Uppsala University

Department of Physics and Astronomy

Bachelor's thesis 15 ECTS

How do iodized nucleotides fragment due to photoactivation?

Ebba Koerfer

Supervisors: Oscar Grånäs and Carl Caleman
Subject reader: Mattias Klintenberg

June 30, 2020

Abstract

Cancer is the second leading cause of death worldwide and affects millions of people every year. Furthermore, the available treatments often lead to severe side effects, thus improving radiation treatment is meaningful. Photoactivation therapy seeks to build in heavy atoms into the DNA of cancer cells, as markers, then activating them to cause secondary radiation that damages the DNA of the targeted cells only. This has been suggested but is not well understood. Hence this study seeks to investigate how a reduced model system of iodine-marked DNA is fragmented due to ionization. Computer simulations with eleven separate starting configurations of the molecule 5-iodocytidine were analyzed, for ionization levels from an average 0.03 up to 0.33 electrons removed per atom (e/N), during 200 femtoseconds (fs). A Python program was written in order to estimate bond sensitivities and identify fragments. While 5-iodocytidine resembles an iodized DNA-base it is still a rather simple model system, far from a double stranded DNA chain, and the simulations were limited to non-targeted ionization and an isolated environment. Results of this thesis include that the sugar "backbone" of 5-iodocytidine seems to be most sensitive to ionization, fragmenting in several pieces after 150-200 fs at ionization levels of 0.30-0.33 e/N, while the rest of the molecule mostly remained intact. These results appear promising since back bone fragmentation is crucial for disrupting cancer cell growth.

Sammanfattning

Cancer är den näst största dödsorsaken i världen och påverkar miljontals människor varje år. Dessutom leder tillgängliga behandlingar ofta till allvarliga bieffekter, därför är det meningsfullt att förbättra strålbehandling. Fotoaktiveringsterapi går ut på att bygga in tunga atomer i cancercellernas DNA, som markörer, och därefter aktivera dem för att orsaka sekundär strålning vilket endast skadar de fokuserade cellerna. Detta har studerats men processerna är inte fullt klarlagda, därför ämnar denna studie att undersöka hur ett förenklat modelsystem av jod-märkt DNA fragmenteras till följd av jonisering. Datorsimuleringar för elva olika begynnelsevillkor av molekylen 5-iodocytidine analyserades, för joniseringsnivåer från ett genomsnitt 0.03 upp till 0.33 borttagna elektroner per atom (e/N), under 200 femtosekunder (fs). Ett Pythonprogram skrevs i syfte att uppskatta bindningarnas känslighet och identifiera fragment. Även om 5-iodocytidine liknar en jodiserad DNA-bas så är det fortfarande ett tämligen enkelt modelsystem, långt ifrån två sammanbunda DNA-strängar. Simuleringarna var dessutom avgränsade till icke-fokuserande jonisering och en isolerad omgivning. Resultat från den här avhandlingen innefattar att socker-"ryggraden" av 5-iodocytidine verkar vara mest känslig för jonisering, och fragmenteras i flera bitar efter 150-200 fs vid joniseringnivåer 0.30-0.33 e/N, medan resten av molekylen oftast förblir intakt. Dessa resultat ser lovande ut eftersom fragmentering av ryggraden är särskilt viktig för att hämma tillväxten av cancerceller.

Contents

1 Introduction

Living organisms consist of cells that carry the hereditary information necessary for protein synthesis and cell replication in order to grow and renew throughout their lifetime. In the nuclei of eukaryotic cells there are long strands of deoxyribonucleic acid (DNA), which together with proteins form so called chromosomes. DNA is composed of four different nucleotides, each with a phosphate and sugar backbone bound to a specific base. Instructions for building a certain protein is encoded by a particular sequence of these bases. Cell replication is possible because each chromosome has two strands of DNA, with matching sequences of nucleotides, in the structure of a double helix [1].

Cell division is necessary for all living organisms, but diseases linked to an abnormal cell replication rate, cancers, exploit this process. Rapid cell growth can proceed to invade other parts of the body and thus spread the disease even more. Approximately ten million people died in 2018 due to cancer, and it is the second leading cause of death worldwide [2]. A common part of cancer treatment today is external high energy radiation therapy, however as the X-rays are shone on the target tumor it also causes damage to surrounding healthy tissue. Alternatively, radioactive iodine treatment (RAI) is currently being used and researched on as medical care for certain types of cancer and other diseases. Injecting a radioactive isotope of iodine with a large decay range, ^{131}I, into patients is one way to treat hyperthyroidism [3, 4] and thyroid cancer. The radioactive substance accumulates naturally in the thyroid and the decay damages nearby cells. A different isotope with lower energy radiation, e.g ^{125}I, can be built into the DNA during cell replication or built into binders that attach to the double helix [5], and so cause damage to nearby nucleotides as it decays. If the decay results in a fragmentation of the DNA strands such that the backbone is broken on both sides of the double helix, the cell cannot replicate anymore and the cancer growth is obstructed. This treatment can however damage healthy cells in the process, as the radioactive iodine can be built into naturally fast growing cells as well [6].

An alternative to RAI has been studied for several decades, but its process is not fully understood yet. The idea consists of using the stable isotope of iodine as a markers, building it into DNA the same way as ^{125}I, then using low energy X-rays to activate the iodine. This produces various secondary radiations which the common elements in DNA are susceptible to - causing the desired ionization and fragmentation. An external radiation source can therefore target the iodine markers specifically, thus preventing the harmful side effects of high energy radiation or decaying substances in non-cancerous cells. Experimental results support the potential of this alternative approach, and a deeper understanding of this so called photoactivation therapy (PAT) could be substantial for improving radiation therapy [6].

In order to understand how photoactivation of iodine-marked DNA can cause sufficient damage to decrease cancer cell growth, it would be interesting to investigate the process from the ground up - in a very simplified model system. The theoretical side of the research will be analyzed in this project, with the goal of later being able to compare

the results to experiments. Using a reduced model system enables the problem statement to be explored under less complicated premises, while ideally still revealing something about the represented system. Simulations were previously made based on quantum mechanical first principles and density functional theory, modelling the aftermath of the ionization of an iodized nucleotide-like molecule; 5-iodocytidine. The simulations return coordinates of the atoms as a function of time, including information about other quantities such as atom charges for each time step. Analyzing this data in a Python program will return heatmaps of mean bond integrity over time for the different relevant bonds in the molecule and several ionization levels. Fragments, sorted by charge and mass, and their mass spectrometry trajectories can then be identified. In the end these results will hopefully answer the following problem statements.

How does a single nucleotide-like molecule marked with a stable iodine atom, 5-iodocytidine, fragment due to different levels of ionization over the molecule?
Which bonds break and what are the distributions of fragments based on charge and mass? Under which circumstances does the backbone break?

2 Background

2.1 DNA structure

Two long chains of nucleotides, connected in the form of a double helix, is the structure of genetic information called deoxyribonucleic acid (DNA). Information on how to construct different proteins are decided by the order of nucleotides in the chain. The second strand in the double helix is an indirect copy of the first one, making it possible for the cell to clone itself by opening up the helix and replicating both chains separately [1]. Figure 1 illustrates the general structure of the DNA polymer with the four unique nucleotides.

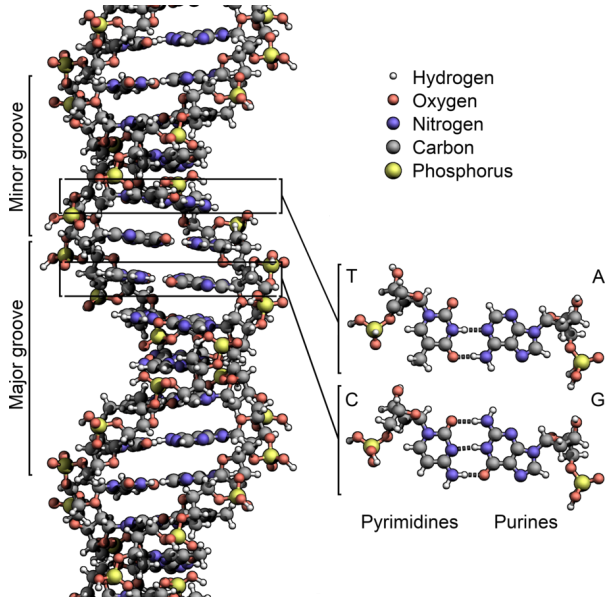


Figure 1: Double helix structure of DNA [7].

Each nucleotide is composed of a phosphate-sugar backbone and one of four different bases called adenine (A), cytosine (C), guanine (G) and thymine (T). These DNA bases are paired together, A and T attach through a hydrogen bond, and C bonds with G likewise [1]. This is how the second strand in the double helix is an inverted copy of the first, as the base pairs correspond to each other. A figurative representation of this structure is shown in figure 2.

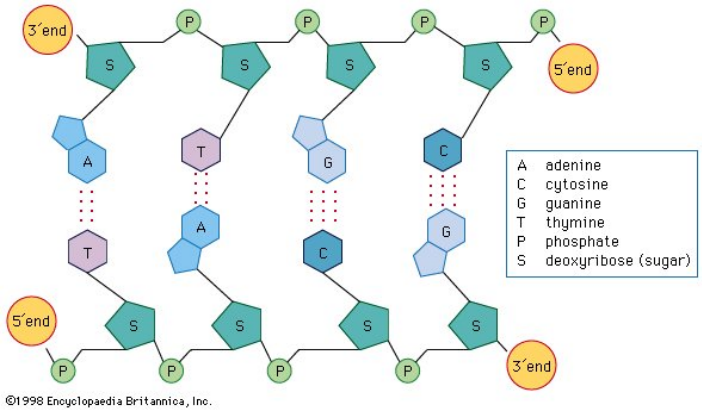


Figure 2: DNA double chain structure and bonding [8].

An even closer look at the skeletal structure of these four nucleotides is depicted in figure 3, where the arrangement of every atom and bond in the molecules are represented. This image shows an example of a certain sequence of bases, namely AGCT, and how the

DNA backbone is bonded to them. Here it is easier to distinguish the formation of the backbone, phosphate-groups alternated with deoxyribose, a pentose sugar [9].

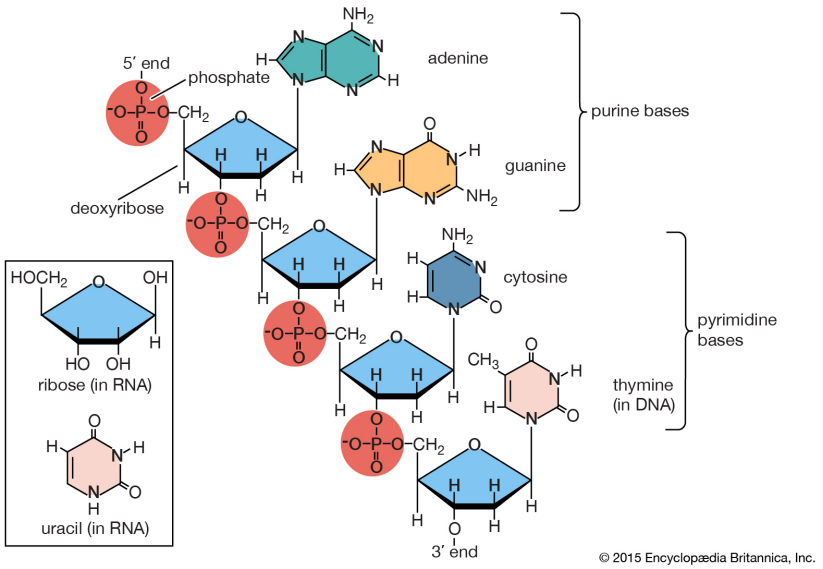


Figure 3: Skeletal structure of DNA nucleotides with bases adenine, cytosine, guanine and thymine [10].

2.2 Radioactive iodine treatment and Photoactivation therapy

Cancer is a term for diseases characterized by an unusually rapid cell growth, and even though there are different methods to relieve symptoms, more permanent treatments are certainly preferred and searched for.

Some types of cancer are treated with high energy external radiation, exposed on a larger area around the tumor, which also causes uncontrolled damage to non-cancerous tissue. Radiation therapy with high energy X-rays is frequently used today, leading to these harmful consequences. Other forms of medical care include internal radiation, such as injecting or implanting a radioactive substance into the body - in the hope of damaging the malignant tumors as the substance decays close to them [4]. Radioactive iodine treatment (RAI) is a term for different methods of internal radiation, including the use of isotope ^{131}I to target the thyroid gland in cases of thyroid cancer or hyperthyroidism as the substance can accumulate there naturally. This isotope of iodine produces high energy radiation, beta and gamma, where the beta decay can have effect in a millimeter range. Hence the cancerous tissue can be damaged to quite a large extent [3].

RAI has been used since the 1940s to treat hyperthyroidism. However, there is a debate among scientists whether RAI causes adverse complications that offsets the benefits in these cases [11]. A study observed patients with thyroid cancer over a 40 year period,

where the patients had been treated with RAI or external beam radiation. The study concludes that even though the treatment is generally well tolerated there are risks of severe side effects. Radiation sickness, bone marrow depression and other dysfunctions are mentioned as possible consequences of RAI [4]. A different study is also discussed here, whose results revealed a greater risk for developing other types of cancer, both solid forms and leukaemia, due to the radioactive decay. Even though there are some positive results for certain patient groups, in regards to treating thyroid cancers, the article concludes that limiting the use of RAI is meaningful considering the harmful side effects [12].

Another approach to internal radiation treatment seeks to target the foundation of the cancer cells. The growth is disrupted form the inside by affecting the DNA directly in the nuclei. There are studies where DNA-binders have been used to achieve this, such as *Iodo-Hoechst derivatives* with built in iodine ^{125}I. This is a radioactive isotope that emits low energy X-rays in a nanometer radius from its position. As these *Iodo-Hoechst derivatives* binds to the DNA double helix, the natural decay of the built in iodine can cause the desired double strand breaks. It seems that the binders can attach close enough to the DNA that despite the relatively small decay volume of ^{125}I, significant damage can be caused [5]. However, since the iodine marked binders in practice would be fed to patients and spread throughout the body there is always a possibility that these radioactive substances are also bound to the DNA of healthy cells. Consequently this would cause undesired damage with possible side effects.

Targeting the cancer cells even closer than with the binders, by building substances into the DNA itself, has been researched on as well. The idea consists of feeding patients nucleotides or short DNA strands marked with a substance, such as thymine bases marked with an iodine isotope, which would allow it to be built into the DNA during cell replication. Since cancer is characterized by an unusually rapid cell growth, this exchange of regular thymine with the iodine marked thymine is more likely to happen in the cancer affected cells. Considering that new DNA is formed during cell division. If a radioactive substance is used, such as ^{125}I, the decay would damage the DNA even more likely than when helix-binders were used. However, these marked nucleotides could be built into the DNA of non-cancerous cells as well, such as bone marrow cells that naturally grow fast. Using radioactive isotopes such as ^{125}I can therefore be toxic to healthy growing tissue [6].

To avoid harming healthy cells a slightly different concept has been considered. Namely building stable high-Z atoms into the DNA and then activating them with low energy X-rays to induce so called Auger cascades. This procedure is called photoactivation therapy (PAT). Human tissue is mostly made of low-Z atoms, such as carbon, oxygen and nitrogen, thus it is possible to target the high-Z atom markers specifically by choosing a certain photon energy. High-Z atoms have a larger photoionization cross section than low-Z atoms for certain photon energies, meaning the probability of emitting a electron from a certain electronic state is higher. Tuning the photon energy to hit the core energy levels and creating a core hole, as an electron is removed, can result in different outcomes

when the core vacancy is filled by another electron from a higher state. The released energy of this transition can result in an emitted photon, with lower energy than the activating radiation, but there is also a chance that the energy is transferred to a third electron and thus ejecting it from the atom. These ejected electrons are called Auger electrons and the process is called the Auger effect [6].

In the case of marking DNA with a non-radioactive high-Z atom, such as a stable isotope of iodine, the Auger effect can be utilized. The external photoactivating radiation could be shone locally on the tumour, and by choosing a certain photon energy the core levels of iodine will most likely absorb it, hence affecting the DNA of the cancer cells locally. A method that causes less damage to healthy tissue compared to external high X-ray treatment and radioactive markers like ^{125}I. When the iodine atoms are activated they then emit a combination of secondary X-rays and a cascade of Auger electrons, which is a complex process that is not fully understood. However, it is known that the Auger electrons and secondary X-rays are likely to be absorbed by the lower-Z atoms, such as carbon, nitrogen and oxygen, which DNA is composed of. Atoms in the DNA close to these iodine markers would thus be ionized and cause fragmentation of the total structure, both single strand breaks and double strand breaks have been found in previous experimental research [6].

Results from several experiments suggest that PAT could improve radiation therapy, as DNA fragmentation in cancer cells can occur with less risk of damaging healthy tissue. There are however several obstacles, one being that the required photoactivating radiation is generated by a synchrotron, which facilities are not adapted for treating patients. Another problem is the lack of understanding how the Auger cascade works and how the DNA is fragmented due to the photoactivation of the iodine [6].

Considering this limited knowledge, it is valuable to investigate how this fragmentation process occurs in detail. Exploring both theory and experiments from the ground up. By analyzing reduced model systems of iodized DNA, such as a single nucleotide, it might give insight into the fragmentation of DNA induced by core level ionization. Studying when and how this photoactivation can cause the backbone to break is especially interesting, since damaging the foundation of the DNA is vital to obstruct its reparation and thus impede replication of the cancer cells. In the future, along with studies and collaborations between different faculties, this could hopefully contribute to improving radiation therapy by alleviating harmful side effects.

2.3 Ionization simulations based on DFT calculations

There have been several experimental studies of PAT using stable iodine as a marker, as described in the previous section, and mostly the results seem to support the idea while also acknowledging that there is a lack of understanding of the process. A group of researchers from different fields, physics, biophysics and medical physics, at Uppsala University have recently started doing theoretical and experimental studies in the hopes of deepening their insight into the process. On the theoretical side methods such as

molecular simulations of different ionization processes can be applied. As of yet no papers have been published by this research group regarding this particular subject. However, an experiment has been performed at the synchrotron source BESSY II in Berlin. These results will be compared with the theoretical work in the near future.

There are developed methods for simulating molecule dynamics in an ionization process, using first-principles, which have been used to understand the fragmentation of peptides for the purpose of improving protein imaging experiments. Initial conditions for the molecular structure are generated from quantum molecular dynamics based on density functional theory (DFT), using Siesta software [13]. DFT is also applied after ionization, in order to generate molecule trajectories. In short, DFT utilizes a density based approach for the electrons which makes the calculations easier than the many-body perspective. Both classical and quantum mechanical interactions are included in DFT, such as classical Coulomb forces but also exchange and correlation effects. Various approximations can simplify the DFT calculations further. Since nuclei have a greater mass than electrons, the Born-Oppenheimer approximation can be applied - meaning that the nuclei are regarded as motionless relative to the electrons and thus decoupling their degrees of freedom [14]. The probability of breaking certain bonds is highly dependent on initial conditions. Due to the Born-Oppenheimer approximation, the electron wave functions are explicitly dependent on electron positions but also implicitly dependent on nuclei positions, thus different initial conditions for the nuclei positions will yield different results. Hence, several simulations with different initial conditions have to be performed to limit this dependence [15]. It is also important to integrate out the initial condition dependence when comparing the simulations with experimental results. Fragmentation of molecules in a synchrotron experiment practically shows outcomes for all starting configurations, as many randomly positioned molecules are shone on with X-rays over long periods of time. As the experiments gather large amounts of statistics, it is relevant to pursue the same in theoretical studies for a compatible comparison.

In order to deal with the large number of electrons in the molecules, certain approximations regarding electron movements are made. Using so called pseudo-potentials, where core electrons are seen as inert to their surrounding, and therefore not explicitly considered in the calculations. Instead the core and core electrons form a pseudo-potential, and the wave functions of the valence electrons are smoothed out close to the core. When photons interact inelastically with atoms in the molecule, the energy can absorbed by an electron which given a certain wavelength can remove the electron entirely - causing ionization. De-excitation can then occur through the release of a photon or by the transition of a higher energy state electron to the vacancy. The pseudo-potentials used do not regard electron holes, nevertheless it is consistent assuming that these vacancies are filled quickly by other electrons before affecting the system [14].

Along with other assumptions and physical discussions, these methods are implemented in the *Siesta*-package and return atomic coordinates as a function of time and charge-distribution as a function of time. Hirshfeld charge is used to calculate the atomic charges, which is based on partitioning molecular density into atomic density contri-

butions that allows less basis-set dependence than other common methods [16]. The simulation results can then be analyzed to distinguish how the molecule was fragmented due to a certain exposure of radiation, or from the point of a specific ionization, by defining when a chemical bond is considered broken and by observing the change in charge distribution. To estimate the prospect of bond-breaking between two atoms, A and B, the chemical bond integrity has previously been defined as

$$\mathscr{B}_I(A,B,t) = \frac{1}{N}\sum_{i=1}^{N}\left(1+e^{\lambda(|d_i[A,B](t)-d_i[A,B](0)|-0.5)}\right)^{-1} \tag{1}$$

where the time-dependent separation of the atoms, for the i-th simulation of the molecule, is denoted $d_i[A,B]$. N is the number of simulations and λ is a smearing parameter, chosen such that regular oscillations between the atoms are not considered a broken bond. If the value of the bond integrity $\mathscr{B}_I$ is close to 1 the atoms are in their initial positions, while a value close to 0 indicates that the separation of the atoms differ significantly from the initial state, meaning the bond can be considered broken. The bond integrity definition is described in more detail in work by Oscar Grånäs, et al [15]. In this work the bond integrity measure was improved upon, in order to avoid the initial condition dependence of the term $d_i[A,B](0)$. This is discussed further in the methodology.

Simulations based on these concepts have been made by Oscar Grånäs at Uppsala University, for a molecule called 5-iodocytidine - which is the base cytosine linked to a five-carbon sugar D-ribose, but with a iodine atom bound to the hexagon. See figure 4. 5-iodocytidine has 30 atoms, and the molecular formula is $C_9H_{12}IN_3O_5$, which means that its total mass is around 369 $A_{r,std}$ (standard atomic weights) [17].

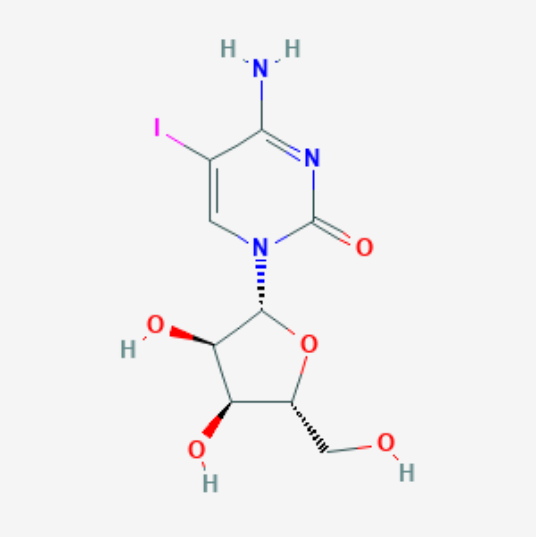


Figure 4: Chemical structure depiction of 5-iodocytidine [18].

In figure 5 the iodized single DNA strand CTG is visualised, alongside 5-iodocytidine, in order to see the structural similarities. Even though 5-iodocytidine lacks the exact sugar-phosphate backbone that cytosine has, it has a very similar sugar structure and can be considered an acceptable model system for a iodized nucleotide.

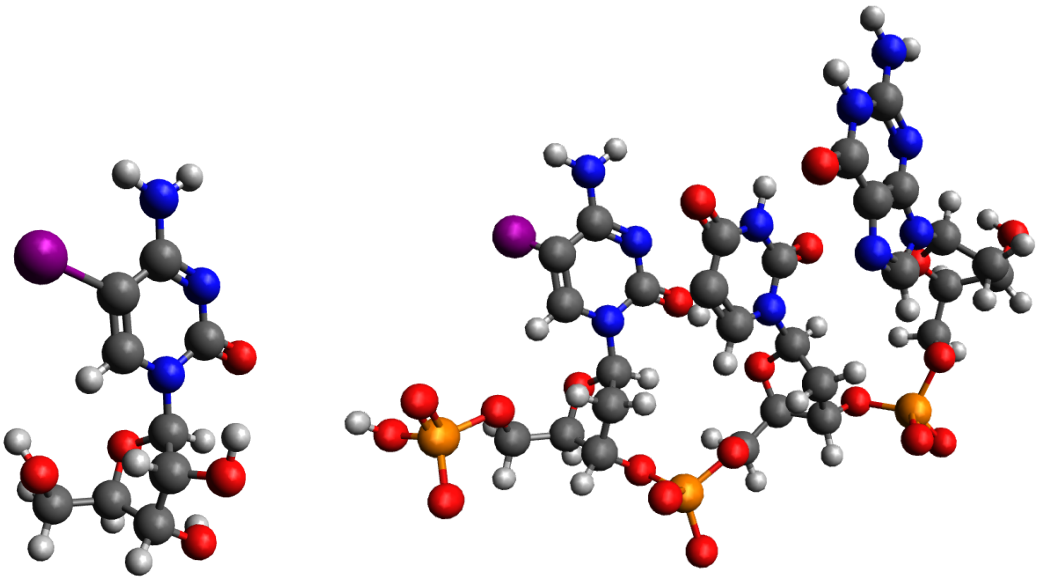


Figure 5: On the left: 5-iodocytidine. On the right: DNA sequence with bases Cytosine-Thymine-Guanine, with a hydrogen substituted for iodine (I) on the cytosine base. (Phosphor (P) - yellow, Carbon (C) - grey, Oxygen (O) - red, Nitrogen (N) - blue, Hydrogen (H) - white, Iodine (I) - purple). Visualised with Avogadro [19].

Since this research is merely in early stages the model system for the iodized DNA was chosen to be simply this 5-iodocytidine, to make the simulation and analysis less complicated. Even more so, the simulation does not account for the properties of the external radiation that DNA is meant to be exposed to - instead the simulation starts at the ionization of the molecule. Aftereffects of the ionization is the main focus in this case, to investigate what fragments appear due to the following cascade effects. It is also important to note that this is a model system in more than one sense, because the process is in fact supposed to occur in a human cell nuclei - not isolated in vacuum. Also if results are compared to synchrotron experiments, the environments are not exactly the same either. The molecules are often surrounded by something, such as cold helium gas or water, and are affected by a weak electric field.

Using *Siesta* software, Oscar Grånäs performed computer simulations of 5-iodocytidine for several scenarios. See the input-files for the simulation program in appendix C. Firstly ten thermalization runs were made, simulating the molecule in room temperature without ionization, thus generating data for each bond's natural thermal oscillations. These ten simulations began with different atom positions, in order to minimize the

dependence on the molecule's initial conditions, and ran for 1000 femtoseconds (fs) to provide data for the statistical analysis. Then various simulations for the ionization process of 5-iodocytidine were made. In detail; eleven separate starting geometries, again to minimize initial value dependence, each with ionization levels of one up to 10 electrons removed from the molecule. The ionization is homogeneous over the molecule, but not evenly spread since the most weakly bound electrons are removed first. This is meant to simulate the effects of Auger cascades, as described in 2.2, which is the believed cause of most of the fragmentation in experiments. In total 110 simulations provide various information about the molecule, including atom positions and charges, during 200 fs after ionization. Data from these simulations were stored in OUT files.

3 Method

Data analysis of Oscar Grånäs's simulations, described in section 2.3, was made automatic by writing code in Python 3 language using the interactive computing platform Jupyter Notebook. The code was written in collaboration with another bachelor student at Uppsala University, Emma Danielsson, who studied fragmentation of another organic molecule. Thus the program can be used for different molecules, if the data files from the simulation carry similar information in the same format. The Python program used to produce the results for this project can be found in appendix A. An overview of the code is shown in figure 6, depicted in a flowchart.

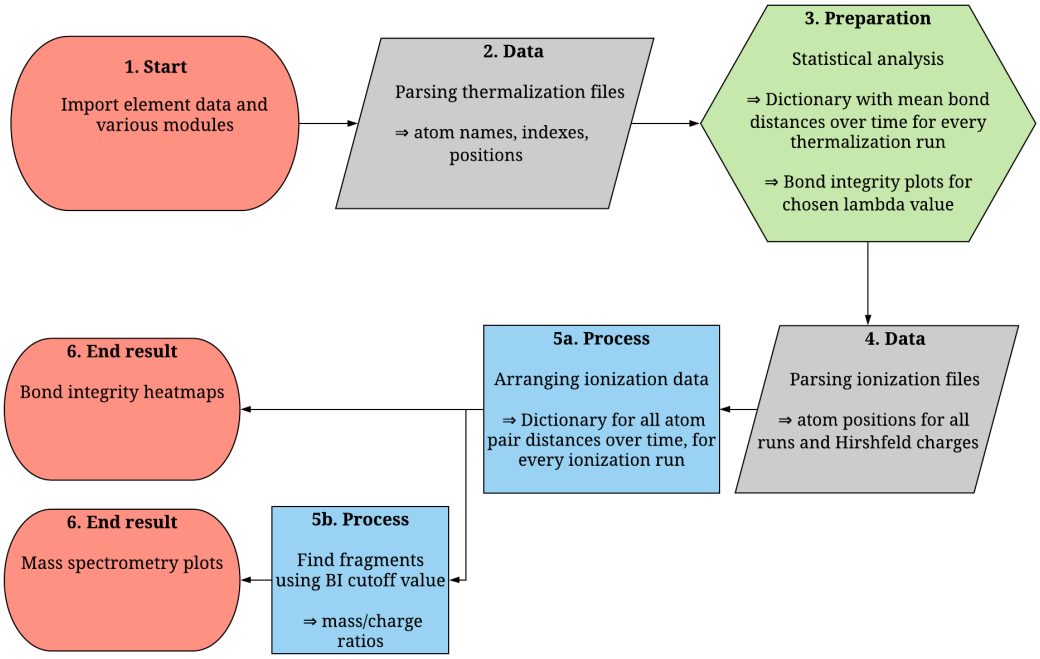


Figure 6: Flowchart of the Python program that generated the desired results from given OUT files, which were previously produced by computer simulations using *Siesta* software [13]. Made on Lucidchart.com.

Initially, common Python modules were imported and used along with a library of necessary functions and classes. These were previously written by several people, among others Oscar Grånäs, in the same research group at Uppsala University. Statistical analysis of the parsed thermalization runs was then executed, where the aim was to find expectation values and standard deviations for the distance between every two atoms that form a bond in the molecule. This corresponds to steps 1-3 in the flowchart, see figure 6. Since the data files from the simulations contained a lot of information but not sorted in a readable way, the program first identified atoms in the molecule and gave them unique indexes. Then all the neighbouring atoms were identified for each atom, by finding other atoms in a 1.8 Å radius. One exception was made, since the iodine-carbon bond (see figure 5) was much longer than the other bonds this bond pair was added manually. This was done instead of increasing the search radius, because that would result in several incorrect neighbours. Thereafter distances between each neighbour pair was calculated for all 1000 fs, and from this data mean bond distances and standard deviations were calculated. These results were then used to find a suitable value for the smearing parameter λ, in the definition of bond integrity, see equation 1. Instead of using this exact definition of bond integrity, a slightly different equation was defined in order to benefit from the statistical results. Bond distance at time zero, $d_i[A,B](0)$, was replaced with the mean value $\mu(d_i[A,B](0))$ from the thermalization runs shifted by one standard deviation $\sigma(d_i[A,B](0))$. The shift was done because the function is not symmetric around the mean value. Using the mean value of bond distances surpasses the use of bond distance at time zero, because the bonds fluctuate and could therefore return significantly varying values for a specific time. The following formula was used

$$\mathscr{B}_I(A,B,t) = \frac{1}{N}\sum_{i=1}^{N}\left(1+e^{\lambda(|d_i[A,B](t)-\mu(d_t[A,B])-\sigma(d_t[A,B])|-0.5)}\right)^{-1}. \tag{2}$$

Having obtained the bond distances from the thermalization runs, different values of λ was used to calculate the bond integrity for a certain bond as a function of bond distance. This was then compared to the general form of the bond integrity function, to see if the thermal oscillations of the bond is kept at a bond integrity value close enough to 1 - since the maximum of the natural oscillations should not be considered a broken bond. A suitable value for the smearing parameter was found to be $\lambda = 10$, by analysing these plots for a sample of bonds. Finally the list of neighbours was pruned in order to remove incorrect and abundant bonds, namely some hydrogen atoms that had more than one neighbour due to being close to several atoms, and also removing doublets of atom pairs.

After finishing the statistical analysis the ionization simulations were parsed and examined. Procedures 4 and 5a in figure 6. In a similar way bond distances as a function of time were calculated from the data files, for all bonds in the molecule that were found in the previous analysis. This was done for every bond and every ionization run, namely for eleven starting geometries with 1-10 electrons removed each, spanning 200 fs. The

least tightly bound electrons were removed first, thus unevenly spread over the molecule. Bond integrity for various bonds, for a chosen starting geometry and ionization level, could then be plotted as a function of time. These results reveal if the bond breaks, if the calculated bond integrity goes to zero at a certain time. Considering the vast amount of data and simulation runs, this information was also gathered in a fewer number of plots by generating heatmaps that allow an extra dimension. This corresponds to one of the sixth and final steps in the flowchart, in figure 6. For chosen interesting bonds, the program produced heatmaps where the mean value of bond integrity, over all starting geometries, is shown in a color gradient as a function of time and ionization level $\overline{z}$. The ionization level is the number of electrons removed per number of atoms in the molecule, (e/N). This visualisation choice was inspired by the results in an article by Oscar Gråsäs, et al [15], and the mean bond integrity data was interpolated to smoothen the graphs. To validate these bond integrity results they were compared to the simulation data directly in Avogadro [19], where a visualisation of the molecule can be observed. In Avogadro's animation extension it was possible to see the trajectories of the atoms for every time step, which confirmed that the program had processed the simulation data correctly.

Finally the bond integrity values for every atom pair, for all ionization runs, was controlled against a chosen limit in order to identify the fragments of the molecule at the last time step. See process 5b in the flowchart. Bonds with a bond integrity lower than 0.5 was considered broken, while higher than 0.5 were considered intact. Fragments were only determined at the end of the run, after 200 fs, due to the possible comparison with experiments - where the timescale of results in a detector is practically infinite compared to a few hundred femtoseconds. This allowed less time consuming calculations as well.

Further, when fragments for a certain simulation run had been found, mass spectrometry plots were made. Namely intensity plots of occurring mass/charge ratios. This is the last end result in the flowchart, step 6 in figure 6. For a specific ionization level, fragments from the eleven separate runs were gathered. The total mass and charge of each fragment was then calculated, using imported element data and Hirshfeld charges from the simulation files, thus providing a set of mass/charge ratios from several runs. Histograms and density plots were then made for these fragment occurrences at values of $\overline{z} = 0.03$-0.33 (e/N), since 5-iodocytidine has 30 atoms and simulations had been made for one up to ten removed electrons. When visualising the results in the histograms and density plots, the bin widths were optimized to fit each ionization level. To represent the results in the most legible and correct way the peak heights were compared with the calculated mass/charge ratios. In the results, chosen bin widths for every ionization level is specified in the caption of figure 19-21. Fragments from all simulation runs with the same $\overline{z}$ were combined into one mass spectrometry plot, since different starting geometries can give varying results. Also it is more suitable when comparing with experimental results, as they show the statistics from a vast amount of trajectories.

4 Results

All following results were produced by the Python program described in section 3, which can also be found in appendix A along with additional functions in appendix B. Each atom in 5-iodocytidine was assigned an index by the program, see figure 7 below, references to these indexes are made throughout the results.

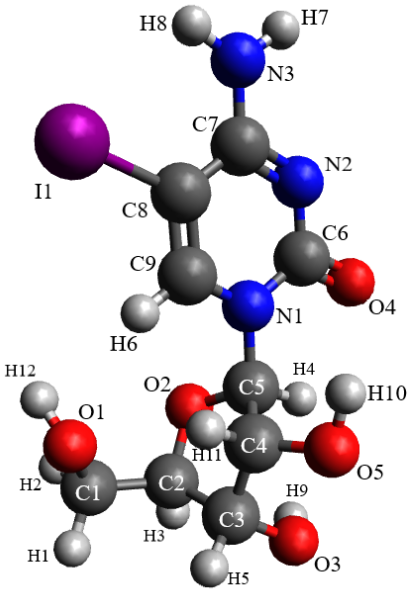
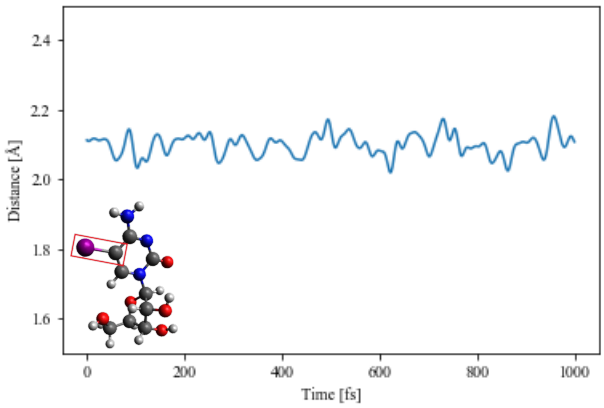


Figure 7: 5-iodocytidine with atom indexes

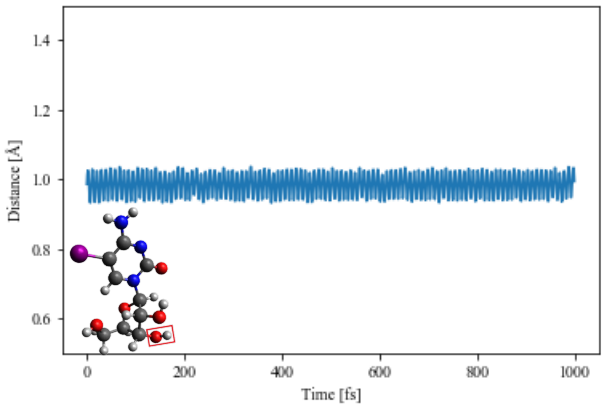
The results are presented in the same order as the workflow of the program, and thus demonstrates the analysis chronologically. A selection of interesting results have been chosen for display here.

4.1 Statistical analysis

Figure 8 displays the bond distance, in Ångströms, as a function of time of two atom pairs during one of the thermalization runs. Similar results were found for all bonds in the molecule, and the mean values for these bond distances - along with a corresponding standard deviation - was calculated.



(a) I1-C8



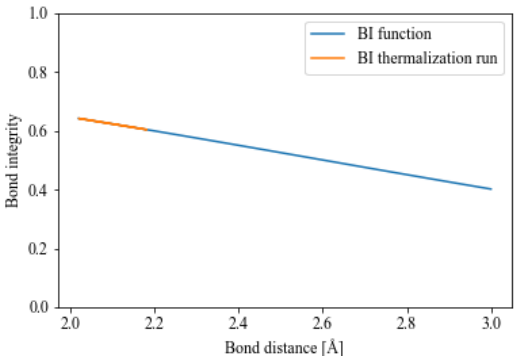
(b) O3-H9

Figure 8: Bond distance (Å) as a function of time (fs) for bonds I1-C8 and O3-H9, the longest bond in the molecule and one of the shortest. Differences in bond strength could be observed by comparing frequency of oscillations and the displacement from mean values.

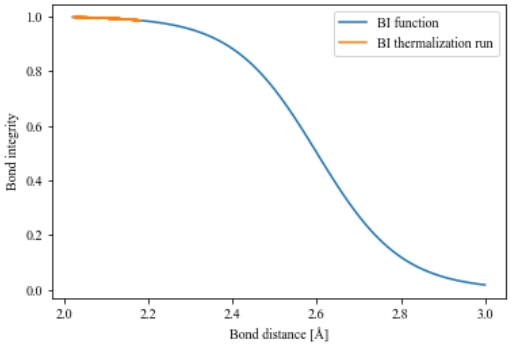
As an example, the calculated mean bond distance for I1-C8 was $\mu = 2.099$ Å with standard deviation $\sigma = 0.001$ Å and for the shorter bond O3-H9 these values were $\mu = 0.985$ Å and $\sigma = 0.0007$ Å. Note that different elements bonded to each other can result in varying bond strengths due to their chemical properties. For instance polarity in the

oxygen-hydrogen bond could provide further strength to the covalent bond.

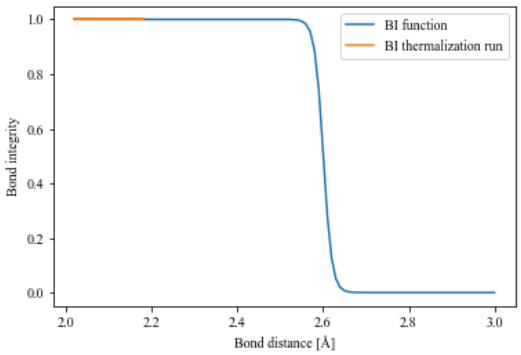
Some different atom pairs were analyzed to find a suitable value for the smearing parameter λ in the definition for bond integrity, see equation 2. In figure 9 and 10 the bond integrity for atom pairs I1-C8 and O3-H9, during one of the 1000 fs long thermalization runs, is depicted as a function of bond distance. Calculations for three different values of λ are shown. This is compared to the general form of the bond integrity function for the same statistical values of mean distance and standard deviation as well as λ.



(a) I1-C8 $\lambda = 1$



(b) I1-C8 $\lambda = 10$



(c) I1-C8 $\lambda = 100$

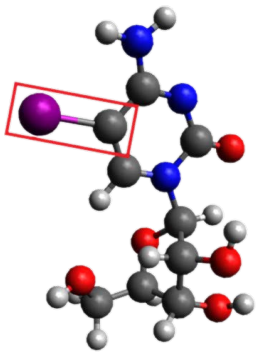
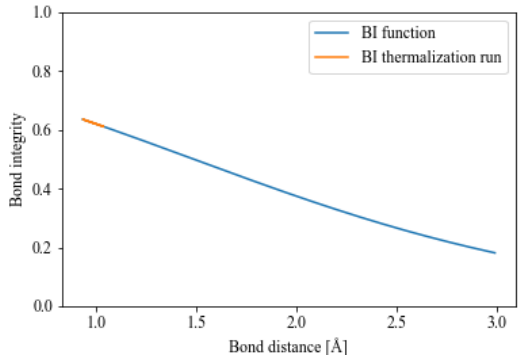
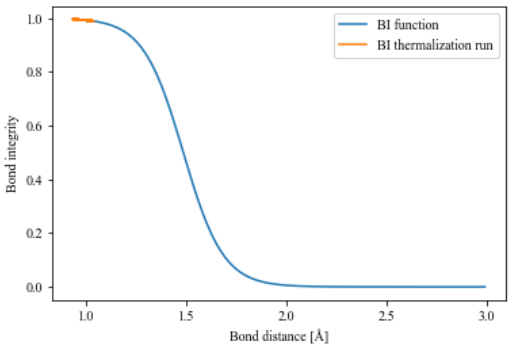


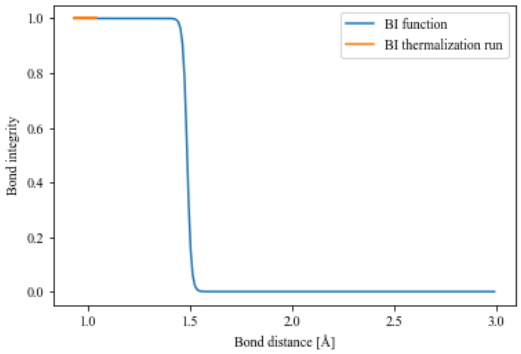
Figure 9: Bond integrity as a function of bond distance for the longest bond I1-C8, from a thermalization run (orange), compared to the general bond integrity function (blue).



(a) O3-H9 $\lambda = 1$



(b) O3-H9 $\lambda = 10$



(c) O3-H9 $\lambda = 100$

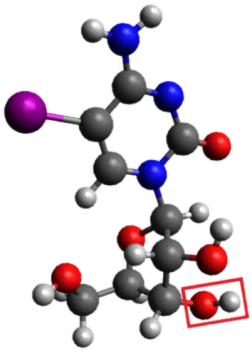


Figure 10: Bond integrity as a function of bond distance for one of the shortest bonds O3-H9, from a thermalization run (orange), compared to the general bond integrity function (blue) for the same statistical values. Here different values for the smearing parameter λ is shown.

Inspection of the graphs in figure 9 and 10 resulted in choosing the smearing parameter for bond integrity to be $\lambda = 10$. The following results were produced by the program, using this parameter value for all bonds in the molecule.

4.2 Bond integrity

Bond integrity for atom pair C3-C4 is plotted in figure 11 as a function of time during a single ionization run of the 110 different simulations, with a certain starting geometry and ionization level.

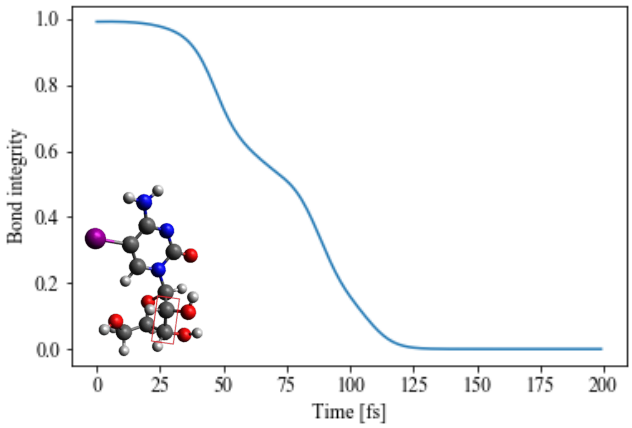
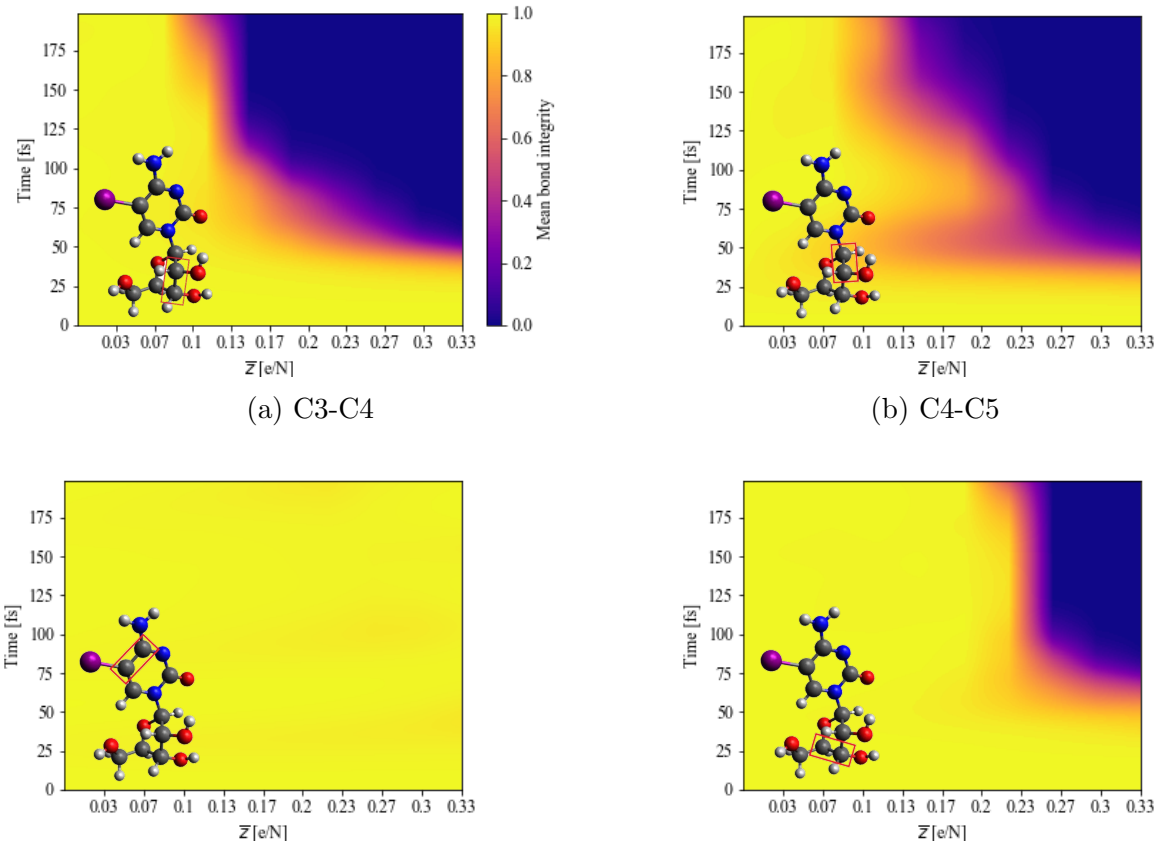


Figure 11: Bond integrity for C3-C4 as a function of time for ionization run with starting geometry 2 and ionization level $\overline{z}$=0.27.

The bond integrity evolution for a specific bond in the molecule varies for every starting geometry and ionization level, thus many results of the same type as depicted in figure 11 are gathered into more detailed graphs. Heatmaps visualising bond integrity averaged over all eleven staring geometries, for every ionization level and time step, are found in figures 13, 15 and 17 for various selected bonds in the molecule. The colorbar displays mean bond integrity values, where blue is 0, indicating a broken bond, and yellow is 1 which means an intact bond. Along with the mean bond integrity plots, some examples of specific ionization runs are shown for three different time steps - from the animation extension in Avogadro [19]. In these figures bond breakage is visualised and resulting fragments can be seen. Some examples of these animations were chosen from various ionization runs. In figure 12 the simulation with one of the starting geometries and the ionization level $\overline{z} = 0.17$ is shown for a few time steps.

(a) t = 177 fs (b) t = 200 fs

Figure 12: Fragmentation of 5-iodocytidine for $\overline{z} = 0.17$ and starting geometry 3. Only the bond C3-C4 is broken in this case, at the very end of the simulation t = 200 fs.



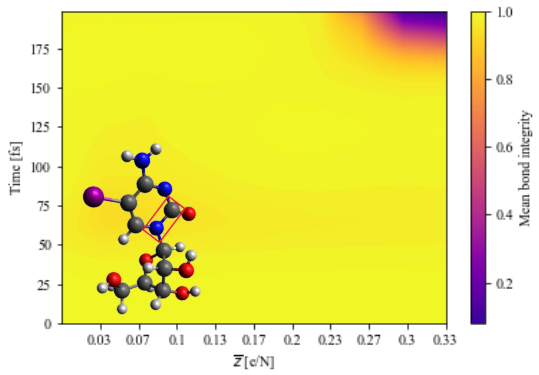
(a) C3-C4 (b) C4-C5

(c) C7-C8 (d) C2-C3

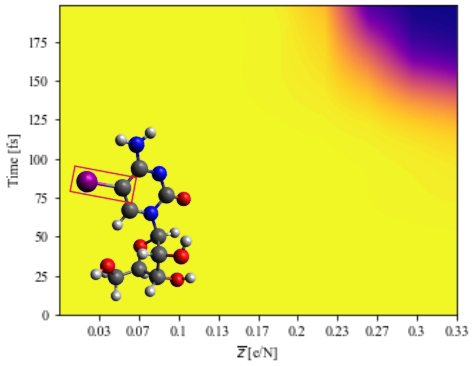
Figure 13: Mean bond integrity as a function of time and ionization level $\overline{z}$ for some of the carbon-carbon bonds in the molecule.

(a) t = 140 fs (b) t = 200 fs

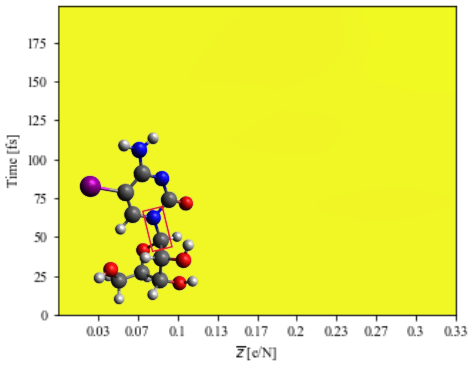
Figure 14: Fragmentation of 5-iodocytidine for $\overline{z} = 0.23$ and starting geometry 0. The last time step, t = 200 fs, is interesting to compare with e.g figure 13 (a) and (d).



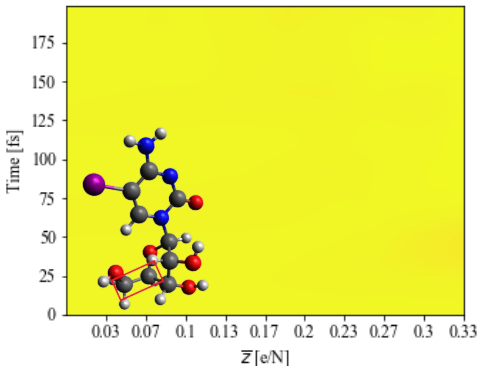
(a) N1-C6



(b) I1-C8

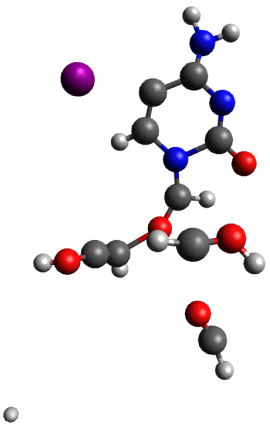


(c) C5-N1

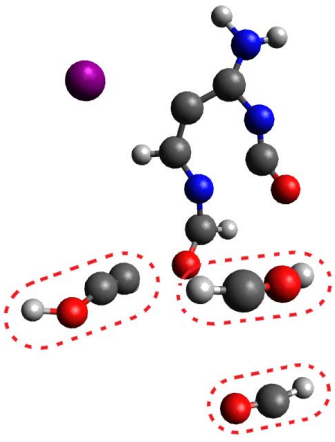


(d) C1-C2

Figure 15: Mean bond integrity as a function of time and ionization level $\overline{z}$ for a selection of interesting bonds, such as C5-N1 in (c) which connects the hexagon structure to the "back bone" part of the molecule - see figure 7.

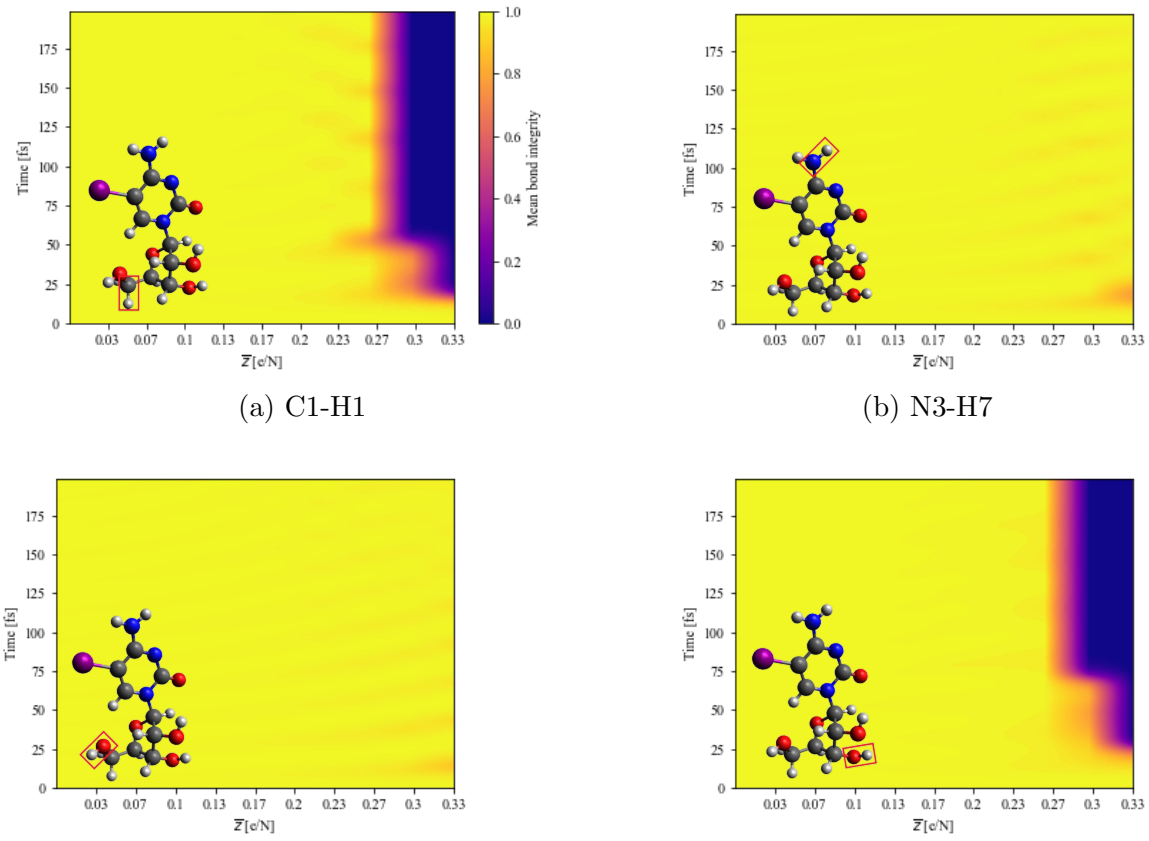


(a) t = 160 fs



(b) t = 200 fs

Figure 16: Fragmentation of 5-iodocytidine for $\overline{z} = 0.33$ and starting geometry 3. The sugar "backbone" is broken into three pieces, and separates from the larger fragment.



(a) C1-H1

(b) N3-H7

(c) O1-H12

(d) O3-H9

Figure 17: Mean bond integrity for some of the atoms bound to a hydrogen.

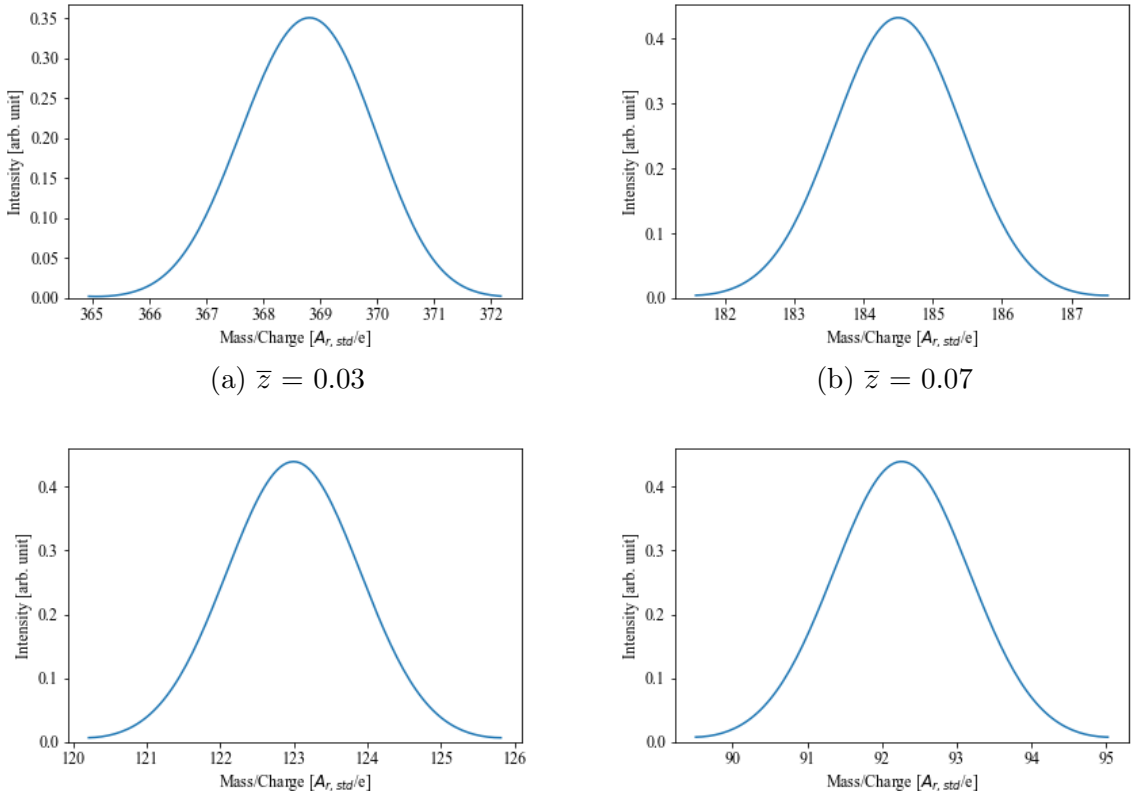
(a) t = 60 fs

(b) t = 200 fs

Figure 18: Fragmentation of 5-iodocytidine for $\overline{z} = 0.33$ and starting geometry 4. Unlike figure 16, the sugar backbone breaks into only two pieces, and a part is still attached to the hexagon structure. Note: some of the hydrogen atoms are out of frame in (b).

4.3 Mass spectrometry

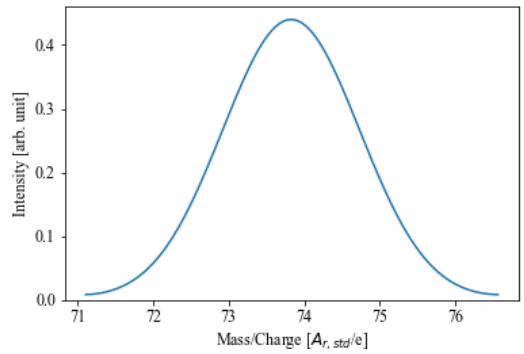
Occurence of fragments with a certain mass over charge ratio, for every starting geometry, was calculated and plotted for every ionization level. This might be comparable with experimental mass spectrometry results. The number of atoms in 5-iodocytidine is N = 30, and the ionization level $\overline{z}$ (e/N) is the average charge per atom - number of electrons removed per number of atoms. Masses are given in standard atomic weights, and the Hirshfeld charges of each fragment are given in units of electron charges. Intensity, here shown as probability density, of fragments has arbitrary units.



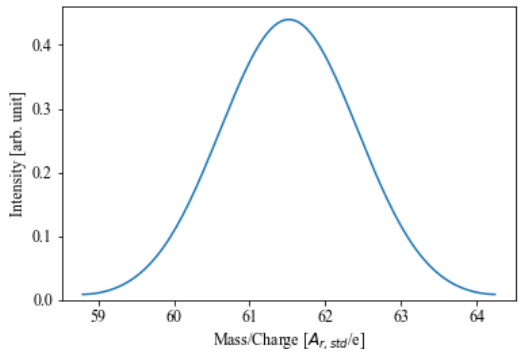
(a) $\overline{z} = 0.03$ (b) $\overline{z} = 0.07$

(c) $\overline{z} = 0.10$ (d) $\overline{z} = 0.13$

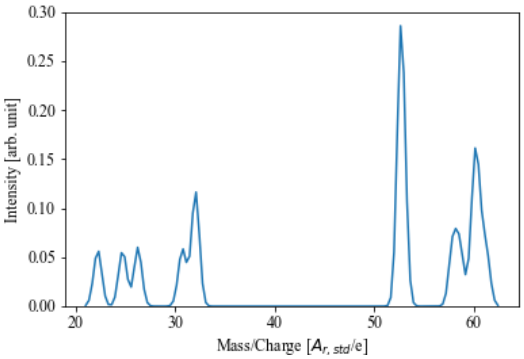
Figure 19: Mass spectrometry plots for one up to four electrons removed, spread over the molecule, mean ionization level $\overline{z}$ (e/N) per atom is shown for each figure. Bin width was set to 0.9 for (a)-(d).



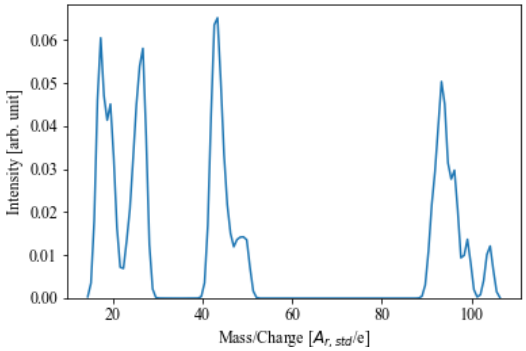
(a) $\overline{z} = 0.17$



(b) $\overline{z} = 0.20$

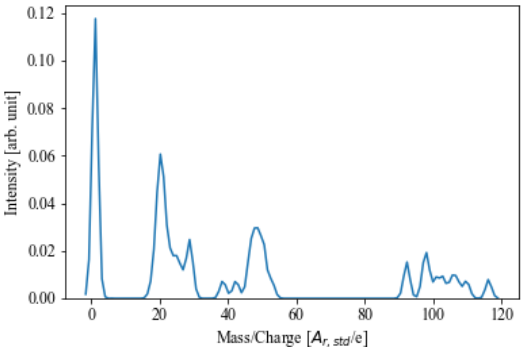


(c) $\overline{z} = 0.23$

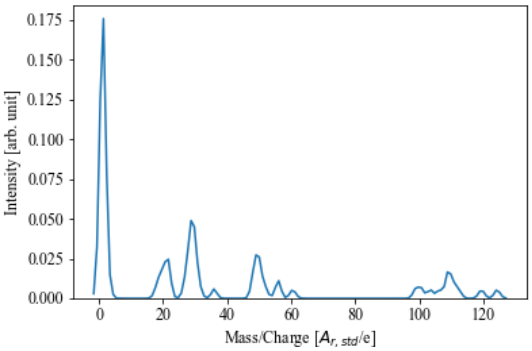


(d) $\overline{z} = 0.27$

Figure 20: Mass spectrometry plots for five up to eight electrons removed from the molecule. Bin width was set to 0.9 for (a)-(b), 0.4 for (c) and 0.8 for (d).



(a) $\overline{z} = 0.30$



(b) $\overline{z} = 0.33$

Figure 21: Mass spectrometry plots for nine and ten electrons removed from the molecule, the highest ionization levels simulated. Bin width was set to 0.9 for (a)-(b).

5 Discussion

The main focus of this thesis was to analyze how DNA is fragmented due to photoactivation, with the motivation that a deeper understanding of these processes might help improve radiation therapy for cancer treatment. However, this study has several limitations in more than one aspect, which will be discussed here along with an analysis of the results as well as a comparison with experimental results. As previously mentioned, this project investigated a considerably reduced model system in order to build up the understanding of the fragmentation process from the ground up. A simplified model made both the simulations, that were made beforehand, and the analysis of them easier to execute. Less amounts of data had to be processed as well, thus extracting desired information was quicker and more straightforward. For the extent of a bachelor's thesis, and considering the lack of similar studies of this form, these limitations were preferred. Model systems also give an indication of further needed studies, which is important for the progression of the research. Furthermore, only 110 simulations were analyzed in this work. More simulations for each ionization level would be needed to really decrease the dependence of starting configurations.

Photoactivation therapy (PAT) involves ionizing high-Z atoms, in this case iodine, built into the nucleotides, in order to cause Auger cascades which fragments the DNA. This is a targeted ionization, meant to hit specific core energy levels of iodine, and thus spare other tissue. The simulations that were analyzed in this study did not account for certain photon energies, they simply start with removing electrons spread across the molecule - this is simpler and also avoids the band gap problem. This is a known flaw of DFT, that the band gap, energy difference of the top of the valance band and the bottom of the conduction band, of materials are underestimated in size. Despite this choice, it is still relevant to investigate how this type of ionization fragments the molecule - as it could be comparable with the effects of the Auger cascades. It is also worth to note that even the idea of PAT has its own limitations, including how the lower energy X-rays would reach the cancer cell nuclei in an actual patient. Perhaps the process is restricted to shallow tumours. As of yet there have been lab trials only, studying isolated cells, and in the future several technical difficulties might arise when implementing the concept in clinical trials. How well the iodized nucleotides would build into the DNA when fed to patients is also an important point. It might not be built into the cancer cells to a sufficient extent. Despite the possible limitations, it is still meaningful to study advancements of PAT as the benefits of a successful treatment would include less risk of side effects. Attempting to save lives of people suffering from a heinous disease is a righteous cause, and improving treatments could also contribute to the society in other ways such as economically.

Analyzing the results of this thesis, it seems that the most sensitive bonds in 5-iodocytidine are three of the carbon-carbon covalent bonds in the sugar structure. Figure 13 suggests that one of these bonds, C3-C4, breaks on average at quite low ionization levels, $\overline{z} =$ 0.17-0.33 e/N, and in the higher ionization cases after merely 50-75 fs. Even at the lowest

of these levels, where the bond breaks, the molecule still remains intact as one fragment. Because even if C3-C4 breaks, the neighbouring bonds such as C2-C3 or C4-C5 are not broken at this level. Figure 12 visualises this result, it is one of the simulations for $\overline{z}$= 0.17 and it is clear that at 200 fs the C3-C4 bond breaks but no fragmentation occurs. Mass spectrometry plots for the lower ionization levels also suggest that up to $\overline{z}$= 0.20 the molecule gets more charged but stays unbroken, see figures 19 and 20.

Other carbon-carbon bonds in the sugar ribose break quite easily as well, possibly because they are purely covalent bonds while for instance a oxygen-hydrogen or oxygen-carbon bond have a polarity that strengthens the bond. Another possible explanation is that the least bound electrons in the molecule are found in the sugar. Thus when the simulations were made, that removed a certain number of least bound electrons, the origin of the ionization was in or close to the sugar structure. It would be interesting to analyze exactly where the electrons are removed in further work, for instance by looking at the change in charge distribution. These susceptible carbon-carbon bonds in the sugar result in the first fragments seen at ionization level $\overline{z} = 0.23$ - which seem to be a HCOH (hydroxymethylene) split from the rest of the molecule. See figure 14, where this fragmentation is visualised, and figure 13 where the mean bond integrity supports this. Examining the mass spectrometry plot for this ionization level, figure 20 (c), can reveal this result in another way. The HCOH group has a mass of approximately 30 standard atomic weights [17], and different charges of this fragment could correspond to the peaks around 20-35 $A_{r,std}/e$. The remaining larger fragment would then match the peaks around 50-65 $A_{r,std}/e$, as its mass would be around 340 standard atomic weights but higher charged (5 or 6 electrons removed from the larger fragment of the total 7 electrons).

It is however important to note that the occurrence of mass/charge ratios in these particular plots are ambiguous. Since fragment occurences are gathered from all the different starting configurations, eleven for each ionization level, this particular representation does not give information for separate simulation runs. Different types of fragments can have the same ratio and therefore it is not possible to directly identify peaks with specific fragments. One example being in the cases where the molecule remains intact for $\overline{z} =$ 0.23, the mass/charge ratio is then approximately 53 $A_{r,std}/e$. In other words within the same region as the larger fragment discussed above. Still, in this case the molecule is small and the information from the bond integrity graphs can provide a clearer picture of the fragmentation process. In future work it would be valuable to calculate the ratios of certain fragments for each peak. This is possible to do since the program can provide specific fragments for each run as well, not just collected into the same mass spectrometry plot as was done here. Doing so would provide very useful information for experimental results, where it can be difficult to analyze their mass spectrometry measurements.

When the ionization level rises to $\overline{z} = 0.27$-0.33, fragmentation occurs to a greater extent as further bonds become more likely to break. For instance the longest bond in the molecule, the iodine-carbon bond, breaks at these ionization levels after approximately

150 fs, see figure 15 (b). Even though the hexagon structure of the molecule seems rather stable, one of the bonds (N1-C6) break at the very end of the simulation, t = 200 fs, for the two highest ionizations. On the other side the nitrogen attaches the hexagon to the sugar ribose, bond C5-N1, which is kept intact throughout all the simulations. See figure 15 for the mean bond integrity and two examples of the highest ionization level runs in figure 16 and 18. Perhaps the stability of the ring comes from the delocalization of the shared electrons in this structure. Ring structures with carbon atoms tend to align p-orbitals and thus the shared electrons in the molecular orbital are delocalized, causing more stable bonds. However, as discussed previously it would be meaningful to analyze the charge distribution in order to make a more substantiated conclusion. While the hexagon ring remains as one fragment at these levels, the sugar "backbone" fragments in several smaller pieces - which pieces is not only dependent on the ionization level but also the starting geometry. Initial condition dependence has been discussed many times in this study, and it is reflected in the results despite there being eleven different runs for each ionization level. Analyzing the mass spectrometry plots for $\overline{z}$ = 0.27-0.33, figure 20 and 21, it is clear that there are more variations in the mass/charge ratios.

Another interesting point is the hydrogen atoms, some of which appear to break very rapidly at around 20-75 fs but only at the very highest values of $\overline{z}$. Only a few hydrogen atoms break free, such as H1 and H2 that were bound to C1, and also H9 from O3, while others oscillate a bit more but remain bound. See figure 17. It is interesting to see that some of the hydrogen atoms that are not separated, such as O1-H12 and N3-H7 in figure 17 (b) and (c), appear to vibrate with a lower frequency and extended displacement. This suggests that even these bonds are affected by the increased ionization level, the bond strength is weakened even though the bond does not break. The mass spectrometry plots for these two levels also suggest that single hydrogen atoms are released, as there are peaks close to 1 $A_{r,std}/e$, see figure 20. It is also evident that it occurs for many of the starting geometries as the peaks are significantly higher than for the other fragments.

Considering all the various results, it appears that when 5-iodocytidine is ionized gradually over the whole molecule it is most sensitive to bond breaks in the five carbon-sugar D-ribose structure. Small fragments of the sugar back bone partially occurs around $\overline{z}$ = 0.23 after 200 fs, while it is significantly fragmented at $\overline{z}$ = 0.30-0.33 after approximately 150-200 fs. While the fragmentation process varies between different initial conditions, results from eleven separate simulations for each ionization level provide some statistics to regard. It seems that the sugar section can fragment in somewhat altering pieces, but certain fragments are more likely to occur and damage to the molecule is highly probable at the highest ionization levels studied here. The most definite result for these $\overline{z}$ values suggest that a few specific hydrogen atoms are separated quite rapidly, this is however not as interesting when investigating how extensive damage to the molecule occurs - such that further on when regarding more accurate model systems could point at double strand breaks of a DNA helix. For the motivation behind this study, it is auspicious that the sugar ribose fragments quite easily since backbone breaks are essential to disrupt

the duplication of cancer cells.

It is also important to discuss the accuracy and credibility of this study. As stated several times previously, many approximations were made for the model system in relation to the motivation, which should be kept in mind as well. Regarding the simulations and method used to produce the results of this thesis, which are based on theory alone, a qualitative discussion of possible errors seems more appropriate than a quantitative analysis. Inaccuracies of the results most likely stem from the simulations, rather than the small rounding errors made in the Python program. General mistakes in the program have not been found, as the results were compared to and matched with the raw data from simulations using Avogadro to visualise a sample of runs. However, there might be an accumulating numerical error in the Hirshfeld charges. When the values are imported from the simulation files only three decimals are included, which could lead to a variation in the mass/charge ratios of the mass spectrometry plots. This can be observed using a more narrow bin width for the lower ionization levels, 0.10-0.20, where the eleven different runs all result in a charged but intact molecule. There is in fact some variation between the mass/charge ratios in these simulations, approximately 0.5 percent, which is why the width of the intensity curves are quite large in figure 19 (a)-(d) and 20 (a)-(b). Even though the single fragments should have the same total mass and charge for these cases they differ slightly. Since the mass is not changed, it is most likely due to the possible numerical error in the imported Hirshfeld charges. Changing the functions that import the charges could improve the accuracy, by storing the values with more than three decimals, and perhaps decrease this error in future work.

The simulations, that were not made in this study, probably have larger systematic errors that outweigh its numerical flaws. One example of a known systematic error in the simulations is the underestimation of bond lengths, by a few percent. As the simulations are bases on DFT calculations, they will inherit the approximations made in that theory - even though one of its drawbacks regarding band gaps was avoided. During the iterative process of calculating electron densities using DFT, the self consistency tolerance was set to 10^{-5} eV which would correspond to a numerical accuracy. While simulating the thermalization runs, random velocities were given and the bonds approximated as harmonic oscillators, and while the internal energy is preserved the temperature fluctuates - which could cause inconsistency over time. Ideally many more simulations should be made for each situation, to decrease the effects of these errors, and the known systematic errors should be considered or compensated for. More simulations would also be needed to gather more statistics. Since the fragmentation process varies significantly from one starting configuration to another, more than eleven separate runs should be made to produce more definite results. Increasing the number of simulations might reveal fragments that were not found in this study, while also providing more certain probabilities for the identified fragments.

6 Outlook

Many improvements of this study can be made, due to the largely reduced model system used here. An interesting next step would be to compare these results with simulations where the iodine is ionized specifically, in order to resemble the targeted photon energies for the core levels of iodine which is the key to photoactivation therapy. In both cases it would also be meaningful to analyze the changes in charge distribution of the molecule during the simulation, to see exactly where the ionization happens initially and how charge is carried. Additionally the simulation of Auger cascades can be improved by recreating the ionization stages in more detail. For instance simulations based on Monte Carlo methods could be used to simulate the cascade effects with several steps of electron excitation probabilities, for short time scales. Iodine could be targeted with a certain photon energy, and the following Auger cascades could be imitated better. Results from synchrotron experiments would also be more comparable then. Expanding the model system would also be desirable, such as analyzing the fragmentation of an iodized short DNA strand, and even further a double helix structure. Simulations and data analysis of these systems would be more complex, but would further deepen the understanding of how and when Auger cascades can cause double strand breaks in DNA. Thus gain useful theoretical results in the pursuit of improving PAT. Generally it would be advantageous to perform more than eleven simulations for every ionization level as well, to obtain more data and enhance the statistics. It is possible that some bonds break after much longer time than 200 fs, so it would be interesting to increase the duration of the simulations in order to identify further fragmentation processes. This is also suitable for comparing the results with experiments, since they detect fragments for a much longer time scale. The representation of the mass spectrometry plots in this work can also be improved to help decipher experimental results. Calculating a ratio of specific fragments for each peak, corresponding to a mass/charge value, could be done by extending the Python program slightly. This information would be very useful for the experiments since it can be difficult to identify the type of fragments that correspond to a peak in the ambiguous mass spectrometry results.

7 Conclusions

The ionization of an iodized nucleotide-like molecule, 5-iodocytidine, had been simulated for one up to ten electrons removed over the molecule - the least tightly bound electrons in the system were removed first. This study then analyzed the fragmentation process for these ionization levels, using Python language to write a program that returns mean bond integrity heatmaps for selected bonds and mass spectrometry plots for the last time step. Specific simulation runs were visualised in Avogadro. Analysis of the results include that the most sensitive bonds in 5-iodocytidine seem to be the carbon-carbon bonds in the sugar ribose. Up to an ionization level of $\overline{z} = 0.20$ e/N no fragmentation occurs, despite one or two single bonds breaking in the sugar backbone. As $\overline{z}$ gradually increased from 0.23 to 0.33, the molecule fragments in more pieces, mainly in the five-carbon sugar structure. Most of the larger fragments separate from each other around 150-200 fs. Occurring fragments generally depend on initial condition, although for $\overline{z} =$ 0.30-0.33 a few hydrogen atoms consistently disperse at 20-75 fs. Additional studies have to be made in order to deepen the understanding of photoactivation therapy. Even so, in this particular study of 5-iodocytidine it appears promising that the sugar backbone structure is most sensitive to fragmentation when the molecule is ionized.

References

[1] Bruce Alberts, et al. *Molecular biology of the cell*. Garland Science, Taylor and Francis Group, 2008

[2] *Cancer*, World Health Organization. [Updated 2018-09-12, read 2020-02-12]. Url: https://www.who.int/en/news-room/fact-sheets/detail/cancer

[3] Murat Faik Erdogan, et al. Radioactive iodine treatment in medullary thyroid carcinoma. *Nuclear Medicine Communications*, 27(4):359-62 (2006)

[4] J. Brierley, et al. Prognostic factors and the effect of treatment with radioactive iodine and external beam radiation on patients with differentiated thyroid cancer seen at a single institution over 40 years. *Clinical Endocrinology*, 63, 418–427 (2005)

[5] Balagurumoorthy, Pichumani Xu, Xiang Wang, Ketai Adelstein, S. Kassis, Amin. (2012). Effect of distance between decaying 125I and DNA on Auger-electron induced double-strand break yield. *International Journal of Radiation Biology*. 88. 998-1008. 10.3109/09553002.2012.706360.

[6] Bayart E, Pouzoulet F, Calmels L, Dadoun J, Allot F, Plagnard J, et al. (2017) Enhancement of IUdR Radiosensitization by Low-Energy Photons Results from Increased and Persistent DNA Damage. PLoS ONE 12(1): e0168395.doi:10.1371/journal.pone.0168395

[7] Richard "Zephyris" Wheeler, *Wikipedia*, image. [Published 2011, Viewed 2020-02-17]. Url: https://en.wikipedia.org/wiki/File:DNA_Structure%2BKey%2BLabelled.pn_NoBB.png

[8] Richard J. Roberts, *Encyclopædia Britannica*, image. [Copyright 1998, Viewed 2020-02-13]. Url: https://www.britannica.com/science/nucleic-acid/images-videos/media/1/421900/2659

[9] Richard J. Roberts, *Nucleic acid*, Encyclopædia Britannica. [Published 2019-08-07, read 2020-02-17]. Url: https://www.britannica.com/science/nucleic-acid

[10] Richard J. Roberts, *Encyclopædia Britannica*, image. [Copyright 2015, Viewed 2020-02-13]. Url: https://www.britannica.com/science/nucleic-acid/images-videos/media /1/421900/2658

[11] Xin Zhang, et al. Regarding the manuscript entitled "Association of Radioactive Iodine Treatment With Cancer Mortality in Patients With Hyperthyroidism". *Eur J Nucl Med Mol Imaging*, 46, 2410–2411 (2019)

[12] Rubino C, et al. Second primary malignancies in thyroid cancer patients. *British Journal of Cancer*, 89, 1638–1644 (2003)

[13] José M. Soler et al. The siesta method for ab-initio order-n materials simulation. *J.Phys.: Condens. Matter* (2002)

[14] Ibrahim E. Dawod. Structural integrity of highly ionized peptides. ISSN:1401-5757, UPTEC F 19035 (2019)

[15] Oscar Gråsäs, et al. Femtosecond bond-breaking and charge dynamics in ultra-charged aminoacids. *J. Chem. Phys.* 151, 144307 (2019)

[16] Hirshfeld, F.L. Bonded-atom fragments for describing molecular charge densities. *Theoret. Chim. Acta* 44, 129–138 (1977)

[17] *Periodic table*, Wikipedia, the free encyclopedia. [Updated 2020-05-04, read 2020-05-11]. Url: https://en.wikipedia.org/wiki/Periodic-table

[18] National Center for Biotechnology Information. PubChem Database. 5-Iodocytidine, CID=159359, https://pubchem.ncbi.nlm.nih.gov/compound/5-Iodocytidine (accessed on May 12, 2020)

[19] Avogadro: an open-source molecular builder and visualization tool. Version 1.2. http://avogadro.cc/

Appendix

A: Main python program

This is a program constructed to analyze the fragmentation of a molecule subjected to ionization, using data files from computer simulations that implemented Siesta software, for two bachelor theses at Uppsala University.

Authors: Ebba Koerfer and Emma Danielsson
Supervisors: Oscar Grånäs and Carl Caleman
Created autumn 2019 - spring 2020

First some useful modules are imported. "analyze_trajectories" contains various functions and classes, some of which we have written and some were already created.

[1]:
```
import numpy as np
import scipy as sp
from statistics import mean, stdev
from analyze_trajectories import *
import matplotlib.pyplot as plt
from elementdata import *
import seaborn as sns

plt.rcParams["font.family"] = "times new roman"    # Changing font style and␣
 ↪size for plots
plt.rc('font', size=12)
```

Files for thermalization runs are parsed, where thermalization list will contain analyze_trajectories objects (from a class specified there).

[2]:
```
runs=['thermalize_1.out','thermalize_2.out','thermalize_3.out','thermalize_4.
 ↪out','thermalize_5.out','thermalize_6.out',
      'thermalize_7.out','thermalize_8.out','thermalize_9.out','thermalize_10.
 ↪out']
thermalization_list=[]
for run in runs:
    time_pos, timeserie, orblegend, specieslegend, numberlegend =␣
 ↪parse_timestep(run)
    thermalization_list.append(time_pos)
```

Naming all atoms in molecule, and giving them unique indexes. Atom names and indexes can be converted easily using the dictionaries returned.

[3]:
```
index_to_atom, atom_to_index=make_atom_dictionary_from_timeserie(time_pos)
```

```
0 O  1 C  2 C  3 O  4 C  5 O  6 C  7 C  8 N  9 C  10 O  11 N  12 C  13 N  14 C  15␣
 ↪C  16 H  17 H  18 H  19 H  20 H  21 H  22 H  23 H  24 H  25 I  26 O  27 H  28 H
29 H
```

Creating a list where every list index corresponds to atom k, and the element is a list with neighbor

atom indexes j. Choosing a certain time (should be same neighbors always) and a maximum radius (Å) for what is a neighbor by looking at the bonds in the molecule structure.

[4]:
```
neighbors_list = get_neighborlist(time_pos[80],1.8)
neighbors_list[25].extend([14])     # The iodine-carbon bond is longer, and is␣
 ↪therefore manually added

print(f'Neighbor list[k][j]: {neighbors_list}')
```

```
Neighbor list[k][j]: [[1, 29], [0, 2, 16, 17], [1, 3, 4, 18], [2, 7], [2, 5, 6,
20], [4, 24], [4, 7, 26, 28], [3, 6, 8, 19], [7, 9, 15], [8, 10, 11], [9], [9,
12], [11, 13, 14], [12, 22, 23], [12, 15], [8, 14, 21], [1], [1], [2], [7], [4],
[15], [13, 23], [13, 22], [5], [14], [6, 27], [26], [6], [0]]
```

We want the mean value and standard deviation of the distance between each atom pair k,j in the system, which is both a mean over all the thermalization runs (with different initial conditions) and over the entire time-period for all of them. We use different lists and dictionaries to keep track of the neigbor indexes and their distances over time as well as for different thermalization runs.

In the end we have many mean values for the distance over time for each neighboring atom pair, for different thermalization runs, so we take the mean value of these mean values and print the results. In the printed results, e.g N1 means the first Nitrogen atom, and H4 means the fourth Hydrogen atom, not the corresponding atom_index (which in this case is index 8 and index 19).

[5]:
```
mean_distances_dict, distance_list = mean_distance_dict(thermalization_list,␣
 ↪index_to_atom, neighbors_list)
                                                        # Function is found␣
 ↪in module analyze_trajectories

for k in range(len(neighbors_list)):
    for j in neighbors_list[k]:
        print(f"Mean distance between {index_to_atom[str(k)]} and␣
 ↪{index_to_atom[str(j)]}: \t"

 ↪f"{mean(mean_distances_dict[str((index_to_atom[str(k)],index_to_atom[str(j)]))])}␣
 ↪Å")
        print(f'Standard deviation:␣
 ↪\t\t
 ↪\t{stdev(mean_distances_dict[str((index_to_atom[str(k)],index_to_atom[str(j)]))])
        }␣Å', end='\n\n')
```

```
Mean distance between O1 and C1:        1.430886085193834 Å
Standard deviation:                     0.0013001530695649522 Å

Mean distance between O1 and H12:       0.9701384298159741 Å
Standard deviation:                     0.0005357301508090135 Å

...
```

As an example we plot the distance between the first atom, C1, and its neighbors (from neighbor_list we get O1, C2, H1 and H2), as a function of time to see how it oscillates around the mean values.

[6]:
```
time = [x for x in range(len(thermalization_list[0]))]
i = 25
for j in neighbors_list[i]:
    fig, ax = plt.subplots()
    ax.plot(time, distance_list[i][str(j)])
    ax.set_ylim([1.5,2.5])
    ax.set(xlabel='Time [fs]', ylabel='Distance [Å]',
           title=f'Distance between atom pair {index_to_atom[str(i)]}␣
 ↪{index_to_atom[str(j)]}')
    #fig.savefig(f'dist{index_to_atom[str(i)]} {index_to_atom[str(j)]}')
    plt.show()
```

Choosing a atom pair of the form "('O1', 'C1')" and a value for the smearing parameter lambda, will return the thermalization run's bond integrity values as a function of bond distance. Then a comparison is made with the general form of the bond integrity function. The function "bond_broken_2", that we wrote, can be found in the module analyze_trajectories.

[7]:
```
atom_pair = "('I1', 'C8')"
atom_i = atom_pair.split("'")[1]
i = atom_to_index[atom_i]
atom_j = atom_pair.split("'")[3]
j = atom_to_index[atom_j]
lamda = 10

bond_dists = distance_list[int(i)][str(j)]
fig, ax = plt.subplots()
ax.plot(bond_dists, bond_broken_2(bond_dists,len(bond_dists),␣
 ↪mean(mean_distances_dict[atom_pair]),
                                   stdev(mean_distances_dict[atom_pair]), lamda),␣
 ↪c='orange')
ax.set(xlabel='Bond distance [Å]', ylabel='Bond integrity', title=f'Bond␣
 ↪integrity for atom pair {atom_i} {atom_j}')
plt.show()

d = np.arange(min(bond_dists),3,0.01)
fig, ax = plt.subplots()
ax.plot(d,bond_broken_2(d,len(d),mean(mean_distances_dict[atom_pair]),
                        stdev(mean_distances_dict[atom_pair]), lamda), label='BI␣
 ↪function')
ax.plot(distance_list[int(i)][str(j)],␣
 ↪bond_broken_2(distance_list[int(i)][str(j)],len(distance_list[int(i)][str(j)]),
```

```
 ↪mean(mean_distances_dict[atom_pair]), stdev(mean_distances_dict[atom_pair]),␣
 ↪lamda),
                                                label='BI thermalization run')
#ax.set_ylim([0,1])
ax.legend()
ax.set(xlabel='Bond distance [Å]', ylabel='Bond integrity', title=f'Bond␣
 ↪integrity for {atom_i} {atom_j} with lambda={lamda}')
fig.savefig(f'lamda{lamda} {atom_i} {atom_j}')
plt.show()
```

Now we want to remove all the doublets of atom pairs in the dictionary mean_distances_dict, as well as removing incorrect bonds (in this case only Hydrogen atoms were an issue as they were close to more than one atom but are in fact only ever bound to one), it is done using the following code:

[8]:
```
all_keys = list(mean_distances_dict.keys())
sorted_keys = [sorted(e) for e in list(mean_distances_dict.keys())]

index=[]
for i, key in enumerate(sorted_keys):
    index.append([i for i, keyi in enumerate(sorted_keys) if key==keyi])

for indexpair in index:
    if len(indexpair) > 1:
        if all_keys[indexpair[1]] in mean_distances_dict:
            del mean_distances_dict[all_keys[indexpair[1]]]

for k in index_to_atom.values():
    neighlist= [n for n in mean_distances_dict.keys() if str(k) in n]
    mainatom=str(k[0])
    if mainatom=="H":
        #print(k)
        for n in neighlist:
            #print(n)
            if n.count("H")>1 and n in mean_distances_dict.keys():
                del mean_distances_dict[n]
print(list(mean_distances_dict.keys()))
```

```
["('O1', 'C1')", "('O1', 'H12')", "('C1', 'C2')", "('C1', 'H1')", "('C1',
'H2')", "('C2', 'O2')", "('C2', 'C3')", "('C2', 'H3')", "('O2', 'C5')", "('C3',
'O3')", "('C3', 'C4')", "('C3', 'H5')", "('O3', 'H9')", "('C4', 'C5')", "('C4',
'O5')", "('C4', 'H11')", "('C5', 'N1')", "('C5', 'H4')", "('N1', 'C6')", ...]
```

Next the ionization runs from the simulation are parsed, in the same way as the thermalization runs - but ordered differently to keep track of which starting geometry and ionization number it is.

[9]:
```
runs2=[['startgeo0_ionization1.out', 'startgeo0_ionization2.out',␣
↪'startgeo0_ionization3.out',
     'startgeo0_ionization4.out', 'startgeo0_ionization5.out',␣
↪'startgeo0_ionization6.out', 'startgeo0_ionization7.out',
     'startgeo0_ionization8.out', 'startgeo0_ionization9.out',␣
↪'startgeo0_ionization10.out'],
     ['startgeo1_ionization1.out', 'startgeo1_ionization2.out',␣
↪'startgeo1_ionization3.out',
     'startgeo1_ionization4.out', 'startgeo1_ionization5.out',␣
↪'startgeo1_ionization6.out', 'startgeo1_ionization7.out',
     'startgeo1_ionization8.out', 'startgeo1_ionization9.out',␣
↪'startgeo1_ionization10.out'],
     ['startgeo2_ionization1.out', 'startgeo2_ionization2.out',␣
↪'startgeo2_ionization3.out',
     'startgeo2_ionization4.out', 'startgeo2_ionization5.out',␣
↪'startgeo2_ionization6.out', 'startgeo2_ionization7.out',
     'startgeo2_ionization8.out', 'startgeo2_ionization9.out',␣
↪'startgeo2_ionization10.out'],
     ['startgeo3_ionization1.out', 'startgeo3_ionization2.out',␣
↪'startgeo3_ionization3.out',
     'startgeo3_ionization4.out', 'startgeo3_ionization5.out',␣
↪'startgeo3_ionization6.out', 'startgeo3_ionization7.out',
     'startgeo3_ionization8.out', 'startgeo3_ionization9.out',␣
↪'startgeo3_ionization10.out'],
     ['startgeo4_ionization1.out', 'startgeo4_ionization2.out',␣
↪'startgeo4_ionization3.out',
     'startgeo4_ionization4.out', 'startgeo4_ionization5.out',␣
↪'startgeo4_ionization6.out', 'startgeo4_ionization7.out',
     'startgeo4_ionization8.out', 'startgeo4_ionization9.out',␣
↪'startgeo4_ionization10.out'],
     ['startgeo5_ionization1.out', 'startgeo5_ionization2.out',␣
↪'startgeo5_ionization3.out',
     'startgeo5_ionization4.out', 'startgeo5_ionization5.out',␣
↪'startgeo5_ionization6.out', 'startgeo5_ionization7.out',
     'startgeo5_ionization8.out', 'startgeo5_ionization9.out',␣
↪'startgeo5_ionization10.out'],
     ['startgeo6_ionization1.out', 'startgeo6_ionization2.out',␣
↪'startgeo6_ionization3.out',
     'startgeo6_ionization4.out', 'startgeo6_ionization5.out',␣
↪'startgeo6_ionization6.out', 'startgeo6_ionization7.out',
     'startgeo6_ionization8.out', 'startgeo6_ionization9.out',␣
↪'startgeo6_ionization10.out'],
     ['startgeo7_ionization1.out', 'startgeo7_ionization2.out',␣
↪'startgeo7_ionization3.out',
```

```
     'startgeo7_ionization4.out', 'startgeo7_ionization5.out',␣
↪'startgeo7_ionization6.out', 'startgeo7_ionization7.out',
     'startgeo7_ionization8.out', 'startgeo7_ionization9.out',␣
↪'startgeo7_ionization10.out'],
     ['startgeo8_ionization1.out', 'startgeo8_ionization2.out',␣
↪'startgeo8_ionization3.out',
     'startgeo8_ionization4.out', 'startgeo8_ionization5.out',␣
↪'startgeo8_ionization6.out', 'startgeo8_ionization7.out',
     'startgeo8_ionization8.out', 'startgeo8_ionization9.out',␣
↪'startgeo8_ionization10.out'],
     ['startgeo9_ionization1.out', 'startgeo9_ionization2.out',␣
↪'startgeo9_ionization3.out',
     'startgeo9_ionization4.out', 'startgeo9_ionization5.out',␣
↪'startgeo9_ionization6.out', 'startgeo9_ionization7.out',
     'startgeo9_ionization8.out', 'startgeo9_ionization9.out',␣
↪'startgeo9_ionization10.out'],
     ['startgeo10_ionization1.out', 'startgeo10_ionization2.out',␣
↪'startgeo10_ionization3.out',
     'startgeo10_ionization4.out', 'startgeo10_ionization5.out',␣
↪'startgeo10_ionization6.out', 'startgeo10_ionization7.out',
     'startgeo10_ionization8.out', 'startgeo10_ionization9.out',␣
↪'startgeo10_ionization10.out']]

ionization_list=[]
for geo in runs2:
    for run in geo:
        time_pos, timeserie, orblegend, specieslegend, numberlegend =␣
↪parse_timestep(run)
        ionization_list.append(time_pos)
```

A certain ionization run can be chosen, and a .xyz file stored which can be opened in e.g Avogadro and watch the animation of the process.

[10]:
```
geo = 0
ion = 7
write_xyz_anim('test.xyz',ionization_list[10*geo + ion-1])
```

Now we want to create a dictionary where the keys are the atom pairs from mean_distances_dict, and their respective values are lists of lists - where the indexes give us a certain startgeometry and ionization run - and then there are bond distances for every time step (200 fs). The dimension of this dictionary is: ion_dict[atom_pair][startgeometry][ionization number][time step]. OBS: since the ionization numbers go from 1-10, the index in the dictionary is in fact ionization number - 1.

[11]:
```
n_geo = 11    # Number of different starting geometries
n_ion = 10    # Number of different ionization numbers (# electrons removed)
ion_dict = {}
for key in list(mean_distances_dict.keys()):
```

```
    index_i = int(atom_to_index[key.split("'")[1]])
    index_j = int(atom_to_index[key.split("'")[3]])
    for geo in range(n_geo):
        for ion in range(n_ion):
            current_run=(n_ion*geo)+ion
            ion_dist_list =␣
↪[dist_timestep(ionization_list[current_run][t],index_i,index_j)
                              for t in range(len(ionization_list[current_run]))]

            if key not in ion_dict:
                ion_dict[key]=[None]*n_geo
                ion_dict[key]=[[None]*n_ion for x in ion_dict[key]]
            else:
                pass
            ion_dict[key][geo][ion]=ion_dist_list
```

Different atom pairs can now be choosen, along with a desired starting geometry and ionization, to plot the bond integrity as a function of time. Statistics from mean_distances_dict is used in the calculation of each bond integrity.

[12]:
```
atom_pairs = ["('I1', 'C8')", "('O1', 'H12')"]
g = 5
ion_run = 8
ion = ion_run - 1
time = [t for t in range(len(ionization_list[0]))]
for atom_pair in atom_pairs:
    fig, ax = plt.subplots()
    ax.plot(time,␣
↪bond_broken_2(ion_dict[atom_pair][g][ion],len(ion_dict[atom_pair][g][ion]),
                                  mean(mean_distances_dict[atom_pair]),␣
↪stdev(mean_distances_dict[atom_pair]),10))
    i = atom_pair.split("'")[1]
    j = atom_pair.split("'")[3]
    ax.set(xlabel='Time [fs]',ylabel='Bond integrity',title=f'Bond integrity for␣
↪atom pair {i} {j} with g={g} and i={ion_run}')
    plt.show()
```

In order to gather more information in one plot, we produce heatmaps where the mean value of bond integrity, over all starting geometries, is shown in a color gradient as a function of time and ionization level $\bar{z}$. The ionization level is the number of electrons removed as a fraction of number of atoms in the molecule, (e/N). Yellow color (bond integrity=1) means an intact bond and blue color (bond integrity=0) suggests a broken bond.

[13]:
```
atom_pair = "('C7', 'C8')"
i = atom_pair.split("'")[1]
j = atom_pair.split("'")[3]
```

```
mean_g_dist = [[] for _ in range(n_geo)]
all_g = [[] for _ in range(n_geo)]
for ion in range(n_ion):
    all_g[ion] = [ion_dict[atom_pair][g][ion] for g in range(n_geo) if␣
 ↪len(ion_dict[atom_pair][g][ion]) == len(time)]

    for t in range(len(time)):
        mean_g_dist[ion].append(mean([all_g[ion][g][t] for g in␣
 ↪range(len(all_g[ion]))]))


z_mesh = np.divide(np.linspace(0,n_ion,10),np.float(30))
time_mesh = [t for t in range(len(ionization_list[0]))]
all_mean_integrity = np.transpose([bond_broken_2(mean_g_dist[current_i],␣
 ↪len(mean_g_dist[current_i]),
                    mean(mean_distances_dict[atom_pair]),␣
 ↪stdev(mean_distances_dict[atom_pair]),10) for current_i in range(10)])


fig, ax = plt.subplots()
p = plt.contourf(z_mesh, time_mesh, all_mean_integrity, levels=100, vmin=0.,␣
 ↪vmax=1.0,
             alpha=1, cmap='plasma')

x_labels = [round(x/30,2) for x in range(1,11)]   #Specifying the ticks on␣
 ↪x-axis, to show simulated values exactly, 1/30-10/30
plt.xticks([x/30 for x in range(1,11)], x_labels)

fig.colorbar(p, ticks=[0,0.2,0.4,0.6,0.8,1], label='Mean bond integrity')
plt.clim(0,1)
plt.rc('font', size=12)
#fig.savefig(f'BI {i} {j}')
ax.set(xlabel='$\overline{z}$ [e/N]', ylabel='Time [fs]', title=f'Mean bond␣
 ↪integrity for atom pair {i} {j}')
plt.show()
```

Next the bond integrity values, the statistical results and a cutoff limit for when we want a bond to be considered broken is used to call a function we created (see module analyze_trajectories), it returns fragments at time t=200 fs. Dimension of this list is: total_fragments[starting geometry][ionization number-1][fragment][atoms in fragment].

[14]:
```
total_fragments = frags_from_dists(mean_distances_dict, atom_to_index, ion_dict,␣
 ↪lamda=10, cutoff_BI=0.5)

# Printing an example from the fragment list
geo = 3
ion = 8
```

```
print(total_fragments[geo][ion-1])
```

```
[['O1', 'C1', 'H12', 'C2', 'H1', 'H2', 'O2', 'H3', 'C5', 'N1', 'H4', 'C9', 'C6',
'O4', 'N2', 'C7', 'N3', 'C8', 'H7', 'H8', 'H6'], ['C3', 'O3', 'C4', 'H5', 'H9',
'O5', 'H11', 'H10'], ['I1']]
```

This next cell creates a dictionary with the Hirsh charges for every atom, a certain starting geometry and ionization, and every timestep. The charges are found in the same files as parsed above, by using the function parse_hirsh (see module analyze_trajectories). The dimensions of the dictionary is: atom_charge_dict[atom name][11*startgeo + ionization-1][time step].

[15]:
```
all_filenames = []
for geo in runs2:
    for run in geo:
        all_filenames.append(run)

atom_charge_dict = {}
for file in all_filenames:
    charges = parse_hirsh(file)
    for atom in range(len(charges[0])):
        if atom_charge_dict.get(index_to_atom[str(atom)]) != None:
            atom_charge_dict[index_to_atom[str(atom)]].append([charges[t][atom]␣
 ↪for t in range(len(charges))])
        else:
            atom_charge_dict[index_to_atom[str(atom)]] = []
            atom_charge_dict[index_to_atom[str(atom)]].append([charges[t][atom]␣
 ↪for t in range(len(charges))])
```

First some element data is imported. Then lists are made for the total masses and charges of each fragment in total_fragments. This is then used to plot the occurence of every mass/charge ratio of fragments for a specific ionization level $\bar{z}$, for every starting geometry - both as a regular histogram and a density plot. The bin widths should be regulated for each run of ionization level, to visualize the data correctly. The list ratios of mass/charge, "mass_charge", can be printed to compare with. Masses are given in standard atomic weights, and charges in electron charges.

[16]:
```
ed=ElementData()

frag_weights=[0]*n_geo
frag_weights=[[0]*n_ion for x in frag_weights]
for geo in range(n_geo):
    for ion in range(n_ion):
        frag_weights[geo][ion]=[0]*len(total_fragments[geo][ion])


for geo in range (len(frag_weights)):
    for ion in range(len(frag_weights[geo])):
        for frag in range(len(frag_weights[geo][ion])):
            for atm in range(len(total_fragments[geo][ion][frag])):
```

```
                frag_weights[geo][ion][frag]+=ed.
 ↪elementweight[total_fragments[geo][ion][frag][atm][0]]


frags_charges=[0]*n_geo
frags_charges=[[0]*n_ion for x in frags_charges]
for geo in range(n_geo):
    for run in range(n_ion):
        frags_charges[geo][run] = [0 for x in total_fragments[geo][run]] #The␣
 ↪same # of frags and charges
        for frag in range(len(total_fragments[geo][run])):
            for atm in total_fragments[geo][run][frag]:
                frags_charges[geo][run][frag] += atom_charge_dict[atm][n_ion*geo␣
 ↪+ run][-1]

ion_run = 10
i = ion_run - 1
mass_charge = []
for g in range(n_geo):
    mass_charge.extend(np.ndarray.tolist(np.divide([m for m in␣
 ↪frag_weights[g][i]], [c for c in frags_charges[g][i]])))


print(f'mass/charge ratios: {mass_charge}')

fig, ax = plt.subplots()
plt.hist(mass_charge, bins=np.arange(min(mass_charge), max(mass_charge) + 0.5, 0.
 ↪9))
ax.set(xlabel='Mass/Charge [$A_{r, std}$/e]', ylabel='Counts [#]', title='Mass␣
 ↪spectrometry for $\overline{z}$'
                                                        f'={ion_run}/30  at
 ↪t={len(time)} fs')
plt.show()

fig, ax = plt.subplots()
sns.kdeplot(mass_charge, bw=0.9)
ax.set(xlabel='Mass/Charge [$A_{r, std}$/e]', ylabel='Intensity [arb. unit]',␣
 ↪title='Mass spectrometry for $\overline{z}$'

 ↪f'={ion_run}/30  at  t={len(time)} fs')
#fig.savefig(f'mass i {ion_run}')
plt.show()
```

```
mass/charge ratios: [55.71781626971851, 29.57910295616718, 31.201182795698926, ...␣
 ↪]
```

B: analyze-trajectories.py

```
#!/usr/bin/env python
import os, sys
import numpy as np
import shutil
import matplotlib.pyplot as plt
from statistics import mean, stdev
from numpy import linalg as LA
from scipy import interpolate
from itertools import combinations

# To analyze preparsed with "the naming convention", run e.g. ./analyze_preparsed.
 ↪py ALA 1 10 1 10 C4 "H10 H11 H12" "Alanine C-H methyl bonds"
class atom:
    def __init__(self):
        self.name=""
        self.rvec=np.zeros(3)
        self.dvec = np.zeros(3) # direct
        self.pdos = np.zeros(1)
        self.sumdos = np.zeros(1)
        self.color = 0 # color index is used to find color from list in plotter,␣
 ↪otherwise it's messier to change.
        self.phonons = []
        self.speciesName = ""
        self.speciesNumber = 0
        self.specnum=0
        self.speciesZNumber = 0
        self.mass = 0.0
        self.hirshfeldcharge=0.0
        self.mulliken_legend=[]
        self.mulliken_charges=[]

    def distance(self,center=np.asarray([0.0,0.0,0.0])):
        return np.linalg.norm(np.subtract(self.rvec,center))
        # return float(np.sqrt(self.rvec[0]**2+self.rvec[1]**2+self.rvec[2]**2))

    def in_cluster(self,maxrad,center=np.asarray([0.0,0.0,0.0]),minrad=0.0):
        return (self.distance(center) <= float(maxrad) and self.distance(center)␣
 ↪>= float(minrad))

class lattice:
     def __init__(self):
         self.bravais=np.zeros((3,3))
         self.reciprocal=np.zeros((3,3))
         self.atoms=[]
         self.lattparam=0.0
         self.indSpecies=[]   # Number of atoms for one individual specie
```

```
        self.numSpecies=0    # Number of species
        self.indSpeciesNames=[]
        self.coordtype=""

def deleteContent(fName): #Clearing the previous input file
            with open(fName, "w"):
                pass

def Date_and_Time():
    from time import gmtime, strftime #Current date and time
    t = strftime("%Y-%m-%d %H:%M:%S", gmtime())
    return t

def parse_text_bond_data(filename):
    bond_integrity=[]
    f=open(filename,'r')
    for i, line in enumerate(f.readlines()):
        if line.split()[-1][-1] is not "]":
            full_line = line.split()
        else:
            full_line = full_line + line.split()
            bond_integrity.append(np.asarray(filter(None,[element.strip('[]') for␣
 ↪element in full_line[1:]])).astype(np.float))
    return np.asarray(bond_integrity)

def get_neighborlist(timestep,rmax):
    neighborlist=[]
    rmin = 0.1 # Do not include self
    for i, atm in enumerate(timestep):
        neighborlist.append(find_atoms_within_radius(timestep,atm.rvec,rmax,rmin))
    return neighborlist

def find_atoms_within_cartesian(cluster,xlim,ylim,zlim):
    indices=[]
    for i, atm in enumerate(cluster):
        within = ((float(xlim[0])<= float(atm.rvec[0]) <= float(xlim[1])) and
                  (float(ylim[0])<= float(atm.rvec[1]) <=float(ylim[1])) and
                  (float(zlim[0])<= float(atm.rvec[2]) <=float(zlim[1])))
        if within:
            indices.append(i)
    return indices

def find_atoms_within_radius(cluster,center,rmax,rmin=0.0):
    indices=[]
    for i, atm in enumerate(cluster):
        if (atm.in_cluster(rmax,center,rmin)):
            indices.append(i)
```

```
    return indices

def get_neighborlist(timestep,rmax):
    neighborlist=[]
    rmin = 0.1 # Do not include self
    for i, atm in enumerate(timestep):
        neighborlist.append(find_atoms_within_radius(timestep,atm.rvec,rmax,rmin))
    return neighborlist

#checking if element is int
def Is_Int(s):
    try:
        int(s)
        return True
    except ValueError:
        return False

#Parsing .ANI file
def parse_ANI(filename):
    f = open(filename, 'r')
    contents = f.readlines()
    f.close()
    atoms=[]
    time_serie=[]
    for i in range(len(contents)):
        if (Is_Int(contents[i])):
            atoms_in_timestep=int(contents[i].split()[0])
            for j in range(i+2,i+2+int(atoms_in_timestep)):
                atoms.append(atom())
                atoms[-1].rvec=[float(contents[j].split()[k]) for k in range(1,4)]
                atoms[-1].name=contents[j].split()[0]
            time_serie.append(atoms)
            atoms=[]
    return atoms_in_timestep, time_serie

def distR(D):
    N = np.loadtxt(D, dtype=np.float, delimiter=',')
    Q = [np.linalg.norm(a-b) for a, b in combinations(N, 2)]
    return Q

def dist_timestep(timestep,atom1,atom2):
    return np.linalg.norm(np.subtract(timestep[atom2].rvec,timestep[atom1].rvec))

def bond_broken_2(dist, T, mean, sigma, lamda):
    B=[]
    for num in range(0,T):
        e = (1 + np.exp(lamda*(dist[num]-mean-sigma-0.5)))**(-1)
        B.append(e)
```

```
    return np.asarray(B)

def parse_hirsh(filename):
    f = open(filename, 'r')
    contents = f.readlines()
    f.close()
    timesteps=[]
    charges=[]
    numatm=0
    for i in range(len(contents)):
        if ("NumberOfAtoms" in contents[i]):
            numatm=contents[i].split()[1]
        if ("Atom #    Qatom  Species" in contents[i]):
            for j in range(i+1,i+1+int(numatm)):
                charges.append(contents[j].split()[1])
            timesteps.append(np.asarray(charges,dtype=float))
            charges=[]
    return timesteps

def mean_distance_dict(thermalization_list, index_to_atom, neighbors_list):
    """Returns a dictionary with keys in the form '(Atom_A_index, Atom_B_index)'␣
 ↪with values in the form of lists, where the elements are mean values for the␣
 ↪distance between atom A and B over time for each thermalization run in␣
 ↪thermalization list (which contains information from parse_timestep function).␣
 ↪index_to_atom is a dictionary made from make_atom_dictionary_from_timeserie()␣
 ↪and neighbors_list from get_neighborlist()"""
    mean_distance_lexi =  # Dictionary with every atom pair_kj, in tuple-form, as␣
 ↪keys and their values are mean distances over
                          # all time positions for every thermalization run

    for run_index in range(len(thermalization_list)):
        distance_list = []
        for k in range(len(neighbors_list)): # atom_k, atom_j = atom pair_kj
            distance_lexi =
            for j in neighbors_list[k]:      # distance_list has dicts for every␣
 ↪atom_k with neighbor atom_j as key and with
                                             # distance_atom pair_kj(t) as values
                distance_lexi[str(j)] =␣
 ↪[dist_timestep(thermalization_list[run_index][t_i],k,j) for t_i in

 ↪range(len(thermalization_list[run_index]))]
            distance_list.append(distance_lexi)
            for j in neighbors_list[k]:
                if mean_distance_lexi.
 ↪get(str((index_to_atom[str(k)],index_to_atom[str(j)]))) != None:
```

```
 ↪mean_distance_lexi[str((index_to_atom[str(k)],index_to_atom[str(j)]))].
 ↪extend([mean(distance_list[k][str(j)])])
                else:

 ↪mean_distance_lexi[str((index_to_atom[str(k)],index_to_atom[str(j)]))] =␣
 ↪[mean(distance_list[k][str(j)])]
    return mean_distance_lexi, distance_list


def frags_from_dists(mean_distances_dict, atom_to_index, ion_dict, lamda,␣
 ↪cutoff_BI):
    """Returns a list of lists for which fragments there are at the last timestep,␣
 ↪given a lambda value and a cutoff value for the bond integrity. The dimensions␣
 ↪of the list are: total_fragments[geo][ion][fragment][atom]."""
    l=lamda #Lambda value
    n_geo = 11
    n_ion = 10
    broken_bonds_dict=
    for bond in list(mean_distances_dict.keys()):
        broken_bonds_dict[bond]=[None]*n_geo
        broken_bonds_dict[bond]=[[None]*n_ion for x in broken_bonds_dict[bond]]
        for geo in range(n_geo):
            for ion in range(n_ion):
                BI =␣
 ↪bond_broken_2(ion_dict[bond][geo][ion],len(ion_dict[bond][geo][ion]),
                                mean(mean_distances_dict[bond]),␣
 ↪stdev(mean_distances_dict[bond]),l)
                if BI[-1] <= cutoff_BI:
                    broken_bonds_dict[bond][geo][ion]="broken"
                else:
                    broken_bonds_dict[bond][geo][ion]="intact"

    total_fragments=[None]*n_geo
    total_fragments=[[None]*n_ion for x in total_fragments]

    for geo in range(n_geo):
        for ion in range(n_ion):
            polyatomic=[]
            monoatomic=[]
            for bond in broken_bonds_dict.keys():
                atoms= [x for x in atom_to_index.keys() if bond.split("'")[1]==x␣
 ↪or bond.split("'")[3]==x]
                if broken_bonds_dict[bond][geo][ion]=="intact":
                    found=False
                    merged=False
                    for j in range (len(polyatomic)):
```

```
                        if (atoms[0] in polyatomic[j]) and (atoms[1] not in␣
↪polyatomic[j]):
                            for k in range(len(polyatomic)):
                                if atoms[1] in polyatomic[k] and atoms[0] not in␣
↪polyatomic[k]:
                                    polyatomic[j].extend(polyatomic[k])
                                    polyatomic[k]=[]
                                    merged=True
                                    break
                            if not merged:
                                polyatomic[j].append(atoms[1])
                            found=True
                        elif (atoms[1] in polyatomic[j]) and (atoms[0] not in␣
↪polyatomic[j]):
                            for k in range(len(polyatomic)):
                                if atoms[0] in polyatomic[k] and atoms[1] not in␣
↪polyatomic[k]:
                                    polyatomic[j].extend(polyatomic[k])
                                    polyatomic[k]=[]
                                    merged=True
                                    break
                            if not merged:
                                polyatomic[j].append(atoms[0])
                            found=True
                        elif (atoms[0] in polyatomic[j]) and (atoms[1] in␣
↪polyatomic[j]):
                            found=True
                        else:
                            pass
                    if found==False: #This is the case where the atoms do not␣
↪occur anywhere in the current version of "polyatomic"
                        polyatomic.append(atoms)
                elif broken_bonds_dict[bond][geo][ion]=="broken":
                    atom0_in_somefrag=False
                    atom1_in_somefrag=False
                    for other_bond in broken_bonds_dict.keys():
                        if broken_bonds_dict[other_bond][geo][ion]=="intact":
                            if (atoms[0]==other_bond.split("'")[1] or␣
↪atoms[0]==other_bond.split("'")[3]):
                                atom0_in_somefrag=True
                            if (atoms[1]==other_bond.split("'")[1] or␣
↪atoms[1]==other_bond.split("'")[3]):
                                atom1_in_somefrag=True
                    if not atom0_in_somefrag and atoms[0] not in monoatomic:
                        monoatomic.append(atoms[0])
                    if not atom1_in_somefrag and atoms[1] not in monoatomic:
                        monoatomic.append(atoms[1])
```

```
                    else:
                        pass
                else:
                    print("broken_bonds_dict["+bond+"]["+geo+"]["+ion+"] was not␣
 ↪assigned a value")

            for i in range (len(monoatomic)):
                monoatomic[i]=[monoatomic[i]]
            empty_indices=[]
            for j in range(len(polyatomic)):
                if not polyatomic[j]:
                    empty_indices.append(j)
            for empty_index in empty_indices:
                del polyatomic[empty_index]
            fragments=[]
            fragments.extend(polyatomic)
            fragments.extend(monoatomic)
            total_fragments[geo][ion]=fragments
    return total_fragments

def write_xyz_anim(filename,timesteps,skipstep=1):
    f = open(filename,'w')
    for i, step in enumerate(timesteps):
        if (np.mod(i,skipstep)<0.5):
            f.write(str(len(step))+"")
            f.write('Timestep: '+str(i*skipstep)+"")
            for atm in step:
                f.write(str(atm.name)+" "+str(atm.rvec[0])+" "+str(atm.rvec[1])+"
 ↪"+str(atm.rvec[2])+"")

def parse_hirsh_from_file(ion,lastion,acid):
    all_mean_hirsh=[]
    all_std_hirsh=[]
    for ionstage in range(ion,lastion):
        hirsh=[]
        print("acid, ionstage: ", str(acid), str(ionstage))
        for geostage in range(geometry,lastgeometry):
            Sim = './startgeo0_ionization1'.format(geostage,ionstage)
            os.chdir(Sim)
            try:
                hirsh.append(parse_hirsh("./stdout"))
            except:
                print("Failed to parse Hirshfeld for: 0/startgeo1_ionization2".
 ↪format(acid, geostage, ionstage))
            os.chdir("..")
        #print np.asarray(hirsh).mean(0).shape
        mean_data_name='0_hirshfeld_charge_1_hirshrun.dat'.format(acid,ionstage)
```

```
        np.savetxt(mean_data_name,np.asarray(hirsh).mean(0))
        mean_data_name='0_stdev_hirshfeld_charge_1_hirshrun.dat'.
 ↪format(acid,ionstage)
        np.savetxt(mean_data_name,np.asarray(hirsh).std(0))
        all_mean_hirsh.append(np.asarray(hirsh).mean(0))
        all_std_hirsh.append(np.asarray(hirsh).std(0))
    return all_mean_hirsh, all_std_hirsh

def parse_eigenvalues(filename):
    f = open(filename, 'r')
    contents = f.readlines()
    f.close()
    timeserie_eig=[]
    timeserie_occ=[]

    for i, line in enumerate(contents):
        if ("Timestep" in line):
            current_step=int(line.split()[1])
            print(current_step)
            num_eigens=int(line.split()[3])
            print(num_eigens)
            eigenvalues=[]
            occupations=[]
            for j in range(i+1,i+num_eigens+1):
                eigenvalues.append(np.asarray(contents[j].split()[0:2],␣
 ↪dtype=float))
                occupations.append(np.asarray(contents[j].split()[3:5],␣
 ↪dtype=float))
            # Transpose to get spin-channels as timeserie[itime][ispin][:]
            timeserie_eig.append(np.transpose(np.asarray(eigenvalues)))
            timeserie_occ.append(np.transpose(np.asarray(occupations)))

    return np.asarray(timeserie_eig), np.asarray(timeserie_occ)

def read_preparsed_hirsh(acid,ion):
    mean_data_name='0_hirshfeld_charge_1_hirshrun.dat'.format(acid,ion)
    mean_hirsh=np.loadtxt(mean_data_name, dtype=np.float)
    mean_data_name='0_stdev_hirshfeld_charge_1_hirshrun.dat'.format(acid,ion)
    std_hirsh=np.loadtxt(mean_data_name, dtype=np.float)
    return mean_hirsh, std_hirsh

def make_atom_dictionary(filename):
    natoms, md_verlet = parse_ANI(filename)
    atomdict=
    name_list=[atm.name for atm in md_verlet[0]]
    for i, atm in enumerate(md_verlet[0]):
        new_atom_number=name_list[0:i].count(atm.name)+1
```

```
        key=str(i)
        value=atm.name+str(new_atom_number)
        atomdict[key]=value
    inverted_dict = dict(map(reversed, atomdict.items()))
    return atomdict, inverted_dict

def make_atom_dictionary_from_timeserie(timeserie):
    atomdict=
    name_list=[atm.name for atm in timeserie[0]]
    #print(name_list), print(type(name_list[0]))
    for i, atm in enumerate(timeserie[0]):
        print(str(i), atm.name)
        new_atom_number=name_list[0:i].count(atm.name)+1
        key=str(i)
        value=atm.name+str(new_atom_number)
        atomdict[key]=value
    inverted_dict = dict(map(reversed, atomdict.items()))
    return atomdict, inverted_dict

def parse_xyz(filename):
    xyz=[]
    f = open(filename, 'r')
    contents = f.readlines()
    f.close()
    for line in contents:
        xyz.append(np.asarray(line.split()[1:4], dtype=float))
    return np.transpose(np.asarray(xyz))

def parse_timestep(filename, outfile=None):
    f = open(filename, 'r')
    contents = f.readlines()
    #här print("filename: "+str(filename))
    print("length of file: "+str(len(contents)))
    f.close()
    numatm=0
    basissize='SZP'
    time_pos=[]
    time_mulliken=[]
    timesteps=[]
    specieslegend=
    numberlegend=
    mulls=[]
    orblegend=[]
    for i in range(len(contents)):
        if ("NumberOfAtoms" in contents[i]):
            numatm=int(contents[i].split()[1])
            print("Number of Atoms: "+str(numatm))
            break
```

```
  for i in range(len(contents)):
      if ("PAO.BasisSize" in contents[i]):
          basissize=str(contents[i].split()[1])
          print("Basis Size: "+str(basissize))
          break

  for i in range(len(contents)):
      if ("SpinPolarized" in contents[i]):
          if ("true") in contents[i]:
             spins=2
          else:
             spins=1
          break
  print("Spin components: "+str(spins))

  for i in range(len(contents)):
      if ("AtomicSpecies" in contents[i]):
          #print "Found AtomicCoord..."
          for j in range(i+1,len(contents)):
              print(str(j-i)+"  "+str(contents[j].split()))
              numberlegend[str(j-i)]=str(contents[j].split()[3])
              if ("AtomicCoordinatesAndAtomicSpecies" in contents[j+1]):
              #    print "Found!"
                  break
          else:
              continue
          break

  for i in range(len(contents)):
      if ("ChemicalSpeciesLabel" in contents[i]):
          for j in range(i+1,len(contents)):
              specieslegend[str(contents[j].split()[0])]=str(contents[j].
↪split()[2])
              if ("ChemicalSpeciesLabel" in contents[j+1]):
                  break
          else:
               continue
          break
  # print numberlegend
  # print specieslegend


  for i in range(len(contents)):
      if ("(Ang)" in contents[i])  and ("outcoor" in contents[i]):
              atoms=[]
              for j in range(i+1,i+numatm+1):
```

```
                    atoms.append(atom())
                    atoms[-1].rvec=[float(contents[j].split()[k]) for k in␣
↪range(0,3)]
                    atoms[-1].name=speciesslegend[numberlegend[str(contents[j].
↪split()[4])]]
                time_pos.append(atoms)
        elif ("(Bohr)" in contents[i]) and ("outcoor" in contents[i]):
                atoms=[]
                for j in range(i+1,i+numatm+1):
                    atoms.append(atom())
                    #print([contents[j].split()[k] for k in range(0,3)])
                    atoms[-1].rvec=np.multiply([float(contents[j].split()[k]) for␣
↪k in range(0,3)], 0.529177249)  # Convert Bohr to Angstrom
                    atoms[-1].name=speciesslegend[numberlegend[str(contents[j].
↪split()[4])]]
                time_pos.append(atoms)

# Approximately two lines per atom times number of spins + overhead of a few lines
    approx_mulliken_size=numatm*(3+spins*2)

    for i in range(len(contents)):
        if ("mulliken: Atomic and Orbital Populations:" in contents[i]):
            mulls, orblegend= parse_mulliken(contents[i:
↪i+approx_mulliken_size],numatm,basissize,spins,outfile)
            time_mulliken.append(mulls)

    return time_pos, time_mulliken, orblegend, speciesslegend, numberlegend
```

C: input.fdt

```
NumberOfAtoms 30
NumberOfSpecies 5

%block  ChemicalSpecieslabel
    1  7  N
    2  6  C
    3  8  O
    4  1  H
    5  53  I
%endblock  ChemicalSpecieslabel

SystemName              XYZ       # Descriptive name of the system
SystemLabel             XYZ       # Short name for naming files
SpinPolarized           false

AtomicCoordinatesFormat NotScaledCartesianAng # Format for coordinates
AtomicCoorFormatOut     Ang

PAO.BasisSize           DZP          # (DZP) Double-z + polarization
XC.functional           GGA
XC.authors              PBE
MeshCutoff              100. Ry      # Mesh cutoff. real space mesh (Ry)

# SCF options
MaxSCFIterations      400            # Maximum number of SCF iter

DM.MixingWeight 0.1      # New DM amount for next SCF cycle
DM.Tolerance 0.00001
DM.NumberPulay 5

UseSaveData           true
DM.UseSaveDM          false           # to use continuation files
MD.UseSaveXV          true

WriteCoorStep           .true.
WriteForces             .true.
# NeglNonOverlapInt     false         # Neglect non-overlap interactions

SolutionMethod          diagon        # OrderN or Diagon
ElectronicTemperature   300 K         # Temp. for Fermi smearing (Ry)

# WriteWaveFunctions      true
# WriteDenchar            true

MD.TypeOfRun          Verlet
MD.InitialTemperature 300 K
```

```
MD.TargetTemperature  300 K
MD.InitialTimeStep    1
MD.FinalTimeStep      1000
MD.LengthTimeStep     1.0 fs
MD.LengthTimeStep     0.024189 fs # 0.5*hbar/Ru

WriteMDHistory       true   # MD, MDE files
WriteMDXmol          true   # ANI file

LatticeConstant         40 Ang
%block LatticeVectors
  1.0000  0.0000  0.0000
  0.0000  1.0000  0.0000
  0.0000  0.0000  1.0000
%endblock LatticeVectors

%block AtomicCoordinatesAndAtomicSpecies
   -0.47843744    4.47965268    1.11932912   3
   -1.22585866    4.46205286    2.32652251   2
   -0.51803596    3.53641261    3.30716254   2
   -0.47976067    2.23553841    2.71779413   3
    0.94064779    3.88178189    3.68384856   2
    1.28040515    3.35929662    4.95915973   3
    1.70978321    3.04540156    2.65677456   2
    0.87436478    1.75214392    2.64054210   2
    1.10445790    0.92235416    1.45324927   1
    1.51106360   -0.50476018    1.61477222   2
    1.75944725   -0.95665884    2.72225222   3
    1.60570341   -1.26321543    0.48142491   1
    1.36367320   -0.75932563   -0.71269647   2
    1.59155611   -1.58259829   -1.76759268   1
    0.86766976    0.58592821   -0.89423439   2
    0.71986441    1.37312706    0.22589322   2
   -1.32426037    5.47924461    2.77093720   4
   -2.25026562    4.05877008    2.15401222   4
   -1.11926507    3.48939714    4.24485668   4
    1.13874967    1.09603677    3.50454164   4
    1.16129728    4.96946080    3.59604739   4
    0.26094469    2.37851578    0.20290395   4
    1.88874673   -2.52925683   -1.56308320   4
    1.35083535   -1.27813236   -2.70683149   4
    0.91872641    3.92813370    5.65606542   4
    0.32356178    1.28233081   -2.79452489   5
    3.04150935    2.88009818    3.04078214   3
    3.44639649    2.19543651    2.48167193   4
    1.59357373    3.54953941    1.66541929   4
   -0.82418095    5.16395551    0.52738346   4
%endblock AtomicCoordinatesAndAtomicSpecies
```