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

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]

$$k = A \times \exp\left[\frac{-\Delta G^\star}{k_B T}\right], \tag{1}$$

where $\Delta G^\star$ could be expanded as

$$\Delta G^\star = \frac{\lambda}{4}\left[1 + \frac{\Delta G^0}{\lambda}\right]^2. \tag{2}$$

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 solvational and internal component of reorganisation energy denoted by $\lambda = \lambda_{\text{sol}} + \lambda_{\text{int}}$ respectively[1]. For 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:

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

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 $2k_j^r k_j^p / k_j^r + k_j^p$, 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.

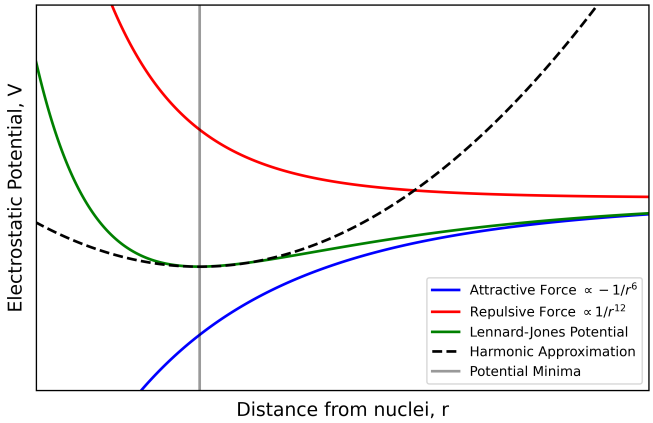

Fig. 1: Illustration of the harmonic approximation used in the derivation of the Marcus rate, valid when the displacement of the individual atoms about the points of equilibria are small where energy is at a minimum.

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

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:

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

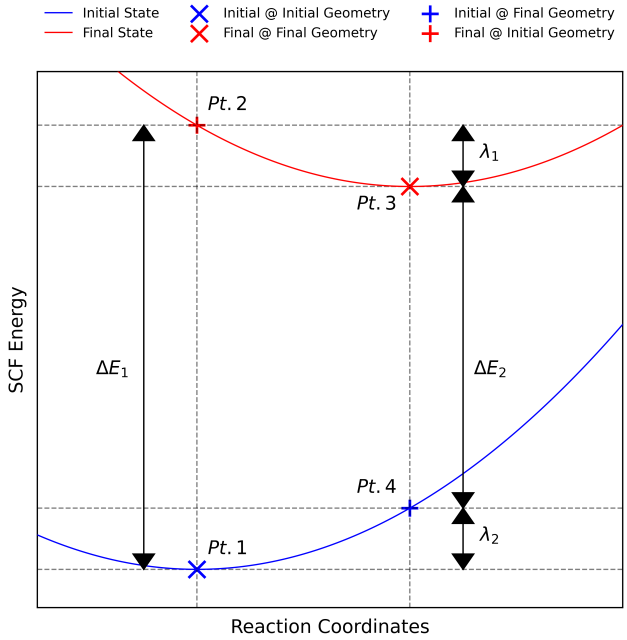

Fig. 2: Illustration of Nelsen's 4-Point method. The two lines 'initial state' and 'final state' would be indicative of the potential energy surface of the molecule at each state. When the molecule is given energy ΔE_1, it moves from Pt 1 to Pt 2. However since Pt 2 would not be at the molecule's optimised 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 database[2]. This class containers 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.

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 examp-le 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 night[3].

```
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()
```

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

[3]The 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, simillar 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:

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

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 use[4] [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 processed 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 molecule[5].

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 frequecies 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 ran 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 SCF energies were extracted and then the final reorganisation energy[6] 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

$$\lambda_{\text{int}} = \frac{h}{2}\sum_j v_j \delta_j^2, \tag{6}$$

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

[4] The 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.

[5] Gaussian16 could terminate at a local minima.

[6] Intuitively 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.

[7] There 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.

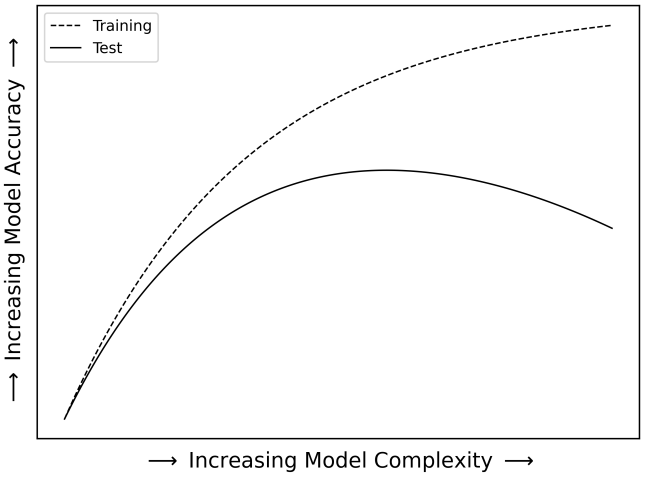

Fig. 3: Illustration of the relationship between the performance of a machine learning model and the complexity of the model. As complexity increases, the model tends to perform better at predicting outcomes. Past a certain point however, further increasing the complexity would lead to over-fitting which 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 energies[10].

[8] Gaussian16 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.

[9] Benzene, napthalene, anthracene, tetracene, pentacene, hexacene *et cetera*.

[10] The 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 the use of CAM-B3LYP over B3LYP seems to contribute to an increase in the calculated reorganisation energies for the transition to the charges 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.

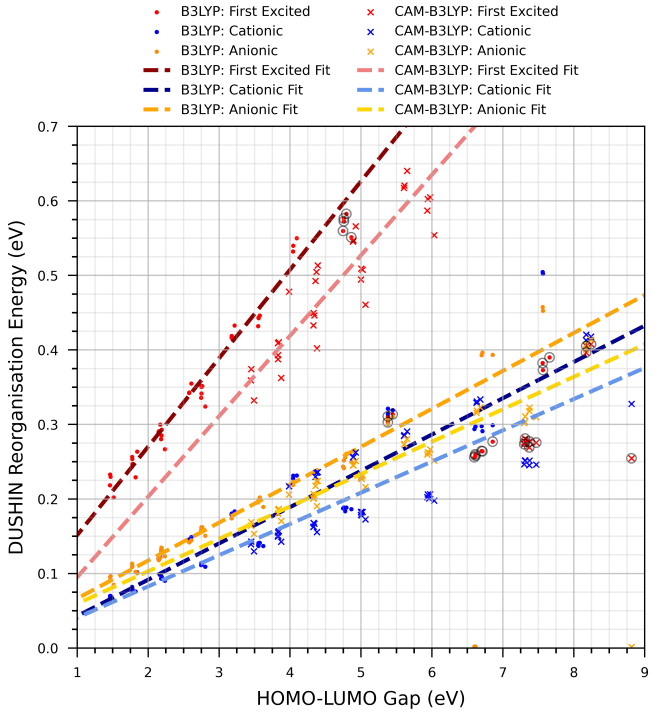

Fig. 4: The dashed lines are the best fit lines through each groups of molecules. The fit was perform with the exclusion of benzene and napthalene, which could be seen circled in the figure. The use of different functionals seems to cause a parallel shift in the computed values.

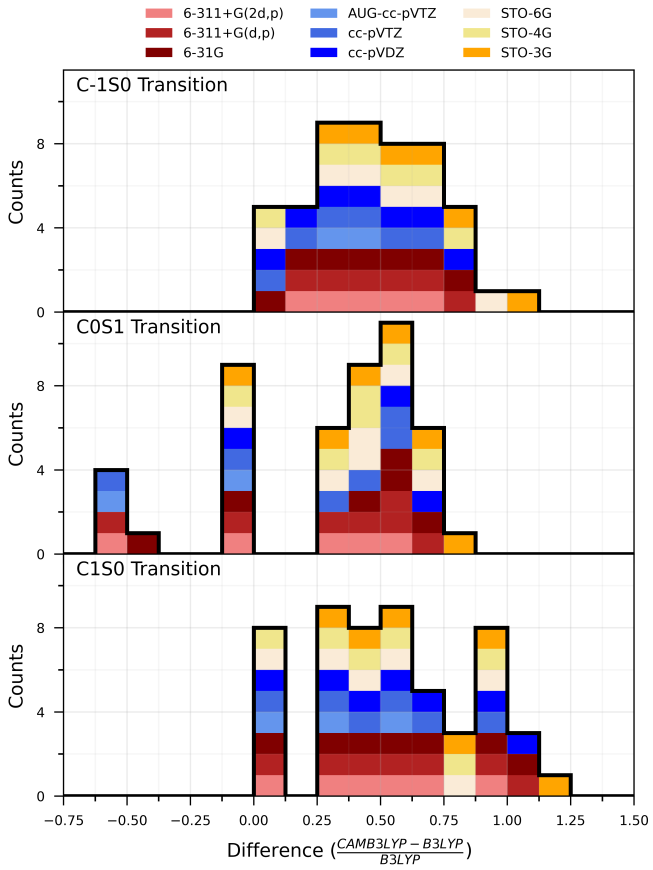

Fig. 5: As seen by the shape of the distribution for the transitions to the charged states (where the mean of the distribution is confidently above 0), the use of CAM-B3LYP would yield higher reorganisation energies compared to the use of the B3LYP functional. The analysis was also color-coded according 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

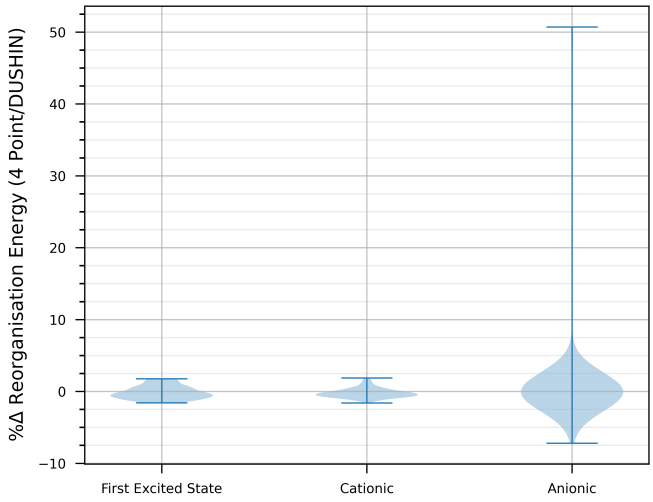

Fig. 6: Illustration of the fractional difference between the DUSHIN reorganisation energies and the 4-point reorganisation energies. Most results agree to within 10% of each other.

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

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

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 into[11].

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.

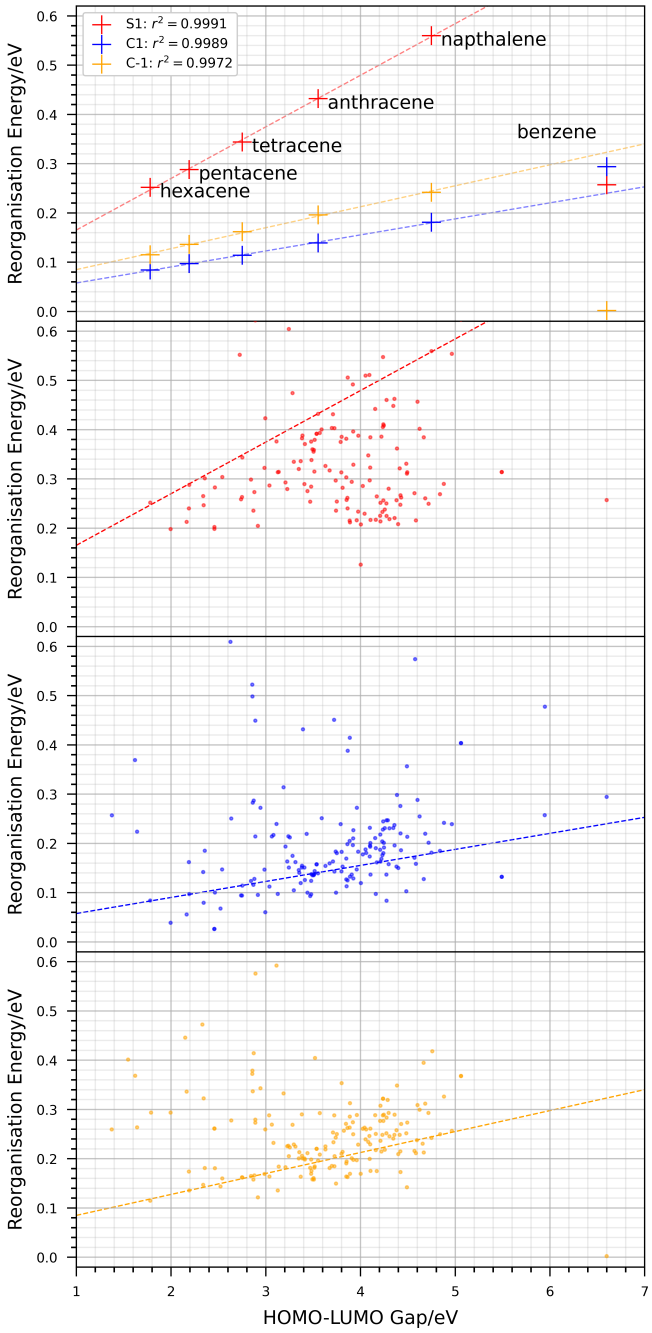

Fig. 7: Summary of the results in Appendix VIII. **Red:** Transitions to first excited-state, **blue:** transitions to cationic states, and **yellow:** transitions to anionic states. The top pane only shows the values for the linear oligoacenes with their best-fit lines along with the r^s-values. The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

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 molecules[12]. 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 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

[11] In 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.

[12] A 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.

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 available[13]. 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 performed[14]. 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.

[13] Due 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.

[14] The 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-Evrenk, 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. 054 106–054 106, 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. Piela, *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. Raghavachari, 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=vbQlDQAAQBAJ

[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. Francl, 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/Er2sjJjYjHFCl9KRP_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=C2C=C1	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=CC3=CC2=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=C2C=CC4=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=C2CC3=CC4=CC=CC=C4C=C31	4.666480627
C1CCC2=CC3=C(CCCC3)C=C2C1	5.681465334
C1C2=CC=CC=C2CC3=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=CC2=C3C4=C1C=CC5=C4C6=C(C=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=CS5	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=CS2)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)C=CC4=C3C5=C(C=C4)SC=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=CC4=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=CC3=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=CC3=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=CC2=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=C2CC3=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=C2CC3=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(=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=C3C=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=CC6=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=CC4=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(CCC4)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(C4=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=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=CC2=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(C2=CC=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=CS5	b3lyp/6-311+G(d,p)	First excited	0.2588624	2.73773757
242	C1=CC2=C(C=CC3=C2C=CC4=C3SC=C4)C5=C1C=CS5	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=C5S4	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=C5S4	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(C5=C3SC=C5)SC=C4	b3lyp/6-311+G(d,p)	First excited	0.3795392	3.515983217
248	C1=CC=C(C(=C1)C#CC2=CC=CS2)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=C3C=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=CS5	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=CS5	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=CS5	b3lyp/6-311+G(d,p)	First excited	0.2540636	3.467819063
265	C1=CC2=C(C=CC3=C2C=CC4=C3C=CS4)C5=C1C=CS5	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=CS4)C5=C(C=C3)C=CS5	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=C5	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=CS5	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=CS5	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=CS5	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=CS5	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=CS4)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=CSC=C54	b3lyp/6-311+G(d,p)	First excited	0.2969552	3.737211787
283	C1=CC2=C(C3=C(C=C2)SC=C3)C4=C1C=CC5=C4C=CS5	b3lyp/6-311+G(d,p)	First excited	0.2566552	4.239533977
284	C1=CC2=C3C=CC=C4C3=C(C=CS4)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=CS3	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=CC4=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=CS4)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=CS5	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=CS2)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=CC5=C4C=CS5)C=C31	b3lyp/6-311+G(d,p)	First excited	0.3933156	3.569861762
298	C1=CC2=CSC=C2C3=CC4=C(C=CC5=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=CC4=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)C(=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\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=C4C=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(/C1=C/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=C4	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=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=C6C=CC=CC6=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)C=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=CC(=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=C4C=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=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=CC4=C5C3=C2C6=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=CC5=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=CC7=CC=CC=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(CCC4)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=C2)C	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=CC3=CC=CC4=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=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=CC1=CC4=C(C3=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=CC3=C2C=CC4=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=C5S4	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=C5S4	b3lyp/6-311+G(d,p)	Cationic	0.136958	4.272731868
311	C1=CC=C2C=C3C(=CC2=C1)C4=C(C5=C3SC=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=CS3	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=CC4=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=C3C=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=CC4=C5C3=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=C1C=C5C(=C4)C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.20522	3.8846975
323	C1=CC=C2C=C3C(=CC2=C1)C4=C(S3)C5=CC=CC=C5S4	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=CC3=C2C4=C(S3)C5=CC=CC=C5S4	b3lyp/6-311+G(d,p)	Cationic	0.2102916	3.921432872
326	C1=CC2=C(C=CC3=C2C=CS3)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#C)C3=CC=CS3	b3lyp/6-311+G(d,p)	Cationic	0.3881448	3.864288961
328	C1=CC=C2C(=C1)C3=C(S2)C=C4C=C5C(=CC4=C3)C=CS5	b3lyp/6-311+G(d,p)	Cationic	0.093186	3.467819063
329	C1=CC2=C(C=CC3=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#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=CS5	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(=CC2=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=CC4=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=CC4=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=CC(=C43)C=C2	b3lyp/6-311+G(d,p)	Cationic	0.7743924	3.888234981
469	C1=CC=C2C3=C4C(=CC=C3)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=CS2	b3lyp/6-311+G(d,p)	Cationic	0.214458	3.062913636
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=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=CS3	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=CC4=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(=C\2/C=CC3=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=C4C=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=C3)/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(=CSS4)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
576	C1=CSC2=CC3=C(C=C21)SC=C3	b3lyp/6-311+G(d,p)	Cationic	0.1581	4.296405774
577	C1=CC2=C3C(=C1)SSC3=CC=C2	b3lyp/6-311+G(d,p)	Cationic	0.1405292	3.351898558
578	C1=CSC2=CC3=C(C=CS3)C=C21	b3lyp/6-311+G(d,p)	Cationic	0.1030936	4.482531656
579	C1=CC2=CSC=C2C3=CSC=C31	b3lyp/6-311+G(d,p)	Cationic	0.2985796	4.38348221
580	C1=CC=C2C(=C1)C3=C(S2)SC=C3	b3lyp/6-311+G(d,p)	Cationic	0.2392952	4.963084737
581	C1=CC2=C(C3=C1C=CS3)SC=C2	b3lyp/6-311+G(d,p)	Cationic	0.2017356	4.718182261
582	C1=CC=C2C(=C1)C3=C(S2)C=CS3	b3lyp/6-311+G(d,p)	Cationic	0.2550184	4.623758751
583	C1=CC2=C(C=CS2)C3=C1C=CS3	b3lyp/6-311+G(d,p)	Cationic	0.2419488	4.879817895
584	C1=CC2=C(C=CS2)C3=C1SC=C3	b3lyp/6-311+G(d,p)	Cationic	0.2885108	4.601717528
585	C1=CC(=S)C=C2C1=CC(=S)C=C2	b3lyp/6-311+G(d,p)	Cationic	0.03906	1.993506156
586	C1=CC=C2C(=C1)C=CC(=S)C2=S	b3lyp/6-311+G(d,p)	Cationic	0.9801828	2.148338944
587	C1=CC=C2C(=S)C=CC(=S)C2=C1	b3lyp/6-311+G(d,p)	Cationic	0.0795584	2.342900356
588	C=C1C2=C(C3=C1C=CS3)SC=C2	b3lyp/6-311+G(d,p)	Cationic	42.8313112	3.518976469
589	C1=CSC2=CC3=CSC=C3C=C21	b3lyp/6-311+G(d,p)	Cationic	0.1049908	3.294482533
590	C1=CC2=C(C=CS2)C3=CSC=C31	b3lyp/6-311+G(d,p)	Cationic	0.1967136	3.918983847
591	C1=CC2=C3C=CC=C3SSC2=C1	b3lyp/6-311+G(d,p)	Cationic	0.6642184	2.329566777
592	C1=CC=C2C(=C1)C=CC3=C2SS3	b3lyp/6-311+G(d,p)	Cationic	0.2196412	3.431083692
615	C1=CSC2=C1SC=C2	b3lyp/6-311+G(d,p)	Cationic	0.4038928	5.059957272
616	C1=CSC2=C1C=CS2	b3lyp/6-311+G(d,p)	Cationic	0.132494	5.489625061
617	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=C2C=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)C=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=CC4=CC=CC=C14	b3lyp/6-311+G(d,p)	Anionic	0.2187608	3.629454697
189	C1CCC2=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=CC2=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=CC6=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=CC4=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=CC4=C5C3=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=CC4=CC5=CC=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=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)C=CC4=CC=CC(=C43)C=C2	b3lyp/6-311+G(d,p)	Anionic	0.213962	3.810138302
238	C1=CC=C2C=C3C=C4C=CC=CC4=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(C3=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=CS5	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=C5S4	b3lyp/6-311+G(d,p)	Anionic	0.2516456	4.273003982
364	C1=CC=C2C=C3C(=CC2=C1)C4=C(C5=C3SC=C5)SC=C4	b3lyp/6-311+G(d,p)	Anionic	0.1844252	3.515983217
365	C1=CC=C(C(=C1)C#CC2=CC=CS2)C#CC3=CC=CS3	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=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(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(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(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(C3=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=CS3	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(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=CC4=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=CC4=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(=C\2/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\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=CC3=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(=C/2\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(=C\2/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=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

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  4 16:27:52 2024

@author: kyebchoo
"""
#%%
# Importing modules

# NUMPY
import numpy as np
from numpy import random

# SCIPY
import scipy as sci
import scipy.constants as constants
import scipy.stats as stats

# PANDAS
import pandas as pd
pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# MATPLOTLIB-PYPLOT
import matplotlib.pyplot as plt
from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)

# SYSTEM CONTROLS
import os
import importlib
from copy import deepcopy
import paramiko

# RDKIT(CHEMISTRY)
import rdkit
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import Draw
from rdkit.Chem import Descriptors
from rdkit.Chem import PandasTools
from rdkit.Chem.Draw import IPythonConsole
from rdkit.Chem import rdDepictor
from rdkit.Chem.Draw import rdMolDraw2D

# import kora.install.rdkit

# DATE AND TIME CONTROLS
from datetime import datetime, timedelta
import time

# OTHERS
from pymongo import MongoClient
import urllib.parse
import pprint
import ast
import cclib
from pathlib import Path
from ase import Atoms
import warnings
from openbabel import pybel
from tqdm import tqdm
#%%

def timestamp() -> str:
    """
    Returns the standardised formatted timestamp for use in later methods, in the form (YYYYMMDDHHMMSS)

    Returns
    -------
    str
        Formatted timestamp.

    """
    return str(datetime.now()).split('.')[0].replace('-', '').replace(' ', '').replace(':', '')

def tag(molecule: str) -> str:
    """
    Returns the standardised formmated tag for use in later methods, in the form of {truncated molecule name}_{
     YYYYMMDDHHMMSS}.
    The code would automatically remove symbols and truncate the name to 8 letters.

    Parameters
    ----------
    molecule : str
        Name of molecule (preferrably IUPAC).
```

```
    Returns
    -------
    str
        Standardised tag.

    """
    molecule = molecule.replace('[', '').replace(']', '').replace('-', '').replace(',', '').replace('.', '').replace('(', '').replace(')', '').replace('_', '')[0:8]
    tag_name = '%s_%s' % (molecule, timestamp())
    return tag_name

def is_notebook() -> bool:
    """
    Method to test if the script is being ran in a Jupyter notebook environment.

    Returns
    -------
    bool
        TRUE if Jupyter notebook detected, and FALSE otherwise.

    """
    try:
        shell = get_ipython().__class__.__name__
        if shell == 'ZMQInteractiveShell':
            return True   # Jupyter notebook or qtconsole
        elif shell == 'TerminalInteractiveShell':
            return False  # Terminal running IPython
        else:
            return False  # Other type (?)
    except NameError:
        return False      # Probably standard Python interpreter

def display_if_notebook(display_item, otherwise: bool = False):
    """
    Display or print out the display_item if script is being ran in a Jupyter notebook environment.

    Parameters
    ----------
    display_item : TYPE
        Any item, whether it be string or table, to be displayed if in a Jupyter notebook environment.
    otherwise : bool, optional
        Whether to display or print out the display_item if not in a Jupyter notebook environment. The default is False.

    Returns
    -------
    None.

    """
    if is_notebook():
        try:
            display(display_item) # ignore error if running in Spyder
        except:
            pass
    else:
        if otherwise:
            display(display_item) # ignore error if running in Spyder

def check_directory(directory: str):
    """
    Method to check if a directory exists, and to create the directory if it does not exist.

    Parameters
    ----------
    directory : str
        Path to directory.

    Returns
    -------
    None.

    """
    if not os.path.exists(directory):
        os.makedirs(directory)
        pass

def convert_atomic_number_to_symbol(atomic_number: int) -> str:
    """
    Convert atomic number to atomic symbols representing the elements.

    Parameters
    ----------
    atomic_number : int
        Atomic number to be converted.

    Returns
```

```
    -------
    str
        Symbol of the corresponding element.

    """
    return Chem.GetPeriodicTable().GetElementSymbol(atomic_number)

def format_basis_set(basis_set: str):
    """
    Method to format the basis set, removing invalid characters when being used as part of a filename.

    Parameters
    ----------
    basis_set : str
        Basis set input.

    Returns
    -------
    TYPE
        Formatted basis set string.

    """
    return basis_set.replace('+', 'p').replace('*', 's').replace('(', '').replace(')', '').replace(',', '')

#%%
class Molecule():

    def __init__(self,
                 molecule_name: str,
                 molecule_id: str,
                 SMILES: str,
                 parent_folder: str = os.getcwd(),
                 debug: bool = False):
        """
        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()'.

        The path structure of the system would be:
            {parent_folder}/Data/{molecule_id}/{molecule_documents}

        Parameters
        ----------
        molecule_name : str
            Name of the molecule, preferrably IUPAC name.
        molecule_id : str
            A unique ID for this specific molecule. File and document naming would be dependent on this ID and thus it must be unique.
        SMILES : str
            SMILES string for the molecule.
        parent_folder : str, optional
            The main folder which you would be working in. All subfolders and files would be generated under the parent folder.
            The default is os.getcwd().
        debug : bool, optional
            Toggle to print additional lines during operation for debugging. The default is False.

        Returns
        -------
        None.

        """
        self._molecule_name = molecule_name
        self._molecule_id = molecule_id
        self._SMILES = SMILES
        self._parent_folder = parent_folder
        self._debug = debug

        check_directory("%s/Data/%s/" % (self._parent_folder, self._molecule_id))

    @classmethod
    def load_molecule(cls,
                      molecule_id: str,
                      parent_folder: str = os.getcwd(),
                      debug: bool = False):
        """
        Alternative initialisation method where the key information is extracted from a previously generated documentation.
        Only the molecule ID and the parent folder has to be declared, additional entries would be searched within the parent folder.

        Parameters
        ----------
        molecule_id : str
            Molecule ID specific to the molecule. This should be previously generated when the molecule was first initialised.
            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 genrated 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', 'Neighbors[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.rdchem.HybridizationType.values[atom.GetHybridization()]).split('.')[-1]
            self.print_if_debug('Atom with index: %s\n%s\n%s\nNeighbours:' % (index, atomic_number, element))
            neighbors = []
            for neighbor in atom.GetNeighbors():
                neighbor_index = neighbor.GetIdx()
```

```
                neighbor_atomic_number = neighbor.GetAtomicNum()
                neighbor_symbol = neighbor.GetSymbol()
                neighbor_bond_type = str(molecule.GetBondBetweenAtoms(atom.GetIdx(), neighbor.GetIdx()).GetBondType())
                if allow_aromatic_bond == True:
                    neighbor_bond_order = str(molecule.GetBondBetweenAtoms(atom.GetIdx(), neighbor.GetIdx()).
    GetBondTypeAsDouble())
                else:
                    neighbor_bond_order = str(kekulized_molecule.GetBondBetweenAtoms(atom.GetIdx(), neighbor.GetIdx()).
    GetBondTypeAsDouble())
                neighbors.append([neighbor_index, neighbor_atomic_number, neighbor_symbol, neighbor_bond_type,
    neighbor_bond_order])
                self.print_if_debug(' %s\n %s\n %s\n %s\n %s' % (str(neighbor_index), str(neighbor_atomic_number),
    neighbor_symbol, str(neighbor_bond_type), str(neighbor_bond_order)))
            dataframe.loc[len(dataframe)] = [index, atomic_number, element, hybridisation, neighbors]
        if self._debug:
            display_if_notebook(dataframe, otherwise = True)
        dataframe.to_excel(r'%s/Data/%s/%s_documentation.xlsx' % (self._parent_folder, self._molecule_id, self.
    _molecule_id))

        with open("%s/Data/%s/%s_documentation.txt" % (self._parent_folder, self._molecule_id, self._molecule_id), "w") as
     documentation:
            documentation.write("%s\n%s\n%s\n%s\n\nCreated on %s" % (self._molecule_name, self._molecule_id, self._SMILES,
     self._parent_folder, str(datetime.now())))
        self.print_if_debug('Molecule documentation created sucessfully.')

    def get_SMILES(self) -> str:
        """
        Returns the SMILES string of the molecule.

        Returns
        -------
        str
            SMILES string of the molecule.

        """
        self.print_if_debug('\nIn function Molecule.get_SMILES():')
        return self._SMILES

    def get_molecule_name(self) -> str:
        """
        Returns the name of the molecule.

        Returns
        -------
        str
            Name of the molecule.

        """
        self.print_if_debug('\nIn function Molecule.get_molecule_name():')
        return self._molecule_name

    def get_molecule_id(self) -> str:
        """
        Returns the ID of the molecule.

        Returns
        -------
        str
            ID of the molecule.

        """
        self.print_if_debug('\nIn function Molecule.get_molecule_id():')
        return self._molecule_id

    def generate_xyz_from_rdkit_mol(self,
                                    random_low: float = 0.00,
                                    random_high: float = 0.00) -> list[str, int]:
        """
        Generates the cartesian x,y,z coordinates using built-in RDKit functions.
        The molecule is optimised using Merck molecular force field methods.

        The returned xyz coordinate block DOES NOT terminate with two blank lines.

        Returns
        -------
        list[str, int]
            List with the [0] first entry being the xyz coordinates, and
            the [1] second entry being the total number of atoms in the molecule.

        """
        self.print_if_debug('\nIn function Molecule.generate_xyz_from_rdkit_mol():')
        Mol = Chem.AddHs(Chem.MolFromSmiles(self._SMILES))
        AllChem.EmbedMolecule(Mol)
        AllChem.MMFFOptimizeMolecule(Mol, maxIters = 1000)

        for c in Mol.GetConformers():
```

```
            # only single conformer is chosen currently, might alter in future
            positions = c.GetPositions()
        atoms_symbols = np.array([at.GetSymbol() for at in Mol.GetAtoms()])
        atoms = Atoms(atoms_symbols, positions=positions)
        num_atoms: int = len(atoms)
        types = atoms.get_chemical_symbols()
        all_atoms = zip(types, atoms.get_positions())
        a_str = ""
        for atom in all_atoms:
            if random_low == 0.00 and random_high == 0.00:
                a_str += atom[0] + " " + " ".join([str(x) for x in atom[1]]) + "\n"
            elif random_low != random_high and random_low < random_high:
                a_str += atom[0] + " " + " ".join([str(x + random.uniform(low = random_low, high = random_high)) for x in atom[1]]) + "\n"
            else:
                raise Exception('Invalid random_low and/or random_high input.')
        # a_str = a_str + "\n\n"
        self.print_if_debug(a_str)
        return a_str, num_atoms

    def generate_xyz_from_pymol_mol(self, pbmol) -> list[str, int]:
        """
        Generates the cartesian x,y,z coordinates from a pymol object.

        Parameters
        ----------
        pbmol : TYPE
            pymol object.

        Returns
        -------
        list[str, int]
            List with the [0] first entry being the xyz coordinates, and
            the [1] second entry being the total number of atoms in the molecule.

        """
        self.print_if_debug('\nIn function Molecule.generate_xyz_from_pymol_mol():')
        warnings.warn('This method has not been tested. If the xyz block generated has a double-blank line termination, it should be truncated as it would affect future processes.')
        # read mol and outputs xyz with a small optimisation
        pbmol.write("xyz", r"%s/Data/%s/%s_pybel_buffer.xyz" % (self._parent_folder, self._molecule_id, self._molecule_id), overwrite = True)
        with open(r"%s/Data/%s/%s_pybel_buffer.xyz" % (self._parent_folder, self._molecule_id, self._molecule_id), "r") as file:
            split: list = file.read().split("\n\n")
            pybabelxyz: str = split[1]
            num_atoms: int = int(split[0])
        self.print_if_debug(pybabelxyz)
        return pybabelxyz, num_atoms

    def generate_xyz_from_xyz_file(self, filename: str) -> list[str, int]:
        """
        Generates the cartesian x,y,z coordinates from a .xyz file. The filename should contain the filename with the .xyz suffix.
        The file should also be located in the molecule file under the folder {parent_folder}/Data/{molecule_id}/.

        Parameters
        ----------
        filename : str, optional
            Filename of the xyz file with the suffix (.xyz).

        Returns
        -------
        list[str, int]
            List with the [0] first entry being the xyz coordinates, and
            the [1] second entry being the total number of atoms in the molecule.

        """
        self.print_if_debug('\nIn function Molecule.generate_xyz_from_xyz_file():')
        warnings.warn('This method has not been tested. If the xyz block generated has a double-blank line termination, it should be truncated as it would affect future processes.')
        if filename == '':
            filename = r"%s/Data/%s/%s" % (self._parent_folder, self._molecule_id, self._molecule_id)
        with open(filename, "r") as file:
            lines: str = file.readlines()[2:]
            mol_xyz: str = "".join(lines)
            num_atoms: int = len(lines)
        # mol_xyz: str = mol_xyz + "\n\n"
        mol_xyz: str = mol_xyz
        self.print_if_debug(mol_xyz)
        return mol_xyz, num_atoms

    def generate_xyz_from_log_file(self,
                                   filename: str,
                                   random_low: float = 0.0000,
                                   random_high: float = 0.0001) -> list[str, int]:
```

```
        """
        Generates the cartesian x,y,z coordinates from a Gaussian16 .log output file. The filename should contain the filename with the .log suffix.
        The file should also be located in the molecule file under the folder {parent_folder}/Data/{molecule_id}/.

        Parameters
        ----------
        filename : str
            Filename of the log file with the suffix (.log).

        Returns
        -------
        list[str, int]
            List with the [0] first entry being the xyz coordinates, and
            the [1] second entry being the total number of atoms in the molecule.

        """
        self.print_if_debug('\nIn function Molecule.generate_xyz_from_log_file():')
        # generate XYZ string from calculation output
        # gaussian_data = cclib.io.ccread(r'%s/Data/%s/%s' % (self._parent_folder, self._molecule_id, filename))
        gaussian_data = cclib.io.ccread(r'%s' % (filename))
        # generate xyz file from gaussian log output
        num_atoms: int = gaussian_data.natom
        coords = gaussian_data.atomcoords
        coords = coords[-1, :, :]
        atomnos = gaussian_data.atomnos
        xyz_str_gauss = ""
        for i in range(0, len(coords)):
            xyz_str_gauss += (str(convert_atomic_number_to_symbol(int(atomnos[i])))
                              + " "
                              + " ".join([str(float(x) + random.uniform(low = random_low, high = random_high)) for x in coords[i]])
                              + "\n")
        self.print_if_debug("There are %i atoms and %i MOs" % (gaussian_data.natom, gaussian_data.nmo))
        self.print_if_debug("Obtained opt geometry from optimised log file for: \n%s" % (filename))
        self.print_if_debug(xyz_str_gauss)
        return xyz_str_gauss, num_atoms

    def generate_molecule_xyz(self,
                              input_object = '',
                              name_extension: str = '',
                              save: bool = True,
                              random_low: float = 0.0000,
                              random_high: float = 0.0001) -> str:
        """
        Generates molecule xyz by detecting the input_object. The input_object can either be:
            1) an rdkit mol class
            2) a pymol class
            3) an xyz file (located within the molecule file) with the structure already in it
            4) a Gaussian16 output .log file with previous calculation of the molecule

        If in the form of a filename, the file must be located within the folder:
            {parent_folder}/Data/{molecule_id}/

        The {name_extension} is the suffix to append at the end of the xyz filename: {molecule_id}_{name_extension}.xyz.

        Parameters
        ----------
        input_object : TYPE, optional
            Input object of the form of either (1) an RDKit molecule, (2) pymol molecule, (3) xyz file, or (4) .log Gaussian16 output file.
            The default is '', where the geometry from the SMILES-generated RDKit molecule would be used.
        name_extension : str, optional
            Optional extension to be added to the end of the .xyz file. The default is ''.
        save : bool, optional
            Toggle to save the xyz geometry as a .xyz file. The default is True.

        Returns
        -------
        str
            Molecule xyz block.

        """
        self.print_if_debug('\nIn function Molecule.generate_molecule_xyz():')
        # 1) an rdkit mol class
        if (type(input_object) is Chem.rdchem.Mol) or (input_object == ''):
            a_str_gauss, num_atoms = self.generate_xyz_from_rdkit_mol(random_low = random_low,
                                                                      random_high = random_high)
            self.print_if_debug('RDKit molecule geometry used.')
        # 2) a pymol class
        try:
            from openbabel import pybel as pb
            if type(input_object) is pb.Molecule:
                a_str_gauss, num_atoms = self.generate_xyz_from_pymol_mol(pbmol = input_object)
                self.print_if_debug('Pymol molecule geometry used.')
        except ImportError:
```

```
            print("openbabel not found")
            pass

        if type(input_object) is str:
            if Path(input_object).is_file():
        # 3) path to an xyz file with the structure already in it
                if Path(input_object).suffix == ".xyz":
                    a_str_gauss, num_atoms = self.generate_xyz_from_xyz_file(filename = input_object)
                    self.print_if_debug('Geometry extracted from %s.' % input_object)
        # 4) path to a file with previous calculation of the molecule
                elif Path(input_object).suffix == ".log":
                    a_str_gauss, num_atoms = self.generate_xyz_from_log_file(filename = input_object,
                                                                             random_low = random_low,
                                                                             random_high = random_high)
                    self.print_if_debug('Geometry extracted from %s.' % input_object)
                else:
                    raise Exception('Input in generate_molecule_xyz not recognised.\n%s' % (input_object))
        if save:
            if name_extension != '':
                name_extension = '_%s' % (name_extension)
            with open(r"%s/Data/%s/%s%s.xyz" % (self._parent_folder, self._molecule_id, self._molecule_id, name_extension)
    , "wt") as textfile:
                a_str = "%s\n" % (str(num_atoms)) + "%s\n" % (self._molecule_name) + a_str_gauss
                textfile.write(a_str)
                self.print_if_debug('xyz geometry saved as; %s_%s.xyz.' % (self._molecule_id, name_extension))
        return a_str_gauss

    def keyword_line_generator(self,
                               functional_basisset: str,
                               calculation_type: str = 'OptFreq',
                               state_TD: int = 0,
                               restart: bool = False,
                               higher_max_cycles: int = 0,
                               set_trust_radius_bohr: float = 0,
                               explicitly_specify_connectivity: bool = True,
                               integral: str = 'SuperFine',
                               overwrite_Freq: list[list, list] = [[], []],
                               overwrite_Opt: list[list, list] = [[], []],
                               overwrite_NoSymm: list[list, list] = [[], []],
                               overwrite_state_TD: list[list, list] = [[], []],
                               overwrite_Geom: list[list, list] = [[], []],
                               overwrite_Integral: list[list, list] = [[], []]
                               ) -> str:
        """
        Method to write the keyword line for the Gaussian16 script. Supports by default the methods:
            (1) OptFreq:
                Optimise molecule followed by a frequency calculation.
            (2) Freq:
                Frequenmcy calculation only.
            (3) Manual:
                Manually input and specify keywords. For example to perform an optimisation job...
                ...with increased MaxCycles of 200 and reduced MaxStep of 0.10Bohr:
                    overwrite_Opt = [[True, 'Opt'], ['MaxCycles=200', MaxStep=10]]
                ...as a restart job:
                    overwrite_Opt = [[True, 'Opt'], ['Restart']]
        Overwrite lines would overwrite any inputs generated beforehand.

        Parameters
        ----------
        functional_basisset : str
            Functional and basisset in the form of <functional>/<basis_set>, e.g. B3LYP/6-31G(d,p).
        calculation_type : str, optional
            Calculation choice of 'OptFreq', 'Opt', or 'Manual'. The default is 'OptFreq'.
        state_TD : int, optional
            Excitation state of the molecule. The default is 0.
        restart : bool, optional
            Whether to include the Restart keyword under the Opt() line. The default is False.
        higher_max_cycles : int, optional
            Manually input an integer number of max cycles. The default is 0, where the Gaussian16 default would be used.
        set_trust_radius_bohr : float, optional
            Manually set the maximum trust radius (step size) in units of Bohr for the optimisation.
            The default is 0, where the Gaussian16 default would be used.
        explicitly_specify_connectivity : bool, optional
            Explicitly specify bonds and connectivity. The default is True.
        overwrite_Freq : list[list, list], optional
            Overwrite line for Freq keyword. The default is [[], []] for no overwrite.
        overwrite_Opt : list[list, list], optional
            Overwrite line for Opt keyword. The default is [[], []] for no overwrite.
        overwrite_NoSymm : list[list, list], optional
            Overwrite line for NoSymm keyword. The default is [[], []] for no overwrite.
        overwrite_state_TD : list[list, list], optional
            Overwrite line for TD keyword. The default is [[], []] for no overwrite.
        overwrite_Geom : list[list, list], optional
            Overwrite line for Geom keyword. The default is [[], []] for no overwrite.

        Raises
```

```
        ------
        ValueError
            Calculation type not recognised, check 'calculation_type' input.

        Returns
        -------
        str
            Keyword line to be passed into the method/function generating the Gaussian runscripts..

        """
        self.print_if_debug('\nIn function Molecule.keyword_line_generator():')
        # keyword_line: list[bool, list[]] = [bool, arguments]
        opt_line = [[False, 'Opt'], []]
        freq_line = [[False, 'Freq'], []]
        nosymm_line = [[False, 'NoSymm'], []]
        TD_line = [[False, 'TD'], []]
        Geom_line = [[False, 'Geom'], []]
        integral_line = [[False, 'Integral'], []]

        keyword_line = '#P'

        if calculation_type.lower() == 'optfreq' or calculation_type.lower() == 'freqopt':
            opt_line[0][0] = True
            freq_line[0][0] = True
            nosymm_line[0][0] = True
        elif calculation_type.lower() == 'opt':
            opt_line[0][0] = True
            nosymm_line[0][0] = True
        elif 'marcus':
            nosymm_line[0][0] = True
            Geom_line = [[True, 'Geom'], ['check']]
            explicitly_specify_connectivity = False
            restart = False
            higher_max_cycles = 0
            set_trust_radius_bohr = 0
            # state_TD = 0
        elif calculation_type.lower() == 'manual':
            print('Using manual input to generate keyword line for Gaussian16.')
        else:
            raise ValueError('\nCalculation type not recognised: %s' % (calculation_type))

        if higher_max_cycles != 0:
            opt_line[1].append('MaxCycles=%s' % (higher_max_cycles))

        if set_trust_radius_bohr != 0:
            set_trust_radius_bohr = round(set_trust_radius_bohr, 2)
            max_step = int(set_trust_radius_bohr/0.01)
            opt_line[1].append('MaxStep=%s' % (max_step))

        if state_TD != 0:
            TD_line[0][0] = True
            TD_line[1] = ['NStates=10', 'Root=%s' % (int(state_TD))]

        if restart == True:
            opt_line[1].append('Restart')
            Geom_line[0][0] = True
            Geom_line[1].append('check')

        if explicitly_specify_connectivity == True:
            Geom_line = [[True, 'Geom'], ['Connectivity']]

        if integral.lower() == 'superfine':
            integral_line = [[True, 'Integral'], ['SuperFine']]
        elif integral.lower() == 'ultrafine':
            integral_line = [[True, 'Integral'], ['UltraFine']]
        else:
            integral_line = [[True, 'Integral'], ['UltraFine']]
            warnings.warn('Intergral not recognised, switching to default = UltraFine')

        if overwrite_Freq != [[], []]:
            freq_line = overwrite_Freq
            warnings.warn('Overwriting Freq line.')
        if overwrite_Opt != [[], []]:
            opt_line = overwrite_Opt
            warnings.warn('Overwriting Opt line.')
        if overwrite_NoSymm != [[], []]:
            nosymm_line = overwrite_NoSymm
            warnings.warn('Overwriting NoSymm line.')
        if overwrite_state_TD != [[], []]:
            TD_line = overwrite_state_TD
            warnings.warn('Overwriting TD line.')
        if overwrite_Geom != [[], []]:
            Geom_line = overwrite_Geom
            warnings.warn('Overwriting Geom line.')
        if overwrite_Integral != [[], []]:
            integral_line = overwrite_Integral
```

```
            warnings.warn('Overwriting Geom line.')

        self.print_if_debug('freq_line: %s\nopt_line: %s\nnosymm_line: %s\nTD_line: %s\nGeom_line: %s' % (freq_line,
    opt_line, nosymm_line, TD_line, Geom_line))

        def keyword_line_argument_writer(keyword: list[[bool, str], list]) -> str:
            """
            Method to format the keyword array blocks into proper lines.

            Parameters
            ----------
            keyword : list[[bool, str], list]
                Keyword array block.

            Raises
            ------
            Exception
                Keyword_line_argument_writer failed to write arguments, check input list.

            Returns
            -------
            str
                Keyword line.

            """
            self.print_if_debug('\nIn function keyword_line_argument_writer():')
            if keyword[0][0] == True:
                keyword_line_argument = '%s' % (keyword[0][1])

                arguments: str = ''
                for i in range(0, len(keyword[1])):
                    if arguments == '':
                        arguments = '%s' % (keyword[1][i])
                    else:
                        arguments = arguments + ',' + '%s' % (keyword[1][i])
                if arguments != '':
                    keyword_line_argument = keyword_line_argument + '(' + arguments + ')'
                return keyword_line_argument
            elif keyword[0][0] == False:
                return ''
            else:
                raise Exception('\nKeyword_line_argument_writer failed to write arguments for %s' % (keyword))

        for keyword in [opt_line, freq_line, nosymm_line, TD_line, Geom_line, integral_line]:
            if keyword_line_argument_writer(keyword = keyword) != '':
                keyword_line = keyword_line + ' ' + keyword_line_argument_writer(keyword = keyword)
        keyword_line = keyword_line + ' ' + functional_basisset
        self.print_if_debug('Keyword line: %s' % (keyword_line))
        return keyword_line

    def get_connectivity(self) -> str:
        """
        Returns the connectivity of the bonds in a molecule as a block to be passed into the Gaussian16 runscript.

        Returns
        -------
        write_text : str
            Connectivity text block.

        """
        self.print_if_debug('\nIn function Molecule.get_connectivity():')
        dataframe = pd.read_excel(r'%s/Data/%s/%s_documentation.xlsx' % (self._parent_folder, self._molecule_id, self.
    _molecule_id))
        if self._debug:
            display_if_notebook(dataframe, otherwise = True)
        write_text: str = ''
        for index in range(0, len(dataframe)):
            parent_index = dataframe.at[index, 'Index'] + 1
            line = '%s' % (parent_index)
            neighbors = dataframe.at[index, 'Neighbors[Index, Atomic Number, Element, Bond, Bond Order]']
            neighbors = ast.literal_eval(neighbors)
            for i in range(0, len(neighbors)):
                neighbor_index = neighbors[i][0] + 1
                bond_type = neighbors[i][4]
                line = line + ' ' + str(neighbor_index) + ' ' + str(bond_type)
            # print(line)
            if write_text == '':
                write_text = line
            else:
                write_text = write_text + '\n' + line
        write_text = write_text
        self.print_if_debug(write_text)
        with open(r'%s/Data/%s/%s_connectivity.txt' % (self._parent_folder, self._molecule_id, self._molecule_id), 'wt')
    as textfile:
            textfile.write(write_text)
            textfile.close()
```

```
        return write_text

    def generate_gjf_file(self,
                          input_object,
                          excited_state: int,
                          charge: int,
                          functional: str,
                          basisset: str,
                          number_of_proc: int,
                          memory: int,
                          calculation_type: str = 'OptFreq',
                          connectivity: bool = True,
                          keyword_line_arguments_input: list[[str, str]] = []) -> str:
        """
        Method to write the Gaussian runscript.

        Parameters
        ----------
        input_object : TYPE
            Input object to be fed into Molecule.generate_molecule_xyz().
        excited_state : int
            Excitation state.
        charge : int
            Charge of molecule.
        functional : str
            Functional to be used in DFT calculation.
        basisset : str
            Basis set to be used in DFT calculation.
        number_of_proc : int
            Number of processors.
        memory : int
            Total amount of memory to be divided between the processors.
        calculation_type : str, optional
            Calculation type. The default is 'OptFreq'.
        connectivity : bool, optional
            Whether to use molecule connectivity. The default is True.
        keyword_line_arguments_input : list[[str, str]], optional
            Overwite lines. The default is [].

        Returns
        -------
        gjf_file : str
            Gaussian16 job script.

        """
        self.print_if_debug('\nIn function Molecule.generate_gjf_file():')
        functional_basisset = '%s/%s' % (functional, basisset)
        keyword_line_arguments: dict = {'functional_basisset': functional_basisset,
                                        'calculation_type': calculation_type,
                                        'state_TD': excited_state,
                                        'restart': False,
                                        'higher_max_cycles': int(1024),
                                        'set_trust_radius_bohr': float(0.30),
                                        'explicitly_specify_connectivity': connectivity,
                                        'overwrite_Freq': [[], []],
                                        'overwrite_Opt': [[], []],
                                        'overwrite_NoSymm': [[], []],
                                        'overwrite_state_TD': [[], []]}

        # keyword_line_arguements_input = [['calculation_type', 'Opt'], ['restart', True], ['higher_max_cycles', 1024]]
        for i in range(0, len(keyword_line_arguments_input)):
            keyword_line_arguments['%s' % (keyword_line_arguments_input[i][0])] = keyword_line_arguments_input[i][1]

        keyword_line = self.keyword_line_generator(functional_basisset = keyword_line_arguments['functional_basisset'],
                                                   calculation_type = keyword_line_arguments['calculation_type'],
                                                   state_TD = keyword_line_arguments['state_TD'],
                                                   restart = keyword_line_arguments['restart'],
                                                   higher_max_cycles = keyword_line_arguments['higher_max_cycles'],
                                                   set_trust_radius_bohr = keyword_line_arguments['set_trust_radius_bohr'
    ],
                                                   explicitly_specify_connectivity = keyword_line_arguments['
    explicitly_specify_connectivity'],
                                                   overwrite_Freq = keyword_line_arguments['overwrite_Freq'],
                                                   overwrite_Opt = keyword_line_arguments['overwrite_Opt'],
                                                   overwrite_NoSymm = keyword_line_arguments['overwrite_NoSymm'],
                                                   overwrite_state_TD = keyword_line_arguments['overwrite_state_TD'])

        name_extension = '%s_%s_C%sS%s' % (functional, format_basis_set(basisset), charge, excited_state)
        Mol_xyz = self.generate_molecule_xyz(input_object = input_object,
                                             name_extension = name_extension,
                                             save = True)

        gjf_file = "%s_%s" % (self._molecule_id, name_extension)

        title_line = '%s %s' % (gjf_file, keyword_line_arguments['calculation_type'])

```

```
        a_str_gaussian = (f"%nprocshared={number_of_proc} \n" +
                          "\n" +
                          f"%mem={memory}GB \n" +
                          f"%chk={gjf_file}.chk \n" +
                          keyword_line + "\n" +
                          "\n" +
                          title_line + "\n"
                          "\n" +
                          "%s %s\n" % (charge, abs(charge) + 1) +
                          Mol_xyz + '\n')

        if connectivity == True:
            a_str_gaussian = (a_str_gaussian +
                              self.get_connectivity() + '\n' +
                              '\n')

        self.print_if_debug('\nGaussian file:\n==============\n%s\n==============' % (a_str_gaussian))
        with open('%s/Data/%s/%s.gjf' % (self._parent_folder, self._molecule_id, gjf_file), mode = "wt") as writefile:
            writefile.write(a_str_gaussian)

        return gjf_file

    def generate_gjf_restart_file(self,
                                  excited_state: int,
                                  charge: int,
                                  functional: str,
                                  basisset: str,
                                  number_of_proc: int,
                                  memory: int,
                                  calculation_type: str = 'OptFreq',
                                  keyword_line_arguments_input: list[[str, str]] = []) -> str:
        self.print_if_debug('\nIn function Molecule.generate_gjf_file():')
        functional_basisset = '%s/%s' % (functional, basisset)
        keyword_line_arguments: dict = {'functional_basisset': functional_basisset,
                                        'calculation_type': calculation_type,
                                        'state_TD': excited_state,
                                        'restart': True,
                                        'higher_max_cycles': int(1024),
                                        'set_trust_radius_bohr': float(0.30),
                                        'explicitly_specify_connectivity': False,
                                        'overwrite_Freq': [[], []],
                                        'overwrite_Opt': [[], []],
                                        'overwrite_NoSymm': [[], []],
                                        'overwrite_state_TD': [[], []]}

        # keyword_line_arguements_input = [['calculation_type', 'Opt'], ['restart', True], ['higher_max_cycles', 1024]]
        for i in range(0, len(keyword_line_arguments_input)):
            keyword_line_arguments['%s' % (keyword_line_arguments_input[i][0])] = keyword_line_arguments_input[i][1]

        keyword_line = self.keyword_line_generator(functional_basisset = keyword_line_arguments['functional_basisset'],
                                                   calculation_type = keyword_line_arguments['calculation_type'],
                                                   state_TD = keyword_line_arguments['state_TD'],
                                                   restart = keyword_line_arguments['restart'],
                                                   higher_max_cycles = keyword_line_arguments['higher_max_cycles'],
                                                   set_trust_radius_bohr = keyword_line_arguments['set_trust_radius_bohr'
    ],
                                                   explicitly_specify_connectivity = keyword_line_arguments['
    explicitly_specify_connectivity'],
                                                   overwrite_Freq = keyword_line_arguments['overwrite_Freq'],
                                                   overwrite_Opt = keyword_line_arguments['overwrite_Opt'],
                                                   overwrite_NoSymm = keyword_line_arguments['overwrite_NoSymm'],
                                                   overwrite_state_TD = keyword_line_arguments['overwrite_state_TD'])

        name_extension = '%s_%s_C%sS%s' % (functional, format_basis_set(basisset), charge, excited_state)

        gjf_file = "%s_%s" % (self._molecule_id, name_extension)

        title_line = '%s %s' % (gjf_file, keyword_line_arguments['calculation_type'])

        a_str_gaussian = (f"%nprocshared={number_of_proc} \n" +
                          "\n" +
                          f"%mem={memory}GB \n" +
                          f"%chk={gjf_file}.chk \n" +
                          keyword_line + "\n" +
                          "\n" +
                          title_line + "\n"
                          "\n" +
                          "%s %s\n" % (charge, abs(charge) + 1))

        self.print_if_debug('\nGaussian file:\n==============\n%s\n==============' % (a_str_gaussian))
        with open('%s/Data/%s/%s.gjf' % (self._parent_folder, self._molecule_id, gjf_file), mode = "wt") as writefile:
            writefile.write(a_str_gaussian)

        return gjf_file

    def generate_gjf_marcus_file(self,
```

```
                                charge_geom: int,
                                charge_eval: int,
                                state_geom: int,
                                state_eval: int,
                                functional: str,
                                basisset: str,
                                number_of_proc: int,
                                memory: int,
                                keyword_line_arguments_input: list[[str, str]] = []) -> str:
        self.print_if_debug('\nIn function Molecule.generate_gjf_marcus_file():')
        functional_basisset = '%s/%s' % (functional, basisset)
        keyword_line_arguments: dict = {'functional_basisset': functional_basisset,
                                        'calculation_type': 'marcus',
                                        'state_TD': state_eval,
                                        'restart': False,
                                        'overwrite_Freq': [[], []],
                                        'overwrite_Opt': [[], []],
                                        'overwrite_NoSymm': [[], []],
                                        'overwrite_state_TD': [[], []]}

        # keyword_line_arguements_input = [['calculation_type', 'Opt'], ['restart', True], ['higher_max_cycles', 1024]]
        for i in range(0, len(keyword_line_arguments_input)):
            keyword_line_arguments['%s' % (keyword_line_arguments_input[i][0])] = keyword_line_arguments_input[i][1]

        keyword_line = self.keyword_line_generator(functional_basisset = keyword_line_arguments['functional_basisset'],
                                                   calculation_type = keyword_line_arguments['calculation_type'],
                                                   state_TD = keyword_line_arguments['state_TD'],
                                                   restart = keyword_line_arguments['restart'],
                                                   overwrite_Freq = keyword_line_arguments['overwrite_Freq'],
                                                   overwrite_Opt = keyword_line_arguments['overwrite_Opt'],
                                                   overwrite_NoSymm = keyword_line_arguments['overwrite_NoSymm'],
                                                   overwrite_state_TD = keyword_line_arguments['overwrite_state_TD'])

        oldchk_name_extension = '%s_%s_C%sS%s' % (functional, format_basis_set(basisset), charge_geom, state_geom)
        newchk_name_extension = '%s_%s_C%sS%s_opt_geom_at_C%sS%s' % (functional, format_basis_set(basisset), charge_geom,
    state_geom, charge_eval, state_eval)

        oldchk_gjf_file = "%s_%s" % (self._molecule_id, oldchk_name_extension)
        newchk_gjf_file = "%s_%s" % (self._molecule_id, newchk_name_extension)

        title_line = '%s %s' % (newchk_gjf_file, keyword_line_arguments['calculation_type'])

        a_str_gaussian = (f"%nprocshared={number_of_proc} \n" +
                          "\n" +
                          f"%mem={memory}GB \n" +
                          f"%oldchk={oldchk_gjf_file}.chk" + '\n' +
                          f"%chk={newchk_gjf_file}.chk \n" +
                          keyword_line + "\n" +
                          "\n" +
                          title_line + "\n"
                          "\n" +
                          "%s %s\n" % (charge_eval, abs(charge_eval) + 1) +
                          '\n')

        self.print_if_debug('\nGaussian file:\n==============\n%s\n==============' % (a_str_gaussian))
        with open('%s/Data/%s/%s.gjf' % (self._parent_folder, self._molecule_id, newchk_gjf_file), mode = "wt") as
    writefile:
            writefile.write(a_str_gaussian)

        return newchk_gjf_file

    def generate_sh_file(self,
                         timeout_hr: int,
                         number_of_proc: int,
                         memory_gjf_gb: int,
                         charge: int,
                         excited_state: int,
                         functional_basisset: str) -> str:
        """
        Method to write the job script for a Gaussian16 job in the HPC.

        Parameters
        ----------
        timeout_hr : int
            Timeout settings in hours.
        number_of_proc : int
            Number of processors.
        memory_gjf_gb : int
            Total memory allocated for the Gaussian job.
        charge : int
            Charge of the molecule.
        excited_state : int
            Excitation state of the molecule.
        functional_basisset : str
            Functional and basis set in the form {functional}_{basis_set}.
```

```
        Returns
        -------
        qsub_script : str
            Submission script.

        """
        self.print_if_debug('\nIn function Molecule.generate_sh_file():')
        timeout = '%sh' % (timeout_hr)
        walltime = '%s:10:00' % (timeout_hr)
        gjf_file = "%s_%s_C%sS%s" % (self._molecule_id, functional_basisset, charge, excited_state)

        qsub_script  =  "#!/bin/sh\n"
        qsub_script +=  "#PBS -l walltime=%s\n" % (walltime)
        qsub_script += ("#PBS -l select=1:ncpus=%s" % (int(number_of_proc)) +
                        ":mem=%sGB" % (int(memory_gjf_gb + 8)) +
                        ":avx=true\n\n")
        qsub_script +=  "module load gaussian/g16-c01-avx\n"
        qsub_script +=  "cp $PBS_O_WORKDIR/" + gjf_file + ".gjf ./\n"
        qsub_script +=  "cp $PBS_O_WORKDIR/" + gjf_file + ".chk ./\n"
        qsub_script +=  "timeout %s g16 %s.gjf \n" % (timeout, gjf_file)
        qsub_script +=  "formchk %s.chk %s.fchk\n" % (gjf_file, gjf_file)
        qsub_script +=  "cp *.log  $PBS_O_WORKDIR\ncp *.chk  $PBS_O_WORKDIR\ncp *.fchk  $PBS_O_WORKDIR"
        with open('%s/Data/%s/%s.sh' % (self._parent_folder, self._molecule_id, gjf_file), mode = "wt", newline="\n") as
    text_file:
            text_file.write(qsub_script)
            text_file.close()
        self.print_if_debug('\nqsub_script:\n==============\n%s\n==============' % (qsub_script))
        return qsub_script

    def generate_sh_marcus_file(self,
                                timeout_hr: int,
                                number_of_proc: int,
                                memory_gjf_gb: int,
                                charge_geom: int,
                                charge_eval: int,
                                state_geom: int,
                                state_eval: int,
                                functional_basisset: str) -> str:
        self.print_if_debug('\nIn function Molecule.generate_sh_marcus_file():')
        timeout = '%sh' % (timeout_hr)
        walltime = '%s:10:00' % (timeout_hr)
        old_gjf_file = "%s_%s_C%sS%s" % (self._molecule_id, functional_basisset, charge_geom, state_geom)
        newchk_name_extension = '%s_C%sS%s_opt_geom_at_C%sS%s' % (functional_basisset, charge_geom, state_geom,
    charge_eval, state_eval)
        gjf_file = "%s_%s" % (self._molecule_id, newchk_name_extension)

        qsub_script  =  "#!/bin/sh\n"
        qsub_script +=  "#PBS -l walltime=%s\n" % (walltime)
        qsub_script += ("#PBS -l select=1:ncpus=%s" % (int(number_of_proc)) +
                        ":mem=%sGB" % (int(memory_gjf_gb + 4)) +
                        ":avx=true\n\n")
        qsub_script +=  "module load gaussian/g16-c01-avx\n"
        qsub_script +=  "cp $PBS_O_WORKDIR/" + old_gjf_file + ".chk ./\n"
        qsub_script +=  "cp $PBS_O_WORKDIR/" + gjf_file + ".gjf ./\n"
        qsub_script +=  "cp $PBS_O_WORKDIR/" + gjf_file + ".chk ./\n"
        qsub_script +=  "timeout %s g16 %s.gjf \n" % (timeout, gjf_file)
        qsub_script +=  "formchk %s.chk %s.fchk\n" % (gjf_file, gjf_file)
        qsub_script +=  "cp *.log  $PBS_O_WORKDIR\ncp *.chk  $PBS_O_WORKDIR\ncp *.fchk  $PBS_O_WORKDIR"
        with open('%s/Data/%s/%s.sh' % (self._parent_folder, self._molecule_id, gjf_file), mode = "wt", newline="\n") as
    text_file:
            text_file.write(qsub_script)
            text_file.close()
        self.print_if_debug('\nqsub_script:\n==============\n%s\n==============' % (qsub_script))
        return qsub_script

    def generate_gjf_sh_files(self,
                              input_object,
                              excited_state: int,
                              charge: int,
                              functional: str,
                              basisset: str,
                              number_of_proc: int,
                              memory: int,
                              timeout_hr: int,
                              calculation_type: str = 'OptFreq',
                              connectivity: bool = True,
                              keyword_line_arguments_input: list[[str, str]] = []):
        """
        Generates both Gaussian16 script and the HPC job script.

        Parameters
        ----------
        input_object : TYPE
            Input object to be fed into Molecule.generate_molecule_xyz().
        excited_state : int
            Excitation state of the molecule.
```

```
        charge : int
            Charge of molecule.
        functional : str
            Functional to be used in DFT calculation.
        basisset : str
            Basis set to be used in DFT calculation.
        number_of_proc : int
            Number of processors.
        memory : int
            Total amount of memory to be divided between the processors.
        timeout_hr : int
            Timeout settings in hours.
        calculation_type : str, optional
            Calculation type. The default is 'OptFreq'.
        connectivity : bool, optional
            Whether to use molecule connectivity. The default is True.
        keyword_line_arguments_input : list[[str, str]], optional
            Overwite lines. The default is [].

        Returns
        -------
        None.

        """
        self.print_if_debug('\nIn function Molecule.generate_gjf_sh_files():')
        gjf = self.generate_gjf_file(input_object = input_object,
                                     excited_state = excited_state,
                                     charge = charge,
                                     functional = functional,
                                     basisset = basisset,
                                     number_of_proc = number_of_proc,
                                     memory = memory,
                                     calculation_type = calculation_type,
                                     connectivity = connectivity,
                                     keyword_line_arguments_input = keyword_line_arguments_input)
        sh = self.generate_sh_file(timeout_hr = timeout_hr,
                                   number_of_proc = number_of_proc,
                                   memory_gjf_gb = memory,
                                   charge = charge,
                                   excited_state = excited_state,
                                   functional_basisset = '%s_%s' % (functional, format_basis_set(basisset)))
        self.print_if_debug(gjf)
        self.print_if_debug(sh)

    def generate_gjf_sh_marcus_files(self,
                                     charge_geom: int,
                                     charge_eval: int,
                                     state_geom: int,
                                     state_eval: int,
                                     functional: str,
                                     basisset: str,
                                     number_of_proc: int,
                                     memory: int,
                                     timeout_hr: int,
                                     calculation_type: str = 'marcus',
                                     keyword_line_arguments_input: list[[str, str]] = []):
        self.print_if_debug('\nIn function Molecule.generate_gjf_sh_marcus_files():')
        gjf = self.generate_gjf_marcus_file(charge_geom = charge_geom,
                                            charge_eval = charge_eval,
                                            state_geom = state_geom,
                                            state_eval = state_eval,
                                            functional = functional,
                                            basisset = basisset,
                                            number_of_proc = number_of_proc,
                                            memory = memory,
                                            keyword_line_arguments_input = keyword_line_arguments_input)
        sh = self.generate_sh_marcus_file(timeout_hr = timeout_hr,
                                          number_of_proc = number_of_proc,
                                          memory_gjf_gb = memory,
                                          charge_geom = charge_geom,
                                          charge_eval = charge_eval,
                                          state_geom = state_geom,
                                          state_eval = state_eval,
                                          functional_basisset = '%s_%s' % (functional, format_basis_set(basisset)))
        self.print_if_debug(gjf)
        self.print_if_debug(sh)

    def generate_gjf_sh_restart_files(self,
                                      excited_state: int,
                                      charge: int,
                                      functional: str,
                                      basisset: str,
                                      number_of_proc: int,
                                      memory: int,
                                      timeout_hr: int,
                                      calculation_type: str = 'OptFreq',
```

```
                                        keyword_line_arguments_input: list[[str, str]] = []):
        self.print_if_debug('\nIn function Molecule.generate_gjf_sh_restart_files():')
        gjf = self.generate_gjf_restart_file(excited_state = excited_state,
                                             charge = charge,
                                             functional = functional,
                                             basisset = basisset,
                                             number_of_proc = number_of_proc,
                                             memory = memory,
                                             calculation_type = calculation_type,
                                             keyword_line_arguments_input = keyword_line_arguments_input)
        sh = self.generate_sh_file(timeout_hr = timeout_hr,
                                   number_of_proc = number_of_proc,
                                   memory_gjf_gb = memory,
                                   charge = charge,
                                   excited_state = excited_state,
                                   functional_basisset = '%s_%s' % (functional, format_basis_set(basisset)))
        self.print_if_debug(gjf)
        self.print_if_debug(sh)

#%%

class MongoDB():

    def __init__(self,
                 username: str,
                 password: str,
                 collection: str,
                 database: str = 'New_Reorganisation_Energy_Calc',
                 connection_string: str = 'mongodb+srv://<user>:<password>@cluster0.wxpvcwv.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0',
                 debug: bool = False):
        """
        Establishing connection to a MongoDB database.

        Parameters
        ----------
        username : str
            Username used to access that database.
        password : str
            Password correcponding to username.
        collection : str
            Name of the collection to be accessed.
        database : str, optional
            Name of the database to be accessed.. The default is 'Reorganisation_Energy_Calculation'.
        connection_string : str, optional
            Connection string to access the database. The default is 'mongodb+srv://<user>:<password>@cluster0.wxpvcwv.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0'.
        debug : bool, optional
            Toggle debug mode. The default is False.

        Returns
        -------
        None.

        """
        self._username = username
        self._password = password
        self._collection = collection
        self._database = database
        self._connection_string = connection_string
        self._debug = debug

    def __repr__(self):
        """
        Official representation

        Returns
        -------
        string
            Official representation of class.
        """
        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)

    def __str__(self):
        """
        Simplified string representation.

        Returns
        -------
        string
            String representation of class.

        """
        return '\n%s\n%s\n%s\n%s\n%s' % (self._username, self._password, self._database, self._collection, self._connection_string)
```

```
    def connect_mongo(self):
        """
        Connects to MongoDB and retyurns the database.

        Returns
        -------
        database : MongoDB Database
            Full database.

        """
        if self._debug:
            print('\nIn function MongoDB.connect_mongo():')
            print('Username: %s\nPassword: %s\nDatabase: %s\nConnection string: %s' % (self._username, self._password,
     self._collection, self._connection_string))
        client = MongoClient(self._connection_string.replace('<user>', self._username).replace('<password>', urllib.parse.
     quote_plus(self._password)))
        database = client[self._database]
        return database

    def pull(self):
        """
        Method to pull the collection from the online MongoDB database.

        Returns
        -------
        df : pandas.core.frame.DataFrame
            Dataframe of all the entries in the collection.

        """
        if self._debug:
            print('\nIn function MongoDB.pull():')
        database = self.connect_mongo()['%s' % (self._collection)].find()
        df =  pd.DataFrame(list(database)).iloc[:, :]
        return df

    def create_post(self, keys: list, values: list) -> dict:
        """
        Takes in keys and values in list/array form and organise it into a post for pushing onto MongoDB database.

        Parameters
        ----------
        keys : list
            List of keys in entry.
        values : list
            List of values corresonding to the keys.

        Raises
        ------
        ValueError
            Length of keys and values does not match.

        Returns
        -------
        dict
            A MongoDB post that can be pushed onto the collection.

        """
        if self._debug:
            print('\nIn function MongoDB.create_post():')
            print('keys: %s\nvalues: %s' % (keys, values))
            print('length (keys): %s\nlength (values): %s' % (len(keys), len(values)))
        if len(keys) != len(values):
            raise ValueError('Length of input is not the same:\n keys = %s\n values = %s' % ((len(keys)), len(values)))
        post: dict = {}
        for i in range(0, len(keys)):
            post[keys[i]] = values[i]
        return post

    def push_one_new_entry(self, keys: list, values: list):
        """
        Method to create a new entry on the online collection.

        Parameters
        ----------
        keys : list
            List of keys.
        values : list
            List of values corresponding to above keys.

        Raises
        ------
        Exception
            Push unsucessfull.

        Returns
```

```
        -------
        None.

        """
        post: dict = self.create_post(keys = keys, values = values)
        if self._debug:
            print('\nIn function MongoDB.push_one_entry():')
            pprint.pprint(post)
        try:
            self.connect_mongo()['%s' % (self._collection)].insert_one(post)
        except:
            raise Exception('\nPush failed.')

    def replace_one_entry(self, keys: list, values: list):
        """
        Method to replace one entry of the online collection.

        Parameters
        ----------
        keys : list
            List of keys.
        values : list
            List of values corresponding to above keys.

        Raises
        ------
        Exception
            Push unsucessfull.

        Returns
        -------
        None.

        """
        post: dict = self.create_post(keys = keys, values = values)
        if self._debug:
            print('\nIn function MongoDB.replace_one_entry():')
            pprint.pprint(post)
        try:
            self.connect_mongo()['%s' % (self._collection)].replace_one({'_id':post['_id']}, post)
        except:
            raise Exception('\nReplace failed.')

    def update_one_entry(self, _id: str, keys: list, values: list):
        """
        Method to only edit one entry on the MongoDB database.

        Parameters
        ----------
        _id : str
            ID of the entry.
        keys : list
            Keys to be changed.
        values : list
            The corresponding values to the keys.

        Raises
        ------
        Exception
            Update failed.

        Returns
        -------
        None.

        """
        post: dict = self.create_post(keys = keys, values = values)
        if self._debug:
            print('\nIn function MongoDB.update_one_entry():')
        try:
            self.connect_mongo()['%s' % (self._collection)].update_one({'_id': _id}, {'$set': post})
        except:
            raise Exception('\nUpdate failed.')

    def export_to_excel(self, path: str = ''):
        """
        Method to export the entire online collection to an Excel file saved locally.

        Parameters
        ----------
        path : str, optional
            Path to write the Excel file. The default is '': the current working directory would be used.

        Raises
        ------
        Exception
```

```
            Failed operation.

        Returns
        -------
        None.

        """
        if path == '':
            path = '%s/%s.xlsx' % (os.getcwd(), self._collection)
        dataframe = self.pull()
        if self._debug:
            print('\nIn function MongoDB.export_to_excel():')
            print('Export to location:\n%s' % (path))
            pprint.pprint(dataframe)
        try:
            dataframe.to_excel(r'%s' % (path))
        except:
            raise Exception('\nExport to excel not sucessfull.')

#%%

class HPC():

    def __init__(self,
                 username: str,
                 password: str,
                 hostname: str = 'login.hpc.ic.ac.uk',
                 debug: bool = False):
        """
        A class to connect to the HPC and to submit operations.

        Parameters
        ----------
        username : str
            Username for the connection.
        password : str
            Password correcponding to the username.
        hostname : str, optional
            Hostname of the HPC. The default is 'login.hpc.ic.ac.uk'.
        debug : bool, optional
            Toggle for debugging. The default is False.

        Returns
        -------
        None.

        """
        self._username = username
        self._password = password
        self._hostname = hostname
        self._debug = debug

    def print_if_debug(self, lines: str):
        """
        Print lines if in debug mode

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

        Returns
        -------
        None.

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

    def qstat(self) -> str:
        """
        Query to return the queue status under the username.

        Returns
        -------
        q_status : str
            String containing the queue status.

        """
        self.print_if_debug('\nIn function HPC.qstat():')
        with paramiko.SSHClient() as client:
            client.load_system_host_keys()
            client.set_missing_host_key_policy(paramiko.WarningPolicy)
            client.connect(hostname = self._hostname,
                           username = self._username,
                           password = self._password)
```

```
            stdin, stdout, stderr = client.exec_command("/opt/pbs/bin/qstat ")

            q_status = stdout.read()

            self.print_if_debug('Standard output:\n%s' % (stdout.read()))
            self.print_if_debug('Standard error:\n%s' % (stderr.read()))

            if stdout.channel.recv_exit_status() != 0:
                print('\nError encountered:')
                print("Standard error:\n%s" % (stderr.read()))
                print("Exit status: {}".format(stdout.channel.recv_exit_status()))
            client.close()
        q_status = str(q_status)[2:-1].replace('\\n', '\n')
        return q_status

    def formatted_qstat(self) -> pd.core.frame.DataFrame:
        """
        Method to return the queue status as a pandas dataframe.

        Returns
        -------
        formatted_q_status : pd.core.frame.DataFrame
            Queue status.

        """
        self.print_if_debug('\nIn function HPC.formatted_qstat():')
        q_status = self.qstat()
        self.print_if_debug('%s' % (q_status))
        q_status = q_status.split('\n')
        q_status = q_status[2:-1]
        formatted_q_status = pd.DataFrame(columns = ['Job ID', 'Class', 'Job Name', 'Status', 'Comment'])
        for i in range(0, len(q_status)):
            job_id = q_status[i][0:14].replace(' ', '')
            job_class = q_status[i][15:30].replace(' ', '')
            job_name = q_status[i][31:51].replace(' ', '')
            job_status = q_status[i][52:59].replace(' ', '')
            job_comment = q_status[i][60:]
            formatted_q_status.loc[len(formatted_q_status)] = [job_id, job_class, job_name, job_status, job_comment]
        return formatted_q_status

    def transfer_from_HPC(self,
                          local_folder: str,
                          remote_folder: str,
                          filename: str):
        """
        Method to transfer files from the HPC to local device.

        Parameters
        ----------
        local_folder : str
            Path to local folder where the file would be saved.
        remote_folder : str
            Path to remote folder where the file is located.
        filename : str
            Filename of the file to be copied.

        Returns
        -------
        None.

        """
        self.print_if_debug('\nIn function HPC.transfer_from_HPC():')
        with paramiko.SSHClient() as client:
            client.load_system_host_keys()
            client.set_missing_host_key_policy(paramiko.WarningPolicy)
            client.connect(hostname = self._hostname,
                           username = self._username,
                           password = self._password)
            sftp_client = client.open_sftp()
            sftp_client.get(r'%s/%s' % (remote_folder, filename),
                            r'%s/%s' % (local_folder, filename))
            sftp_client.close()
            client.close()
        self.print_if_debug('\nTransfer sucessfull.')

    def submit_job_gaussian(self,
                            local_folder: str,
                            remote_folder: str,
                            filename: str):
        """
        Method to submit a Gaussian16 job in the HPC.
        The required scripts (.gjf and .sh) should be located in the local folder.

        Parameters
        ----------
        local_folder : str
```

```
            Local folder where the .gjf and .sh job scripts are located.
        remote_folder : str
            Remote folder where scripts will be transferred to.
        filename : str
            Full filename of the scripts. Example {filename}.sh and {filename}.gjf.

        Returns
        -------
        None.

        """
        self.print_if_debug('\nIn function HPC.submit_job_gaussian():')
        with paramiko.SSHClient() as client:
            client.load_system_host_keys()
            client.set_missing_host_key_policy(paramiko.WarningPolicy)

            # Establish SSH connection
            client.connect(hostname = self._hostname,
                           username = self._username,
                           password = self._password)

            # Establish SFTP connection ftp_client=ssh.open_sftp()
            stdin, stdout, stderr = client.exec_command("mkdir -p " + remote_folder)
            with client.open_sftp() as sftp0:
                # Push job submission script to a particular path on the cluster
                sftp0.put(r'%s/%s.sh' % (local_folder, filename), r'%s/%s.sh' % (remote_folder, filename))
                sftp0.put(r'%s/%s.gjf' % (local_folder, filename), r'%s/%s.gjf' % (remote_folder, filename))
            # Submit our Grid Engine job by running a remote 'qsub' command over SSH
            stdin, stdout, stderr = client.exec_command(f"cd {remote_folder} \n " +
                                                        f"chmod +rwx {filename}.sh\n" +
                                                        f"/opt/pbs/bin/qsub {filename}.sh \n")
            # Show the standard output and error of our job
            stdout_copy = deepcopy(stdout.read())
            self.print_if_debug('\nStandard output:\n%s' % (stdout_copy))
            self.print_if_debug('\nStandard error:\n%s' % (stderr.read()))
            print(stdout.channel.recv_exit_status())
            if stdout.channel.recv_exit_status() != 0:
                print('\nError encountered:')
                print("\nStandard error:\n%s" % (stderr.read()))
                print("Exit status: {}".format(stdout.channel.recv_exit_status()))
            client.close()
        job_id = int(''.join(filter(str.isdigit, str(stdout_copy))))
        return job_id

    def restart_job_gaussian(self,
                             local_folder: str,
                             remote_folder: str,
                             filename: str):
        self.print_if_debug('\nIn function HPC.restart_job_gaussian():')
        with paramiko.SSHClient() as client:
            client.load_system_host_keys()
            client.set_missing_host_key_policy(paramiko.WarningPolicy)

            # Establish SSH connection
            client.connect(hostname = self._hostname,
                           username = self._username,
                           password = self._password)

            # Establish SFTP connection ftp_client=ssh.open_sftp()
            stdin, stdout, stderr = client.exec_command("mkdir -p " + remote_folder)
            with client.open_sftp() as sftp0:
                # Push job submission script to a particular path on the cluster
                sftp0.put(r'%s/%s.sh' % (local_folder, filename), r'%s/%s.sh' % (remote_folder, filename))
                sftp0.put(r'%s/%s.gjf' % (local_folder, filename), r'%s/%s.gjf' % (remote_folder, filename))
                sftp0.put(r'%s/%s.chk' % (local_folder, filename), r'%s/%s.chk' % (remote_folder, filename))
            # Submit our Grid Engine job by running a remote 'qsub' command over SSH
            stdin, stdout, stderr = client.exec_command(f"cd {remote_folder} \n " +
                                                        f"chmod +rwx {filename}.sh\n" +
                                                        f"/opt/pbs/bin/qsub {filename}.sh \n")
            # Show the standard output and error of our job
            stdout_copy = deepcopy(stdout.read())
            self.print_if_debug('\nStandard output:\n%s' % (stdout_copy))
            self.print_if_debug('\nStandard error:\n%s' % (stderr.read()))
            print(stdout.channel.recv_exit_status())
            if stdout.channel.recv_exit_status() != 0:
                print('\nError encountered:')
                print("\nStandard error:\n%s" % (stderr.read()))
                print("Exit status: {}".format(stdout.channel.recv_exit_status()))
            client.close()
        job_id = int(''.join(filter(str.isdigit, str(stdout_copy))))
        return job_id

    def transfer_chk_fchk_log_files_from_HPC(self,
                                             local_folder: str,
                                             remote_folder: str,
                                             filename: str):
```

```
        """
        Transfer the calculation files from the HPC to local folder.

        Parameters
        ----------
        local_folder : str
            Local folder where the files will be saved.
        remote_folder : str
            Remote folder where the files will be transferred from.
        filename : str
            Full filename of the files. Example {filename}.fchk, {filename}.chk, and {filename}.log.

        Returns
        -------
        None.

        """
        self.print_if_debug('\nIn function HPC.transfer_chk_fchk_log_files_from_HPC():')
        try:
            self.print_if_debug('Transferring log files from HPC:\nlocal folder = %s\nremote folder = %s\nfile = %s.log' %
    (local_folder, remote_folder, filename))
            self.transfer_from_HPC(local_folder = local_folder,
                                   remote_folder = remote_folder,
                                   filename = r'%s.log' % (filename))
        except:
            print('\nFailed to transfer log files.')
        try:
            self.print_if_debug('Transferring chk files from HPC:\nlocal folder = %s\nremote folder = %s\nfile = %s.chk' %
    (local_folder, remote_folder, filename))
            self.transfer_from_HPC(local_folder = local_folder,
                                   remote_folder = remote_folder,
                                   filename = r'%s.chk' % (filename))
        except:
            print('\nFailed to transfer chk files.')
        try:
            self.print_if_debug('Transferring fchk files from HPC:\nlocal folder = %s\nremote folder = %s\nfile = %s.fchk'
    % (local_folder, remote_folder, filename))
            self.transfer_from_HPC(local_folder = local_folder,
                                   remote_folder = remote_folder,
                                   filename = r'%s.fchk' % (filename))
        except:
            print('\nFailed to transfer fchk files.')

    def copy_fchk_log_files_for_dushin(self,
                                       remote_dushin_folder: str,
                                       remote_log_folder: str,
                                       filename_initial_state:str,
                                       filename_final_state: str):
        """
        Transfer .fchk files and .log files between two folders in the HPC.

        Parameters
        ----------
        remote_dushin_folder : str
            Folder where DUSHIN would be ran.
        remote_log_folder : str
            Folder where the fchk and log files are saved.
        filename_initial_state : str
            Filename of the initial state calculation, e.g. in the form {filename_initial_state}.log.
        filename_final_state : str
            Filename of the final state calculation, e.g. in the form {filename_final_state}.log.

        Returns
        -------
        None.

        """
        self.print_if_debug('\nIn function HPC.copy_fchk_log_files_for_dushin():')

        self.print_if_debug('Copying fchk files for:\n %s.fchk and\n %s.fchk' % (filename_initial_state,
    filename_final_state))
        with paramiko.SSHClient() as client:
            client.load_system_host_keys()
            client.set_missing_host_key_policy(paramiko.WarningPolicy)
            client.connect(hostname = self._hostname,
                           username = self._username,
                           password = self._password)
            stdin, stdout, stderr = client.exec_command(f"cd {remote_log_folder} \n " +
                                                        f"cp {filename_initial_state}.fchk {remote_dushin_folder} \n" +
                                                        f"cp {filename_final_state}.fchk {remote_dushin_folder} \n")
            # Show the standard output and error of our job
            self.print_if_debug('Standard output:\n%s' % (stdout.read()))
            self.print_if_debug('Standard error:\n%s' % (stderr.read()))
            if stdout.channel.recv_exit_status() != 0:
                print('\nError encountered:')
                print("Standard error:\n%s" % (stderr.read()))
```

```
                print("Exit status: {}".format(stdout.channel.recv_exit_status()))
            client.close()
        self.print_if_debug('fchk files copied sucessfully')

        self.print_if_debug('Copying log files for:\n %s.log and\n %s.log' % (filename_initial_state, filename_final_state))
        with paramiko.SSHClient() as client:
            client.load_system_host_keys()
            client.set_missing_host_key_policy(paramiko.WarningPolicy)
            client.connect(hostname = self._hostname,
                           username = self._username,
                           password = self._password)
            stdin, stdout, stderr = client.exec_command(f"cd {remote_log_folder} \n " +
                                                        f"cp {filename_initial_state}.log {remote_dushin_folder} \n" +
                                                        f"cp {filename_final_state}.log {remote_dushin_folder} \n")
            # Show the standard output and error of our job
            self.print_if_debug('Standard output:\n%s' % (stdout.read()))
            self.print_if_debug('Standard error:\n%s' % (stderr.read()))
            if stdout.channel.recv_exit_status() != 0:
                print('\nError encountered:')
                print("Standard error:\n%s" % (stderr.read()))
                print("Exit status: {}".format(stdout.channel.recv_exit_status()))
            client.close()
        self.print_if_debug('log files copied sucessfully')

    def submit_job_dushin(self,
                          remote_dushin_folder: str,
                          remote_bin_path: str,
                          initial_state: str,
                          final_state: str,
                          mol_id_functional_basisset: str):
        self.print_if_debug('\nIn function HPC.submit_job_dushin():')
        filename_initial_state = '%s_%s' % (mol_id_functional_basisset, initial_state)
        filename_final_state = '%s_%s' % (mol_id_functional_basisset, final_state)
        self.print_if_debug('\nDushin started for %s %s %s.' % (mol_id_functional_basisset, initial_state, final_state))
        with paramiko.SSHClient() as client:
            client.load_system_host_keys()
            client.set_missing_host_key_policy(paramiko.WarningPolicy)
            client.connect(hostname = self._hostname,
                           username = self._username,
                           password = self._password)

            stdin, stdout, stderr = client.exec_command(
                f"cd {remote_dushin_folder} \n " +
                f"export PATH=$PATH:{remote_bin_path} \n" +
                f"find -type f -name '*.sh.*' -delete \n" +
                f"cd {remote_dushin_folder} \n" +

                f"split_log.sh {filename_initial_state}.log \n" +
                f"mv {filename_initial_state}.log org_{filename_initial_state}.log\n" +
                f"mv freq_1_{filename_initial_state}.log {filename_initial_state}.log \n" +

                f"split_log.sh {filename_final_state}.log \n" +
                f"mv {filename_final_state}.log org_{filename_final_state}.log\n" +
                f"mv freq_1_{filename_final_state}.log {filename_final_state}.log \n" +

                f"find -type f -name '*freq_2*' -delete \n" +
                f"rundushin.sh {mol_id_functional_basisset} {initial_state} {final_state}")

            if stdout.channel.recv_exit_status() != 0:
                print('\nError encountered:')
                print("\nStandard error:\n%s" % (stderr.read()))
                print("Exit status: {}".format(stdout.channel.recv_exit_status()))
            client.close()
            self.print_if_debug('DUSHIN submission sucessful')

#%%

class SMILES_database():

    def __init__(self,
                 username: str,
                 password: str,
                 collection: str = 'SMILES_Database',
                 database: str = 'New_Reorganisation_Energy_Calc',
                 connection_string: str = 'mongodb+srv://<user>:<password>@cluster0.wxpvcwv.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0',
                 debug: bool = False):
        """
        Database of SMILES stored through a MongoDB connection.

        Parameters
        ----------
        username : str
            Username to connect to MongoDB databsse.
        password : str
```

```
            Password to connect to MongoDB database.
        collection : str, optional
            Name of the collection on the database. The default is 'SMILES_Database'.
        database : str, optional
            Name of the database. The default is 'New_Reorganisation_Energy_Calc'.
        connection_string : str, optional
            Connection string to connect to MongoDB. The default is 'mongodb+srv://<user>:<password>@cluster0.wxpvcwv.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0'.
        debug : bool, optional
            Toggle for debugging. The default is False.

        Returns
        -------
        None.

        """
        self._username = username
        self._password = password
        self._collection = collection
        self._database = database
        self._connection_string = connection_string
        self._debug = debug
        self._MongoDB = MongoDB(username = self._username,
                                password = self._password,
                                collection = self._collection,
                                database = self._database,
                                connection_string = self._connection_string,
                                debug = self._debug)

    def __repr__(self):
        """
        Official representation

        Returns
        -------
        string
            Official representation of class.
        """
        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)

    def __str__(self):
        """
        Simplified string representation.

        Returns
        -------
        string
            String representation of class.

        """
        return '\n%s\n%s\n%s\n%s\n%s' % (self._username, self._password, self._database, self._collection, self._connection_string)

    def pull_SMILES_database(self):
        """
        Get the SMILES database as a pandas dataframe.

        Raises
        ------
        Exception
            Failed to fetch SMILES database.

        Returns
        -------
        dataframe : pandas.core.frame.DataFrame
            SMILES database as a pandas dataframe.

        """
        if self._debug:
            print('\nIn function SMILES_database.pull_SMILES_database():')
        try:
            dataframe = self._MongoDB.pull()
            return dataframe
        except:
            raise Exception('Failed to fetch SMILES database.')

    def export_to_excel(self, path: str = ''):
        """
        Downloads a copy of the SMILES database as an .xlsx file.

        Parameters
        ----------
        path : str, optional
            Full path of the folder where the file would be saved. The default is ''.

```

```
        Raises
        ------
        Exception
            Export to excel not sucessfull..

        Returns
        -------
        None.

        """
        if path == '':
            path = '%s/SMILES_database.xlsx' % (os.getcwd())
        dataframe = self.pull_SMILES_database()
        if self._debug:
            print('\nIn function SMILES_database.export_to_excel():')
            print('Export to location:\n%s' % (path))
            pprint.pprint(dataframe)
        try:
            dataframe.to_excel(r'%s' % (path))
        except:
            raise Exception('\nExport to excel not sucessfull.')
        pass

    def check_against_database(self, iupac_name: str, SMILES: str = '') -> list[str, str]:
        """
        Check the molecule name and SMILES against database. If both molecule name and SMILES are declared, checks for a match.
        If only the Name is declared, checks whether the molecule is in the database, and returns the corresponding SMILES.

        Parameters
        ----------
        iupac_name : str
            Molecule name.
        SMILES : str, optional
            SMILES string. The default is ''.

        Raises
        ------
        Exception
            SMILES string not declared and not found in database.
        Exception
            Input SMILES does not match database.

        Returns
        -------
        list[str, str]
            The molecule name and SMILES string if no error is encountered.

        """
        if self._debug:
            print('\nIn function SMILES_database.in_database():')
            print('SMILES: %s\niupac_name: %s' % (SMILES, iupac_name))
        dataframe = self.pull_SMILES_database()
        # if SMILES not declared, ...
        if SMILES == '':
            # ... try looking for SMILES in database, ...
            try:
                SMILES = dataframe.at[dataframe[dataframe['_id'] == iupac_name].index[0], 'SMILES']
                print('Molecule found in database.')
                return iupac_name, SMILES
            # ... if not found in database, raise exception
            except:
                raise Exception('SMILES string not declared and not found in database.')
        # if SMILES declared, ...
        else:
            try:
                # ... compare SMILES to database, ...
                SMILES_found_in_database = dataframe.at[dataframe[dataframe['_id'] == iupac_name].index[0], 'SMILES']
                print('Molecule found in database.')
            except:
                SMILES_found_in_database = 'not found'
                print('Molecule not found in database.')
            # ... if SMILES from input and database does not match, raise exception, ...
            if SMILES_found_in_database != SMILES and SMILES_found_in_database != 'not found':
                raise Exception('Input SMILES does not match database:\nInput: %s\nDatabase: %s' % (SMILES, SMILES_found_in_database))
            # otherwise if SMILES not found in database, append and write to database
            elif SMILES_found_in_database == 'not found':
                self._MongoDB.push_one_new_entry(keys = ['_id', 'SMILES'], values = [iupac_name, SMILES])
                print('Molecule added to database:\nMolecule Name: %s\nSMILES: %s' % (iupac_name, SMILES))
                return iupac_name, SMILES
            else:
                print('Molecule matched with database')
                return iupac_name, SMILES
```

```
#%%

class Funnel():

    def __init__(self,
                 path: str = r'%s\Funnel.xlsx' % (os.getcwd()),
                 debug: bool = False):
        self._funnel = pd.DataFrame(columns = ['MMFFO', 'Functional', 'Basis Set 1', 'Basis Set 2', 'Basis Set 3'])
        self._path = path
        self._debug = debug

    def retrieve(self):
        return self._funnel

    def add_workflow(self,
                     MMFFO: str = 'Yes',
                     functional: str = 'B3LYP',
                     basis_set_1: str = '6-31G(d)',
                     basis_set_2: str = '6-311+G(d,p)',
                     basis_set_3: str = 'No'):
        if self._debug:
            print('\nIn function Funnel.add_workflow():')
            print('MMFFO = %s\nFunctional = %s\nBasis Set 1 = %s\nBasis Set 2 = %s\nBasis Set 3 = %s' % (MMFFO, functional
    , basis_set_1, basis_set_2, basis_set_3))
        self._funnel.loc[len(self._funnel)] = [MMFFO, functional, basis_set_1, basis_set_2, basis_set_3]

    def workflow_indices(self):
        return list(self._funnel.index)

    def MMFFO(self, index):
        return self._funnel.at[index, 'MMFFO']

    def functional(self, index):
        return self._funnel.at[index, 'Functional']

    def basis_set_1(self, index):
        return self._funnel.at[index, 'Basis Set 1']

    def basis_set_2(self, index):
        return self._funnel.at[index, 'Basis Set 2']

    def basis_set_3(self, index):
        return self._funnel.at[index, 'Basis Set 3']

    def export_to_excel(self, path: str = ''):
        if path == '':
            path = self._path
        dataframe = self.retrieve()
        if self._debug:
            print('\nIn function Funnel.export_to_excel():')
            print('Export to location:\n%s' % (path))
            pprint.pprint(dataframe)
        try:
            dataframe.to_excel(r'%s' % (path))
        except:
            raise Exception('\nExport to excel not sucessfull.')

    def load_from_excel(self, path: str = ''):
        if path == '':
            path = self._path
        if self._debug:
            print('\nIn function Funnel.load_from_excel():')
            print('Load from location:\n%s' % (path))
        try:
            self._funnel = pd.read_excel(path)[['MMFFO', 'Functional', 'Basis Set 1', 'Basis Set 2', 'Basis Set 3']]
        except:
            raise Exception('Read unsucessfull, unable to load from folder %s' % (path))

#%%

class Tracker():

    def __init__(self,
                 username: str,
                 password: str,
                 collection: str = 'Tracker',
                 database: str = 'New_Reorganisation_Energy_Calc',
                 connection_string: str = 'mongodb+srv://<user>:<password>@cluster0.wxpvcwv.mongodb.net/?retryWrites=true&
    w=majority&appName=Cluster0',
                 debug: bool = False):
        self._username = username
        self._password = password
        self._collection = collection
        self._database = database
        self._connection_string = connection_string
        self._debug = debug
```

```
        self._MongoDB = MongoDB(username = self._username,
                                password = self._password,
                                collection = self._collection,
                                database = self._database,
                                connection_string = self._connection_string,
                                debug = self._debug)

    def print_if_debug(self, line):
        if self._debug == True:
            print(line)

    def pull_tracker(self):
        self.print_if_debug('\nIn function Tracker.pull_tracker():')
        try:
            dataframe = self._MongoDB.pull()
            return dataframe
        except:
            raise Exception('Failed to fetch tracker database.')

    def export_to_excel(self, parent_folder: str = ''):
        if parent_folder == '':
            parent_folder = r'%s' % (os.getcwd())

        dataframe = self.pull_tracker()
        self.print_if_debug('\nIn function Tracker.export_to_excel():')
        self.print_if_debug('Export to location:\n%s' % (parent_folder))
        self.print_if_debug(dataframe)
        try:
            dataframe.to_excel(r'%s/Tracker.xlsx' % (parent_folder))
        except:
            raise Exception('\nExport to excel not sucessfull.')

    def add_entry(self,
                  iupac_name: str,
                  SMILES: str,
                  MMFFO: str,
                  functional: str,
                  basis_set_1: str,
                  basis_set_2: str,
                  basis_set_3: str):
        self.print_if_debug('\nIn function Tracker.add_entry():')
        dataframe = self.pull_tracker()
        duplicates = dataframe.loc[((dataframe['IUPAC Name'] == iupac_name) &
                                    (dataframe['SMILES'] == SMILES) &
                                    (dataframe['MMFFO'] == MMFFO) &
                                    (dataframe['Functional'] == functional) &
                                    (dataframe['Basis Set 1'] == basis_set_1) &
                                    (dataframe['Basis Set 2'] == basis_set_2) &
                                    (dataframe['Basis Set 3'] == basis_set_3))]
        molecule_id = tag(iupac_name)

        def create_empty_entry(molecule_id,
                               iupac_name,
                               SMILES,
                               MMFFO,
                               functional,
                               basis_set_1,
                               basis_set_2,
                               basis_set_3,
                               username):
            self._MongoDB.push_one_new_entry(keys = ['_id', 'IUPAC Name', 'SMILES', 'MMFFO',
                                                     'Functional', 'Basis Set 1', 'Basis Set 2', 'Basis Set 3',
                                                     'C0S0: BS1', 'C0S0: BS2', 'C0S0: BS3',
                                                     'C0S1: BS1', 'C0S1: BS2', 'C0S1: BS3',
                                                     'C0S1: BS1 - C0S0 geometry', 'C0S1: BS2 - C0S0 geometry', 'C0S1: BS3 - C0S0 geometry', 'C0S0: BS1 - C0S1 geometry', 'C0S0: BS2 - C0S1 geometry', 'C0S0: BS3 - C0S1 geometry',
                                                     'C1S0: BS1', 'C1S0: BS2', 'C1S0: BS3',
                                                     'C1S0: BS1 - C0S0 geometry', 'C1S0: BS2 - C0S0 geometry', 'C1S0: BS3 - C0S0 geometry', 'C0S0: BS1 - C1S0 geometry', 'C0S0: BS2 - C1S0 geometry', 'C0S0: BS3 - C1S0 geometry',
                                                     'C-1S0: BS1', 'C-1S0: BS2', 'C-1S0: BS3',
                                                     'C-1S0: BS1 - C0S0 geometry', 'C-1S0: BS2 - C0S0 geometry', 'C-1S0: BS3 - C0S0 geometry', 'C0S0: BS1 - C-1S0 geometry', 'C0S0: BS2 - C-1S0 geometry', 'C0S0: BS3 - C-1S0 geometry',
                                                     'User', 'Comment', 'Activity Log', 'Last Updated'],
                                           values = [molecule_id, iupac_name, SMILES, MMFFO,
                                                     functional, basis_set_1, basis_set_2, basis_set_3,
                                                     'Not Started', 'Not Started', 'Not Started',
                                                     'Not Started', 'Not Started', 'Not Started',
                                                     'Not Started', 'Not Started', 'Not Started', 'Not Started', 'Not Started', 'Not Started',
                                                     'Not Started', 'Not Started', 'Not Started',
                                                     'Not Started', 'Not Started', 'Not Started', 'Not Started', 'Not Started', 'Not Started',
                                                     'Not Started', 'Not Started', 'Not Started',
                                                     'Not Started', 'Not Started', 'Not Started', 'Not Started', 'Not Started', 'Not Started',
```

```
                                                    username, 'No comment', 'Entry created on %s' % (str(datetime.now()
    ).split('.')[0]), timestamp()]
                                            )
            self.print_if_debug('\nEntry sucessfully created.')
            # self._MongoDB.push_one_new_entry(keys = ['_id', 'IUPAC Name', 'SMILES', 'MMFFO',
            #                                          'Functional', 'Basis Set 1', 'Basis Set 2', 'Basis Set 3',
            #                                          'C0S0: BS1', 'C0S0: BS2', 'C0S0: BS3',
            #                                          'C0S1: BS1', 'C0S1: BS2', 'C0S1: BS3', 'C0S1-BS1: DUSHIN', 'C0S1-
    BS2: DUSHIN', 'C0S1-BS3: DUSHIN',
            #                                          'C1S0: BS1', 'C1S0: BS2', 'C1S0: BS3', 'C1S0-BS1: DUSHIN', 'C1S0-
    BS2: DUSHIN', 'C1S0-BS3: DUSHIN',
            #                                          'C-1S0: BS1', 'C-1S0: BS2', 'C-1S0: BS3', 'C-1S0-BS1: DUSHIN', 'C-1
    S0-BS2: DUSHIN', 'C-1S0-BS3: DUSHIN',
            #                                          'User', 'Comment', 'Activity Log', 'Last Updated'],
            #                                  values = [molecule_id, iupac_name, SMILES, MMFFO,
            #                                            functional, basis_set_1, basis_set_2, basis_set_3,
            #                                            'Not Started', 'Not Started', 'Not Started',
            #                                            'Not Started', 'Not Started', 'Not Started', 'Not Started', 'Not
    Started', 'Not Started',
            #                                            'Not Started', 'Not Started', 'Not Started', 'Not Started', 'Not
    Started', 'Not Started',
            #                                            'Not Started', 'Not Started', 'Not Started', 'Not Started', 'Not
    Started', 'Not Started',
            #                                            self._username, 'No comment', 'Entry created on %s' % (str(
    datetime.now()).split('.')[0]), timestamp()]
            #                                           )
            # self.print_if_debug('\nEntry sucessfully created.')

        if len(duplicates) == 0:
            create_empty_entry(molecule_id = molecule_id,
                               iupac_name = iupac_name,
                               SMILES = SMILES,
                               MMFFO = MMFFO,
                               functional = functional,
                               basis_set_1 = basis_set_1,
                               basis_set_2 = basis_set_2,
                               basis_set_3 = basis_set_3,
                               username = self._username)
        else:
            print('\nDuplicate found:')
            print(duplicates)
            print('\nDo you want to proceed with adding entry? (Y/N)')
            reply = input()
            if reply == 'Y' or reply == 'y':
                create_empty_entry(molecule_id = molecule_id,
                                   iupac_name = iupac_name,
                                   SMILES = SMILES,
                                   MMFFO = MMFFO,
                                   functional = functional,
                                   basis_set_1 = basis_set_1,
                                   basis_set_2 = basis_set_2,
                                   basis_set_3 = basis_set_3,
                                   username = self._username)
            del reply
        # self.export_to_excel()
        return molecule_id

    def update_entry(self,
                     _id: str,
                     keys: list,
                     values: list):
        self.print_if_debug('\nIn function Tracker.update_entry():')
        # dataframe = self.pull_tracker()
        keys.append('Last Updated')
        values.append('%s' % (timestamp()))
        # if _id not in dataframe['_id'].values:
            # raise ValueError('_id not found in database: %s' % (_id))
        if len(keys) != len(values):
            raise ValueError('Length of keys and values does not match.')
        self._MongoDB.update_one_entry(_id = _id,
                                       keys = keys,
                                       values = values)
        # self.export_to_excel()

#%%

class Results:

    def __init__(self,
                 username: str,
                 password: str,
                 collection: str = 'Results',
                 database: str = 'New_Reorganisation_Energy_Calc',
                 connection_string: str = 'mongodb+srv://<user>:<password>@cluster0.wxpvcwv.mongodb.net/?retryWrites=true&
     w=majority&appName=Cluster0',
                 debug: bool = False):
```

```
        self._username = username
        self._password = password
        self._collection = collection
        self._database = database
        self._connection_string = connection_string
        self._debug = debug

        self._MongoDB = MongoDB(username = self._username,
                                password = self._password,
                                collection = self._collection,
                                database = self._database,
                                connection_string = self._connection_string,
                                debug = self._debug)

    def print_if_debug(self, line):
        if self._debug == True:
            print(line)

    def pull_results(self):
        if self._debug:
            self.print_if_debug('\nIn function Results.pull_results():')
        try:
            dataframe = self._MongoDB.pull()
            return dataframe
        except:
            raise Exception('Failed to fetch results database.')

    def export_to_excel(self, parent_folder: str = ''):
        if parent_folder == '':
            parent_folder = r'%s' % (os.getcwd())

        dataframe = self.pull_results()
        if self._debug:
            self.print_if_debug('\nIn function Results.export_to_excel():')
            self.print_if_debug('Export to location:\n%s' % (parent_folder))
            pprint.pprint(dataframe)
        try:
            dataframe.to_excel(r'%s/Results.xlsx' % (parent_folder))
        except:
            raise Exception('\nExport to excel not sucessfull.')

    def create_empty_entry(self,
                           molecule_id: str,
                           iupac_name: str,
                           SMILES: str,
                           MMFFO: str,
                           functional: str,
                           basis_set: str,
                           initial_charge_and_state: str,
                           final_charge_and_state: str):
        # try:
        self._MongoDB.push_one_new_entry(keys = ['_id', 'IUPAC Name', 'SMILES', 'MMFFO', 'Functional', 'Basis Set',
                                                 'Inital Charge and State', 'Final Charge and State', 'Alpha Occ Orbitals List', 'Alpha Virt Orbitals List', 'HOMO-LUMO Gap',
                                                 'Initial CS at Initial Optimised Geometry', 'Initial CS at Final Optimised Geometry', 'Final CS at Initial Optimised Geometry', 'Final CS at Final Optimised Geometry',
                                                 'Vibrational Frequencies', 'Displacements', 'Reorganisation Energies', 'Parsed SCF Summary', 'Initial Atoms Summary', 'Final Atoms Summary',
                                                 '4-Point Reorganisation Energy', 'DUSHIN Reorganisation Energy',
                                                 'Difference (4-point & DUSHIN)', 'Difference Percentage (4-point & DUSHIN)',
                                                 'Comment', 'Last Updated'],
                                          values = ['%s_%s_%s_%s_%s' % (molecule_id, functional, format_basis_set(basis_set), initial_charge_and_state, final_charge_and_state), iupac_name, SMILES, MMFFO, functional, basis_set,
                                                    initial_charge_and_state, final_charge_and_state, [], [], 0,
                                                    None, None, None, None,
                                                    [], [], [], '', '', '',
                                                    None, None,
                                                    None, None,
                                                    None, None])
        self.print_if_debug('Entry sucessfully created.')
        # except:
        #     self.print_if_debug('Entry already exist.')

    def update_entry(self,
                     molecule_id: str,
                     keys: str,
                     values: str):
        self.print_if_debug('\nIn function Results.update_entry():')
        # dataframe = self.pull_results()
        keys.append('Last Updated')
        values.append('%s' % (timestamp()))
        # if _id not in dataframe['_id'].values:
            # raise ValueError('_id not found in database: %s' % (_id))
        if len(keys) != len(values):
            raise ValueError('Length of keys and values does not match.')
```

```
        self._MongoDB.update_one_entry(_id = molecule_id,
                                       keys = keys,
                                       values = values)
        # self.export_to_excel()

#%%

class ReorganisationEnergy():

    def __init__(self,
                 HPC_username: str,
                 HPC_password: str,
                 HPC_hostname: str,
                 MongoDB_username: str,
                 MongoDB_password: str,
                 MongoDB_database: str = 'New_Reorganisation_Energy_Calc',
                 MongoDB_SMILES_database_collection: str = 'SMILES_Database',
                 MongoDB_tracker_collection: str = 'Tracker',
                 MongoDB_results_collection: str = 'Results',
                 MongoDB_connection_string: str = 'mongodb+srv://<user>:<password>@cluster0.wxpvcwv.mongodb.net/?
     retryWrites=true&w=majority&appName=Cluster0',
                 local_folder: str = r'%s' % (os.getcwd()),
                 remote_folder: str = r'/rds/general/ephemeral/user/your_username/ephemeral/test',
                 debug: str = False):

        self._debug = debug
        self._current_working_directory = os.getcwd()

        self._SMILES_database = SMILES_database(username = MongoDB_username,
                                                password = MongoDB_password,
                                                collection = MongoDB_SMILES_database_collection,
                                                database = MongoDB_database,
                                                connection_string = MongoDB_connection_string,
                                                debug = self._debug)

        self._HPC_connection = HPC(username = HPC_username,
                                   password = HPC_password,
                                   hostname = HPC_hostname,
                                   debug = self._debug)

        self._local_folder = local_folder
        self._remote_folder = remote_folder

        self._funnel = Funnel(path = r'%s/Funnel.xlsx' % (local_folder),
                              debug = self._debug)

        self._tracker = Tracker(username = MongoDB_username,
                                password = MongoDB_password,
                                collection = MongoDB_tracker_collection,
                                database = MongoDB_database,
                                connection_string = MongoDB_connection_string,
                                debug = self._debug)

        self._results = Results(username = MongoDB_username,
                                password = MongoDB_password,
                                collection = MongoDB_results_collection,
                                database = MongoDB_database,
                                connection_string = MongoDB_connection_string,
                                debug = self._debug)

    def print_if_debug(self, lines:str):
        if self._debug:
            print(lines)

    def get_tracker(self):
        # self._tracker.export_to_excel(parent_folder = self._local_folder)
        return self._tracker.pull_tracker()

    def get_results(self):
        # self._tracker.export_to_excel(parent_folder = self._local_folder)
        return self._results.pull_results()

    def download_tracker_as_excel(self):
        self._tracker.export_to_excel(parent_folder = self._local_folder)

    def download_results_as_excel(self):
        self._results.export_to_excel(parent_folder = self._local_folder)

    def get_funnel(self):
        return self._funnel.retrieve()

    def get_SMILES_database(self):
        return self._SMILES_database.pull_SMILES_database()

    def get_HPC_qstat(self):
        return self._HPC_connection.formatted_qstat()
```

```
    def add_workflow(self,
                     MMFFO: str = 'Yes',
                     functional: str = 'B3LYP',
                     basis_set_1: str = '6-31G(d)',
                     basis_set_2: str = '6-311+G(d,p)',
                     basis_set_3: str = 'No'):
        self.print_if_debug('\nIn function ReorganisationEnergy.add_workflow():')
        try:
            self._funnel.load_from_excel()
            print('\nPrevious Workflow found and extracted.')
        except:
            pass
        self._funnel.add_workflow(MMFFO = MMFFO,
                                  functional = functional,
                                  basis_set_1 = basis_set_1,
                                  basis_set_2 = basis_set_2,
                                  basis_set_3 = basis_set_3)
        self._funnel.export_to_excel()
        print('\nWorkflow added sucessfully. Current workflow:')
        print(self._funnel.retrieve())

    def add_molecule(self, iupac_name: str, SMILES: str = ''):
        self.print_if_debug('\nIn function ReorganisationEnergy.add_molecule():')
        try:
            self._funnel.load_from_excel()
            print('\nWorkflow:')
            print(self._funnel.retrieve())
        except:
            raise Exception('No workflow found.')
        iupac_name, SMILES = self._SMILES_database.check_against_database(iupac_name = iupac_name,
                                                                          SMILES = SMILES)
        for index in self._funnel.workflow_indices():
            molecule_id = self._tracker.add_entry(iupac_name = iupac_name,
                                                  SMILES = SMILES,
                                                  MMFFO = self._funnel.MMFFO(index = index),
                                                  functional = self._funnel.functional(index = index),
                                                  basis_set_1 = self._funnel.basis_set_1(index = index),
                                                  basis_set_2 = self._funnel.basis_set_2(index = index),
                                                  basis_set_3 = self._funnel.basis_set_3(index = index))
            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)]:
                self._results.create_empty_entry(molecule_id = molecule_id,
                                                 iupac_name = iupac_name,
                                                 SMILES = SMILES,
                                                 MMFFO = self._funnel.MMFFO(index = index),
                                                 functional = self._funnel.functional(index = index),
                                                 basis_set = basis_set,
                                                 initial_charge_and_state = 'C0S0',
                                                 final_charge_and_state = 'C0S1')
                self._results.create_empty_entry(molecule_id = molecule_id,
                                                 iupac_name = iupac_name,
                                                 SMILES = SMILES,
                                                 MMFFO = self._funnel.MMFFO(index = index),
                                                 functional = self._funnel.functional(index = index),
                                                 basis_set = basis_set,
                                                 initial_charge_and_state = 'C0S0',
                                                 final_charge_and_state = 'C1S0')
                self._results.create_empty_entry(molecule_id = molecule_id,
                                                 iupac_name = iupac_name,
                                                 SMILES = SMILES,
                                                 MMFFO = self._funnel.MMFFO(index = index),
                                                 functional = self._funnel.functional(index = index),
                                                 basis_set = basis_set,
                                                 initial_charge_and_state = 'C0S0',
                                                 final_charge_and_state = 'C-1S0')
            molecule = Molecule(molecule_name = iupac_name,
                                molecule_id = molecule_id,
                                SMILES = SMILES,
                                parent_folder = self._local_folder)
            molecule.create_molecule_documentation(allow_aromatic_bond = True)
            self.print_if_debug('Molecule workflow added.')

    def batch_add_molecule(self, filepath: str):
        self.print_if_debug('\nIn function ReorganisationEnergy.batch_add_molecule():')
        dataframe = pd.read_excel(filepath)
        for line in range(0, len(dataframe)):
            iupac_name = dataframe.at[line, 'iupac_name']
            SMILES = dataframe.at[line, 'SMILES']
            self.add_molecule(iupac_name = iupac_name, SMILES = SMILES)
            print('Batch write: %s added.' % (iupac_name))

    def get_entries_from_molecule_id(self, molecule_id: str):
        dataframe = self.get_tracker()
        return dataframe.loc[dataframe['_id'] == '%s' % (molecule_id)]
```

```
    def submit_molecule_for_calculation(self,
                                        input_object,
                                        molecule_id: str,
                                        excitation_state: int,
                                        charge: int,
                                        functional: str,
                                        basisset: str,
                                        number_of_proc: int,
                                        memory: int,
                                        timeout_hr: int,
                                        connectivity: bool = True,
                                        keyword_line_arguments_input: list[[str, str]] = []):
        molecule = Molecule.load_molecule(molecule_id = molecule_id,
                                          parent_folder = self._local_folder,
                                          debug = self._debug)
        molecule.generate_gjf_sh_files(input_object = input_object,
                                       excited_state = excitation_state,
                                       charge = charge,
                                       functional = functional,
                                       basisset = basisset,
                                       number_of_proc = number_of_proc,
                                       memory = memory,
                                       timeout_hr = timeout_hr,
                                       connectivity = connectivity,
                                       keyword_line_arguments_input = keyword_line_arguments_input)
        job_id = self._HPC_connection.submit_job_gaussian(local_folder = r'%s/Data/%s' % (self._local_folder, molecule_id)
    ,
                                                          remote_folder = self._remote_folder,
                                                          filename = r'%s_%s_%s_C%sS%s' % (molecule_id, functional,
    format_basis_set(basisset), charge, excitation_state))
        return job_id

    def restart_molecule_calculation(self,
                                     molecule_id: str,
                                     excitation_state: int,
                                     charge: int,
                                     functional: str,
                                     basisset: str,
                                     number_of_proc: int,
                                     memory: int,
                                     timeout_hr: int,
                                     keyword_line_arguments_input: list[[str, str]] = []):
        molecule = Molecule.load_molecule(molecule_id = molecule_id,
                                          parent_folder = self._local_folder,
                                          debug = self._debug)
        molecule.generate_gjf_sh_restart_files(excited_state = excitation_state,
                                               charge = charge,
                                               functional = functional,
                                               basisset = basisset,
                                               number_of_proc = number_of_proc,
                                               memory = memory,
                                               timeout_hr = timeout_hr,
                                               keyword_line_arguments_input = keyword_line_arguments_input)
        job_id = self._HPC_connection.restart_job_gaussian(local_folder = r'%s/Data/%s' % (self._local_folder, molecule_id
    ),
                                                           remote_folder = self._remote_folder,

                                                           filename = r'%s_%s_%s_C%sS%s' % (molecule_id, functional,
    format_basis_set(basisset), charge, excitation_state))
        return job_id

    def submit_molecule_for_marcus_calculation(self,
                                               molecule_id: str,
                                               charge_geom: int,
                                               charge_eval: int,
                                               state_geom: int,
                                               state_eval: int,
                                               functional: str,
                                               basisset: str,
                                               number_of_proc: int,
                                               memory: int,
                                               timeout_hr: int,
                                               keyword_line_arguments_input: list[[str, str]] = []):
        molecule = Molecule.load_molecule(molecule_id = molecule_id,
                                          parent_folder = self._local_folder,
                                          debug = self._debug)
        molecule.generate_gjf_sh_marcus_files(charge_geom = charge_geom,
                                              charge_eval = charge_eval,
                                              state_geom = state_geom,
                                              state_eval = state_eval,
                                              functional = functional,
                                              basisset = basisset,
                                              number_of_proc = number_of_proc,
                                              memory = memory,
                                              timeout_hr = timeout_hr,
                                              keyword_line_arguments_input = keyword_line_arguments_input)
```

```
        job_id = self._HPC_connection.submit_job_gaussian(local_folder = r'%s/Data/%s' % (self._local_folder, molecule_id)
    ,
                                                           remote_folder = self._remote_folder,
                                                           filename = r'%s_%s_%s_C%sS%s_opt_geom_at_C%sS%s' % (molecule_id,
     functional, format_basis_set(basisset), charge_geom, state_geom, charge_eval, state_eval))
        return job_id

    def transfer_chk_fchk_log_files_from_HPC(self,
                                             molecule_id: str,
                                             charge: int,
                                             excitation_state: int,
                                             functional: str,
                                             basisset: str):
        self._HPC_connection.transfer_chk_fchk_log_files_from_HPC(local_folder = r'%s/Data/%s' % (self._local_folder,
    molecule_id),
                                                                  remote_folder = r'%s' % (self._remote_folder),
                                                                  filename = r'%s_%s_%s_C%sS%s' % (molecule_id, functional
    , format_basis_set(basisset), charge, excitation_state))

    def transfer_chk_fchk_log_marcus_files_from_HPC(self,
                                                    molecule_id: str,
                                                    charge_geom: int,
                                                    charge_eval: int,
                                                    state_geom: int,
                                                    state_eval: int,
                                                    functional: str,
                                                    basisset: str):
        self._HPC_connection.transfer_chk_fchk_log_files_from_HPC(local_folder = r'%s/Data/%s' % (self._local_folder,
    molecule_id),
                                                                  remote_folder = r'%s' % (self._remote_folder),
                                                                  filename = r'%s_%s_%s_C%sS%s_opt_geom_at_C%sS%s' % (
    molecule_id, functional, format_basis_set(basisset), charge_geom, state_geom, charge_eval, state_eval))

    def memory_assignment(self, molecule_id: str):
        SMILES = self.get_entries_from_molecule_id(molecule_id = molecule_id)['SMILES'].values[0]
        molecular_weight = Descriptors.ExactMolWt(Chem.MolFromSmiles(SMILES))
        out: dict = {'number_of_proc': 0,
                     'memory': 0,
                     'timeout_hr': 0}
        if molecular_weight <= 50:                                              # ethene
            out['number_of_proc'] = 8    # 8
            out['memory']         = 24   # 8 * 3GB
            out['timeout_hr']     = 3    # 3 hr
        elif molecular_weight > 50 and molecular_weight <= 100:                 # benzene
            out['number_of_proc'] = 16   # 16
            out['memory']         = 48   # 16 * 3GB
            out['timeout_hr']     = 5    # 7 hr
        elif molecular_weight > 100 and molecular_weight <= 200:                # napthalene, anthracene
            out['number_of_proc'] = 32   # 32
            out['memory']         = 96   # 32 * 3GB
            out['timeout_hr']     = 7   # 10 hr
        elif molecular_weight > 200 and molecular_weight <= 300:                # tetracene, pentacene
            out['number_of_proc'] = 64   # 64
            out['memory']         = 256  # 64 * 4GB
            out['timeout_hr']     = 9   # 15 hr
        elif molecular_weight > 300 and molecular_weight <= 400:                # hexacene, heptacene
            out['number_of_proc'] = 80   # 80
            out['memory']         = 320  # 80 * 4GB
            out['timeout_hr']     = 11   # 23 hr
        elif molecular_weight > 400:                                            # Octacene and above
            out['number_of_proc'] = 90   # 90
            out['memory']         = 360  # 90 * 4GB
            out['timeout_hr']     = 15   # 48 hr
        self.print_if_debug(out)
        return out

    def add_activity_log(self, molecule_id: str, line: str):
        filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
        entry = filtered_line.loc[:, 'Activity Log'].values[0]
        new_entry = entry + '\n' + line
        self._tracker.update_entry(_id = molecule_id, keys = ['Activity Log'], values = [new_entry])
        self.print_if_debug('Activity log updated for %s:' % (molecule_id))
        self.print_if_debug('-> %s' % (line))

    def add_comment(self, user: str, molecule_id: str, line: str):
        filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
        entry = filtered_line.loc[:, 'Comment'].values[0]
        if entry == 'No comment':
            new_entry = '%s: %s' % (user, line)
        else:
            new_entry = '%s\n%s: %s' % (entry, user, line)
        self._tracker.update_entry(_id = molecule_id, keys = ['Comment'], values = [new_entry])
        self.print_if_debug('Comment updated for %s:' % (molecule_id))
        self.print_if_debug('-> %s' % (line))

    def comment_result(self,
```

```
                    _id: str,
                    user: str,
                    string: str):
        results = self.get_results()
        current_comment = results.loc[results['_id'] == _id, 'Comment'].values[0]
        if current_comment != None:
            comment = (current_comment + '\n' +
                    '%s: ' % (user) + string)
        else:
            comment = '%s: ' % (user) + string
        self._results.update_entry(molecule_id = _id,
                                   keys = ['Comment'],
                                   values = [comment])

    def gaussian_normal_termination(self,
                                    molecule_id: str,
                                    functional: str,
                                    basis_set: str,
                                    charge: int,
                                    excitation_state: int):
        # try:
        #     with open(r'%s/Data/%s/%s_%s_%s_C%sS%s.log' % (self._parent_folder, molecule_id, molecule_id, functional,
        #                                                    format_basis_set(basis_set), charge, excitation_state)) as
     file:
        #         word = file.readlines()[-1].split(' ')
        #         if 'Normal' in word:
        #             return True
        #         else:
        #             return False
        # except:
        #     raise Exception('\nUnable to open file:\n %s/Data/%s/%s_%s_%s_C%sS%s.log' % (self._parent_folder,
     molecule_id, molecule_id, functional,
        #                                                                                  format_basis_set(basis_set),
     charge, excitation_state))
        try:
            with open(r'%s/Data/%s/%s_%s_%s_C%sS%s.log' % (self._local_folder, molecule_id, molecule_id, functional,
                                                           format_basis_set(basis_set), charge, excitation_state)) as file
     :
                whole_file = file.read()
                lines = whole_file.split('\n')
                normal_count = 0
                warning_count = 0
                comment_string = ''
                for line in lines:
                    if 'Normal' in line:
                        normal_count += 1
                        self.print_if_debug('\nNormal termination statement %s: %s' % (normal_count, line))
                        comment_string = comment_string = '\nAuto_Generated: %s C%sS%s Normal termination statement %s: %s
     ' % (basis_set, charge, excitation_state, normal_count, line)
                        # self.add_comment(user = 'Auto-Generated',
                        #                  molecule_id = molecule_id,
                        #                  line = '%s C%sS%s Normal termination statement %s: %s' % (basis_set, charge,
     excitation_state, normal_count, line))
                    if '**** Warning!!' in line:
                        warning_count += 1
                        self.print_if_debug('\nWarning statement %s: %s' % (warning_count, line))
                        comment_string = comment_string + '\nAuto_Generated: %s C%sS%s Warning statement %s: %s' % (
     basis_set, charge, excitation_state, warning_count, line)
                        # self.add_comment(user = 'Auto-Generated',
                        #                  molecule_id = molecule_id,
                        #                  line = '%s C%sS%s Warning statement %s: %s' % (basis_set, charge,
     excitation_state, warning_count, line))
                if normal_count == 2:
                    self.print_if_debug('Normal termination count = 2, SUCESSFULL OPT and FREQ calculation.')
                    comment_string = comment_string + '\nAuto-Generated: %s C%sS%s Normal termination count = 2,
     SUCESSFULL OPT and FREQ calculation.' % (basis_set, charge, excitation_state)
                    comment_string = comment_string.replace('\nAuto-Generated: ', '', 1)
                    self.add_comment(user = 'Auto-Generated',
                                     molecule_id = molecule_id,
                                     line = comment_string)
                    return 'Completed'
                elif normal_count == 1:
                    self.print_if_debug('Normal termination count = 1, SUCESSFULL OPT and FAILED FREQ calculation.')
                    comment_string = comment_string + '\nAuto-Generated: %s C%sS%s Normal termination count = 1,
     SUCESSFULL OPT and FAILED FREQ calculation.' % (basis_set, charge, excitation_state)
                    comment_string = comment_string.replace('\nAuto-Generated: ', '', 1)
                    self.add_comment(user = 'Auto-Generated',
                                     molecule_id = molecule_id,
                                     line = comment_string)
                    return 'Failed'
                else:
                    self.print_if_debug('Normal termination count = 0, FAILED OPT and FREQ calculation.')
                    comment_string = comment_string + '\nAuto-Generated: %s C%sS%s Normal termination count = 0, FAILED
     OPT and FREQ calculation.' % (basis_set, charge, excitation_state)
                    comment_string = comment_string.replace('\nAuto-Generated: ', '', 1)
                    self.add_comment(user = 'Auto-Generated',
```

```
                                    molecule_id = molecule_id,
                                    line = comment_string)
                return 'Failed'
        except:
            raise Exception('\nUnable to open file:\n %s/Data/%s/%s_%s_%s_C%sS%s.log' % (self._local_folder, molecule_id, molecule_id, functional,
                                                                                         format_basis_set(basis_set), charge, excitation_state))
    def gaussian_marcus_normal_termination(self,
                                           molecule_id: str,
                                           functional: str,
                                           basis_set: str,
                                           charge_geom: int,
                                           charge_eval: int,
                                           state_geom: int,
                                           state_eval: int):
        try:
            with open(r'%s/Data/%s/%s_%s_%s_C%sS%s_opt_geom_at_C%sS%s.log' % (self._local_folder, molecule_id, molecule_id, functional, format_basis_set(basis_set), charge_geom, state_geom, charge_eval, state_eval)) as file:
                whole_file = file.read()
                lines = whole_file.split('\n')
                normal_count = 0
                warning_count = 0
                comment_string = ''
                for line in lines:
                    if 'Normal' in line:
                        normal_count += 1
                        self.print_if_debug('\nNormal termination statement %s: %s' % (normal_count, line))
                        comment_string = comment_string + '\nAuto-Generated: %s C%sS%s_opt_geom_at_C%sS%s Normal termination statement %s: %s' % (basis_set, charge_geom, state_geom, charge_eval, state_eval, normal_count, line)
                        # self.add_comment(user = 'Auto-Generated',
                        #                  molecule_id = molecule_id,
                        #                  line = '%s C%sS%s_opt_geom_at_C%sS%s Normal termination statement %s: %s' % (basis_set, charge_geom, state_geom, charge_eval, state_eval, normal_count, line))
                    if '**** Warning!!' in line:
                        warning_count += 1
                        self.print_if_debug('\nWarning statement %s: %s' % (warning_count, line))
                        comment_string = comment_string + '\nAuto-Generated: %s C%sS%s_opt_geom_at_C%sS%s Warning statement %s: %s' % (basis_set, charge_geom, state_geom, charge_eval, state_eval, warning_count, line)
                        # self.add_comment(user = 'Auto-Generated',
                        #                  molecule_id = molecule_id,
                        #                  line = '%s C%sS%s_opt_geom_at_C%sS%s Warning statement %s: %s' % (basis_set, charge_geom, state_geom, charge_eval, state_eval, warning_count, line))
                if normal_count == 1:
                    self.print_if_debug('Normal termination count = 1, SUCESSFULL calculation.')
                    comment_string = comment_string + '\nAuto-Generated: %s C%sS%s_opt_geom_at_C%sS%s Normal termination count = 1, SUCESSFULL calculation.' % (basis_set, charge_geom, state_geom, charge_eval, state_eval)
                    comment_string = comment_string.replace('\nAuto-Generated: ', '', 1)
                    self.add_comment(user = 'Auto-Generated',
                                     molecule_id = molecule_id,
                                     line = comment_string)
                    return 'Completed'
                else:
                    self.print_if_debug('Normal termination count = 0, FAILED OPT and FREQ calculation.')
                    comment_string = comment_string + '\nAuto-Generated: %s C%sS%s_opt_geom_at_C%sS%s Normal termination count = 0, FAILED OPT and FREQ calculation.' % (basis_set, charge_geom, state_geom, charge_eval, state_eval)
                    comment_string = comment_string.replace('\nAuto-Generated: ', '', 1)
                    self.add_comment(user = 'Auto-Generated',
                                     molecule_id = molecule_id,
                                     line = comment_string)
                    return 'Failed'
        except:
            raise Exception('\nUnable to open file:\n %s/Data/%s/%s_%s_%s_C%sS%s_opt_geom_at_C%sS%s.log' % (self._local_folder, molecule_id, molecule_id, functional, format_basis_set(basis_set), charge_geom, state_geom, charge_eval, state_eval))

    def query_stage_started(self,
                            molecule_id: str,
                            charge: int,
                            excitation_state: int,
                            basis_set: int) -> bool:
        filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
        entry = filtered_line.loc[:, 'C%sS%s: BS%s' % (charge, excitation_state, basis_set)].values[0]
        if entry == 'Not Started':
            return False
        else:
            return True

    def count_running_optimisations(self, alternative: str = '') -> int:
        tracker = self.get_tracker()[['C0S0: BS1', 'C0S0: BS2', 'C0S0: BS3',
                                      'C0S1: BS1 - C0S0 geometry', 'C0S1: BS2 - C0S0 geometry', 'C0S1: BS3 - C0S0 geometry',
                                      'C1S0: BS1 - C0S0 geometry', 'C1S0: BS2 - C0S0 geometry', 'C1S0: BS3 - C0S0 geometry',
                                      'C-1S0: BS1 - C0S0 geometry', 'C-1S0: BS2 - C0S0 geometry', 'C-1S0: BS3 - C0S0 geometry',
```

```
                                    'C0S1: BS1', 'C0S1: BS2', 'C0S1: BS3',
                                    'C0S0: BS1 - C0S1 geometry', 'C0S0: BS2 - C0S1 geometry', 'C0S0: BS3 - C0S1 geometry
    ',
                                    'C1S0: BS1', 'C1S0: BS2', 'C1S0: BS3',
                                    'C0S0: BS1 - C1S0 geometry', 'C0S0: BS2 - C1S0 geometry', 'C0S0: BS3 - C1S0 geometry
    ',
                                    'C-1S0: BS1', 'C-1S0: BS2', 'C-1S0: BS3',
                                    'C0S0: BS1 - C-1S0 geometry', 'C0S0: BS2 - C-1S0 geometry', 'C0S0: BS3 - C-1S0
    geometry']]
        if alternative == '':
            raise Exception('\nMethod depreciated, use count_running_optimisations_alt() instead.')
            return pd.DataFrame(tracker == 'Started').sum().sum()
        else:
            return pd.DataFrame(tracker == alternative).sum().sum()

    def count_running_optimisations_alt(self, exclude: list[str] = ['Not Started', 'Completed', 'Failed']):
        tracker = self.get_tracker()
        count = len(tracker) * 30
        for i in range(0, len(exclude)):
            count -= self.count_running_optimisations(alternative = exclude[i])
        return count

    def get_tracker_not_started(self):
        tracker = self.get_tracker()
        out = pd.DataFrame([])
        for index in range(0, len(tracker)):
            line = tracker.iloc[[index]]
            filtered_line = line[['C0S0: BS1', 'C0S0: BS2', 'C0S0: BS3',
                                  'C0S1: BS1 - C0S0 geometry', 'C0S1: BS2 - C0S0 geometry', 'C0S1: BS3 - C0S0 geometry',
                                  'C1S0: BS1 - C0S0 geometry', 'C1S0: BS2 - C0S0 geometry', 'C1S0: BS3 - C0S0 geometry',
                                  'C-1S0: BS1 - C0S0 geometry', 'C-1S0: BS2 - C0S0 geometry', 'C-1S0: BS3 - C0S0 geometry'
    ,
                                  'C0S1: BS1', 'C0S1: BS2', 'C0S1: BS3',
                                  'C0S0: BS1 - C0S1 geometry', 'C0S0: BS2 - C0S1 geometry', 'C0S0: BS3 - C0S1 geometry',
                                  'C1S0: BS1', 'C1S0: BS2', 'C1S0: BS3',
                                  'C0S0: BS1 - C1S0 geometry', 'C0S0: BS2 - C1S0 geometry', 'C0S0: BS3 - C1S0 geometry',
                                  'C-1S0: BS1', 'C-1S0: BS2', 'C-1S0: BS3',
                                  'C0S0: BS1 - C-1S0 geometry', 'C0S0: BS2 - C-1S0 geometry', 'C0S0: BS3 - C-1S0 geometry'
    ]]
            if any(filtered_line.isin(['Not Started']).values[0]):
                if len(out) == 0:
                    out = pd.DataFrame(line)
                else:
                    out = pd.concat([out, line])
        return out

    def get_tracker_failed(self):
        tracker = self.get_tracker()
        out = pd.DataFrame([])
        for index in range(0, len(tracker)):
            line = tracker.iloc[[index]]
            filtered_line = line[['C0S0: BS1', 'C0S0: BS2', 'C0S0: BS3',
                                  'C0S1: BS1 - C0S0 geometry', 'C0S1: BS2 - C0S0 geometry', 'C0S1: BS3 - C0S0 geometry',
                                  'C1S0: BS1 - C0S0 geometry', 'C1S0: BS2 - C0S0 geometry', 'C1S0: BS3 - C0S0 geometry',
                                  'C-1S0: BS1 - C0S0 geometry', 'C-1S0: BS2 - C0S0 geometry', 'C-1S0: BS3 - C0S0 geometry'
    ,
                                  'C0S1: BS1', 'C0S1: BS2', 'C0S1: BS3',
                                  'C0S0: BS1 - C0S1 geometry', 'C0S0: BS2 - C0S1 geometry', 'C0S0: BS3 - C0S1 geometry',
                                  'C1S0: BS1', 'C1S0: BS2', 'C1S0: BS3',
                                  'C0S0: BS1 - C1S0 geometry', 'C0S0: BS2 - C1S0 geometry', 'C0S0: BS3 - C1S0 geometry',
                                  'C-1S0: BS1', 'C-1S0: BS2', 'C-1S0: BS3',
                                  'C0S0: BS1 - C-1S0 geometry', 'C0S0: BS2 - C-1S0 geometry', 'C0S0: BS3 - C-1S0 geometry'
    ]]
            if any(filtered_line.isin(['Failed']).values[0]):
                if len(out) == 0:
                    out = pd.DataFrame(line)
                else:
                    out = pd.concat([out, line])
        return out

    def get_tracker_running_only(self):
        tracker = self.get_tracker()
        out = pd.DataFrame()
        for index in range(0, len(tracker)):
            line = tracker.iloc[[index]]
            filtered_line = line[['C0S0: BS1', 'C0S0: BS2', 'C0S0: BS3',
                                  'C0S1: BS1 - C0S0 geometry', 'C0S1: BS2 - C0S0 geometry', 'C0S1: BS3 - C0S0 geometry',
                                  'C1S0: BS1 - C0S0 geometry', 'C1S0: BS2 - C0S0 geometry', 'C1S0: BS3 - C0S0 geometry',
                                  'C-1S0: BS1 - C0S0 geometry', 'C-1S0: BS2 - C0S0 geometry', 'C-1S0: BS3 - C0S0 geometry'
    ,
                                  'C0S1: BS1', 'C0S1: BS2', 'C0S1: BS3',
                                  'C0S0: BS1 - C0S1 geometry', 'C0S0: BS2 - C0S1 geometry', 'C0S0: BS3 - C0S1 geometry',
                                  'C1S0: BS1', 'C1S0: BS2', 'C1S0: BS3',
                                  'C0S0: BS1 - C1S0 geometry', 'C0S0: BS2 - C1S0 geometry', 'C0S0: BS3 - C1S0 geometry',
                                  'C-1S0: BS1', 'C-1S0: BS2', 'C-1S0: BS3',
```

```
                                ’C0S0: BS1 - C-1S0 geometry’, ’C0S0: BS2 - C-1S0 geometry’, ’C0S0: BS3 - C-1S0 geometry’
    ]]
            count = 0
            scan_for = [’Not Started’, ’Completed’, ’Failed’]
            for item in scan_for:
                count += pd.DataFrame(filtered_line == item).sum().sum()
            if count != 12: # start count from 12 and minus, and if more than 0 add, or != 12
                if len(out) == 0:
                    out = pd.DataFrame(line)
                else:
                    out = pd.concat([out, line])
        return out

    def tracker_summarise_running(self):
        running_only = self.get_tracker_running_only()
        scan = [’C0S0: BS1’, ’C0S0: BS2’, ’C0S0: BS3’,
                ’C0S1: BS1 - C0S0 geometry’, ’C0S1: BS2 - C0S0 geometry’, ’C0S1: BS3 - C0S0 geometry’,
                ’C1S0: BS1 - C0S0 geometry’, ’C1S0: BS2 - C0S0 geometry’, ’C1S0: BS3 - C0S0 geometry’,
                ’C-1S0: BS1 - C0S0 geometry’, ’C-1S0: BS2 - C0S0 geometry’, ’C-1S0: BS3 - C0S0 geometry’,
                ’C0S1: BS1’, ’C0S1: BS2’, ’C0S1: BS3’,
                ’C0S0: BS1 - C0S1 geometry’, ’C0S0: BS2 - C0S1 geometry’, ’C0S0: BS3 - C0S1 geometry’,
                ’C1S0: BS1’, ’C1S0: BS2’, ’C1S0: BS3’,
                ’C0S0: BS1 - C1S0 geometry’, ’C0S0: BS2 - C1S0 geometry’, ’C0S0: BS3 - C1S0 geometry’,
                ’C-1S0: BS1’, ’C-1S0: BS2’, ’C-1S0: BS3’,
                ’C0S0: BS1 - C-1S0 geometry’, ’C0S0: BS2 - C-1S0 geometry’, ’C0S0: BS3 - C-1S0 geometry’]
        out = pd.DataFrame(columns = [’molecule_id’, ’run’, ’run_id’])
        for index in range(0, len(running_only)):
            line = running_only.iloc[index, :]
            for item in scan:
                if line[item] != ’Not Started’ and line[item] != ’Completed’ and line[item] != ’Failed’:
                    out.loc[len(out)] = [line[’_id’], item, line[item]]
        return out


    def next_step_for_molecule_id_with_charge_state(self,
                                                    molecule_id: str,
                                                    charge: int,
                                                    excitation_state: int):
        raise Exception(’Depreciated. Use next_step_for_molecule_id_with_basis_set.’)
        # charge_state = ’C%sS%s’ % (charge, excitation_state)
        # filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
        # if filtered_line.loc[:, ’%s: BS1’ % (charge_state)].values[0] == ’Not Started’:
        #     job_parameters = self.memory_assignment(molecule_id = molecule_id)
        #     job_id = self.submit_molecule_for_calculation(input_object = ’’,            # use RDKit geometry
        #                                                   molecule_id = molecule_id,
        #                                                   excitation_state = excitation_state,
        #                                                   charge = charge,
        #                                                   functional = filtered_line.loc[:, ’Functional’].values[0],
        #                                                   basisset = filtered_line.loc[:, ’Basis Set 1’].values[0],
        #                                                   number_of_proc = int(job_parameters[’number_of_proc’]),
        #                                                   memory = int(job_parameters[’memory’]),
        #                                                   timeout_hr = int(job_parameters[’timeout_hr’]),
        #                                                   connectivity = True,
        #                                                   keyword_line_arguments_input = [])
        #     self._tracker.update_entry(_id = molecule_id,
        #                                keys = [’%s: BS1’ % (charge_state)],
        #                                values = [job_id])
        #     self.add_activity_log(molecule_id = molecule_id,
        #                           line = ’%s BS1 optimisation started: %s’ % (charge_state, str(datetime.now()).split
    (’.’)[0]))
        #     self.print_if_debug((’%s %s: BS1 started.’ % (molecule_id, charge_state)))
        #     return True
        # elif filtered_line.loc[:, ’%s: BS1’ % (charge_state)].values[0] == ’Completed’ and filtered_line.loc[:, ’%s: BS2
    ’ % (charge_state)].values[0] == ’Not Started’:
        #     job_parameters = self.memory_assignment(molecule_id = molecule_id)
        #     job_id = self.submit_molecule_for_calculation(input_object = ’%s/Data/%s/%s_%s_%s_%s.log’ % (self.
    _local_folder, molecule_id, molecule_id,
        #                                                                                                  filtered_line.
    loc[:, ’Functional’].values[0],
        #
    format_basis_set(filtered_line.loc[:, ’Basis Set 1’].values[0]),
        #                                                                                                  charge_state),
                # use BS1 geometry
        #                                                   molecule_id = molecule_id,
        #                                                   excitation_state = excitation_state,
        #                                                   charge = charge,
        #                                                   functional = filtered_line.loc[:, ’Functional’].values[0],
        #                                                   basisset = filtered_line.loc[:, ’Basis Set 2’].values[0],
        #                                                   number_of_proc = int(job_parameters[’number_of_proc’]),
        #                                                   memory = int(job_parameters[’memory’]),
        #                                                   timeout_hr = int(job_parameters[’timeout_hr’]),
        #                                                   connectivity = True,
        #                                                   keyword_line_arguments_input = [])
        #     self._tracker.update_entry(_id = molecule_id,
        #                                keys = [’%s: BS2’ % (charge_state)],
        #                                values = [job_id])
```

```
        #     self.add_activity_log(molecule_id = molecule_id,
        #                           line = '%s BS2 optimisation started: %s' % (charge_state, str(datetime.now()).split
('.')[0]))
        #     self.print_if_debug(('%s %s: BS2 started.' % (molecule_id, charge_state)))
        #     return True
        # elif filtered_line.loc[:, '%s: BS2' % (charge_state)].values[0] == 'Completed' and filtered_line.loc[:, '%s: BS3
' % (charge_state)].values[0] == 'Not Started':
        #     job_parameters = self.memory_assignment(molecule_id = molecule_id)
        #     job_id = self.submit_molecule_for_calculation(input_object = '%s/Data/%s/%s_%s_%s_%s.log' % (self.
_local_folder, molecule_id, molecule_id,
        #                                                                                                  filtered_line.
loc[:, 'Functional'].values[0],
        #
format_basis_set(filtered_line.loc[:, 'Basis Set 2'].values[0]),
        #                                                                                                  charge_state),
                # use BS2 geometry
        #                                             molecule_id = molecule_id,
        #                                             excitation_state = excitation_state,
        #                                             charge = charge,
        #                                             functional = filtered_line.loc[:, 'Functional'].values[0],
        #                                             basisset = filtered_line.loc[:, 'Basis Set 2'].values[0],
        #                                             number_of_proc = int(job_parameters['number_of_proc']),
        #                                             memory = int(job_parameters['memory']),
        #                                             timeout_hr = int(job_parameters['timeout_hr']),
        #                                             connectivity = True,
        #                                             keyword_line_arguments_input = [])
        #     self._tracker.update_entry(_id = molecule_id,
        #                                keys = ['%s: BS3' % (charge_state)],
        #                                values = [job_id])
        #     self.add_activity_log(molecule_id = molecule_id,
        #                           line = '%s BS3 optimisation started: %s' % (charge_state, str(datetime.now()).split
('.')[0]))
        #     self.print_if_debug(('%s %s: BS3 started.' % (molecule_id, charge_state)))
        #     return True
        # else:
        #     return False

    def restart_gaussian_for_molecule(self,
                                      molecule_id: str,
                                      basis_set: int,
                                      excitation_state: int,
                                      charge: int):
        filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
        job_parameters = self.memory_assignment(molecule_id = molecule_id)
        job_id = self.restart_molecule_calculation(molecule_id = molecule_id,
                                                   excitation_state = excitation_state,
                                                   charge = charge,
                                                   functional = filtered_line.loc[:, 'Functional'].values[0],
                                                   basisset = filtered_line.loc[:, 'Basis Set %s' % (basis_set)].values
[0],
                                                   number_of_proc = int(job_parameters['number_of_proc']),
                                                   memory = int(job_parameters['memory']),
                                                   timeout_hr = int(job_parameters['timeout_hr']),
                                                   keyword_line_arguments_input = [])
        self._tracker.update_entry(_id = molecule_id,
                                   keys = ['C%sS%s: BS%s' % (charge, excitation_state, basis_set)],
                                   values = [job_id])
        self.add_activity_log(molecule_id = molecule_id,
                              line = 'C%sS%s BS%s optimisation restarted: %s' % (charge, excitation_state, basis_set, str(
datetime.now()).split('.')[0]))
        self.print_if_debug(('%s C%sS%s: BS%s started.' % (charge, excitation_state, molecule_id, basis_set)))

    def restart_failed_cases(self,
                             stop_time = '',
                             max_jobs_parallel: int = 45):
        if stop_time == '':
            stop_time = datetime.now() + timedelta(minutes = 15)
        if datetime.now() >= stop_time:
            return
        current_running = self.count_running_optimisations_alt()
        to_be_run = self.get_tracker_failed().reset_index()
        for index in range(0, len(to_be_run)):
            molecule_id = to_be_run.at[index, '_id']
            filtered_line = to_be_run.loc[to_be_run['_id'] == molecule_id]
            columns = filtered_line.columns[filtered_line.isin(['Failed']).any()].tolist()
            for header in columns:
                if current_running <= max_jobs_parallel:
                    header = header.split(' - ') # 'C0S1: BS3 - opt geom at C0S0'.split(' - ') -->  ['C0S1: BS3', 'opt
geom at C0S0']
                    if len(header) == 1: # only 'C0S0: BS1'
                        charge_state, basis_set = header[0].split(': ') # ['C0S1', 'BS3']
                        basis_set = int(basis_set.replace('BS', ''))
                        # basis_set = filtered_line['Basis Set %s' % (basis_set)].values[0]
                        state = int(charge_state.split('S')[1])
                        charge = int(charge_state.split('S')[0].replace('C', ''))
                        self.restart_gaussian_for_molecule(molecule_id = molecule_id,
```

```
                                                    basis_set = basis_set,
                                                    excitation_state = state,
                                                    charge = charge)
                        current_running += 1
                    else: # ['C0S1: BS3', 'opt geom at C0S0']
                        raise Exception('Not coded')
        self.download_tracker_as_excel()



    def next_step_for_molecule_id_with_basis_set(self,
                                                 molecule_id: str,
                                                 basis_set: int):
        filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
        job_parameters = self.memory_assignment(molecule_id = molecule_id)
        # start ground state ################################################
        if filtered_line.loc[:, 'C0S0: BS%s' % (basis_set)].values[0] == 'Not Started': # start ground state
            charge = 0
            excitation_state = 0
            charge_state = 'C%sS%s' % (charge, excitation_state)
            if basis_set == 1: # basis set 1
                job_id = self.submit_molecule_for_calculation(input_object = '',            # use RDKit geometry
                                                              molecule_id = molecule_id,
                                                              excitation_state = excitation_state,
                                                              charge = charge,
                                                              functional = filtered_line.loc[:, 'Functional'].values[0],
                                                              basisset = filtered_line.loc[:, 'Basis Set %s' % (basis_set)
    ].values[0],
                                                              number_of_proc = int(job_parameters['number_of_proc']),
                                                              memory = int(job_parameters['memory']),
                                                              timeout_hr = int(job_parameters['timeout_hr']),
                                                              connectivity = True,
                                                              keyword_line_arguments_input = [])
            else: # not basis set 1: use geometry of previous basis set optimisation
                previous_basis_set = int(basis_set - 1)
                file_path_string = '%s/Data/%s/%s_%s_%s_%s.log' % (self._local_folder, molecule_id, molecule_id,
                                                                   filtered_line.loc[:, 'Functional'].values[0],
                                                                   format_basis_set(filtered_line.loc[:, 'Basis Set %s' %
    (previous_basis_set)].values[0]),
                                                                   charge_state)
                job_id = self.submit_molecule_for_calculation(input_object = r'%s' % (file_path_string), # use previous
    basis set geometry
                                                              molecule_id = molecule_id,
                                                              excitation_state = excitation_state,
                                                              charge = charge,
                                                              functional = filtered_line.loc[:, 'Functional'].values[0],
                                                              basisset = filtered_line.loc[:, 'Basis Set %s' % (basis_set)
    ].values[0],
                                                              number_of_proc = int(job_parameters['number_of_proc']),
                                                              memory = int(job_parameters['memory']),
                                                              timeout_hr = int(job_parameters['timeout_hr']),
                                                              connectivity = True,
                                                              keyword_line_arguments_input = [])
            self._tracker.update_entry(_id = molecule_id,
                                       keys = ['C0S0: BS%s' % (basis_set)],
                                       values = [job_id])
            self.add_activity_log(molecule_id = molecule_id,
                                  line = 'C0S0 BS%s optimisation started: %s' % (basis_set, str(datetime.now()).split('.')
    [0]))
            self.print_if_debug(('%s C0S0: BS%s started.' % (molecule_id, basis_set)))
            return True
        # C0S0 optimal geometry completed ###################################
        elif filtered_line.loc[:, 'C0S0: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'C0S1: BS%
    s' % (basis_set)].values[0] == 'Not Started':
            # start C0S1 ########################################################
            charge = 0
            excitation_state = 1
            charge_state = 'C%sS%s' % (charge, excitation_state)
            file_path_string = '%s/Data/%s/%s_%s_%s_C0S0.log' % (self._local_folder, molecule_id, molecule_id,
                                                                 filtered_line.loc[:, 'Functional'].values[0],
                                                                 format_basis_set(filtered_line.loc[:, 'Basis Set %s' % (
    basis_set)].values[0]))
            job_id = self.submit_molecule_for_calculation(input_object = r'%s' % (file_path_string), # use C0S0 geometry
                                                          molecule_id = molecule_id,
                                                          excitation_state = excitation_state,
                                                          charge = charge,
                                                          functional = filtered_line.loc[:, 'Functional'].values[0],
                                                          basisset = filtered_line.loc[:, 'Basis Set %s' % (basis_set)].
    values[0],
                                                          number_of_proc = int(job_parameters['number_of_proc']),
                                                          memory = int(job_parameters['memory']),
                                                          timeout_hr = int(job_parameters['timeout_hr']),
                                                          connectivity = True,
                                                          keyword_line_arguments_input = [])
            self._tracker.update_entry(_id = molecule_id,
                                       keys = ['C0S1: BS%s' % (basis_set)],
```

```
                                    values = [job_id])
            self.add_activity_log(molecule_id = molecule_id,
                                  line = 'C0S1 BS%s optimisation started: %s' % (basis_set, str(datetime.now()).split('.')
[0]))
            self.print_if_debug(('%s C0S0: BS%s started.' % (molecule_id, basis_set)))
            return True
        elif filtered_line.loc[:, 'C0S0: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'C1S0: BS%
s' % (basis_set)].values[0] == 'Not Started':
            # start C1S0 ########################################################
            charge = 1
            excitation_state = 0
            charge_state = 'C%sS%s' % (charge, excitation_state)
            file_path_string = '%s/Data/%s/%s_%s_%s_C0S0.log' % (self._local_folder, molecule_id, molecule_id,
                                                                 filtered_line.loc[:, 'Functional'].values[0],
                                                                 format_basis_set(filtered_line.loc[:, 'Basis Set %s' % (
basis_set)].values[0]))
            job_id = self.submit_molecule_for_calculation(input_object = r'%s' % (file_path_string), # use C0S0 geometry
                                                          molecule_id = molecule_id,
                                                          excitation_state = excitation_state,
                                                          charge = charge,
                                                          functional = filtered_line.loc[:, 'Functional'].values[0],
                                                          basisset = filtered_line.loc[:, 'Basis Set %s' % (basis_set)].
values[0],
                                                          number_of_proc = int(job_parameters['number_of_proc']),
                                                          memory = int(job_parameters['memory']),
                                                          timeout_hr = int(job_parameters['timeout_hr']),
                                                          connectivity = True,
                                                          keyword_line_arguments_input = [])
            self._tracker.update_entry(_id = molecule_id,
                                       keys = ['C1S0: BS%s' % (basis_set)],
                                       values = [job_id])
            self.add_activity_log(molecule_id = molecule_id,
                                  line = 'C1S0 BS%s optimisation started: %s' % (basis_set, str(datetime.now()).split('.')
[0]))
            self.print_if_debug(('%s C1S0: BS%s started.' % (molecule_id, basis_set)))
            return True
        elif filtered_line.loc[:, 'C0S0: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'C-1S0: BS
%s' % (basis_set)].values[0] == 'Not Started':
            # start C-1S0 ########################################################
            charge = -1
            excitation_state = 0
            charge_state = 'C%sS%s' % (charge, excitation_state)
            file_path_string = '%s/Data/%s/%s_%s_%s_C0S0.log' % (self._local_folder, molecule_id, molecule_id,
                                                                 filtered_line.loc[:, 'Functional'].values[0],
                                                                 format_basis_set(filtered_line.loc[:, 'Basis Set %s' % (
basis_set)].values[0]))
            job_id = self.submit_molecule_for_calculation(input_object = r'%s' % (file_path_string), # use C0S0 geometry
                                                          molecule_id = molecule_id,
                                                          excitation_state = excitation_state,
                                                          charge = charge,
                                                          functional = filtered_line.loc[:, 'Functional'].values[0],
                                                          basisset = filtered_line.loc[:, 'Basis Set %s' % (basis_set)].
values[0],
                                                          number_of_proc = int(job_parameters['number_of_proc']),
                                                          memory = int(job_parameters['memory']),
                                                          timeout_hr = int(job_parameters['timeout_hr']),
                                                          connectivity = True,
                                                          keyword_line_arguments_input = [])
            self._tracker.update_entry(_id = molecule_id,
                                       keys = ['C-1S0: BS%s' % (basis_set)],
                                       values = [job_id])
            self.add_activity_log(molecule_id = molecule_id,
                                  line = 'C-1S0 BS%s optimisation started: %s' % (basis_set, str(datetime.now()).split('.'
)[0]))
            self.print_if_debug(('%s C-1S0: BS%s started.' % (molecule_id, basis_set)))
            return True
        elif filtered_line.loc[:, 'C0S0: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'C0S1: BS%
s - C0S0 geometry' % (basis_set)].values[0] == 'Not Started':
            # start C0S0 optimal geometry at C0S1 energy ######################
            job_id = self.submit_molecule_for_marcus_calculation(molecule_id = molecule_id,
                                                                 charge_geom = 0,
                                                                 charge_eval = 0,
                                                                 state_geom = 0,
                                                                 state_eval = 1,
                                                                 functional = filtered_line.loc[:, 'Functional'].values
[0],
                                                                 basisset = filtered_line.loc[:, 'Basis Set %s' % (
basis_set)].values[0],
                                                                 number_of_proc = int(job_parameters['number_of_proc']),
                                                                 memory = int(job_parameters['memory']),
                                                                 timeout_hr = int(job_parameters['timeout_hr']),
                                                                 keyword_line_arguments_input = [])
            self._tracker.update_entry(_id = molecule_id,
                                       keys = ['C0S1: BS%s - C0S0 geometry' % (basis_set)],
                                       values = [job_id])
            self.add_activity_log(molecule_id = molecule_id,
```

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

```
                                                                functional = filtered_line.loc[:, 'Functional'].values
    [0],
                                                                basisset = filtered_line.loc[:, 'Basis Set %s' % (
    basis_set)].values[0],
                                                                number_of_proc = int(job_parameters['number_of_proc']),
                                                                memory = int(job_parameters['memory']),
                                                                timeout_hr = int(job_parameters['timeout_hr']),
                                                                keyword_line_arguments_input = [])
            self._tracker.update_entry(_id = molecule_id,
                                       keys = ['C0S0: BS%s - C1S0 geometry' % (basis_set)],
                                       values = [job_id])
            self.add_activity_log(molecule_id = molecule_id,
                                  line = 'C0S0: BS%s - C1S0 geometry calculation started: %s' % (basis_set, str(datetime.
    now()).split('.')[0]))
            self.print_if_debug(('%s C0S0: BS%s - C1S0 geometry started.' % (molecule_id, basis_set)))
            return True
        # C-1S0 optimal geometry completed ###################################
        elif filtered_line.loc[:, 'C-1S0: BS%s' % (basis_set)].values[0] == 'Completed' and filtered_line.loc[:, 'C0S0: BS
    %s - C-1S0 geometry' % (basis_set)].values[0] == 'Not Started':
            # start C-1S0 optimal geometry at C0S0 energy ####################
            job_id = self.submit_molecule_for_marcus_calculation(molecule_id = molecule_id,
                                                                charge_geom = -1,
                                                                charge_eval = 0,
                                                                state_geom = 0,
                                                                state_eval = 0,
                                                                functional = filtered_line.loc[:, 'Functional'].values
    [0],
                                                                basisset = filtered_line.loc[:, 'Basis Set %s' % (
    basis_set)].values[0],
                                                                number_of_proc = int(job_parameters['number_of_proc']),
                                                                memory = int(job_parameters['memory']),
                                                                timeout_hr = int(job_parameters['timeout_hr']),
                                                                keyword_line_arguments_input = [])
            self._tracker.update_entry(_id = molecule_id,
                                       keys = ['C0S0: BS%s - C-1S0 geometry' % (basis_set)],
                                       values = [job_id])
            self.add_activity_log(molecule_id = molecule_id,
                                  line = 'C0S0: BS%s - C-1S0 geometry calculation started: %s' % (basis_set, str(datetime.
    now()).split('.')[0]))
            self.print_if_debug(('%s C0S0: BS%s - C-1S0 geometry started.' % (molecule_id, basis_set)))
            return True
        else:
            return False


    def next_step_for_molecule_id(self, molecule_id: str):
        filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
        if len(filtered_line) == 0:
            raise Exception('molecule_id not found in tracker.')
        job_submitted: bool = False
        # basis set 1
        if filtered_line.loc[:, 'C0S0: BS1 - C-1S0 geometry'].values[0] != 'Completed':
            job_submitted = self.next_step_for_molecule_id_with_basis_set(molecule_id = molecule_id,
                                                                          basis_set = 1)
        # basis set 2
        if job_submitted == False and (filtered_line.loc[:, 'C0S0: BS2 - C-1S0 geometry'].values[0] != 'Completed' or
     filtered_line.loc[:, 'C0S0: BS2 - C1S0 geometry'].values[0] != 'Completed' or filtered_line.loc[:, 'C0S0: BS2 - C0S1
     geometry'].values[0] != 'Completed') and filtered_line.loc[:, 'C0S0: BS1'].values[0] == 'Completed':
            job_submitted = self.next_step_for_molecule_id_with_basis_set(molecule_id = molecule_id,
                                                                          basis_set = 2)
        # basis set 3
        if job_submitted == False and (filtered_line.loc[:, 'C0S0: BS3 - C-1S0 geometry'].values[0] != 'Completed' or
     filtered_line.loc[:, 'C0S0: BS3 - C1S0 geometry'].values[0] != 'Completed' or filtered_line.loc[:, 'C0S0: BS3 - C0S1
     geometry'].values[0] != 'Completed') and filtered_line.loc[:, 'C0S0: BS2'].values[0] == 'Completed':
            job_submitted = self.next_step_for_molecule_id_with_basis_set(molecule_id = molecule_id,
                                                                          basis_set = 3)
        if job_submitted == True:
            return True
        else:
            return False

    def check_qstat_and_transfer_files(self,
                                       stop_time = ''):
        if stop_time == '':
            stop_time = datetime.now() + timedelta(minutes = 15)
        else:
            stop_time = datetime.now() + timedelta(minutes = stop_time)
        if datetime.now() >= stop_time:
            return
        qstat = self.get_HPC_qstat()
        running_only = self.tracker_summarise_running()
        for i in range(0, len(running_only)):
            if datetime.now() >= stop_time:
                return
            if str(running_only.at[i, 'run_id']) not in qstat['Job ID'].values:
                molecule_id = running_only.at[i, 'molecule_id']
```

```
                filtered_line = self.get_entries_from_molecule_id(molecule_id = molecule_id)
                charge_state, basis_set = running_only.at[i, 'run'].split(': ')
                charge = charge_state.split('S')[0].replace('C', '')
                excitation_state = charge_state.split('S')[1]
                if basis_set == 'BS1' or basis_set == 'BS2' or basis_set == 'BS3':
                    if basis_set == 'BS1':
                        basis_set = 'Basis Set 1'
                    elif basis_set == 'BS2':
                        basis_set = 'Basis Set 2'
                    elif basis_set == 'BS3':
                        basis_set = 'Basis Set 3'
                    print(charge_state, basis_set, charge, excitation_state)
                    self.transfer_chk_fchk_log_files_from_HPC(molecule_id = molecule_id,
                                                              charge = charge,
                                                              excitation_state = excitation_state,
                                                              functional = filtered_line['Functional'].values[0],
                                                              basisset = filtered_line[basis_set].values[0])
                    convergence = self.gaussian_normal_termination(molecule_id = molecule_id,
                                                                   functional = filtered_line['Functional'].values[0],
                                                                   basis_set = format_basis_set(filtered_line['%s' % (
basis_set)].values[0]),
                                                                   charge = charge,
                                                                   excitation_state = excitation_state)
                    self._tracker.update_entry(_id = molecule_id,
                                               keys = ['%s' % (running_only.at[i, 'run'])],
                                               values = [convergence])
                    self.add_activity_log(molecule_id = molecule_id,
                                          line = '%s BS%s files transferred from HPC' % (charge_state, basis_set.split(' '
)[-1]))
                else:
                    geom_charge_state = basis_set.split(' - ')[1].split(' ')[0]
                    basis_set = basis_set.split(' - ')[0]
                    if basis_set == 'BS1':
                        basis_set = 'Basis Set 1'
                    elif basis_set == 'BS2':
                        basis_set = 'Basis Set 2'
                    elif basis_set == 'BS3':
                        basis_set = 'Basis Set 3'
                    charge_eval = charge
                    state_eval = excitation_state
                    charge_geom = geom_charge_state.split('S')[0].replace('C', '')
                    state_geom = geom_charge_state.split('S')[1]
                    self.transfer_chk_fchk_log_marcus_files_from_HPC(molecule_id = molecule_id,
                                                                     charge_geom = charge_geom,
                                                                     charge_eval = charge_eval,
                                                                     state_geom = state_geom,
                                                                     state_eval = state_eval,
                                                                     functional = filtered_line['Functional'].values[0],
                                                                     basisset = filtered_line[basis_set].values[0])
                    convergence = self.gaussian_marcus_normal_termination(molecule_id = molecule_id,
                                                                           functional = filtered_line['Functional'].values
[0],
                                                                           basis_set = filtered_line[basis_set].values[0],
                                                                           charge_geom = charge_geom,
                                                                           charge_eval = charge_eval,
                                                                           state_geom = state_geom,
                                                                           state_eval = state_eval)
                    self._tracker.update_entry(_id = molecule_id,
                                               keys = ['%s' % (running_only.at[i, 'run'])],
                                               values = [convergence])
                    self.add_activity_log(molecule_id = molecule_id,
                                          line = '%s BS%s at C%sS%s geometry files transferred from HPC' % (charge_state,
charge_geom, state_geom, basis_set.split(' ')[-1]))
        # self.get_tracker()

    def automate_submissions(self,
                             stop_time = '',
                             max_jobs_parallel: int = 40):
        if stop_time == '':
            stop_time = datetime.now() + timedelta(minutes = 15)
        if datetime.now() >= stop_time:
            return
        current_running = self.count_running_optimisations_alt()
        to_be_run = self.get_tracker_not_started().reset_index()
        for index in range(0, len(to_be_run)):
            molecule_id = to_be_run.at[index, '_id']
            job_submitted = True
            while job_submitted == True and current_running <= max_jobs_parallel:
                job_submitted = self.next_step_for_molecule_id(molecule_id = molecule_id)
                if job_submitted == True:
                    current_running += 1
                if datetime.now() >= stop_time:
                    # self.download_tracker_as_excel()
                    # raise Exception('\nSubmission terminated automatically, stop_time acheived. Total running currently
= %s.' % (self.count_running_optimisations_alt()))
                    return
```

```
        self.print_if_debug('\nTotal running currently = %s' % (current_running))
        self.download_tracker_as_excel()

    def automate_submission_for_period_of_time(self,
                                               total_time_minutes: int = 10,
                                               pause_between_runs_minutes: int = 2,
                                               max_jobs_parallel: int = 40,
                                               jobs_reserved_for_restart: int = 3):
        time_end = datetime.now() + timedelta(minutes = total_time_minutes)
        run = True
        while datetime.now() <= time_end - timedelta(minutes = pause_between_runs_minutes) and run == True:
            with open(r'%s/Currently Running.txt' % (self._local_folder), 'wt') as file:
                string = 'Started: %s\nEnding: %s' % (datetime.now(), time_end)
                file.write(string)
                file.close()
            if len(self.get_tracker_not_started()) == 0:
                run = False
            self.check_qstat_and_transfer_files(stop_time = time_end)
            self.automate_submissions(stop_time = time_end,
                                      max_jobs_parallel = max_jobs_parallel - jobs_reserved_for_restart)
            #self.restart_failed_cases(stop_time = time_end,
            #                          max_jobs_parallel = max_jobs_parallel)
            self.update_results_all(stop_time = time_end)
            if datetime.now() <= time_end - timedelta(minutes = pause_between_runs_minutes):
                pd.set_option('display.expand_frame_repr', False)
                print(test_reorg_e.get_HPC_qstat())
                pd.set_option('display.expand_frame_repr', True)
                print('\nEntering sleep mode for %s minutes.' % (pause_between_runs_minutes))
                print('Safe to terminate kernal. Program ending at %s.' % (time_end))
                wait_time = pause_between_runs_minutes * 60
                sleep_start = datetime.now()
                for i in tqdm(range(0, int(wait_time/5)), position = 0, leave = True):
                    while datetime.now() < sleep_start + timedelta(seconds = i * 5):
                        time.sleep(5)
        self.download_tracker_as_excel()
        print('\nSubmission terminated automatically. Total running currently = %s.' % (self.
    count_running_optimisations_alt()))

    def get_orbitals(self,
                     molecule_id: str,
                     filename: str):
        with open(r'%s/Data/%s/%s.log' % (self._local_folder, molecule_id, filename.replace('.log', '').replace('fchk', ''
    ).replace('chk', '')), 'r') as file:
            whole_file = file.read()
            lines = whole_file.split('\n')
            number_of_lines_in_whole_file = len(lines)
            population_analysis_start_line = 0
            population_analysis_end_line = 0

            found_start = False
            current_line = number_of_lines_in_whole_file
            while found_start == False:
                current_line -= 1
                if 'Population analysis using the SCF Density.' in lines[current_line]:
                    found_start = True
                    population_analysis_start_line = current_line - 2

            found_end = False
            current_line = population_analysis_start_line + 5
            while found_end == False:
                current_line += 1
                if 'Alpha' in lines[current_line]:
                    population_analysis_end_line = current_line
                    # print(lines[current_line])
                else:
                    found_end = True

            alpha_occ_eigenvalues = []
            alpha_virt_eigenvalues = []

            parsed_lines = lines[population_analysis_start_line:population_analysis_end_line]
            parsed_lines_txt = '\n'.join(parsed_lines)

            for line in parsed_lines:
                if 'Alpha  occ. eigenvalues --' in line:
                    line = line.replace(' Alpha  occ. eigenvalues --  ', '').split(' ')
                    for energy in line:
                        if energy != '':
                            alpha_occ_eigenvalues.append(float(energy))
                if 'Alpha virt. eigenvalues --   ' in line:
                    line = line.replace('Alpha virt. eigenvalues --  ', '').split(' ')
                    for energy in line:
                        if energy != '':
                            alpha_virt_eigenvalues.append(float(energy))
            alpha_occ_eigenvalues = list(np.flip(alpha_occ_eigenvalues))
            alpha_virt_eigenvalues = alpha_virt_eigenvalues
```

```
        HOMO_LUMO_gap = float(alpha_virt_eigenvalues[0]) - float(alpha_occ_eigenvalues[0])

        return alpha_occ_eigenvalues, alpha_virt_eigenvalues, HOMO_LUMO_gap, parsed_lines_txt

    def get_SCF_energy_one(self,
                           molecule_id: str,
                           filename:str):
        with open(r"%s/Data/%s/%s.log" % (self._local_folder, molecule_id, filename.replace('.log', '').replace('fchk', '').replace('chk', '')), 'r') as file:
            whole_file = file.read()
            lines = whole_file.split('\n')
            lines_to_print = 0
            continue_printing_until_blank = False
            summary = ''
            for line in lines:
                if 'SCF Done' in line:
                    # lines_to_print = 1
                    continue_printing_until_blank = True
                    final_SCF_energy = float(line.split('=')[1].split('A.U.')[0].replace(' ', ''))
                if 'Cycle  ' in line:
                    continue_printing_until_blank = True
                if 'Item               Value     Threshold  Converged?' in line:
                    lines_to_print = 5
                if lines_to_print > 0:
                    summary += line + '\n'
                    lines_to_print -= 1
                    if lines_to_print == 0:
                        summary += '\n'
                elif continue_printing_until_blank:
                    if line == '':
                        continue_printing_until_blank = False
                        summary += '\n'
                    else:
                        summary += line + '\n'
            self.print_if_debug(summary)

            warning = ''
            if 'Item               Value     Threshold  Converged?' in summary:
                convergence = []
                parsed_summary = summary.split('Item               Value     Threshold  Converged?\n')[-1]
                for line in parsed_summary.split('\n'):
                    if line != '':
                        word = line.split('     ')[-1].replace(' ', '')
                        convergence.append(word)
                self.print_if_debug(convergence)
                if convergence == ['YES', 'YES', 'YES', 'YES']:
                    self.print_if_debug('convergence acheived')
                else:
                    # raise Exception('\nConvergence not acheived:\n%s\n%s' % ('Item; Value; Threshold; Converged?', parsed_summary))
                    warning = '\n%s\nConvergence not acheived:\n%s\n%s' % (filename.replace('.log', '').replace('fchk', '').replace('chk', ''), 'Item; Value; Threshold; Converged?', parsed_summary)
                    warnings.warn(warning)
                    summary = 'WARNING: CONVERGENCE NOT ACHEIVED\n' + summary

            self.print_if_debug('Final SCF energy = %s' % (final_SCF_energy))
            file.close()
        return final_SCF_energy, summary, warning

    def get_SCF_energy_excited_one(self,
                                   molecule_id: str,
                                   filename:str):
        with open(r"%s/Data/%s/%s.log" % (self._local_folder, molecule_id, filename.replace('.log', '').replace('fchk', '').replace('chk', '')), 'r') as file:
            whole_file = file.read()
            lines = whole_file.split('\n')
            lines_to_print = 0
            continue_printing_until_blank = False
            summary = ''
            for line in lines:
                if 'Excited State   1' in line:
                    # lines_to_print = 1
                    continue_printing_until_blank = True
                if 'Total Energy, E(TD-HF/TD-DFT)' in line:
                    final_SCF_energy = float(line.split('=')[1].replace(' ', ''))
                if 'Cycle  ' in line:
                    continue_printing_until_blank = True
                if 'Item               Value     Threshold  Converged?' in line:
                    lines_to_print = 5
                if lines_to_print > 0:
                    summary += line + '\n'
                    lines_to_print -= 1
                    if lines_to_print == 0:
                        summary += '\n'
                elif continue_printing_until_blank:
```

```
                    if line == '':
                        continue_printing_until_blank = False
                        summary += '\n'
                    else:
                        summary += line + '\n'
            self.print_if_debug(summary)

            warning = ''
            if 'Item               Value     Threshold  Converged?' in summary:
                convergence = []
                parsed_summary = summary.split('Item               Value     Threshold  Converged?\n')[-1]
                for line in parsed_summary.split('\n'):
                    if line != '':
                        word = line.split('     ')[-1].replace(' ', '')
                        convergence.append(word)
                self.print_if_debug(convergence)
                if convergence == ['YES', 'YES', 'YES', 'YES']:
                    self.print_if_debug('convergence acheived')
                else:
                    # raise Exception('\nConvergence not acheived:\n%s\n%s' % ('Item; Value; Threshold; Converged?', parsed_summary))
                    warning = '\n%s\nConvergence not acheived:\n%s\n%s' % (filename.replace('.log', '').replace('fchk', '').replace('chk', ''), 'Item; Value; Threshold; Converged?', parsed_summary)
                    warnings.warn(warning)
                    summary = 'WARNING: CONVERGENCE NOT ACHEIVED\n' + summary

            self.print_if_debug('Final SCF energy = %s' % (final_SCF_energy))
            file.close()
        return final_SCF_energy, summary, warning


    def get_SCF_energies(self,
                         molecule_id:str,
                         initial_charge_state: str,
                         final_charge_state: str,
                         formatted_functional_basisset: str):
        if final_charge_state != 'COS1':
            initial_SCF_at_initial_geom, summary1, warning1 = self.get_SCF_energy_one(molecule_id = molecule_id,
                                                                                      filename = '%s_%s_%s' % (molecule_id, formatted_functional_basisset, initial_charge_state))
            final_SCF_at_final_geom, summary2, warning2 = self.get_SCF_energy_one(molecule_id = molecule_id,
                                                                                  filename = '%s_%s_%s' % (molecule_id, formatted_functional_basisset, final_charge_state))
            initial_SCF_at_final_geom, summary3, warning3 = self.get_SCF_energy_one(molecule_id = molecule_id,
                                                                                    filename = '%s_%s_%s_opt_geom_at_%s' % (molecule_id, formatted_functional_basisset, final_charge_state, initial_charge_state))
            final_SCF_at_initial_geom, summary4, warning4 = self.get_SCF_energy_one(molecule_id = molecule_id,
                                                                                    filename = '%s_%s_%s_opt_geom_at_%s' % (molecule_id, formatted_functional_basisset, initial_charge_state, final_charge_state))
        else:
            initial_SCF_at_initial_geom, summary1, warning1 = self.get_SCF_energy_one(molecule_id = molecule_id,
                                                                                      filename = '%s_%s_%s' % (molecule_id, formatted_functional_basisset, initial_charge_state))
            final_SCF_at_final_geom, summary2, warning2 = self.get_SCF_energy_excited_one(molecule_id = molecule_id,
                                                                                          filename = '%s_%s_%s' % (molecule_id, formatted_functional_basisset, final_charge_state))
            initial_SCF_at_final_geom, summary3, warning3 = self.get_SCF_energy_one(molecule_id = molecule_id,
                                                                                    filename = '%s_%s_%s_opt_geom_at_%s' % (molecule_id, formatted_functional_basisset, final_charge_state, initial_charge_state))
            final_SCF_at_initial_geom, summary4, warning4 = self.get_SCF_energy_excited_one(molecule_id = molecule_id,
                                                                                            filename = '%s_%s_%s_opt_geom_at_%s' % (molecule_id, formatted_functional_basisset, initial_charge_state, final_charge_state))
            pass

        summary = ('initial_SCF_at_initial_geom\n' +
                   '\n' +
                   summary1 + '\n' +
                   'final_SCF_at_final_geom\n' +
                   '\n' +
                   summary2 + '\n' +
                   'initial_SCF_at_final_geom\n' +
                   '\n' +
                   summary3 + '\n' +
                   'final_SCF_at_initial_geom\n' +
                   '\n' +
                   summary4 + '\n')
        warning = ''
        for warn in [warning1, warning2, warning3, warning4]:
            if warn != '':
                warning += warn + '\n'


        return initial_SCF_at_initial_geom, initial_SCF_at_final_geom, final_SCF_at_initial_geom, final_SCF_at_final_geom, summary, warning

    def do_DUSHIN_calculation(self,
```

```
                            molecule_id: str,
                            filename_initial_state: str,
                            filename_final_state: str):
        self._HPC_connection.copy_fchk_log_files_for_dushin(remote_dushin_folder = r'%s/Logs_for_dushin/' % (self._remote_folder),
                                                            remote_log_folder = self._remote_folder,
                                                            filename_initial_state = filename_initial_state,
                                                            filename_final_state = filename_final_state)
        initial_state = filename_initial_state.split('_')[-1]
        final_state = filename_final_state.split('_')[-1]
        if filename_initial_state.replace('_%s' % (initial_state), '') == filename_final_state.replace('_%s' % (final_state), ''):
            mol_id_functional_basisset = filename_initial_state.replace('_%s' % (initial_state), '')
        else:
            raise Exception('\nIssue with mol_id_functional_basisset.')
        self._HPC_connection.submit_job_dushin(remote_dushin_folder = r'%s/Logs_for_dushin/' % (self._remote_folder),
                                               remote_bin_path = r'%s/Logs_for_dushin/' % (self._remote_folder),
                                               initial_state = initial_state,
                                               final_state = final_state,
                                               mol_id_functional_basisset = mol_id_functional_basisset)
        self._HPC_connection.transfer_from_HPC(local_folder = r'%s/Data/%s/%s_%s/' % (self._local_folder, molecule_id, initial_state, final_state),
                                               remote_folder = r'%s/Logs_for_dushin/' % (self._remote_folder),
                                               filename = 'dushin%s_%s_%s.log' % (mol_id_functional_basisset, initial_state, final_state))
        with open(r'%s/Data/%s/%s_%s/dushin%s_%s_%s.log' % (self._local_folder, molecule_id, initial_state, final_state, mol_id_functional_basisset, initial_state, final_state), 'r') as file:
            dushin_output = file.readlines()
            split_index_top = dushin_output.index(" Displacement: in terms of nc of 1 THEN of nc of 2\n") + 2
            dushin_output = dushin_output[split_index_top::]
            split_index_bot = dushin_output.index("\n")
            dushin_output = dushin_output[0:split_index_bot]
            if (dushin_output[-1] != " "):
                dushin_output = dushin_output[:-1]
            dushin_results = pd.DataFrame(columns = ['Freq', 'Q/cm', 'lam/eV'])
            for line in dushin_output:
                # print(line)
                freq1 = float(line.split('freq=')[1].split('Q')[0].replace(' ', ''))
                freq2 = float(line.split('freq=')[2].split('Q')[0].replace(' ', ''))
                Q1 = float(line.split('Q=')[1].split('lam')[0].replace(' ', ''))
                Q2 = float(line.split('Q=')[2].split('lam')[0].replace(' ', ''))
                lam1 = float(line.split('lam=')[1].split('freq=')[0].replace(' ', '')) * constants.physical_constants['inverse meter-electron volt relationship'][0] * 100
                lam2 = float(line.split('lam=')[2].split('\n')[0].replace(' ', '')) * constants.physical_constants['inverse meter-electron volt relationship'][0] * 100
                if freq1 != 0 or Q1 != 0 or lam1 != 0:
                    dushin_results.loc[len(dushin_results)] = [freq1, Q1, lam1]
                if freq2 != 0 or Q2 != 0 or lam2 != 0:
                    dushin_results.loc[len(dushin_results)] = [freq2, Q2, lam2]
        dushin_summarised = dushin_results.sort_values('Freq', axis = 'index').reset_index(drop = True)
        dushin_summarised.to_excel(r'%s/Data/%s/%s_%s/dushin%s_%s_%s.xlsx' % (self._local_folder, molecule_id, initial_state, final_state, mol_id_functional_basisset, initial_state, final_state))
        self.print_if_debug(dushin_summarised)
        imaginary_frequencies = dushin_results[['Freq']].lt(0).any(axis = 'index').values[0]
        return dushin_summarised, imaginary_frequencies

    def get_atoms_summary(self,
                          molecule_id: str,
                          filename: str):
        summary = pd.DataFrame(columns = ['Atomic Mass', 'Atomic Number', 'Coordinates', 'Charge', 'Number of Bonds', 'Hybridisation'])
        mol = next(pybel.readfile("log", r"%s/Data/%s/%s.log" % (self._local_folder, molecule_id, filename.replace('.log', '').replace('fchk', '').replace('chk', ''))))
        for entry in mol.atoms:
            self.print_if_debug([entry.atomicmass, entry.atomicnum, entry.coords, entry.formalcharge,  entry.degree, entry.hyb])
            summary.loc[len(summary)] = [entry.atomicmass, entry.atomicnum, entry.coords, entry.formalcharge,  entry.degree, entry.hyb]
        del mol
        return summary

    def organise_results_one(self,
                             molecule_id: str,
                             # new_entry: bool = False,
                             download_results: bool = False,
                             tracker = '',
                             results = '',
                             basis_set_list: list = [1, 2, 3],
                             transition_list: list = ['C0S0_C0S1', 'C0S0_C1S0', 'C0S0_C-1S0'],
                             overwrite_: bool = False):
        if type(tracker) == str:
            tracker = self.get_entries_from_molecule_id(molecule_id = molecule_id)
        else:
            tracker = tracker.loc[tracker['_id'] == molecule_id]
        if type(results) == str:
            results = self.get_results()
```

```
        else:
            tracker = tracker.loc[tracker['_id'] == molecule_id]
        for basis_set in basis_set_list:
            for transitions in transition_list:
                keys = []
                values = []
                initial = transitions.split('_')[0]
                final = transitions.split('_')[1]
                sub_directory = r'%s/Data/%s/%s/' % (self._local_folder, molecule_id, transitions)
                check_directory(r'%s' % (sub_directory))

                if overwrite_ == False:
                    if results.loc[results['_id'] == '%s_%s_%s_%s' % (molecule_id, tracker.loc[tracker['_id'] == molecule_id, 'Functional'].values[0], format_basis_set(tracker.loc[tracker['_id'] == molecule_id, 'Basis Set %s' % (basis_set)].values[0]), transitions), 'Last Updated'].values[0] == None:
                        if (tracker['%s: BS%s' % (initial, basis_set)].values[0] == 'Completed') and (tracker['%s: BS%s' % (final, basis_set)].values[0] == 'Completed') and (tracker['%s: BS%s - %s geometry' % (final, basis_set, initial)].values[0] == 'Completed') and (tracker['%s: BS%s - %s geometry' % (initial, basis_set, final)].values[0] == 'Completed'):
                            run = True
                        else:
                            run = False
                    elif tracker.loc[tracker['_id'] == molecule_id, 'Last Updated'].values[0] > results.loc[results['_id'] == '%s_%s_%s_%s' % (molecule_id, tracker.loc[tracker['_id'] == molecule_id, 'Functional'].values[0], format_basis_set(tracker.loc[tracker['_id'] == molecule_id, 'Basis Set %s' % (basis_set)].values[0]), transitions), 'Last Updated'].values[0]:
                        if (tracker['%s: BS%s' % (initial, basis_set)].values[0] == 'Completed') and (tracker['%s: BS%s' % (final, basis_set)].values[0] == 'Completed') and (tracker['%s: BS%s - %s geometry' % (final, basis_set, initial)].values[0] == 'Completed') and (tracker['%s: BS%s - %s geometry' % (initial, basis_set, final)].values[0] == 'Completed'):
                            run = True
                        else:
                            run = False
                    else:
                        run = False
                else:
                    if (tracker['%s: BS%s' % (initial, basis_set)].values[0] == 'Completed') and (tracker['%s: BS%s' % (final, basis_set)].values[0] == 'Completed') and (tracker['%s: BS%s - %s geometry' % (final, basis_set, initial)].values[0] == 'Completed') and (tracker['%s: BS%s - %s geometry' % (initial, basis_set, final)].values[0] == 'Completed'):
                        run = True
                    else:
                        run = False

                if run == True:
                    print(r'%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), final))
                    # Orbitals and HOMO-LUMO handling
                    alpha_occ_eigenvalues, alpha_virt_eigenvalues, HOMO_LUMO_gap, parsed_lines_txt = self.get_orbitals(molecule_id = molecule_id, filename = r'%s_%s_%s_COSO' % (molecule_id, tracker['Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0])))
                    with open(r'%s/GS_Orbitals.txt' % (sub_directory), 'wt') as file:
                        file.write(parsed_lines_txt)
                        file.close()
                    keys.append('Alpha Occ Orbitals List')
                    values.append(alpha_occ_eigenvalues)
                    keys.append('Alpha Virt Orbitals List')
                    values.append(alpha_virt_eigenvalues)
                    keys.append('HOMO-LUMO Gap')
                    values.append(HOMO_LUMO_gap * constants.physical_constants['Hartree energy in eV'][0])

                    # SCF energies
                    initial_SCF_at_initial_geom, initial_SCF_at_final_geom, final_SCF_at_initial_geom, final_SCF_at_final_geom, summary, warning = self.get_SCF_energies(molecule_id = molecule_id,
                                                        initial_charge_state = initial,
                                                        final_charge_state = final,
                                                        formatted_functional_basisset = '%s_%s' % (tracker['Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0])))
                    with open(r'%s/%s_%s_%s_SCF_Summary.txt' % (sub_directory, molecule_id, tracker['Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0])), 'wt') as file:
                        file.write(summary)
                        file.close()
                    keys.append('Initial CS at Initial Optimised Geometry')
                    values.append(initial_SCF_at_initial_geom)
                    keys.append('Initial CS at Final Optimised Geometry')
                    values.append(initial_SCF_at_final_geom)
                    keys.append('Final CS at Initial Optimised Geometry')
                    values.append(final_SCF_at_initial_geom)
                    keys.append('Final CS at Final Optimised Geometry')
                    values.append(final_SCF_at_final_geom)
                    keys.append('Parsed SCF Summary')
                    values.append(summary)
```

```
                four_point = initial_SCF_at_final_geom - initial_SCF_at_initial_geom + final_SCF_at_initial_geom - final_SCF_at_final_geom
                keys.append('4-Point Reorganisation Energy')
                values.append(four_point * constants.physical_constants['Hartree energy in eV'][0])
                if warning != '':
                    self.comment_result(_id = '%s_%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), initial, final),
                                        user = 'Auto-Generated',
                                        string = warning)

                # DUSHIN
                filename_initial_state = r'%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), initial)
                filename_final_state = r'%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), final)
                dushin_summarised, imaginary_frequencies = self.do_DUSHIN_calculation(molecule_id = molecule_id,
                                                                                      filename_initial_state = filename_initial_state,
                                                                                      filename_final_state = filename_final_state)
                keys.append('Vibrational Frequencies')
                values.append(list(dushin_summarised['Freq'].values))
                keys.append('Displacements')
                values.append(list(dushin_summarised['Q/cm'].values))
                keys.append('Reorganisation Energies')
                values.append(list(dushin_summarised['lam/eV'].values))
                keys.append('DUSHIN Reorganisation Energy')
                values.append(dushin_summarised['lam/eV'].sum())

                # atoms summary
                initial_atoms_summary = self.get_atoms_summary(molecule_id = molecule_id,
                                                               filename = r'%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), initial))
                final_atoms_summary = self.get_atoms_summary(molecule_id = molecule_id,
                                                             filename = r'%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), final))
                keys.append('Initial Atoms Summary')
                values.append(initial_atoms_summary.to_string())
                keys.append('Final Atoms Summary')
                values.append(final_atoms_summary.to_string())

                # simple energy analysis
                difference = (four_point * constants.physical_constants['Hartree energy in eV'][0]) - (dushin_summarised['lam/eV'].sum())
                frac_diff = difference/(four_point * constants.physical_constants['Hartree energy in eV'][0])
                keys.append('Difference (4-point & DUSHIN)')
                values.append(difference)
                keys.append('Difference Percentage (4-point & DUSHIN)')
                values.append('%.3f' % (frac_diff * 100))

                self._results.update_entry(molecule_id = '%s_%s_%s_%s_%s' % (molecule_id, tracker['Functional'].values[0], format_basis_set(tracker['Basis Set %s' % (basis_set)].values[0]), initial, final),
                                           keys = keys,
                                           values = values)

    def update_results_all(self,
                           stop_time: str = ''):
        if stop_time == '':
            stop_time = datetime.now() + timedelta(minutes = 15)
        if datetime.now() >= stop_time:
            return
        tracker = self.get_tracker()
        results = self.get_results()
        for molecule_id in tracker['_id'].values:
            functional = tracker.loc[tracker['_id'] == molecule_id, 'Functional'].values[0]
            if results.loc[results['_id'] == '%s_%s_%s_C0S0_C0S1' % (molecule_id, functional, format_basis_set(tracker.loc[tracker['_id'] == molecule_id, 'Basis Set %s' % (1)].values[0])), 'Last Updated'].values[0] == None:
                self.organise_results_one(molecule_id = molecule_id,
                                          # new_entry = False,
                                          download_results = False,
                                          tracker = tracker,
                                          results = results,
                                          basis_set_list = [1, 2, 3])
            else:
                if tracker.loc[tracker['_id'] == molecule_id, 'Last Updated'].values[0] > results.loc[results['_id'] == '%s_%s_%s_C0S0_C0S1' % (molecule_id, functional, format_basis_set(tracker.loc[tracker['_id'] == molecule_id, 'Basis Set %s' % (1)].values[0])), 'Last Updated'].values[0]:
                    self.organise_results_one(molecule_id = molecule_id,
                                              # new_entry = False,
                                              download_results = False,
                                              tracker = tracker,
                                              results = results,
                                              basis_set_list = [1, 2, 3])
        self.download_results_as_excel()
```