
2425                  HPC_password: str,
2426                  HPC_hostname: str,
2427                  MongoDB_username: str,
2428                  MongoDB_password: str,
2429                  MongoDB_database: str = 'New_Reorganisation_Energy_Calc',
2430                  MongoDB_SMILES_database_collection: str = 'SMILES_Database',
2431                  MongoDB_tracker_collection: str = 'Tracker',
2432                  MongoDB_results_collection: str = 'Results',
2433                  MongoDB_connection_string: str = 'mongodb+srv://<user>:<password>@cluster0.wxpvcwv.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0',
2434                  local_folder: str = r'%s' % (os.getcwd()),
2435                  remote_folder: str = r'/rds/general/ephemeral/user/your_username/ephemeral/test',
2436                  debug: str = False):
2437
2438         self._debug = debug
2439         self._current_working_directory = os.getcwd()
2440
2441         self._SMILES_database = SMILES_database(username = MongoDB_username,
2442                                               password = MongoDB_password,
2443                                               collection = MongoDB_SMILES_database_collection,
2444                                               database = MongoDB_database,
2445                                               connection_string = MongoDB_connection_string,
2446                                               debug = self._debug)
2447
2448         self._HPC_connection = HPC(username = HPC_username,
2449                                   password = HPC_password,
2450                                   hostname = HPC_hostname,
2451                                   debug = self._debug)
2452
2453         self._local_folder = local_folder
2454         self._remote_folder = remote_folder
2455
2456         self._funnel = Funnel(path = r'%s/Funnel.xlsx' % (local_folder),
2457                               debug = self._debug)
2458
2459         self._tracker = Tracker(username = MongoDB_username,
2460                                password = MongoDB_password,
2461                                collection = MongoDB_tracker_collection,
2462                                database = MongoDB_database,
2463                                connection_string = MongoDB_connection_string,
2464                                debug = self._debug)
2465
2466         self._results = Results(username = MongoDB_username,
2467                                password = MongoDB_password,
2468                                collection = MongoDB_results_collection,
2469                                database = MongoDB_database,
2470                                connection_string = MongoDB_connection_string,
2471                                debug = self._debug)
2472
2473     def print_if_debug(self, lines:str):
2474         if self._debug:
2475             print(lines)
2476
2477     def get_tracker(self):
2478         # self._tracker.export_to_excel(parent_folder = self._local_folder)
2479         return self._tracker.pull_tracker()
2480
2481     def get_results(self):
2482         # self._tracker.export_to_excel(parent_folder = self._local_folder)
2483         return self._results.pull_results()
2484
2485     def download_tracker_as_excel(self):
2486         self._tracker.export_to_excel(parent_folder = self._local_folder)
2487
2488     def download_results_as_excel(self):
2489         self._results.export_to_excel(parent_folder = self._local_folder)
2490
2491     def get_funnel(self):
2492         return self._funnel.retrieve()
2493
2494     def get_SMILES_database(self):
2495         return self._SMILES_database.pull_SMILES_database()
2496
2497     def get_HPC_qstat(self):
2498         return self._HPC_connection.formatted_qstat()

```

```

2499     def add_workflow(self,
2500         MMFFO: str = 'Yes',
2501             functional: str = 'B3LYP',
2502                 basis_set_1: str = '6-31G(d)',
2503                     basis_set_2: str = '6-311+G(d,p)',
2504                         basis_set_3: str = 'No'):
2505             self.print_if_debug('\nIn function ReorganisationEnergy.add_workflow():')
2506             try:
2507                 self._funnel.load_from_excel()
2508                 print('\nPrevious Workflow found and extracted.')
2509             except:
2510                 pass
2511             self._funnel.add_workflow(MMFFO,
2512                 functional = functional,
2513                     basis_set_1 = basis_set_1,
2514                         basis_set_2 = basis_set_2,
2515                             basis_set_3 = basis_set_3)
2516             self._funnel.export_to_excel()
2517             print('\nWorkflow added sucessfully. Current workflow:')
2518             print(self._funnel.retrieve())
2519
2520     def add_molecule(self, iupac_name: str, SMILES: str = ''):
2521         self.print_if_debug('\nIn function ReorganisationEnergy.add_molecule():')
2522         try:
2523             self._funnel.load_from_excel()
2524             print('\nWorkflow:Y')
2525             print(self._funnel.retrieve())
2526         except:
2527             raise Exception('No workflow found.')
2528         iupac_name, SMILES = self._SMILES_database.check_against_database(iupac_name = iupac_name,
2529                                         SMILES = SMILES)
2530         for index in self._funnel.workflow_indices():
2531             molecule_id = self._tracker.add_entry(iupac_name = iupac_name,
2532                 SMILES = SMILES,
2533                     MMFFO = self._funnel.MMFFO(index = index),
2534                         functional = self._funnel.functional(index = index),
2535                             basis_set_1 = self._funnel.basis_set_1(index = index),
2536                                 basis_set_2 = self._funnel.basis_set_2(index = index),
2537                                     basis_set_3 = self._funnel.basis_set_3(index = index))
2538             for basis_set in [self._funnel.basis_set_1(index = index), self._funnel.basis_set_2(index = index), self._funnel.basis_set_3(index = index)]:
2539                 self._results.create_empty_entry(molecule_id = molecule_id,
2540                     iupac_name = iupac_name,
2541                         SMILES = SMILES,
2542                             MMFFO = self._funnel.MMFFO(index = index),
2543                                 functional = self._funnel.functional(index = index),
2544                                     basis_set = basis_set,
2545                                         initial_charge_and_state = 'COSO',
2546                                             final_charge_and_state = 'COS1')
2547             self._results.create_empty_entry(molecule_id = molecule_id,
2548                 iupac_name = iupac_name,
2549                     SMILES = SMILES,
2550                         MMFFO = self._funnel.MMFFO(index = index),
2551                             functional = self._funnel.functional(index = index),
2552                                 basis_set = basis_set,
2553                                     initial_charge_and_state = 'COSO',
2554                                         final_charge_and_state = 'C1SO')
2555             self._results.create_empty_entry(molecule_id = molecule_id,
2556                 iupac_name = iupac_name,
2557                     SMILES = SMILES,
2558                         MMFFO = self._funnel.MMFFO(index = index),
2559                             functional = self._funnel.functional(index = index),
2560                                 basis_set = basis_set,
2561                                     initial_charge_and_state = 'COSO',
2562                                         final_charge_and_state = 'C-1SO')
2563             molecule = Molecule(molecule_name = iupac_name,
2564                 molecule_id = molecule_id,
2565                     SMILES = SMILES,
2566                         parent_folder = self._local_folder)
2567             molecule.create_molecule_documentation(allow_aromatic_bond = True)
2568             self.print_if_debug('Molecule workflow added.')
2569
2570     def batch_add_molecule(self, filepath: str):
2571         self.print_if_debug('\nIn function ReorganisationEnergy.batch_add_molecule():')
2572         dataframe = pd.read_excel(filepath)
2573         for line in range(0, len(dataframe)):
2574             iupac_name = dataframe.at[line, 'iupac_name']
2575             SMILES = dataframe.at[line, 'SMILES']
2576             self.add_molecule(iupac_name = iupac_name, SMILES = SMILES)
2577             print('Batch write: %s added.' % (iupac_name))
2578
2579     def get_entries_from_molecule_id(self, molecule_id: str):
2580         dataframe = self.get_tracker()
2581         return dataframe.loc[dataframe['_id'] == '%s' % (molecule_id)]
2582
2583

```

```

2584     def submit_molecule_for_calculation(self,
2585                                         input_object,
2586                                         molecule_id: str,
2587                                         excitation_state: int,
2588                                         charge: int,
2589                                         functional: str,
2590                                         basisset: str,
2591                                         number_of_proc: int,
2592                                         memory: int,
2593                                         timeout_hr: int,
2594                                         connectivity: bool = True,
2595                                         keyword_line_arguments_input: list[[str, str]] = []):
2596         molecule = Molecule.load_molecule(molecule_id = molecule_id,
2597                                           parent_folder = self._local_folder,
2598                                           debug = self._debug)
2599         molecule.generate_gjf_sh_files(input_object = input_object,
2600                                         excited_state = excitation_state,
2601                                         charge = charge,
2602                                         functional = functional,
2603                                         basisset = basisset,
2604                                         number_of_proc = number_of_proc,
2605                                         memory = memory,
2606                                         timeout_hr = timeout_hr,
2607                                         connectivity = connectivity,
2608                                         keyword_line_arguments_input = keyword_line_arguments_input)
2609         job_id = self._HPC_connection.submit_job_gaussian(local_folder = r'%s/Data/%s' % (self._local_folder, molecule_id)
2610 ,
2611                                         remote_folder = self._remote_folder,
2612                                         filename = r'%s_%s_%s_C%sS%s' % (molecule_id, functional,
2613                                         format_basis_set(basisset), charge, excitation_state))
2614         return job_id
2615
2616     def restart_molecule_calculation(self,
2617                                         molecule_id: str,
2618                                         excitation_state: int,
2619                                         charge: int,
2620                                         functional: str,
2621                                         basisset: str,
2622                                         number_of_proc: int,
2623                                         memory: int,
2624                                         timeout_hr: int,
2625                                         keyword_line_arguments_input: list[[str, str]] = []):
2626         molecule = Molecule.load_molecule(molecule_id = molecule_id,
2627                                           parent_folder = self._local_folder,
2628                                           debug = self._debug)
2629         molecule.generate_gjf_sh_restart_files(excited_state = excitation_state,
2630                                         charge = charge,
2631                                         functional = functional,
2632                                         basisset = basisset,
2633                                         number_of_proc = number_of_proc,
2634                                         memory = memory,
2635                                         timeout_hr = timeout_hr,
2636                                         keyword_line_arguments_input = keyword_line_arguments_input)
2637         job_id = self._HPC_connection.restart_job_gaussian(local_folder = r'%s/Data/%s' % (self._local_folder, molecule_id
2638 ),
2639                                         remote_folder = self._remote_folder,
2640                                         filename = r'%s_%s_%s_C%sS%s' % (molecule_id, functional,
2641                                         format_basis_set(basisset), charge, excitation_state))
2642         return job_id
2643
2644     def submit_molecule_for_marcus_calculation(self,
2645                                         molecule_id: str,
2646                                         charge_geom: int,
2647                                         charge_eval: int,
2648                                         state_geom: int,
2649                                         state_eval: int,
2650                                         functional: str,
2651                                         basisset: str,
2652                                         number_of_proc: int,
2653                                         memory: int,
2654                                         timeout_hr: int,
2655                                         keyword_line_arguments_input: list[[str, str]] = []):
2656         molecule = Molecule.load_molecule(molecule_id = molecule_id,
2657                                           parent_folder = self._local_folder,
2658                                           debug = self._debug)
2659         molecule.generate_gjf_sh_marcus_files(charge_geom = charge_geom,
2660                                         charge_eval = charge_eval,
2661                                         state_geom = state_geom,
2662                                         state_eval = state_eval,
2663                                         functional = functional,
2664                                         basisset = basisset,
2665                                         number_of_proc = number_of_proc,
                                         memory = memory,
                                         timeout_hr = timeout_hr,
                                         keyword_line_arguments_input = keyword_line_arguments_input)

```

```

2666     job_id = self._HPC_connection.submit_job_gaussian(local_folder = r'%s/Data/%s' % (self._local_folder, molecule_id)
2667
2668         ,
2669         remote_folder = self._remote_folder,
2670         filename = r'%s_%s_%s_C%sS%s_opt_geom_at_C%sS%s' % (molecule_id,
2671         functional, format_basis_set(basisset), charge_geom, state_geom, charge_eval, state_eval))
2672     return job_id
2673
2674
2675
2676
2677 def transfer_chk_fchk_log_files_from_HPC(self,
2678                                         molecule_id: str,
2679                                         charge: int,
2680                                         excitation_state: int,
2681                                         functional: str,
2682                                         basisset: str):
2683     self._HPC_connection.transfer_chk_fchk_log_files_from_HPC(local_folder = r'%s/Data/%s' % (self._local_folder,
2684     molecule_id),
2685
2686         ,
2687         remote_folder = r'%s' % (self._remote_folder),
2688         filename = r'%s_%s_%s_C%sS%s' % (molecule_id, functional
2689         , format_basis_set(basisset), charge, excitation_state))
2690
2691
2692 def transfer_chk_fchk_log_marcus_files_from_HPC(self,
2693                                         molecule_id: str,
2694                                         charge_geom: int,
2695                                         charge_eval: int,
2696                                         state_geom: int,
2697                                         state_eval: int,
2698                                         functional: str,
2699                                         basisset: str):
2700     self._HPC_connection.transfer_chk_fchk_log_files_from_HPC(local_folder = r'%s/Data/%s' % (self._local_folder,
2701     molecule_id),
2702
2703         ,
2704         remote_folder = r'%s' % (self._remote_folder),
2705         filename = r'%s_%s_%s_C%sS%s_opt_geom_at_C%sS%s' % (
2706     molecule_id, functional, format_basis_set(basisset), charge_geom, state_geom, charge_eval, state_eval))
2707
2708
2709 def memory_assignment(self, molecule_id: str):
2710     SMILES = self.get_entries_from_molecule_id(molecule_id = molecule_id)['SMILES'].values[0]
2711     molecular_weight = Descriptors.ExactMolWt(Chem.MolFromSmiles(SMILES))
2712     out: dict = {'number_of_proc': 0,
2713                 'memory': 0,
2714                 'timeout_hr': 0}
2715     if molecular_weight <= 50:                                              # ethene
2716         out['number_of_proc'] = 8      # 8
2717         out['memory'] = 24      # 8 * 3GB
2718         out['timeout_hr'] = 3      # 3 hr
2719     elif molecular_weight > 50 and molecular_weight <= 100:           # benzene
2720         out['number_of_proc'] = 16     # 16
2721         out['memory'] = 48      # 16 * 3GB
2722         out['timeout_hr'] = 5      # 7 hr
2723     elif molecular_weight > 100 and molecular_weight <= 200:          # napthalene, anthracene
2724         out['number_of_proc'] = 32     # 32
2725         out['memory'] = 96      # 32 * 3GB
2726         out['timeout_hr'] = 7      # 10 hr
2727     elif molecular_weight > 200 and molecular_weight <= 300:          # tetracene, pentacene
2728         out['number_of_proc'] = 64     # 64
2729         out['memory'] = 256      # 64 * 4GB
2730         out['timeout_hr'] = 9      # 15 hr
2731     elif molecular_weight > 300 and molecular_weight <= 400:          # hexacene, heptacene
2732         out['number_of_proc'] = 80     # 80
2733         out['memory'] = 320      # 80 * 4GB
2734         out['timeout_hr'] = 11      # 23 hr
2735     elif molecular_weight > 400:                                         # Octacene and above
2736         out['number_of_proc'] = 90     # 90
2737         out['memory'] = 360      # 90 * 4GB
2738         out['timeout_hr'] = 15      # 48 hr
2739     self.print_if_debug(out)
2740     return out
2741
2742
2743
2744 def add_activity_log(self, molecule_id: str, line: str):
2745     filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
2746     entry = filtered_line.loc[:, 'Activity Log'].values[0]
2747     new_entry = entry + '\n' + line
2748     self._tracker.update_entry(_id = molecule_id, keys = ['Activity Log'], values = [new_entry])
2749     self.print_if_debug('Activity log updated for %s:' % (molecule_id))
2750     self.print_if_debug('> %s' % (line))
2751
2752
2753
2754 def add_comment(self, user: str, molecule_id: str, line: str):
2755     filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
2756     entry = filtered_line.loc[:, 'Comment'].values[0]
2757     if entry == 'No comment':
2758         new_entry = '%s: %s' % (user, line)
2759     else:
2760         new_entry = '%s\n%s: %s' % (entry, user, line)
2761     self._tracker.update_entry(_id = molecule_id, keys = ['Comment'], values = [new_entry])
2762     self.print_if_debug('Comment updated for %s:' % (molecule_id))
2763     self.print_if_debug('> %s' % (line))
2764
2765
2766 def comment_result(self,

```

```

2746             _id: str,
2747             user: str,
2748             string: str):
2749     results = self.get_results()
2750     current_comment = results.loc[results['_id'] == _id, 'Comment'].values[0]
2751     if current_comment != None:
2752         comment = (current_comment + '\n' +
2753                     '%s: ' % (user) + string)
2754     else:
2755         comment = '%s: ' % (user) + string
2756     self._results.update_entry(molecule_id = _id,
2757                                 keys = ['Comment'],
2758                                 values = [comment])
2759
2760 def gaussian_normal_termination(self,
2761                                 molecule_id: str,
2762                                 functional: str,
2763                                 basis_set: str,
2764                                 charge: int,
2765                                 excitation_state: int):
2766     # try:
2767     #     with open(r'%s/Data/%s/%s_%s_%s_C%sS%.log' % (self._parent_folder, molecule_id, molecule_id, functional,
2768     #                                                     format_basis_set(basis_set), charge, excitation_state)) as file:
2769     #         word = file.readlines()[-1].split(' ')
2770     #         if 'Normal' in word:
2771     #             return True
2772     #         else:
2773     #             return False
2774     # except:
2775     #     raise Exception('\nUnable to open file:\n %s/Data/%s/%s_%s_%s_C%sS%.log' % (self._parent_folder,
molecule_id, molecule_id, functional,
2776     #                                                     format_basis_set(basis_set), charge, excitation_state))
2777     try:
2778         with open(r'%s/Data/%s/%s_%s_%s_C%sS%.log' % (self._local_folder, molecule_id, molecule_id, functional,
format_basis_set(basis_set), charge, excitation_state)) as file
2779     :
2780         whole_file = file.read()
2781         lines = whole_file.split('\n')
2782         normal_count = 0
2783         warning_count = 0
2784         comment_string = ''
2785         for line in lines:
2786             if 'Normal' in line:
2787                 normal_count += 1
2788                 self.print_if_debug('\nNormal termination statement %s: %s' % (normal_count, line))
2789                 comment_string = comment_string + '\nAuto_Generated: %s C%sS% Normal termination statement %s: %s' %
(basis_set, charge, excitation_state, normal_count, line)
2790                 # self.add_comment(user = 'Auto-Generated',
2791                 #                   molecule_id = molecule_id,
2792                 #                   line = '%s C%sS% Normal termination statement %s: %s' % (basis_set, charge,
excitation_state, normal_count, line))
2793             if '*** Warning!!' in line:
2794                 warning_count += 1
2795                 self.print_if_debug('\nWarning statement %s: %s' % (warning_count, line))
2796                 comment_string = comment_string + '\nAuto_Generated: %s C%sS% Warning statement %s: %s' %
(basis_set, charge, excitation_state, warning_count, line)
2797                 # self.add_comment(user = 'Auto-Generated',
2798                 #                   molecule_id = molecule_id,
2799                 #                   line = '%s C%sS% Warning statement %s: %s' % (basis_set, charge,
excitation_state, warning_count, line))
2800             if normal_count == 2:
2801                 self.print_if_debug('Normal termination count = 2, SUCESSFULL OPT and FREQ calculation.')
2802                 comment_string = comment_string + '\nAuto_Generated: %s C%sS% Normal termination count = 2,
SUCESSFULL OPT and FREQ calculation.' % (basis_set, charge, excitation_state)
2803                 comment_string = comment_string.replace('\nAuto-Generated: ', '', 1)
2804                 self.add_comment(user = 'Auto-Generated',
2805                                 molecule_id = molecule_id,
2806                                 line = comment_string)
2807             return 'Completed'
2808         elif normal_count == 1:
2809             self.print_if_debug('Normal termination count = 1, SUCESSFULL OPT and FAILED FREQ calculation.')
2810             comment_string = comment_string + '\nAuto_Generated: %s C%sS% Normal termination count = 1,
SUCESSFULL OPT and FAILED FREQ calculation.' % (basis_set, charge, excitation_state)
2811             comment_string = comment_string.replace('\nAuto-Generated: ', '', 1)
2812             self.add_comment(user = 'Auto-Generated',
2813                             molecule_id = molecule_id,
2814                             line = comment_string)
2815             return 'Failed'
2816         else:
2817             self.print_if_debug('Normal termination count = 0, FAILED OPT and FREQ calculation.')
2818             comment_string = comment_string + '\nAuto_Generated: %s C%sS% Normal termination count = 0, FAILED
OPT and FREQ calculation.' % (basis_set, charge, excitation_state)
2819             comment_string = comment_string.replace('\nAuto-Generated: ', '', 1)
2820             self.add_comment(user = 'Auto-Generated',

```

```

2821                     molecule_id = molecule_id,
2822                     line = comment_string)
2823             return 'Failed'
2824     except:
2825         raise Exception('\nUnable to open file:\n %s/Data/%s/%s_%s_%s_C%sS%.log' % (self._local_folder, molecule_id,
2826                                         functional,
2827                                         format_basis_set(basis_set),
2828                                         charge, excitation_state))
2829 def gaussian_marcus_normal_termination(self,
2830                                         molecule_id: str,
2831                                         functional: str,
2832                                         basis_set: str,
2833                                         charge_geom: int,
2834                                         charge_eval: int,
2835                                         state_geom: int,
2836                                         state_eval: int):
2837     try:
2838         with open(r'%s/Data/%s/%s_%s_%s_C%sS%.opt_geom_at_C%sS%.log' % (self._local_folder, molecule_id, molecule_id
2839 , functional, format_basis_set(basis_set), charge_geom, state_geom, charge_eval, state_eval)) as file:
2840             whole_file = file.read()
2841             lines = whole_file.split('\n')
2842             normal_count = 0
2843             warning_count = 0
2844             comment_string = ''
2845             for line in lines:
2846                 if 'Normal' in line:
2847                     normal_count += 1
2848                     self.print_if_debug('\nNormal termination statement %s: %s' % (normal_count, line))
2849                     comment_string = comment_string + '\nAuto-Generated: %s C%sS%.opt_geom_at_C%sS% Normal
2850 termination statement %s: %s' % (basis_set, charge_geom, state_geom, charge_eval, state_eval, normal_count, line)
2851                     # self.add_comment(user = 'Auto-Generated',
2852                     #                     molecule_id = molecule_id,
2853                     #                     line = '%s C%sS%.opt_geom_at_C%sS% Normal termination statement %s: %s' %
2854 basis_set, charge_geom, state_geom, charge_eval, state_eval, normal_count, line))
2855                     if '**** Warning!!' in line:
2856                         warning_count += 1
2857                         self.print_if_debug('\nWarning statement %s: %s' % (warning_count, line))
2858                         comment_string = comment_string + '\nAuto-Generated: %s C%sS%.opt_geom_at_C%sS% Warning
2859 statement %s: %s' % (basis_set, charge_geom, state_geom, charge_eval, state_eval, warning_count, line)
2860                         # self.add_comment(user = 'Auto-Generated',
2861                         #                     molecule_id = molecule_id,
2862                         #                     line = '%s C%sS%.opt_geom_at_C%sS% Warning statement %s: %s' %
2863 charge_geom, state_geom, charge_eval, state_eval, warning_count, line))
2864                     if normal_count == 1:
2865                         self.print_if_debug('Normal termination count = 1, SUCESSFULL calculation.')
2866                         comment_string = comment_string + '\nAuto-Generated: %s C%sS%.opt_geom_at_C%sS% Normal termination
2867 count = 1, SUCESSFULL calculation.' % (basis_set, charge_geom, state_geom, charge_eval, state_eval)
2868                         comment_string = comment_string.replace('\nAuto-Generated: ', '', 1)
2869                         self.add_comment(user = 'Auto-Generated',
2870                                         molecule_id = molecule_id,
2871                                         line = comment_string)
2872                     return 'Completed'
2873                 else:
2874                     self.print_if_debug('Normal termination count = 0, FAILED OPT and FREQ calculation.')
2875                     comment_string = comment_string + '\nAuto-Generated: %s C%sS%.opt_geom_at_C%sS% Normal termination
2876 count = 0, FAILED OPT and FREQ calculation.' % (basis_set, charge_geom, state_geom, charge_eval, state_eval)
2877                     comment_string = comment_string.replace('\nAuto-Generated: ', '', 1)
2878                     self.add_comment(user = 'Auto-Generated',
2879                                     molecule_id = molecule_id,
2880                                     line = comment_string)
2881             return 'Failed'
2882     except:
2883         raise Exception('\nUnable to open file:\n %s/Data/%s/%s_%s_%s_C%sS%.opt_geom_at_C%sS%.log' % (self.
2884 _local_folder, molecule_id, molecule_id, functional, format_basis_set(basis_set), charge_geom, state_geom, charge_eval
2885 , state_eval))
2886 def query_stage_started(self,
2887                         molecule_id: str,
2888                         charge: int,
2889                         excitation_state: int,
2890                         basis_set: int) -> bool:
2891     filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
2892     entry = filtered_line.loc[:, 'C%sS%: BS%' % (charge, excitation_state, basis_set)].values[0]
2893     if entry == 'Not Started':
2894         return False
2895     else:
2896         return True
2897
2898 def count_running_optimisations(self, alternative: str = '') -> int:
2899     tracker = self.get_tracker()[['COS0: BS1', 'COS0: BS2', 'COS0: BS3',
2900                                 'COS1: BS1 - COS0 geometry', 'COS1: BS2 - COS0 geometry', 'COS1: BS3 - COS0 geometry
2901 ',
2902                                 'C1S0: BS1 - COS0 geometry', 'C1S0: BS2 - COS0 geometry', 'C1S0: BS3 - COS0 geometry
2903 ',
2904                                 'C-1S0: BS1 - COS0 geometry', 'C-1S0: BS2 - COS0 geometry', 'C-1S0: BS3 - COS0
2905 geometry',
2906

```

```

2893     'COS1: BS1', 'COS1: BS2', 'COS1: BS3',
2894     'COS0: BS1 - COS1 geometry', 'COS0: BS2 - COS1 geometry', 'COS0: BS3 - COS1 geometry'
2895     ,
2896     'C1S0: BS1', 'C1S0: BS2', 'C1S0: BS3',
2897     'COS0: BS1 - C1S0 geometry', 'COS0: BS2 - C1S0 geometry', 'COS0: BS3 - C1S0 geometry'
2898     ,
2899     'C-1S0: BS1', 'C-1S0: BS2', 'C-1S0: BS3',
2900     'COS0: BS1 - C-1S0 geometry', 'COS0: BS2 - C-1S0 geometry', 'COS0: BS3 - C-1S0
geometry']]])
2901     if alternative == '':
2902         raise Exception('\nMethod deprecated, use count_running_optimisations_alt() instead.')
2903     return pd.DataFrame(tracker == 'Started').sum().sum()
2904   else:
2905     return pd.DataFrame(tracker == alternative).sum().sum()
2906
2907 def count_running_optimisations_alt(self, exclude: list[str] = ['Not Started', 'Completed', 'Failed']):
2908   tracker = self.get_tracker()
2909   count = len(tracker) * 30
2910   for i in range(0, len(exclude)):
2911     count -= self.count_running_optimisations(alternative = exclude[i])
2912   return count
2913
2914 def get_tracker_not_started(self):
2915   tracker = self.get_tracker()
2916   out = pd.DataFrame([])
2917   for index in range(0, len(tracker)):
2918     line = tracker.iloc[[index]]
2919     filtered_line = line[['COS0: BS1', 'COS0: BS2', 'COS0: BS3',
2920                           'COS1: BS1 - COS0 geometry', 'COS1: BS2 - COS0 geometry', 'COS1: BS3 - COS0 geometry',
2921                           'C1S0: BS1 - COS0 geometry', 'C1S0: BS2 - COS0 geometry', 'C1S0: BS3 - COS0 geometry',
2922                           'C-1S0: BS1 - COS0 geometry', 'C-1S0: BS2 - COS0 geometry', 'C-1S0: BS3 - COS0 geometry',
2923                           'COS0: BS1 - C1S0 geometry', 'COS0: BS2 - C1S0 geometry', 'COS0: BS3 - C1S0 geometry',
2924                           'C-1S0: BS1', 'C-1S0: BS2', 'C-1S0: BS3',
2925                           'COS0: BS1 - C-1S0 geometry', 'COS0: BS2 - C-1S0 geometry', 'COS0: BS3 - C-1S0 geometry']]
2926   ]
2927   if any(filtered_line.isin(['Not Started']).values[0]):
2928     if len(out) == 0:
2929       out = pd.DataFrame(line)
2930     else:
2931       out = pd.concat([out, line])
2932   return out
2933
2934 def get_tracker_failed(self):
2935   tracker = self.get_tracker()
2936   out = pd.DataFrame([])
2937   for index in range(0, len(tracker)):
2938     line = tracker.iloc[[index]]
2939     filtered_line = line[['COS0: BS1', 'COS0: BS2', 'COS0: BS3',
2940                           'COS1: BS1 - COS0 geometry', 'COS1: BS2 - COS0 geometry', 'COS1: BS3 - COS0 geometry',
2941                           'C1S0: BS1 - COS0 geometry', 'C1S0: BS2 - COS0 geometry', 'C1S0: BS3 - COS0 geometry',
2942                           'C-1S0: BS1 - COS0 geometry', 'C-1S0: BS2 - COS0 geometry', 'C-1S0: BS3 - COS0 geometry',
2943                           'COS1: BS1', 'COS1: BS2', 'COS1: BS3',
2944                           'COS0: BS1 - COS1 geometry', 'COS0: BS2 - COS1 geometry', 'COS0: BS3 - COS1 geometry',
2945                           'C1S0: BS1', 'C1S0: BS2', 'C1S0: BS3',
2946                           'COS0: BS1 - C1S0 geometry', 'COS0: BS2 - C1S0 geometry', 'COS0: BS3 - C1S0 geometry',
2947                           'C-1S0: BS1', 'C-1S0: BS2', 'C-1S0: BS3',
2948                           'COS0: BS1 - C-1S0 geometry', 'COS0: BS2 - C-1S0 geometry', 'COS0: BS3 - C-1S0 geometry']]
2949   ]
2950   if any(filtered_line.isin(['Failed']).values[0]):
2951     if len(out) == 0:
2952       out = pd.DataFrame(line)
2953     else:
2954       out = pd.concat([out, line])
2955   return out
2956
2957 def get_tracker_running_only(self):
2958   tracker = self.get_tracker()
2959   out = pd.DataFrame()
2960   for index in range(0, len(tracker)):
2961     line = tracker.iloc[[index]]
2962     filtered_line = line[['COS0: BS1', 'COS0: BS2', 'COS0: BS3',
2963                           'COS1: BS1 - COS0 geometry', 'COS1: BS2 - COS0 geometry', 'COS1: BS3 - COS0 geometry',
2964                           'C1S0: BS1 - COS0 geometry', 'C1S0: BS2 - COS0 geometry', 'C1S0: BS3 - COS0 geometry',
2965                           'C-1S0: BS1 - COS0 geometry', 'C-1S0: BS2 - COS0 geometry', 'C-1S0: BS3 - COS0 geometry',
2966                           'COS1: BS1', 'COS1: BS2', 'COS1: BS3',
2967                           'COS0: BS1 - COS1 geometry', 'COS0: BS2 - COS1 geometry', 'COS0: BS3 - COS1 geometry',
2968                           'C1S0: BS1', 'C1S0: BS2', 'C1S0: BS3',
2969                           'COS0: BS1 - C1S0 geometry', 'COS0: BS2 - C1S0 geometry', 'COS0: BS3 - C1S0 geometry',
2970                           'C-1S0: BS1', 'C-1S0: BS2', 'C-1S0: BS3',
```

```

2970
2971     ]
2972     count = 0
2973     scan_for = ['Not Started', 'Completed', 'Failed']
2974     for item in scan_for:
2975         count += pd.DataFrame(filtered_line == item).sum().sum()
2976     if count != 12: # start count from 12 and minus, and if more than 0 add, or != 12
2977         if len(out) == 0:
2978             out = pd.DataFrame(line)
2979         else:
2980             out = pd.concat([out, line])
2981     return out
2982
2983 def tracker_summarise_running(self):
2984     running_only = self.get_tracker_running_only()
2985     scan = ['COSO: BS1', 'COSO: BS2', 'COSO: BS3',
2986             'COS1: BS1 - COSO geometry', 'COS1: BS2 - COSO geometry', 'COS1: BS3 - COSO geometry',
2987             'C1SO: BS1 - COSO geometry', 'C1SO: BS2 - COSO geometry', 'C1SO: BS3 - COSO geometry',
2988             'C-1SO: BS1 - COSO geometry', 'C-1SO: BS2 - COSO geometry', 'C-1SO: BS3 - COSO geometry',
2989             'COS1: BS1', 'COS1: BS2', 'COS1: BS3',
2990             'COSO: BS1 - COS1 geometry', 'COSO: BS2 - COS1 geometry', 'COSO: BS3 - COS1 geometry',
2991             'C1SO: BS1', 'C1SO: BS2', 'C1SO: BS3',
2992             'COSO: BS1 - C1SO geometry', 'COSO: BS2 - C1SO geometry', 'COSO: BS3 - C1SO geometry',
2993             'C-1SO: BS1', 'C-1SO: BS2', 'C-1SO: BS3',
2994             'COSO: BS1 - C-1SO geometry', 'COSO: BS2 - C-1SO geometry', 'COSO: BS3 - C-1SO geometry']
2995     out = pd.DataFrame(columns = ['molecule_id', 'run', 'run_id'])
2996     for index in range(0, len(running_only)):
2997         line = running_only.iloc[index, :]
2998         for item in scan:
2999             if line[item] != 'Not Started' and line[item] != 'Completed' and line[item] != 'Failed':
3000                 out.loc[len(out)] = [line['_id'], item, line[item]]
3001     return out
3002
3003 def next_step_for_molecule_id_with_charge_state(self,
3004                                                 molecule_id: str,
3005                                                 charge: int,
3006                                                 excitation_state: int):
3007     raise Exception('Deprecated. Use next_step_for_molecule_id_with_basis_set.')
3008     # charge_state = 'C%sS%' % (charge, excitation_state)
3009     # filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
3010     # if filtered_line.loc[:, '%s: BS1' % (charge_state)].values[0] == 'Not Started':
3011     #     job_parameters = self.memory_assignment(molecule_id = molecule_id)
3012     #     job_id = self.submit_molecule_for_calculation(input_object = '',
3013     #                                                     # use RDKit geometry
3014     #                                                     molecule_id = molecule_id,
3015     #                                                     excitation_state = excitation_state,
3016     #                                                     charge = charge,
3017     #                                                     functional = filtered_line.loc[:, 'Functional'].values[0],
3018     #                                                     basisset = filtered_line.loc[:, 'Basis Set 1'].values[0],
3019     #                                                     number_of_proc = int(job_parameters['number_of_proc']),
3020     #                                                     memory = int(job_parameters['memory']),
3021     #                                                     timeout_hr = int(job_parameters['timeout_hr']),
3022     #                                                     connectivity = True,
3023     #                                                     keyword_line_arguments_input = [])
3024     #     self._tracker.update_entry(_id = molecule_id,
3025     #                               keys = ['%s: BS1' % (charge_state)],
3026     #                               values = [job_id])
3027     #     self.add_activity_log(molecule_id = molecule_id,
3028     #                           line = '%s BS1 optimisation started: %s' % (charge_state, str(datetime.now()).split
3029     #                                                               ('.')[0]))
3030     #     self.print_if_debug('%s %s: BS1 started.' % (molecule_id, charge_state))
3031     #     return True
3032     # elif filtered_line.loc[:, '%s: BS1' % (charge_state)].values[0] == 'Completed' and filtered_line.loc[:, '%s: BS2
3033     # % (charge_state)].values[0] == 'Not Started':
3034     #     job_parameters = self.memory_assignment(molecule_id = molecule_id)
3035     #     job_id = self.submit_molecule_for_calculation(input_object = '%s/Data/%s/%s_%s_%s.log' % (self.
3036     # _local_folder, molecule_id, molecule_id,
3037     #                                         # filtered_line.
3038     #                                         loc[:, 'Functional'].values[0],
3039     #                                         # format_basis_set(filtered_line.loc[:, 'Basis Set 1'].values[0]),
3040     #                                         # use BS1 geometry
3041     #                                         molecule_id = molecule_id,
3042     #                                         excitation_state = excitation_state,
3043     #                                         charge = charge,
3044     #                                         functional = filtered_line.loc[:, 'Functional'].values[0],
3045     #                                         basisset = filtered_line.loc[:, 'Basis Set 2'].values[0],
3046     #                                         number_of_proc = int(job_parameters['number_of_proc']),
3047     #                                         memory = int(job_parameters['memory']),
3048     #                                         timeout_hr = int(job_parameters['timeout_hr']),
3049     #                                         connectivity = True,
3050     #                                         keyword_line_arguments_input = [])
3051     #     self._tracker.update_entry(_id = molecule_id,
3052     #                               keys = ['%s: BS2' % (charge_state)],
3053     #                               values = [job_id])

```

```

3049     #     self.add_activity_log(molecule_id = molecule_id,
3050     #                         line = '%s BS2 optimisation started: %s' % (charge_state, str(datetime.now()).split
3051     # ('.')[0]))
3052     #     self.print_if_debug('%s %s: BS2 started.' % (molecule_id, charge_state))
3053     #     return True
3054     # elif filtered_line.loc[:, 's: BS2' % (charge_state)].values[0] == 'Completed' and filtered_line.loc[:, 's: BS3
3055     # % (charge_state)].values[0] == 'Not Started':
3056     #     job_parameters = self.memory_assignment(molecule_id = molecule_id)
3057     #     job_id = self.submit_molecule_for_calculation(input_object = '%s/Data/%s/%s_%s_%s.log' % (self.
3058     _local_folder, molecule_id, molecule_id,
3059     #                                         filtered_line.
3060     loc[:, 'Functional'].values[0],
3061     #     format_basis_set(filtered_line.loc[:, 'Basis Set 2'].values[0]),
3062     #             charge_state),
3063     #             # use BS2 geometry
3064     #             molecule_id = molecule_id,
3065     #             excitation_state = excitation_state,
3066     #             charge = charge,
3067     #             functional = filtered_line.loc[:, 'Functional'].values[0],
3068     #             basisset = filtered_line.loc[:, 'Basis Set 2'].values[0],
3069     #             number_of_proc = int(job_parameters['number_of_proc']),
3070     #             memory = int(job_parameters['memory']),
3071     #             timeout_hr = int(job_parameters['timeout_hr']),
3072     #             connectivity = True,
3073     #             keyword_line_arguments_input = [])
3074     #     self._tracker.update_entry(_id = molecule_id,
3075     #                             keys = ['s: BS3' % (charge_state)],
3076     #                             values = [job_id])
3077     #     self.add_activity_log(molecule_id = molecule_id,
3078     #                         line = '%s BS3 optimisation started: %s' % (charge_state, str(datetime.now()).split
3079     # ('.')[0]))
3080     #     self.print_if_debug('%s %s: BS3 started.' % (molecule_id, charge_state))
3081     #     return True
3082     # else:
3083     #     return False
3084
3085 def restart_gaussian_for_molecule(self,
3086                                     molecule_id: str,
3087                                     basis_set: int,
3088                                     excitation_state: int,
3089                                     charge: int):
3090     filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
3091     job_parameters = self.memory_assignment(molecule_id = molecule_id)
3092     job_id = self.restart_molecule_calculation(molecule_id = molecule_id,
3093                                                 excitation_state = excitation_state,
3094                                                 charge = charge,
3095                                                 functional = filtered_line.loc[:, 'Functional'].values[0],
3096                                                 basisset = filtered_line.loc[:, 'Basis Set %s' % (basis_set)].values
3097     [0],
3098     number_of_proc = int(job_parameters['number_of_proc']),
3099     memory = int(job_parameters['memory']),
3100     timeout_hr = int(job_parameters['timeout_hr']),
3101     keyword_line_arguments_input = [])
3102     self._tracker.update_entry(_id = molecule_id,
3103     keys = ['C%sS%s: BS%s' % (charge, excitation_state, basis_set)],
3104     values = [job_id])
3105     self.add_activity_log(molecule_id = molecule_id,
3106     line = 'C%sS%s BS%s optimisation restarted: %s' % (charge, excitation_state, basis_set, str(
3107     datetime.now().split('.')[0]))
3108     self.print_if_debug('%s C%sS%s: BS%s started.' % (charge, excitation_state, molecule_id, basis_set)))
3109
3110 def restart_failed_cases(self,
3111                         stop_time = '',
3112                         max_jobs_parallel: int = 45):
3113     if stop_time == '':
3114         stop_time = datetime.now() + timedelta(minutes = 15)
3115     if datetime.now() >= stop_time:
3116         return
3117     current_running = self.count_running_optimisations_alt()
3118     to_be_run = self.get_tracker_failed().reset_index()
3119     for index in range(0, len(to_be_run)):
3120         molecule_id = to_be_run.at[index, '_id']
3121         filtered_line = to_be_run.loc[to_be_run['_id'] == molecule_id]
3122         columns = filtered_line.columns[filtered_line.isin(['Failed']).any()].tolist()
3123         for header in columns:
3124             if current_running <= max_jobs_parallel:
3125                 header = header.split(' - ') # 'COS1: BS3 - opt geom at COS0'.split(' - ') --> ['COS1: BS3', 'opt
3126                 geom at COS0']
3127                 if len(header) == 1: # only 'COS0: BS1'
3128                     charge_state, basis_set = header[0].split(':') # ['COS1', 'BS3']
3129                     basis_set = int(basis_set.replace('BS', ''))
3130                     # basis_set = filtered_line[' Basis Set %s' % (basis_set)].values[0]
3131                     state = int(charge_state.split('S')[1])
3132                     charge = int(charge_state.split('S')[0].replace('C', ''))
3133                     self.restart_gaussian_for_molecule(molecule_id = molecule_id,
3134

```



```

3203             values = [job_id])
3204         self.add_activity_log(molecule_id = molecule_id,
3205                               line = 'COS1 BS%s optimisation started: %s' % (basis_set, str(datetime.now()).split('.')[0]))
3206         self.print_if_debug('%s COS0: BS%s started.' % (molecule_id, basis_set))
3207         return True
3208     elif filtered_line.loc[:, 'COS0: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'C1S0: BS%s' % (basis_set)].values[0] == 'Not Started':
3209         # start C1S0 #####
3210         charge = 1
3211         excitation_state = 0
3212         charge_state = 'C%sS%' % (charge, excitation_state)
3213         file_path_string = '%s/Data/%s/%s_%s_COS0.log' % (self._local_folder, molecule_id, molecule_id,
3214                                                               filtered_line.loc[:, 'Functional'].values[0],
3215                                                               format_basis_set(filtered_line.loc[:, 'Basis Set %s' % (basis_set)].values[0]))
3216         job_id = self.submit_molecule_for_calculation(input_object = r'%s' % (file_path_string), # use COS0 geometry
3217                                                       molecule_id = molecule_id,
3218                                                       excitation_state = excitation_state,
3219                                                       charge = charge,
3220                                                       functional = filtered_line.loc[:, 'Functional'].values[0],
3221                                                       basisset = filtered_line.loc[:, 'Basis Set %s' % (basis_set)].values[0],
3222                                                       number_of_proc = int(job_parameters['number_of_proc']),
3223                                                       memory = int(job_parameters['memory']),
3224                                                       timeout_hr = int(job_parameters['timeout_hr']),
3225                                                       connectivity = True,
3226                                                       keyword_line_arguments_input = [])
3227         self._tracker.update_entry(_id = molecule_id,
3228                                   keys = ['C1S0: BS%s' % (basis_set)],
3229                                   values = [job_id])
3230         self.add_activity_log(molecule_id = molecule_id,
3231                               line = 'C1S0 BS%s optimisation started: %s' % (basis_set, str(datetime.now()).split('.')[0]))
3232     self.print_if_debug('%s C1S0: BS%s started.' % (molecule_id, basis_set))
3233     return True
3234     elif filtered_line.loc[:, 'COS0: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'C-1S0: BS%s' % (basis_set)].values[0] == 'Not Started':
3235         # start C-1S0 #####
3236         charge = -1
3237         excitation_state = 0
3238         charge_state = 'C%sS%' % (charge, excitation_state)
3239         file_path_string = '%s/Data/%s/%s_%s_COS0.log' % (self._local_folder, molecule_id, molecule_id,
3240                                                               filtered_line.loc[:, 'Functional'].values[0],
3241                                                               format_basis_set(filtered_line.loc[:, 'Basis Set %s' % (basis_set)].values[0]))
3242         job_id = self.submit_molecule_for_calculation(input_object = r'%s' % (file_path_string), # use COS0 geometry
3243                                                       molecule_id = molecule_id,
3244                                                       excitation_state = excitation_state,
3245                                                       charge = charge,
3246                                                       functional = filtered_line.loc[:, 'Functional'].values[0],
3247                                                       basisset = filtered_line.loc[:, 'Basis Set %s' % (basis_set)].values[0],
3248                                                       number_of_proc = int(job_parameters['number_of_proc']),
3249                                                       memory = int(job_parameters['memory']),
3250                                                       timeout_hr = int(job_parameters['timeout_hr']),
3251                                                       connectivity = True,
3252                                                       keyword_line_arguments_input = [])
3253         self._tracker.update_entry(_id = molecule_id,
3254                                   keys = ['C-1S0: BS%s' % (basis_set)],
3255                                   values = [job_id])
3256         self.add_activity_log(molecule_id = molecule_id,
3257                               line = 'C-1S0 BS%s optimisation started: %s' % (basis_set, str(datetime.now()).split('.')[0]))
3258     self.print_if_debug('%s C-1S0: BS%s started.' % (molecule_id, basis_set))
3259     return True
3260     elif filtered_line.loc[:, 'COS0: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'COS1: BS%s - COS0 geometry' % (basis_set)].values[0] == 'Not Started':
3261         # start COS0 optimal geometry at COS1 energy #####
3262         job_id = self.submit_molecule_for_marcus_calculation(molecule_id = molecule_id,
3263                                                               charge_geom = 0,
3264                                                               charge_eval = 0,
3265                                                               state_geom = 0,
3266                                                               state_eval = 1,
3267                                                               functional = filtered_line.loc[:, 'Functional'].values[0],
3268                                                               basisset = filtered_line.loc[:, 'Basis Set %s' % (basis_set)].values[0],
3269                                                               number_of_proc = int(job_parameters['number_of_proc']),
3270                                                               memory = int(job_parameters['memory']),
3271                                                               timeout_hr = int(job_parameters['timeout_hr']),
3272                                                               keyword_line_arguments_input = [])
3273         self._tracker.update_entry(_id = molecule_id,
3274                                   keys = ['COS1: BS%s - COS0 geometry' % (basis_set)],
3275                                   values = [job_id])
3276         self.add_activity_log(molecule_id = molecule_id,

```

```

3277             line = 'C0S1: BS%s - C0S0 geometry calculation started: %s' % (basis_set, str(datetime.
3278         now()).split('.')[0]))
3279         self.print_if_debug('%s C0S1: BS%s - C0S0 geometry started.' % (molecule_id, basis_set))
3280         return True
3281     elif filtered_line.loc[:, 'C0S0: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'C1S0: BS%
3282 s - C0S0 geometry' % (basis_set)].values[0] == 'Not Started':
3283         # start C0S0 optimal geometry at C1S0 energy #####
3284         job_id = self.submit_molecule_for_marcus_calculation(molecule_id = molecule_id,
3285                                                 charge_geom = 0,
3286                                                 charge_eval = 1,
3287                                                 state_geom = 0,
3288                                                 state_eval = 0,
3289                                                 functional = filtered_line.loc[:, 'Functional'].values
3290                                                 [0],
3291                                                 basisset = filtered_line.loc[:, 'Basis Set %s' % (
3292                                                 basis_set)].values[0],
3293                                                 number_of_proc = int(job_parameters['number_of_proc']),
3294                                                 memory = int(job_parameters['memory']),
3295                                                 timeout_hr = int(job_parameters['timeout_hr']),
3296                                                 keyword_line_arguments_input = [])
3297         self._tracker.update_entry(_id = molecule_id,
3298                                     keys = ['C1S0: BS%s - C0S0 geometry' % (basis_set)],
3299                                     values = [job_id])
3300         self.add_activity_log(molecule_id = molecule_id,
3301                             line = 'C1S0: BS%s - C0S0 geometry calculation started: %s' % (basis_set, str(datetime.
3302 now()).split('.')[0]))
3303         self.print_if_debug('%s C1S0: BS%s - C0S0 geometry started.' % (molecule_id, basis_set))
3304         return True
3305     elif filtered_line.loc[:, 'C0S0: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'C-1S0: BS%
3306 s - C0S0 geometry' % (basis_set)].values[0] == 'Not Started':
3307         # start C0S0 optimal geometry at C-1S0 energy #####
3308         job_id = self.submit_molecule_for_marcus_calculation(molecule_id = molecule_id,
3309                                                 charge_geom = 0,
3310                                                 charge_eval = -1,
3311                                                 state_geom = 0,
3312                                                 state_eval = 0,
3313                                                 functional = filtered_line.loc[:, 'Functional'].values
3314                                                 [0],
3315                                                 basisset = filtered_line.loc[:, 'Basis Set %s' % (
3316                                                 basis_set)].values[0],
3317                                                 number_of_proc = int(job_parameters['number_of_proc']),
3318                                                 memory = int(job_parameters['memory']),
3319                                                 timeout_hr = int(job_parameters['timeout_hr']),
3320                                                 keyword_line_arguments_input = [])
3321         self._tracker.update_entry(_id = molecule_id,
3322                                     keys = ['C-1S0: BS%s - C0S0 geometry' % (basis_set)],
3323                                     values = [job_id])
3324         self.add_activity_log(molecule_id = molecule_id,
3325                             line = 'C-1S0: BS%s - C0S0 geometry calculation started: %s' % (basis_set, str(datetime.
3326 now()).split('.')[0]))
3327         self.print_if_debug('%s C-1S0: BS%s - C0S0 geometry started.' % (molecule_id, basis_set))
3328         return True
3329     # C0S1 optimal geometry completed #####
3330     elif filtered_line.loc[:, 'C0S1: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'C0S0: BS%
3331 s - C0S1 geometry' % (basis_set)].values[0] == 'Not Started':
3332         # start C0S1 optimal geometry at C0S0 energy #####
3333         job_id = self.submit_molecule_for_marcus_calculation(molecule_id = molecule_id,
3334                                                 charge_geom = 0,
3335                                                 charge_eval = 0,
3336                                                 state_geom = 1,
3337                                                 state_eval = 0,
3338                                                 functional = filtered_line.loc[:, 'Functional'].values
3339                                                 [0],
3340                                                 basisset = filtered_line.loc[:, 'Basis Set %s' % (
3341                                                 basis_set)].values[0],
3342                                                 number_of_proc = int(job_parameters['number_of_proc']),
3343                                                 memory = int(job_parameters['memory']),
3344                                                 timeout_hr = int(job_parameters['timeout_hr']),
3345                                                 keyword_line_arguments_input = [])
3346         self._tracker.update_entry(_id = molecule_id,
3347                                     keys = ['C0S0: BS%s - C0S1 geometry' % (basis_set)],
3348                                     values = [job_id])
3349         self.add_activity_log(molecule_id = molecule_id,
3350                             line = 'C0S0: BS%s - C0S1 geometry calculation started: %s' % (basis_set, str(datetime.
3351 now()).split('.')[0]))
3352         self.print_if_debug('%s C0S0: BS%s - C0S1 geometry started.' % (molecule_id, basis_set))
3353         return True
3354     # C1S0 optimal geometry completed #####
3355     elif filtered_line.loc[:, 'C1S0: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'C0S0: BS%
3356 s - C1S0 geometry' % (basis_set)].values[0] == 'Not Started':
3357         # start C1S0 optimal geometry at C0S0 energy #####
3358         job_id = self.submit_molecule_for_marcus_calculation(molecule_id = molecule_id,
3359                                                 charge_geom = 1,
3360                                                 charge_eval = 0,
3361                                                 state_geom = 0,
3362                                                 state_eval = 0,
3363                                                 functional = filtered_line.loc[:, 'Functional'].values
3364                                                 [0],
3365                                                 basisset = filtered_line.loc[:, 'Basis Set %s' % (
3366                                                 basis_set)].values[0],
3367                                                 number_of_proc = int(job_parameters['number_of_proc']),
3368                                                 memory = int(job_parameters['memory']),
3369                                                 timeout_hr = int(job_parameters['timeout_hr']),
3370                                                 keyword_line_arguments_input = [])
3371         self._tracker.update_entry(_id = molecule_id,
3372                                     keys = ['C0S0: BS%s - C1S0 geometry' % (basis_set)],
3373                                     values = [job_id])
3374         self.add_activity_log(molecule_id = molecule_id,
3375                             line = 'C0S0: BS%s - C1S0 geometry calculation started: %s' % (basis_set, str(datetime.
3376 now()).split('.')[0]))
3377         self.print_if_debug('%s C0S0: BS%s - C1S0 geometry started.' % (molecule_id, basis_set))
3378         return True

```

```

3349     [0],
3350     basis_set]).values[0],
3351
3352     self._tracker.update_entry(_id = molecule_id,
3353                               keys = ['COSO: BS%s - C1SO geometry' % (basis_set)],
3354                               values = [job_id])
3355     self.add_activity_log(molecule_id = molecule_id,
3356                           line = 'COSO: BS%s - C1SO geometry calculation started: %s' % (basis_set, str(datetime.
3357 now()).split('.')[0]))
3358     self.print_if_debug('%s COSO: BS%s - C1SO geometry started.' % (molecule_id, basis_set))
3359     return True
3360
3361 # C-1SO optimal geometry completed #####
3362 elif filtered_line.loc[:, 'C-1SO: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'COSO: BS
3363 %s - C-1SO geometry' % (basis_set)].values[0] == 'Not Started':
3364     # start C-1SO optimal geometry at COSO energy #####
3365     job_id = self.submit_molecule_for_marcus_calculation(molecule_id = molecule_id,
3366                                                       charge_geom = -1,
3367                                                       charge_eval = 0,
3368                                                       state_geom = 0,
3369                                                       state_eval = 0,
3370                                                       functional = filtered_line.loc[:, 'Functional'].values
3371     [0],
3372     basis_set]).values[0],
3373
3374
3375     self._tracker.update_entry(_id = molecule_id,
3376                               keys = ['COSO: BS%s - C-1SO geometry' % (basis_set)],
3377                               values = [job_id])
3378     self.add_activity_log(molecule_id = molecule_id,
3379                           line = 'COSO: BS%s - C-1SO geometry calculation started: %s' % (basis_set, str(datetime.
3380 now()).split('.')[0]))
3381     self.print_if_debug('%s COSO: BS%s - C-1SO geometry started.' % (molecule_id, basis_set))
3382     return True
3383 else:
3384     return False
3385
3386
3387 def next_step_for_molecule_id(self, molecule_id: str):
3388     filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
3389     if len(filtered_line) == 0:
3390         raise Exception('molecule_id not found in tracker.')
3391     job_submitted: bool = False
3392     # basis set 1
3393     if filtered_line.loc[:, 'COSO: BS1 - C-1SO geometry'].values[0] != 'Completed':
3394         job_submitted = self.next_step_for_molecule_id_with_basis_set(molecule_id = molecule_id,
3395                                                                       basis_set = 1)
3396     # basis set 2
3397     if job_submitted == False and (filtered_line.loc[:, 'COSO: BS2 - C-1SO geometry'].values[0] != 'Completed' or
3398                                    filtered_line.loc[:, 'COSO: BS2 - C1SO geometry'].values[0] != 'Completed' or filtered_line.loc[:, 'COSO: BS2 - COS1
3399 geometry'].values[0] != 'Completed') and filtered_line.loc[:, 'COSO: BS1'].values[0] == 'Completed':
3400         job_submitted = self.next_step_for_molecule_id_with_basis_set(molecule_id = molecule_id,
3401                                                                       basis_set = 2)
3402     # basis set 3
3403     if job_submitted == False and (filtered_line.loc[:, 'COSO: BS3 - C-1SO geometry'].values[0] != 'Completed' or
3404                                    filtered_line.loc[:, 'COSO: BS3 - C1SO geometry'].values[0] != 'Completed' or filtered_line.loc[:, 'COSO: BS3 - COS1
3405 geometry'].values[0] != 'Completed') and filtered_line.loc[:, 'COSO: BS2'].values[0] == 'Completed':
3406         job_submitted = self.next_step_for_molecule_id_with_basis_set(molecule_id = molecule_id,
3407                                                                       basis_set = 3)
3407     if job_submitted == True:
3408         return True
3409     else:
3410         return False
3411
3412 def check_qstat_and_transfer_files(self,
3413                                     stop_time = ''):
3414     if stop_time == '':
3415         stop_time = datetime.now() + timedelta(minutes = 15)
3416     else:
3417         stop_time = datetime.now() + timedelta(minutes = stop_time)
3418     if datetime.now() >= stop_time:
3419         return
3420     qstat = self.get_HPC_qstat()
3421     running_only = self.tracker_summarise_running()
3422     for i in range(0, len(running_only)):
3423         if datetime.now() >= stop_time:
3424             return
3425         if str(running_only.at[i, 'run_id']) not in qstat['Job ID'].values:
3426             molecule_id = running_only.at[i, 'molecule_id']

```

```

3424     filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
3425     charge_state, basis_set = running_only.at[i, 'run'].split(': ')
3426     charge = charge_state.split('S')[0].replace('C', '')
3427     excitation_state = charge_state.split('S')[1]
3428     if basis_set == 'BS1' or basis_set == 'BS2' or basis_set == 'BS3':
3429         if basis_set == 'BS1':
3430             basis_set = 'Basis Set 1'
3431         elif basis_set == 'BS2':
3432             basis_set = 'Basis Set 2'
3433         elif basis_set == 'BS3':
3434             basis_set = 'Basis Set 3'
3435         print(charge_state, basis_set, charge, excitation_state)
3436         self.transfer_chk_fchk_log_files_from_HPC(molecule_id = molecule_id,
3437                                         charge = charge,
3438                                         excitation_state = excitation_state,
3439                                         functional = filtered_line['Functional'].values[0],
3440                                         basisset = filtered_line[basis_set].values[0])
3441         convergence = self.gaussian_normal_termination(molecule_id = molecule_id,
3442                                         functional = filtered_line['Functional'].values[0],
3443                                         basis_set = format_basis_set(filtered_line['%s' % (
3444                                         basis_set)].values[0]),
3445                                         charge = charge,
3446                                         excitation_state = excitation_state)
3447         self._tracker.update_entry(_id = molecule_id,
3448                                     keys = ['%s' % (running_only.at[i, 'run'])],
3449                                     values = [convergence])
3450         self.add_activity_log(molecule_id = molecule_id,
3451                               line = '%s BS%s files transferred from HPC' % (charge_state, basis_set.split(' ')
3452                                         )[-1]))
3453     else:
3454         geom_charge_state = basis_set.split(' - ')[1].split(' ')[0]
3455         basis_set = basis_set.split(' - ')[0]
3456         if basis_set == 'BS1':
3457             basis_set = 'Basis Set 1'
3458         elif basis_set == 'BS2':
3459             basis_set = 'Basis Set 2'
3460         elif basis_set == 'BS3':
3461             basis_set = 'Basis Set 3'
3462         charge_eval = charge
3463         state_eval = excitation_state
3464         charge_geom = geom_charge_state.split('S')[0].replace('C', '')
3465         state_geom = geom_charge_state.split('S')[1]
3466         self.transfer_chk_fchk_log_marcus_files_from_HPC(molecule_id = molecule_id,
3467                                         charge_geom = charge_geom,
3468                                         charge_eval = charge_eval,
3469                                         state_geom = state_geom,
3470                                         state_eval = state_eval,
3471                                         functional = filtered_line['Functional'].values[0],
3472                                         basisset = filtered_line[basis_set].values[0])
3473         convergence = self.gaussian_marcus_normal_termination(molecule_id = molecule_id,
3474                                         functional = filtered_line['Functional'].values
3475                                         [0],
3476                                         basis_set = filtered_line[basis_set].values[0],
3477                                         charge_geom = charge_geom,
3478                                         charge_eval = charge_eval,
3479                                         state_geom = state_geom,
3480                                         state_eval = state_eval)
3481         self._tracker.update_entry(_id = molecule_id,
3482                                     keys = ['%s' % (running_only.at[i, 'run'])],
3483                                     values = [convergence])
3484         self.add_activity_log(molecule_id = molecule_id,
3485                               line = '%s BS%s at C%sS%s geometry files transferred from HPC' % (charge_state,
3486                                         charge_geom, state_geom, basis_set.split(' ')[:-1]))
3487     # self.get_tracker()
3488
3489 def automate_submissions(self,
3490                         stop_time = '',
3491                         max_jobs_parallel: int = 40):
3492     if stop_time == '':
3493         stop_time = datetime.now() + timedelta(minutes = 15)
3494     if datetime.now() >= stop_time:
3495         return
3496     current_running = self.count_running_optimisations_alt()
3497     to_be_run = self.get_tracker_not_started().reset_index()
3498     for index in range(0, len(to_be_run)):
3499         molecule_id = to_be_run.at[index, '_id']
3500         job_submitted = True
3501         while job_submitted == True and current_running <= max_jobs_parallel:
3502             job_submitted = self.next_step_for_molecule_id(molecule_id = molecule_id)
3503             if job_submitted == True:
3504                 current_running += 1
3505             if datetime.now() >= stop_time:
3506                 # self.download_tracker_as_excel()
3507                 # raise Exception('\nSubmission terminated automatically, stop_time acheived. Total running currently
3508 = %s.' % (self.count_running_optimisations_alt()))
3509                 return

```

```

3505     self.print_if_debug('\nTotal running currently = %s' % (current_running))
3506     self.download_tracker_as_excel()
3507
3508     def automate_submission_for_period_of_time(self,
3509                                                 total_time_minutes: int = 10,
3510                                                 pause_between_runs_minutes: int = 2,
3511                                                 max_jobs_parallel: int = 40,
3512                                                 jobs_reserved_for_restart: int = 3):
3513         time_end = datetime.now() + timedelta(minutes = total_time_minutes)
3514         run = True
3515         while datetime.now() <= time_end - timedelta(minutes = pause_between_runs_minutes) and run == True:
3516             with open(r'%s/Currently Running.txt' % (self._local_folder), 'wt') as file:
3517                 string = 'Started: %s\nEnding: %s' % (datetime.now(), time_end)
3518                 file.write(string)
3519                 file.close()
3520             if len(self.get_tracker_not_started()) == 0:
3521                 run = False
3522             self.check_qstat_and_transfer_files(stop_time = time_end)
3523             self.automate_submissions(stop_time = time_end,
3524                                       max_jobs_parallel = max_jobs_parallel - jobs_reserved_for_restart)
3525             #self.restart_failed_cases(stop_time = time_end,
3526             #                           max_jobs_parallel = max_jobs_parallel)
3527             self.update_results_all(stop_time = time_end)
3528             if datetime.now() <= time_end - timedelta(minutes = pause_between_runs_minutes):
3529                 pd.set_option('display.expand_frame_repr', False)
3530                 print(test_reorg_e.get_HPC_qstat())
3531                 pd.set_option('display.expand_frame_repr', True)
3532                 print('\nEnter sleep mode for %s minutes.' % (pause_between_runs_minutes))
3533                 print('Safe to terminate kernel. Program ending at %s.' % (time_end))
3534                 wait_time = pause_between_runs_minutes * 60
3535                 sleep_start = datetime.now()
3536                 for i in tqdm(range(0, int(wait_time/5)), position = 0, leave = True):
3537                     while datetime.now() < sleep_start + timedelta(seconds = i * 5):
3538                         time.sleep(5)
3539                 self.download_tracker_as_excel()
3540                 print('\nSubmission terminated automatically. Total running currently = %s.' % (self.
3541 count_running_optimisations_alt()))
3542
3543     def get_orbitals(self,
3544                      molecule_id: str,
3545                      filename: str):
3546         with open(r'%s/Data/%s/%s.log' % (self._local_folder, molecule_id, filename.replace('.log', '')).replace('fchk', '').
3547 .replace('chk', ''), 'r') as file:
3548             whole_file = file.read()
3549             lines = whole_file.split('\n')
3550             number_of_lines_in_whole_file = len(lines)
3551             population_analysis_start_line = 0
3552             population_analysis_end_line = 0
3553
3554             found_start = False
3555             current_line = number_of_lines_in_whole_file
3556             while found_start == False:
3557                 current_line -= 1
3558                 if 'Population analysis using the SCF Density.' in lines[current_line]:
3559                     found_start = True
3560                     population_analysis_start_line = current_line - 2
3561
3562             found_end = False
3563             current_line = population_analysis_start_line + 5
3564             while found_end == False:
3565                 current_line += 1
3566                 if 'Alpha' in lines[current_line]:
3567                     population_analysis_end_line = current_line
3568                     # print(lines[current_line])
3569                 else:
3570                     found_end = True
3571
3572             alpha_occ_eigenvalues = []
3573             alpha_virt_eigenvalues = []
3574
3575             parsed_lines = lines[population_analysis_start_line:population_analysis_end_line]
3576             parsed_lines_txt = '\n'.join(parsed_lines)
3577
3578             for line in parsed_lines:
3579                 if 'Alpha occ. eigenvalues -- ' in line:
3580                     line = line.replace('Alpha occ. eigenvalues -- ', '').split(' ')
3581                     for energy in line:
3582                         if energy != '':
3583                             alpha_occ_eigenvalues.append(float(energy))
3584
3585                 if 'Alpha virt. eigenvalues -- ' in line:
3586                     line = line.replace('Alpha virt. eigenvalues -- ', '').split(' ')
3587                     for energy in line:
3588                         if energy != '':
3589                             alpha_virt_eigenvalues.append(float(energy))
3590             alpha_occ_eigenvalues = list(np.flip(alpha_occ_eigenvalues))
3591             alpha_virt_eigenvalues = alpha_virt_eigenvalues

```

```

3589     HOMO_LUMO_gap = float(alpha_virt_eigenvalues[0]) - float(alpha_occ_eigenvalues[0])
3590
3591     return alpha_occ_eigenvalues, alpha_virt_eigenvalues, HOMO_LUMO_gap, parsed_lines_txt
3592
3593
3594     def get_SCF_energy_one(self,
3595         molecule_id: str,
3596         filename:str):
3597         with open(r"%s/Data/%s/%s.log" % (self._local_folder, molecule_id, filename.replace('.log', '').replace('fchk', '').
3598             replace('chk', '')), 'r') as file:
3599             whole_file = file.read()
3600             lines = whole_file.split('\n')
3601             lines_to_print = 0
3602             continue_printing_until_blank = False
3603             summary = ''
3604             for line in lines:
3605                 if 'SCF Done' in line:
3606                     # lines_to_print = 1
3607                     continue_printing_until_blank = True
3608                     final_SCF_energy = float(line.split('=')[1].split('A.U.')[0].replace(' ', ''))
3609                 if 'Cycle ' in line:
3610                     continue_printing_until_blank = True
3611                 if 'Item          Value      Threshold  Converged?' in line:
3612                     lines_to_print = 5
3613                 if lines_to_print > 0:
3614                     summary += line + '\n'
3615                     lines_to_print -= 1
3616                     if lines_to_print == 0:
3617                         summary += '\n'
3618                 elif continue_printing_until_blank:
3619                     if line == '':
3620                         continue_printing_until_blank = False
3621                     summary += '\n'
3622                 else:
3623                     summary += line + '\n'
3624             self.print_if_debug(summary)
3625
3626             warning = ''
3627             if 'Item          Value      Threshold  Converged?' in summary:
3628                 convergence = []
3629                 parsed_summary = summary.split('Item          Value      Threshold  Converged?\n')[-1]
3630                 for line in parsed_summary.split('\n'):
3631                     if line != '':
3632                         word = line.split('      ')[-1].replace(' ', '')
3633                         convergence.append(word)
3634                 self.print_if_debug(convergence)
3635                 if convergence == ['YES', 'YES', 'YES', 'YES']:
3636                     self.print_if_debug('convergence acheived')
3637                 else:
3638                     # raise Exception('\nConvergence not acheived:\n%s\n%s' % ('Item; Value; Threshold; Converged?', parsed_summary))
3639                     warning = '\n%s\nConvergence not acheived:\n%s\n%s' % (filename.replace('.log', '').replace('fchk', '').
3640             .replace('chk', ''), 'Item; Value; Threshold; Converged?', parsed_summary)
3641                     warnings.warn(warning)
3642                     summary = 'WARNING: CONVERGENCE NOT ACHEIVED\n' + summary
3643
3644             self.print_if_debug('Final SCF energy = %s' % (final_SCF_energy))
3645             file.close()
3646             return final_SCF_energy, summary, warning
3647
3648     def get_SCF_energy_excited_one(self,
3649         molecule_id: str,
3650         filename:str):
3651         with open(r"%s/Data/%s/%s.log" % (self._local_folder, molecule_id, filename.replace('.log', '').replace('fchk', '').
3652             replace('chk', '')), 'r') as file:
3653             whole_file = file.read()
3654             lines = whole_file.split('\n')
3655             lines_to_print = 0
3656             continue_printing_until_blank = False
3657             summary = ''
3658             for line in lines:
3659                 if 'Excited State  1' in line:
3660                     # lines_to_print = 1
3661                     continue_printing_until_blank = True
3662                 if 'Total Energy, E(TD-HF/TD-DFT)' in line:
3663                     final_SCF_energy = float(line.split('=')[1].replace(' ', ''))
3664                 if 'Cycle ' in line:
3665                     continue_printing_until_blank = True
3666                 if 'Item          Value      Threshold  Converged?' in line:
3667                     lines_to_print = 5
3668                 if lines_to_print > 0:
3669                     summary += line + '\n'
3670                     lines_to_print -= 1
3671                     if lines_to_print == 0:
3672                         summary += '\n'
3673                 elif continue_printing_until_blank:

```

```

3671     if line == '':
3672         continue_printing_until_blank = False
3673         summary += '\n'
3674     else:
3675         summary += line + '\n'
3676     self.print_if_debug(summary)
3677
3678     warning = ''
3679     if 'Item'           Value      Threshold  Converged? in summary:
3680         convergence = []
3681         parsed_summary = summary.split('Item'           Value      Threshold  Converged?\n)[-1]
3682         for line in parsed_summary.split('\n'):
3683             if line != '':
3684                 word = line.split('    ')[-1].replace(' ', '')
3685                 convergence.append(word)
3686             self.print_if_debug(convergence)
3687             if convergence == ['YES', 'YES', 'YES', 'YES']:
3688                 self.print_if_debug('convergence acheived')
3689             else:
3690                 # raise Exception('\nConvergence not acheived:\n%s\n%s' % ('Item; Value; Threshold; Converged?', parsed_summary))
3691                 warning = '\n%s\nConvergence not acheived:\n%s\n%s' % (filename.replace('.log', '').replace('fchk', ''),
3692 ).replace('chk', ''), 'Item; Value; Threshold; Converged?', parsed_summary)
3693             warnings.warn(warning)
3694             summary = 'WARNING: CONVERGENCE NOT ACHEIVED\n' + summary
3695
3696             self.print_if_debug('Final SCF energy = %s' % (final_SCF_energy))
3697             file.close()
3698         return final_SCF_energy, summary, warning
3699
3700 def get_SCF_energies(self,
3701     molecule_id:str,
3702     initial_charge_state: str,
3703     final_charge_state: str,
3704     formatted_functional_basisset: str):
3705     if final_charge_state != 'COS1':
3706         initial_SCF_at_initial_geom, summary1, warning1 = self.get_SCF_energy_one(molecule_id = molecule_id,
3707                                         filename = '%s_%s_%s' % (molecule_id,
3708 , formatted_functional_basisset, initial_charge_state))
3709         final_SCF_at_final_geom, summary2, warning2 = self.get_SCF_energy_one(molecule_id = molecule_id,
3710                                         filename = '%s_%s_%s' % (molecule_id,
3711 formatted_functional_basisset, final_charge_state))
3712         initial_SCF_at_final_geom, summary3, warning3 = self.get_SCF_energy_one(molecule_id = molecule_id,
3713                                         filename = '%s_%s_%s_opt_geom_at_%s' %
3714 (molecule_id, formatted_functional_basisset, final_charge_state, initial_charge_state))
3715         final_SCF_at_initial_geom, summary4, warning4 = self.get_SCF_energy_one(molecule_id = molecule_id,
3716                                         filename = '%s_%s_%s_opt_geom_at_%s' %
3717 (molecule_id, formatted_functional_basisset, initial_charge_state, final_charge_state))
3718     else:
3719         initial_SCF_at_initial_geom, summary1, warning1 = self.get_SCF_energy_one(molecule_id = molecule_id,
3720                                         filename = '%s_%s_%s' % (molecule_id,
3721 , formatted_functional_basisset, initial_charge_state))
3722         final_SCF_at_final_geom, summary2, warning2 = self.get_SCF_energy_excited_one(molecule_id = molecule_id,
3723                                         filename = '%s_%s_%s' % (
3724 molecule_id, formatted_functional_basisset, final_charge_state))
3725         initial_SCF_at_final_geom, summary3, warning3 = self.get_SCF_energy_one(molecule_id = molecule_id,
3726                                         filename = '%s_%s_%s_opt_geom_at_%s' %
3727 (molecule_id, formatted_functional_basisset, final_charge_state, initial_charge_state))
3728         final_SCF_at_initial_geom, summary4, warning4 = self.get_SCF_energy_excited_one(molecule_id = molecule_id,
3729                                         filename = '%s_%s_%s_opt_geom_at_%s' %
3730 (molecule_id, formatted_functional_basisset, initial_charge_state, final_charge_state))
3731     pass
3732
3733     summary = ('initial_SCF_at_initial_geom\n' +
3734         '\n' +
3735         summary1 + '\n' +
3736         'final_SCF_at_final_geom\n' +
3737         '\n' +
3738         summary2 + '\n' +
3739         'initial_SCF_at_final_geom\n' +
3740         '\n' +
3741         summary3 + '\n' +
3742         'final_SCF_at_initial_geom\n' +
3743         '\n' +
3744         summary4 + '\n')
3745     warning = ''
3746     for warn in [warning1, warning2, warning3, warning4]:
3747         if warn != '':
3748             warning += warn + '\n'
3749
3750     return initial_SCF_at_initial_geom, initial_SCF_at_final_geom, final_SCF_at_initial_geom, final_SCF_at_final_geom,
3751     summary, warning
3752
3753 def do_DUSHIN_calculation(self,

```

```

3746     molecule_id: str,
3747     filename_initial_state: str,
3748     filename_final_state: str):
3749         self._HPC_connection.copy_fchk_log_files_for_dushin(remote_dushin_folder = r'%s/Logs_for_dushin/' % (self.
3750         _remote_folder),
3751                                     remote_log_folder = self._remote_folder,
3752                                     filename_initial_state = filename_initial_state,
3753                                     filename_final_state = filename_final_state)
3754         initial_state = filename_initial_state.split('_')[-1]
3755         final_state = filename_final_state.split('_')[-1]
3756         if filename_initial_state.replace('_%s' % (initial_state), '') == filename_final_state.replace('_%s' % (
3757             final_state), ''):
3758             mol_id_functional_basisset = filename_initial_state.replace('_%s' % (initial_state), '')
3759         else:
3760             raise Exception('\nIssue with mol_id_functional_basisset.')
3761         self._HPC_connection.submit_job_dushin(remote_dushin_folder = r'%s/Logs_for_dushin/' % (self._remote_folder),
3762                                             remote_bin_path = r'%s/Logs_for_dushin/' % (self._remote_folder),
3763                                             initial_state = initial_state,
3764                                             final_state = final_state,
3765                                             mol_id_functional_basisset = mol_id_functional_basisset)
3766         self._HPC_connection.transfer_from_HPC(local_folder = r'%s/Data/%s/%s_%s/' % (self._local_folder, molecule_id,
3767         initial_state, final_state),
3768                                     remote_folder = r'%s/Logs_for_dushin/' % (self._remote_folder),
3769                                     filename = 'dushin%s_%s_%s.log' % (mol_id_functional_basisset,
3770         initial_state, final_state))
3771         with open(r'%s/Data/%s/%s_%s/dushin%s_%s_%s.log' % (self._local_folder, molecule_id, initial_state, final_state,
3772         mol_id_functional_basisset, initial_state, final_state), 'r') as file:
3773             dushin_output = file.readlines()
3774             split_index_top = dushin_output.index(" Displacement: in terms of nc of 1 THEN of nc of 2\n") + 2
3775             dushin_output = dushin_output[split_index_top::]
3776             split_index_bot = dushin_output.index("\n")
3777             dushin_output = dushin_output[0:split_index_bot]
3778             if (dushin_output[-1] != "\n"):
3779                 dushin_output = dushin_output[:-1]
3780             dushin_results = pd.DataFrame(columns = ['Freq', 'Q/cm', 'lam/eV'])
3781             for line in dushin_output:
3782                 # print(line)
3783                 freq1 = float(line.split('freq')[1].split('Q')[0].replace(' ', '')) * 100
3784                 freq2 = float(line.split('freq')[2].split('Q')[0].replace(' ', '')) * 100
3785                 Q1 = float(line.split('Q=')[1].split('lam')[0].replace(' ', '')) * constants.physical_constants['
3786                 inverse meter-electron volt relationship'][0]
3787                 Q2 = float(line.split('Q=')[2].split('lam')[0].replace(' ', '')) * constants.physical_constants['
3788                 inverse meter-electron volt relationship'][0]
3789                 lam1 = float(line.split('lam=')[1].split('freq')[0].replace(' ', '')) * constants.physical_constants['
3790                 inverse meter-electron volt relationship'][0] * 100
3791                 lam2 = float(line.split('lam=')[2].split('\n')[0].replace(' ', '')) * constants.physical_constants['
3792                 inverse meter-electron volt relationship'][0] * 100
3793                 if freq1 != 0 or Q1 != 0 or lam1 != 0:
3794                     dushin_results.loc[len(dushin_results)] = [freq1, Q1, lam1]
3795                 if freq2 != 0 or Q2 != 0 or lam2 != 0:
3796                     dushin_results.loc[len(dushin_results)] = [freq2, Q2, lam2]
3797             dushin_summarised = dushin_results.sort_values(['Freq', axis = 'index').reset_index(drop = True)
3798             dushin_summarised.to_excel(r'%s/Data/%s/%s_%s_%s.xlsx' % (self._local_folder, molecule_id,
3799             initial_state, final_state, mol_id_functional_basisset, initial_state, final_state))
3800             self.print_if_debug(dushin_summarised)
3801             imaginary_frequencies = dushin_results[['Freq']].lt(0).any(axis = 'index').values[0]
3802             return dushin_summarised, imaginary_frequencies
3803
3804     def get_atoms_summary(self,
3805                           molecule_id: str,
3806                           filename: str):
3807         summary = pd.DataFrame(columns = ['Atomic Mass', 'Atomic Number', 'Coordinates', 'Charge', 'Number of Bonds', '
3808         Hybridisation'])
3809         mol = next(pybel.readfile("log", r'%s/Data/%s/%s.log' % (self._local_folder, molecule_id, filename.replace('.log',
3810         '').replace('fchk', '').replace('chk', ''))))
3811         for entry in mol.atoms:
3812             self.print_if_debug([entry.atomicmass, entry.atomicnum, entry.coords, entry.formalcharge, entry.degree, entry
3813             .hyb])
3814             summary.loc[len(summary)] = [entry.atomicmass, entry.atomicnum, entry.coords, entry.formalcharge, entry.
3815             degree, entry.hyb]
3816             del mol
3817         return summary
3818
3819     def organise_results_one(self,
3820                           molecule_id: str,
3821                           # new_entry: bool = False,
3822                           download_results: bool = False,
3823                           tracker = '',
3824                           results = '',
3825                           basis_set_list: list = [1, 2, 3],
3826                           transition_list: list = ['COSO_COS1', 'COSO_C1SO', 'COSO_C-1SO'],
3827                           overwrite_: bool = False):
3828         if type(tracker) == str:
3829             tracker = self.get_entries_from_molecule_id(molecule_id = molecule_id)
3830         else:
3831             tracker = tracker.loc[tracker['_id'] == molecule_id]
3832         if type(results) == str:
3833             results = self.get_results()

```

```

3820
3821     tracker = tracker.loc[tracker['_id'] == molecule_id]
3822     for basis_set in basis_set_list:
3823         for transitions in transition_list:
3824             keys = []
3825             values = []
3826             initial = transitions.split('_')[0]
3827             final = transitions.split('_')[1]
3828             sub_directory = r'%s/Data/%s/%s/' % (self._local_folder, molecule_id, transitions)
3829             check_directory(r'%s' % (sub_directory))
3830
3831             if overwrite_ == False:
3832                 if results.loc[results['_id']] == '%s_%s_%s_%s' % (molecule_id, tracker.loc[tracker['_id']] ==
3833                     molecule_id, 'Functional').values[0], format_basis_set(tracker.loc[tracker['_id']] ==
3834                     molecule_id, 'Basis Set %s' % (basis_set)).values[0]), transitions), 'Last Updated'].values[0] == None:
3835                     if (tracker['%s: BS%' % (initial, basis_set)].values[0] == 'Completed') and (tracker['%s: BS%' %
3836                     (final, basis_set)].values[0] == 'Completed') and (tracker['%s: BS% - %s geometry' % (final, basis_set, initial)].values[0] ==
3837                     'Completed') and (tracker['%s: BS% - %s geometry' % (initial, basis_set, final)].values[0] == 'Completed'):
3838                         run = True
3839                     else:
3840                         run = False
3841
3842                 elif tracker.loc[tracker['_id']] == molecule_id, 'Last Updated'].values[0] > results.loc[results['_id']]
3843                     == '%s_%s_%s_%s' % (molecule_id, tracker.loc[tracker['_id']] == molecule_id, 'Functional').values[0], format_basis_set(
3844                     tracker.loc[tracker['_id']] == molecule_id, 'Basis Set %s' % (basis_set)).values[0]), transitions), 'Last Updated'].values[0]:
3845                     if (tracker['%s: BS%' % (initial, basis_set)].values[0] == 'Completed') and (tracker['%s: BS%' %
3846                     (final, basis_set)].values[0] == 'Completed') and (tracker['%s: BS% - %s geometry' % (final, basis_set, initial)].values[0] ==
3847                     'Completed') and (tracker['%s: BS% - %s geometry' % (initial, basis_set, final)].values[0] == 'Completed'):
3848                         run = True
3849                     else:
3850                         run = False
3851
3852                     if (tracker['%s: BS%' % (initial, basis_set)].values[0] == 'Completed') and (tracker['%s: BS%' %
3853                     (final, basis_set)].values[0] == 'Completed') and (tracker['%s: BS% - %s geometry' % (final, basis_set, initial)].values[0] ==
3854                     'Completed') and (tracker['%s: BS% - %s geometry' % (initial, basis_set, final)].values[0] == 'Completed'):
3855                         run = True
3856                     else:
3857                         run = False
3858
3859             if run == True:
3860                 print(r'%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values[0], format_basis_set(tracker['Basis
3861                     Set %s' % (basis_set)].values[0]), final))
3862                 # Orbitals and HOMO-LUMO handling
3863                 alpha_occ_eigenvalues, alpha_virt_eigenvalues, HOMO_LUMO_gap, parsed_lines_txt = self.get_orbitals(
3864                     molecule_id, filename = r'%s_%s_%s_COSO' % (molecule_id, tracker['Functional'].values[0],
3865                     format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0])))
3866                 with open(r'%s/GS_Orbitals.txt' % (sub_directory), 'wt') as file:
3867                     file.write(parsed_lines_txt)
3868                     file.close()
3869                 keys.append('Alpha Occ Orbitals List')
3870                 values.append(alpha_occ_eigenvalues)
3871                 keys.append('Alpha Virt Orbitals List')
3872                 values.append(alpha_virt_eigenvalues)
3873                 keys.append('HOMO-LUMO Gap')
3874                 values.append(HOMO_LUMO_gap * constants.physical_constants['Hartree energy in eV'][0])
3875
3876                 # SCF energies
3877                 initial_SCF_at_initial_geom, initial_SCF_at_final_geom, final_SCF_at_initial_geom,
3878                 final_SCF_at_final_geom, summary, warning = self.get_SCF_energies(molecule_id = molecule_id,
3879
3880                     initial_charge_state = initial,
3881
3882                     final_charge_state = final,
3883
3884                     formatted_functional_basisset = '%s_%s' % (tracker['Functional'].values[0],
3885                     format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0])))
3886                     with open(r'%s/%s_%s_%s_SCF_Summary.txt' % (sub_directory, molecule_id, tracker['Functional'].values[0],
3887                     format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0])), 'wt') as file:
3888                         file.write(summary)
3889                         file.close()
3890                 keys.append('Initial CS at Initial Optimised Geometry')
3891                 values.append(initial_SCF_at_initial_geom)
3892                 keys.append('Initial CS at Final Optimised Geometry')
3893                 values.append(initial_SCF_at_final_geom)
3894                 keys.append('Final CS at Initial Optimised Geometry')
3895                 values.append(final_SCF_at_initial_geom)
3896                 keys.append('Final CS at Final Optimised Geometry')
3897                 values.append(final_SCF_at_final_geom)
3898                 keys.append('Parsed SCF Summary')
3899                 values.append(summary)
3900

```

```

3882     four_point = initial_SCF_at_final_geom - initial_SCF_at_initial_geom + final_SCF_at_initial_geom -
3883     final_SCF_at_final_geom
3884     keys.append('4-Point Reorganisation Energy')
3885     values.append(four_point * constants.physical_constants['Hartree energy in eV'][0])
3886     if warning != '':
3887         self.comment_result(_id = '%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values[0]),
3888                             format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), initial, final,
3889                             user = 'Auto-Generated',
3890                             string = warning)
3891
3892     # DUSHIN
3893     filename_initial_state = r'%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values[0],
3894     format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), initial)
3895     filename_final_state = r'%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values[0],
3896     format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), final)
3897     dushin_summarised, imaginary_frequencies = self.do_DUSHIN_calculation(molecule_id = molecule_id,
3898                                                                           filename_initial_state =
3899                                                                           filename_final_state,
3900                                                                           filename_final_state)
3901     keys.append('Vibrational Frequencies')
3902     values.append(list(dushin_summarised['Freq'].values))
3903     keys.append('Displacements')
3904     values.append(list(dushin_summarised['Q/cm'].values))
3905     keys.append('Reorganisation Energies')
3906     values.append(list(dushin_summarised['lam/eV'].values))
3907     keys.append('DUSHIN Reorganisation Energy')
3908     values.append(dushin_summarised['lam/eV'].sum())
3909
3910     # atoms summary
3911     initial_atoms_summary = self.get_atoms_summary(molecule_id = molecule_id,
3912                                                     filename = r'%s_%s_%s_%s' % (molecule_id, tracker['
3913 Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), initial))
3914     final_atoms_summary = self.get_atoms_summary(molecule_id = molecule_id,
3915                                                     filename = r'%s_%s_%s_%s' % (molecule_id, tracker['
3916 Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), final))
3917     keys.append('Initial Atoms Summary')
3918     values.append(initial_atoms_summary.to_string())
3919     keys.append('Final Atoms Summary')
3920     values.append(final_atoms_summary.to_string())
3921
3922     # simple energy analysis
3923     difference = (four_point * constants.physical_constants['Hartree energy in eV'][0]) - (
3924     dushin_summarised['lam/eV'].sum())
3925     frac_diff = difference/(four_point * constants.physical_constants['Hartree energy in eV'][0])
3926     keys.append('Difference (4-point & DUSHIN)')
3927     values.append(difference)
3928     keys.append('Difference Percentage (4-point & DUSHIN)')
3929     values.append('%.3f' % (frac_diff * 100))
3930
3931     self._results.update_entry(molecule_id = '%s_%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values
3932 [0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), initial, final),
3933                                         keys = keys,
3934                                         values = values)
3935
3936 def update_results_all(self,
3937                         stop_time: str = ''):
3938     if stop_time == '':
3939         stop_time = datetime.now() + timedelta(minutes = 15)
3940     if datetime.now() >= stop_time:
3941         return
3942     tracker = self.get_tracker()
3943     results = self.get_results()
3944     for molecule_id in tracker['_id'].values:
3945         functional = tracker.loc[tracker['_id'] == molecule_id, 'Functional'].values[0]
3946         if results.loc[results['_id'] == '%s_%s_%s_COSO_COS1' % (molecule_id, functional, format_basis_set(tracker.loc
3947 [tracker['_id'] == molecule_id, 'Basis Set %s' % (1)].values[0])), 'Last Updated'].values[0] == None:
3948             self.organise_results_one(molecule_id = molecule_id,
3949                                     # new_entry = False,
3950                                     download_results = False,
3951                                     tracker = tracker,
3952                                     results = results,
3953                                     basis_set_list = [1, 2, 3])
3954         else:
3955             if tracker.loc[tracker['_id'] == molecule_id, 'Last Updated'].values[0] > results.loc[results['_id'] == '%
3956 s_%s_%s_COSO_COS1' % (molecule_id, functional, format_basis_set(tracker.loc[tracker['_id'] == molecule_id, 'Basis Set
3957 %s' % (1)].values[0])), 'Last Updated'].values[0]:
3958                 self.organise_results_one(molecule_id = molecule_id,
3959                                         # new_entry = False,
3960                                         download_results = False,
3961                                         tracker = tracker,
3962                                         results = results,
3963                                         basis_set_list = [1, 2, 3])
3964
3965     self.download_results_as_excel()

```