IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia do Barreiro

GONÇALO MIGUEL FONSECA MONTEIRO

ANALYSIS OF RESIDUE INTERACTION NETWORKS IN MOLECULAR DYNAMICS SIMULATIONS

Relatório de Dissertação do Mestrado em

Engenharia Biológica e Química

ORIENTADORAS

Professora Doutora Marta Justino

Professora Doutora Raquel Barreira

SUPERVISOR

Doutor Gonçalo Justino

Dezembro 2022

GONÇALO MIGUEL FONSECA MONTEIRO

ANALYSIS OF RESIDUE INTERACTION NETWORKS IN MOLECULAR DYNAMICS SIMULATIONS

JÚRI

Presidente: Professora Doutora Maria de Lurdes de Figueiredo Gameiro, Escola Superior de Tecnologia do Barreiro, Instituto Politécnico de Setúbal

Supervisor: Doutor José Gonçalo Deira Duarte de Campos Justino, Centro de Química Estrutural, Instituto Superior Técnico, Universidade de Lisboa

Vogal: Doutor Miguel Machuqueiro, Faculdade de Ciências, Universidade de Lisboa

Dezembro 2022

AGRADECIMENTOS

Gostaria de aproveitar esta oportunidade para, em primeiro lugar, agradecer à Escola Superior de Tecnologia do Barreiro, ESTB/IPS por disponibilizar os recursos necessários para a realização deste projeto. Mesmo com uma pandemia pelo meio que acabou por condicionar o normal funcionamento as coisas acabaram por se encaminhar e permitiu chegar a este ponto do Mestrado.

Não menos importante uma palavra de reconhecimento à Escola Superior de Tecnologia de Setúbal, ESTS/IPS por todos os conhecimentos que me foi passado ao longo dos meus 3 anos de licenciatura e que certamente me facilitou a longo de todo o processo de elaboração deste projeto e do Mestrado.

Quero deixar também uma grande palavra de agradecimento ao Professor Doutor Gonçalo Justino (ESTB/IPS), que desde o primeiro momento sempre me ajudou e acompanhou na elaboração deste projeto. Mesmo não estando este tema no conjunto de temas apresentados, quando questionado sobre a possibilidade de fazer um projeto na área da programação se prontificou a encontrar uma solução que fosse ao encontro dos meus interesses e que fizesse sentido no contexto do Mestrado, permitindo não só aumentar a motivação na realização do mesmo bem como o crescimento profissional uma vez que consegui fazer a ligação com a área em que trabalho atualmente. Um sincero obrigado.

Um agradecimento para a Professora Lurdes Gameiro, coordenadora do Mestrado em Engenharia Biológica e Química, por todo o ajuda e apoio prestando durante estes anos de Mestrado.

Um obrigado especial também para a Professora Doutora Marta Justino e a Professora Doutora Raquel Barreira que enquanto orientadoras deste projeto asseguraram o bom funcionamento do mesmo e permitiram que o mesmo se realizasse.

Um obrigado também para a Accenture, empresa em qual trabalho atualmente, e mais concretamente para o projeto em que estou incluído, por toda a facilidade prestada aquando da necessidade de tratar assuntos relacionados com o Mestrado, nunca foram colocadas limitações ou imposições.

Em último lugar, mas não menos importante, um grande agradecimento aos meus familiares e aos meus amigos próximos por todo o apoio, em especial aos meus pais Sónia e Paulo, ao meu irmão André e à minha namorada Sara.

Este trabalho foi parcialmente financiado pelo projeto PTDC/QUI-QAN/32242/2017, financiado pela Fundação para a Ciência e a Tecnologia.

RESUMO

É sabido que a estrutura de uma proteína não é estática, mas sim dinâmica. Porém, a análise dinâmica da sua estrutura não é de todo fácil, muito por via das dimensões dos sistemas a analisar. Por outro lado, a variedade de fatores que pode causar perturbações num sistema biológico também dificulta a análise. Nesse sentido, as simulações de dinâmica molecular (MD) apresentam-se como uma ferramenta muito importante pois geram toda uma biblioteca de dados a partir de uma determinada simulação. A análise detalhada destes dados é um processo complexo e demorado que requer diversas ferramentas computacionais.

Neste projeto foi utilizado a MDAnalysis, uma biblioteca *open source* orientada a objetos para análise estrutural e temporal de resultados de dinâmica molecular, em particular de trajetórias de simulação e de estruturas de proteínas. Está escrita na linguagem Python, com algum código de desempenho crítico em C, recorrendo ainda a funcionalidades do pacote NumPy. Adicionalmente, foi criado um *script* em Python que tentou facilitar e otimizar o processo com via a obter uma interface que permita ao utilizador selecionar o tipo interações a analisar, necessitando apenas de fornecer como input os ficheiros *gro* e *xtc* da dinâmica molecular de uma simulação de sistema biológico. O *script* criado, por sua vez, faz todo o processo de seleção, tratamento, análise e representação dos resultados com maior interesse da simulação fornecida sobre a forma de ficheiro *.pdf,* bem como em formato *.csv* com os dados detalhados da análise de cada simulação.

O *script* desenvolvido foi aplicado ao estudo da capacidade da transferrina, uma proteína envolvida no transporte de ferro, associado ao anião carbonato, transportar também vanádio. Foram analisadas as ligações para todos os tipos de interações possíveis envolvendo os metais Fe(III) e V(III) e o anião sinergístico carbonato, nos diferentes estados de protonação possíveis, de modo a validar o funcionamento do código desenvolvido.

Foi observado que as conformações mais fechadas da proteína estabelecem mais interacções do que a conformação aberta, sendo a distância média na conformação fechada inferior à na conformação relaxada. O anião favorecido através da análise das interações entre os metais e os aniões para todas as proteínas, revelou ser o ácido carbónico, juntamente com o Fe(III) uma vez que este estabelece mais interações do que V(III). Estes resultados correspondem ao que é conhecido da literatura par esta proteína, permitindo validar o funcionamento do código desenvolvido. Estes resultados permitem concluir que o código origina resultados fiáveis.

PALAVRAS-CHAVE: Dinâmica Molecular, POO - Programação Orientada a Objetos, Python, Redes de interação de aminoácidos, MDAnalysis.

ABSTRACT

It is known that the structure of the protein is not static, but its analysis is not at all easy, due to the dimensions of the studies, the variety of factors that can cause disturbances in a biological system also make these analyzes difficult. In this sense, molecular dynamics (MD) simulations are a very important tool because they generate a whole library of data from a given simulation. Uhis is a complex and time-consuming process that requires several computational tools to perform a detailed analysis.

MDAnalysis was used in this project, it's an *open source* object-oriented library for structural and temporal analysis of molecular dynamics (MD), simulation's trajectories and structures analysis of individual proteins. It's written in Python language with some critical code in C, it also uses features from the NumPy package. Additionally, a python *script* was created in order to improve and optimize the process with an interactive user interface that would allow selecting the type of interactions to be analyzed, only needing to provide the .gro and .xtc files of the molecular dynamics of a molecular dynamics simulation as input. The script created makes the entire process of selection, treatment, analysis and representation easier. The results are provided in the form of .pdf as well as .csv.

The developed script was applied to the study of the ability of transferrin, a protein involved in the transport of iron, associated with the carbonate anion, to also transport vanadium, as well as its ability to use other synergistic anions. Interactions were analyzed for all types of possible interactions involving metals - Fe(III) and V(III) - and the different protonation states of carbonate, in order to validate the script.

It was observed that the closed and relaxed conformations of the protein were more likely to establish bonds than the open conformation. When there were coincident interactions, the average distance in the more closed conformation values were inferior to those of the relaxed conformation. The favored anion through the analysis of interactions between metals and anions for all proteins turned out to be carbonic acid, together with Fe(III) since this establishes more interactions than V(III). Additionally, in the case of the open conformation, interactions with cysteine residues were observed, although average values of distance of interactions are very high. These results indicate that the developed code is working as expected.

KEYWORDS: Molecular Dynamics, Object-oriented Programming, Python, Aminoacid interaction networks, MDAnalysis.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

SÍMBOLOS E ABREVIATURAS

ALA	Alanina
ARG	Arginina
ASN	Asparagina
ASP	Aspartato
CL	Ião cloreto
COA	Ião carbonato, CO_3^{2-}
COB	Ião hidrogenocarbonato, HCO_3^-
COC	Ácido carbónico, H_2CO_3
CPU	Unidade central de processamento (*Central Process Unit*)
CSV	Ficheiro do tipo *Comma Separated Values*
CYS	Cisteína
FE3	Ião ferro(III)
GLY	Glicina
GLU	Glutamato
GLN	Glutamina
GPU	Unidade de processamento gráfico (*Graphics Processing Unit*)
HIS	Histidina
ILE	Isoleucina
LEU	Leucina
LYS	Lysina
MD	Dinâmica Molecular (*Molecular Dynamics*)
MET	Metionia

MPI	Interface de passagem de mensagem (*Messaging Passing Interface*)
NA	Ião sódio´
PDB	Banco de dados de proteínas (*Protein Data Bank*)
PHE	Fenilalanina
PRO	Prolina
SER	Serina
TYR	Tirosina
THR	Treonina
TRP	Triptofano
VAL	Valina

ERRATA

Nas figuras 7 a 20 e nas figuras A-1 a A-51, no título do eixo das ordenadas, onde se lê “Distance (nm)” deve ler-se “Distance (Å)”.

1. INTRODUÇÃO

O estudo da estrutura macromolecular é um ponto chave na compreensão da biologia. A função biológica depende de interações moleculares, e estas são uma consequência de estruturas macromoleculares. Desde as determinações iniciais da estrutura de proteínas na década de 50, tanto no mundo das proteínas quanto no mundo dos ácidos nucleicos, o aumento no conhecimento de como as estruturas macromoleculares são construídas tem sido contínuo. Atualmente, o banco de dados de proteínas (PDB) contém mais de 110.000 entradas, incluindo mais de 100.000 proteínas, 2.800 ácidos nucleicos e aproximadamente 20.000 pequenas moléculas complexadas com macromoléculas [1].

Assim sendo o conhecimento de regras moleculares bem como o conhecimento estrutural permitem a compreensão de fenómenos biológicos básicos, tais como mecanismos de regulação enzimática, transporte através de membranas, construção de grandes estruturas como ribossomas, entre outros. O estudo de interações proteína-proteína é um dos campos com mais evolução nos últimos tempos na bioquímica. No entanto, apesar de sua enorme utilidade, as estruturas armazenadas no banco de dados PDB fornecem apenas uma visão parcial da estrutura 3D. Tanto a proteína quanto os ácidos nucleicos são entidades flexíveis e a sua dinâmica pode desempenhar um papel fundamental na determinação da sua funcionalidade. As proteínas sofrem mudanças conformacionais significativas durante o desempenho da sua função. Como regra, a formação de qualquer complexo proteico implica algum tipo de reorganização estrutural. Isso pode ser facilmente verificado apenas comparando algumas entradas PDB de uma determinada proteína que diferem apenas num pequeno ligando [1].

No caso dos ácidos nucleicos, as mudanças conformacionais são ainda mais complexas. O B-DNA padrão tem uma estrutura relativamente simples em comparação com proteínas ou RNAs complexos. No entanto, é uma molécula extremamente plástica que sofre grandes mudanças conformacionais para se adaptar aos seus parceiros de interação [2]. A ligação de fatores de transcrição ao DNA, por exemplo, não é apenas dependente do reconhecimento da sequência de DNA, mas também uma consequência direta da capacidade da molécula de DNA de se adaptar à superfície da proteína. A abordagem tradicional para entender a influência da conformação na função macromolecular é acumular estruturas experimentais que cobrem o espaço conformacional [1]. Isso levou à criação de estruturas cristalinas para macromoléculas em vários ambientes, ou macromoléculas complexadas com diferentes moléculas, e contribui para a enorme redundância observada nas estruturas resolvidas experimentalmente [1].

1.1. DINÂMICA MOLECULAR

A simulação por dinâmica molecular (MD) foi desenvolvida pela primeira vez no final dos anos 70, e surgiu da necessidade de simular centenas de átomos em sistemas com relevância biológica, desde proteínas inteiras até grandes complexos macromoleculares como nucleossomas ou ribossomas [3]. As simulações de sistemas de grande escala estão cada vez mais otimizadas e neste momento apresentam tempos de processamento muito menores do que no início. Simulações de aproximadamente 500.000 átomos são comuns e relativamente rápidos desde que os recursos computacionais apropriados estejam disponíveis [1]. Esta notável melhoria é em grande parte uma consequência do uso de computação de alto desempenho e da simplicidade do algoritmo de dinâmica molecular básico, representado na Figura 1, em que a partir das forças entre partículas de um sistema, caracterizadas através do *force field*[1], é possível calcular a aceleração de cada partícula e, consequentemente, a posição de cada partícula ao longo do tempo [4] .

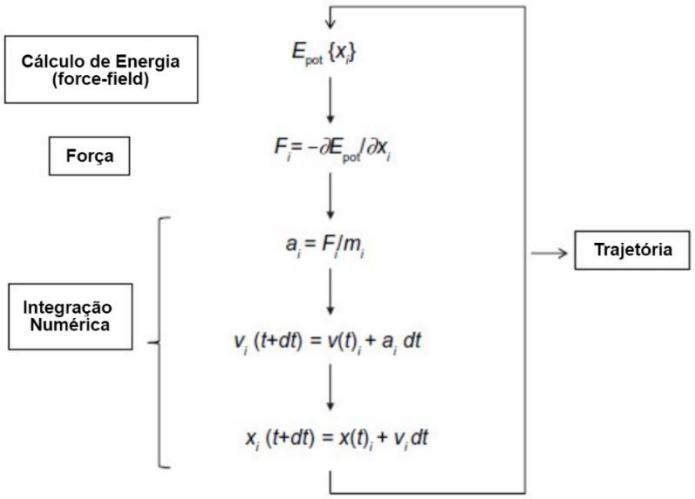

Figura 1 - Algoritmo Genérico de Simulação por Dinâmica Molecular. Adaptado de Hospital *et al.* (2015). E_{pot} - energia potencial; t – tempo da simulação; dt - tempo de iteração; x -coordenadas dos átomos, F- componente da força; a – acelaração; m – massa atómica; v – velocidade.

O resultado da simulação, a trajetória, é uma lista de coordenadas de átomos para cada tempo de simulação (ou instantâneo). A representação do solvente é uma questão-chave na definição do sistema. Várias abordagens foram testadas, mas foi demonstrado que o mais eficaz é também o mais simples, a representação explícita das moléculas do solvente, embora tenha como consequência o aumento do tamanho dos sistemas simulados. O solvente explícito é capaz de recuperar a maioria dos efeitos de solvatação do solvente real, incluindo os de origem entrópica, como o efeito hidrofóbico [5]. Assim que o sistema é construído, as forças que atuam sobre cada átomo são obtidas por equações a partir do *force field*, onde a

[1] Um *force field* é o conjunto de equações e parâmetros utilizados para estimar forças entre átomos (partículas) nas moléculas e também entre moléculas [16].

energia potencial é deduzida a partir da estrutura molecular. A simplicidade da representação do *force field* deve-se a características moleculares, em particular ao facto de as biomoléculas estudadas por MD, como as proteínas e os ácidos nucleicos, serem poliméricas [6] . Assim, a partir de um conjunto de parâmetros reduzido, que cubram os monómeros, é possível simular vastíssimos conjuntos de biomoléculas.

Uma vez obtidas as forças que atuam sobre os átomos individuais, a lei do movimento de Newton é usada para calcular as acelerações e velocidades e atualizar as posições dos átomos. As equações de movimento são essencialmente a segunda lei de Newton ($\vec{F} = \boldsymbol{M}\vec{q}$) Aqui, $\vec{q}$ é o vetor de coordenadas, $\boldsymbol{M}$ é a matriz de massa do sistema e $\vec{F}$ é o vetor de forças atuando nas coordenadas. As forças conservativas são calculadas a partir do gradiente da energia do sistema. A dinâmica molecular resolve essas equações por integração de tempo, ou seja, é guardado o registo para cada *frame* da simulação. Este exercício equivale a calcular as forças atuantes e a distribuição de massa em cada instante e de seguida resolver a aceleração e as velocidades dos corpos em movimento. Os graus de liberdade nas equações de movimento são tipicamente a posição de todos os átomos envolvidos na simulação. Para cada molécula, esses graus de liberdade podem ser formulados alternativamente como coordenadas internas [7].

A dinâmica molecular tem muitas aplicações na modelação de proteínas, sendo particularmente utilizada para simular complexos proteína-ligando. A facilidade de manipulação de fatores permite explorar estados próximos ao equilíbrio. Em ambiente fechado, e a uma temperatura suficientemente alta, os estudos de dinâmica molecular podem superar as barreiras de energia que são proporcionais a essa temperatura, podendo ser usada como uma ferramenta de validação para a estrutura da proteína, avaliando a estabilidade da molécula de proteína ao longo do tempo de simulação. Para um complexo proteína-ligando, a dinâmica molecular pode ser usada para estudar a geometria e a força das interações entre as duas moléculas [7].

Conforme referido anteriormente, o desenvolvimento tecnológico fortaleceu em grande parte os estudos computacionais de dinâmica molecular. A atual geração de computadores beneficia de processamentos em paralelo, o que permite diminuir os tempos de execução drasticamente. Ao longo do tempo surgiram numerosos pacotes de *software* para simulação por MD, estando os mais populares resumidos na Tabela 1.

Os códigos de simulação mais populares (AMBER, CHARMM, GROMACS, NAMD) são compatíveis há muito tempo com a interface paralela MPI (*messaging passing interface*), quando um grande número de núcleos de computador pode ser usado simultaneamente, o que pode reduzir bastante o tempo de computação (Habenicht, Paddison et al. 2012). A estratégia geral é utilizar a localização das interações, ou seja, distribuir o sistema a simular entre os processadores. Esta estratégia é chamada de decomposição espacial, e assim apenas um pequeno fragmento do sistema deve ser simulado em cada processador [1]. A divisão mais eficiente não se baseia na lista de partículas, mas sim na sua posição no espaço. A comunicação entre processadores também é reduzida, uma vez que apenas aqueles que simulam regiões vizinhas necessitam de compartilhar informações. A maioria dos principais códigos MD já foi também preparada para GPU (Graphics Processing Unit), existindo inclusive

até mesmo código escritos especificamente para serem usados exclusivamente em GPUs. A simulação em GPUs isolados ou combinada com MPI é, atualmente, a estratégia padrão para simulações de MD de alto rendimento [8].

Tabela 1 - Pacotes de *software* de dinâmica molecular mais comuns. Adaptado de Loschwitz *et al*. (2020)

Software	Interface	Tipo de Licença	Website
AMBER	CLI	Proprietary	https://ambermd.org/
CHARMM	CLI / GUI	Proprietary	https://www.charmm.org
Desmond	GUI	Academic	https://www.deshawresearch.com/resources_desmond.html
GROMACS	CLI	Open-source	http://www.gromacs.org/
NAMD	CLI	Academic	https://www.ks.uiuc.edu/Research/namd/
LAMMPS	CLI / GUI	Open-source	https://lammps.sandia.gov/

Os pacotes de *software* diferem na disponibilidade de algoritmos implementados, influenciando diretamente também o seu desempenho computacional, e no seu suporte relativamente a vários *force fields* (FFs) [9]. No caso de GROMACS e NAMD, pode-se escolher entre uma ampla variedade de FF's padrão e personalizados (por exemplo, todos os tipos de FFs CHARMM, OPLS-AA e AMBER), enquanto os mecanismos MD CHARMM e AMBER suportam apenas os seus próprios FFs. Além disso, alguns pacotes de software MD não estão disponíveis gratuitamente, como por exemplo o AMBER; porém, o seu pacote de análise correspondente AmberTools está disponível gratuitamente para fins académicos. Por fim, aquando da escolha do *software* a usar, o utilizador deve também considerar quais as ferramentas de análise que são fornecidas pelos programas. Por exemplo, o GROMACS tem a grande vantagem de vir com um conjunto diversificado de ferramentas de análise, abrangendo tanto análises de propriedades de proteínas quanto de membranas, que podem ser facilmente modificadas e redistribuídas. Adicionalmente o NAMD é implementado no VMD, o que lhe confere recursos de visualização poderosos, uma interface gráfica (GUI), seleções dinâmicas de átomos e permite a leitura de vários formatos de arquivo, incluindo aqueles obtidos por todos os pacotes MD comuns [8].

1.2. MDANALYSIS

MDAnalysis é uma ferramenta orientada a objetos, escrita e compilada em Python, e que permite analisar trajetórias de dinâmica molecular geradas em CHARMM, Gromacs, Amber, e NAMD, entre outros. Tem a capacidade de ler ficheiros de vários formatos, incluindo ficheiros *pdb* e trajectórias. Tem também a possibilidade fazer a seleção de átomos para utilização conjunta com Gromacs, CHARMM, VMD e PyMol.

Esta ferramenta permite a leitura de trajetórias de dinâmica molecular, bem como a consulta das coordenadas atómicas através de matrizes em NumPy. Apresenta-se como uma ferramenta flexível e relativamente rápida para tarefas de análise complexas. Esta flexibilidade permite, por exemplo, manipular trajetórias e escrever as mesmas em vários formatos.

A MDAnalysis é um pacote Python que fornece classes para analisar dados de trajetórias de dinâmica molecular. Uma vez que é orientado a objetos, trata átomos, grupos de átomos e trajetórias como objetos diferentes.

O MDAnalysis utiliza bibliotecas numéricas/algébricas rápidas como ATLAS, LAPACK ou MKL para melhorar a velocidade e poderosas bibliotecas Python como NumPy e SciPy que fornecem classes e funções otimizadas para componentes importantes de código científico, como matrizes multidimensionais ou rotinas de álgebra linear [10, 11] . A MDAnalysis facilita o desenvolvimento rápido de código com base em Python, uma linguagem de programação poderosa e extensível, fácil de aprender, que foi considerada particularmente útil na comunidade científica.

Cada objeto tem um número de operações definidas em si mesmo (também conhecidas como “métodos”) e também contém valores que descrevem o objeto (“atributos”). Por exemplo, um objeto "AtomGroup” possui um método/função center_of_mass() que calcula o centro de massa do grupo de átomos. Um outro exemplo é o atributo chamado "residues" que devolve todos os resíduos que pertencem ao grupo. Usando métodos de seleção como o "select_atoms()" podem criar-se novos objetos [10].

A classe Universe contém informações topológicas e estruturais e mantém uma lista de todos os átomos do sistema, através de uma instância “AtomGroup” chamada “atoms”, no sistema. Os métodos selectAtoms() fornecem uma interface para o mecanismo de seleção e devolvem uma instância AtomGroup. Uma trajetória MD é obtida através do atributo trajetória, que é uma instância de uma classe Reader (NA, Overview over MDAnalysis – MDAnalyses Documentation).

O MDAnalysis também possui um sub-módulo de análise, que contém várias classes e ferramentas predefinidas.

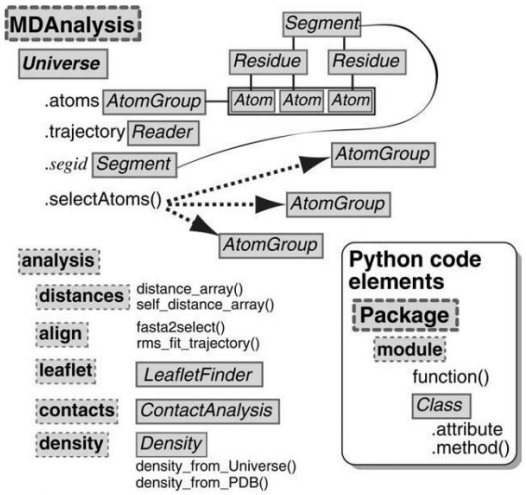

Figura 2 - Organização Estrutural da Biblioteca MDAnalysis. Adaptado de Michaud-Agrawal *et al.* (2011).

A classe Universe contém um AtomGroup de todas as instâncias Atom que podem ser acedidas através do atributo Universe.atoms. Este apresenta vários AtomGroups gerados automaticamente, um para cada identificador de segmento da topologia (Thomas, Oliver et al. 2011). Podem ser obtidas informações das instâncias individuais do átomo bem como de um AtomGroup inteiro fazendo a *query* aos atributos ou aos métodos. A classe Timestep representa o estado atual do sistema num determinado período da trajetória.

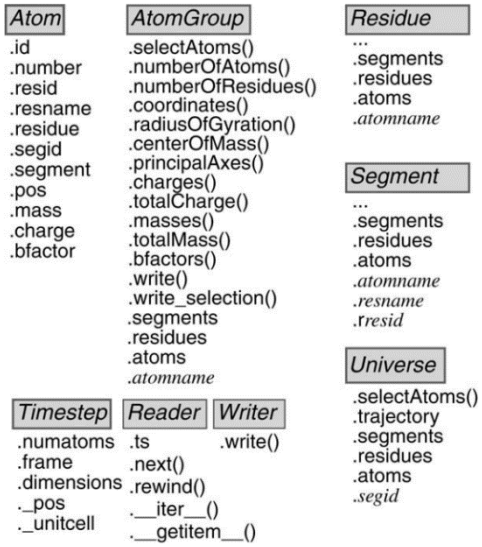

Figura 3 - Layout das Classes Mais Importantes em MDAnalysis. Adaptado de Michaud-Agrawal *et al*. (2011).

1.3. TRANSFERRINA DE SORO HUMANA

A transferrina humana, hTf, é uma proteína plasmática responsável pelo transporte de ferro, sob a forma de ião Fe(III), no organismos. O transporte através de uma proteína impede que este metal participe em reações de oxidação, que poderiam contribuir para o desequilíbrio da homeostasia redox [12]. O ião Fe(III) é normalmente coordenado pelos resíduos Asp63, Tyr95, Tyr188 e His249, bem como por dois átomos de oxigénio do anião carbonato, que desempenha aqui um papel sinergístico, permitindo manter uma geometria de coordenação octaédrica, e reforçando a ligação do metal à proteína através do estabelecimento de interações variadas com a proteína (Figura 4). A transferrina pode também ligar outros metais, como V(III), recorrendo ao carbonato como anião sinergístico.

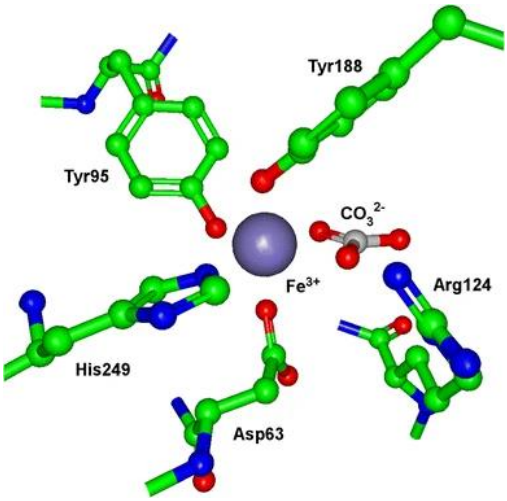

Figura 4 - Local de ligação do ferro no lobo N da transferrina humana (hTf). Adaptado de Mathies *et al*. (2015).

A interação do metal e do anião carbonato com a proteína pode ser feita através de diferentes tipos de interações intermoleculares, envolvendo não só interações eletrostáticas do tipo *salt bridge*, mas possivelmente interações com os resíduos aromáticos de aminoácidos presentes no local de ligação. Assim, escolheu-se o sistema hTf – Fe(III)/V(III) – carbonato como caso de estudo para aplicar o código a desenvolver.

1.4. OBJETIVOS

O objetivo deste trabalho foi desenvolver um *script* capaz de analisar trajetórias obtidas por simulação de dinâmica molecular e de identificar as principais interações intermoleculares (não covalentes) no sistema estudado. Como caso de estudo, foi aplicado a simulações da transferrina humana, simulada na presença de Fe(III) ou V(III), com o anião sinergístico carbonato presente nas diferentes formas possíveis de protonação.

As simulações de dinâmica molecular foram desenvolvidas in house por outros membros do grupo de investigação como parte de um projeto mais abrangente centrado na caracterização das interações da transferrina com diferentes metais.”

2. SCRIPT

Uma vez que a ferramenta do MDAnalysis dispunha das funcionalidades consideradas fundamentais para complementar a criação de um *script* em Python, avançou-se com essa abordagem. Assim, foi desenvolvido um *script* na linguagem de programação orientada a objetos Python, cujo objetivo passou pela criação de uma interface para o utilizador que lhe desse a possibilidade de escolher o tipo de interações que deseja analisar e, consecutivamente, gerar ficheiros *pdf* com detalhes das distâncias e dos ângulos das interações que cumpram os parâmetros definidos como aceitáveis. A ideia passou sempre por ter um *script* de fácil compreensão para qualquer que seja o utilizador. Este foi desenvolvido pronto a poder correr em qualquer diretoria em que seja colocado pois, aquando da primeira execução, o *script* cria as pastas necessárias à boa realização da execução. Por parte do utilizador apenas é necessário garantir que os ficheiros de posições (formato *gro*) e de trajetórias (formato *xtc*) estão na mesma diretoria que o próprio *script*. É também criado um ficheiro de registo (log) de cada execução, onde vão sendo escritos em tempo real todas as ações que o *script* executa, bem como as respetivas informações e data/hora correspondentes. Isto permite não só fazer *debug* ou análise de erro das corridas, mas também análise de *performance* através do tempo de execução de cada tarefa. O tema de "*housekeeping*"/manutenção foi também tido em conta, uma vez que são criados e mantidos na diretoria onde são executadas pastas denominadas de "*Archive*", "*Logs*" e "*Runs*" onde são guardadas versões anteriores dos resultados obtidos em formato *zip* para minimizar o espaço ocupado, mantendo somente na diretoria onde está o *script* os dados da execução que se encontra a correr.

2.1. INTERFACE

A interação inicial com o *script* criado é feita através da seleção das interações a analisar (Figura 5), que permite selecionar a análise de interações π - π, de interações π - anião/catião, e de interações π - HX. Cada seleção leva a um menu que permite seleções mais específicas, conforme apresentado na Figura 5

Estas interações são particularmente importantes na coordenação do metal na transferrina, em que o local de ligação do metal é constituído por vários resíduos aromáticos e aniónicos, incluindo dois resíduos tipicamente neutros que apresentam aqui uma cadeia lateral com carga negativa – resíduo de tirosina, na forma de fenolato, e resíduo de histidina, na forma de imidazolato, conforme apresentado Figura 4 da secção 1.3

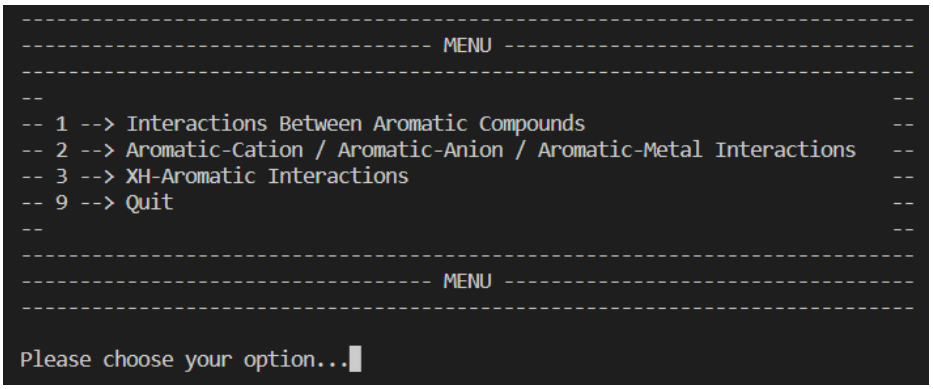

Figura 5 - Menu Inicial das Interações com Anéis Aromáticos.

A seguinte representação permite entender a maneira como a lógica do código foi desenvolvida bem como entender os "caminhos" que podem ser percorridos aquando da execução do *script* (todos os menus estão contidos no capítulo 2 a 6 dos anexos). Na figura está representada a lógica que deve ser seguida de modo a iniciar o estudo de uma determinada interação.

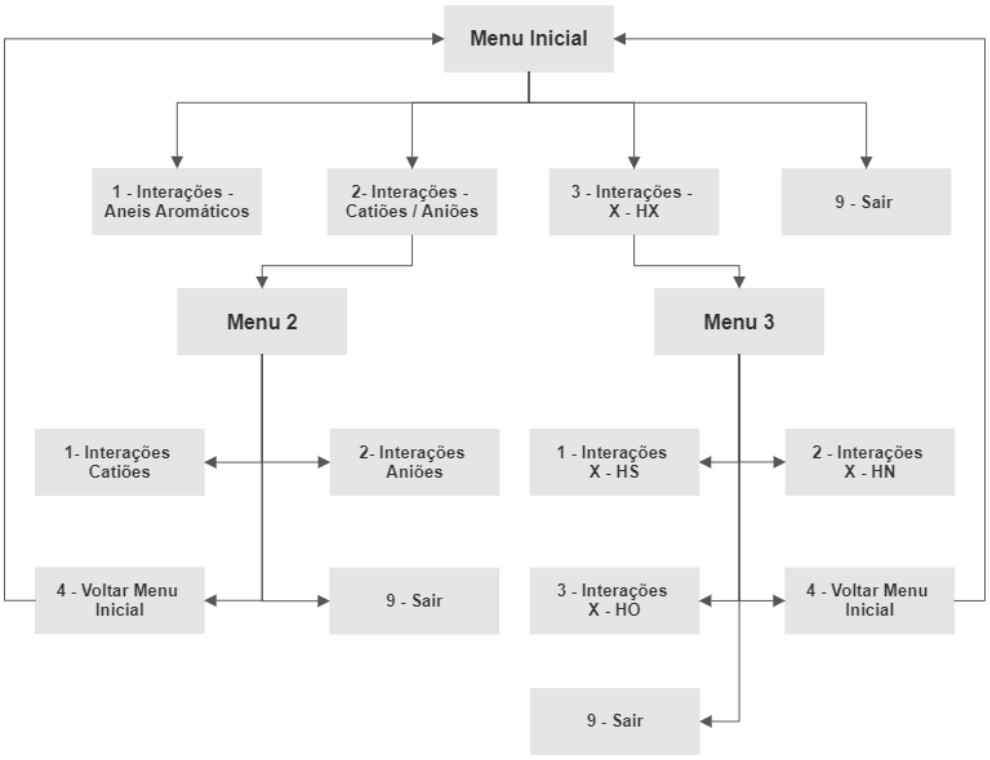

Figura 6 – Organização da interface do utilizador.

2.2. PACOTES E MÓDULOS

Uma das vantagens consideradas aquando da escolha do Python enquanto linguagem de programação para este projeto foi a possibilidade de fazer a importação de pacotes e módulos para facilitar ou otimizar o *script*.

É possível obter acesso ao código de um outro módulo pelo processo de importação. A declaração *import* é a forma mais comum de invocar o processo de consulta do código desse mesmo módulo. Quando um módulo é importado pela primeira vez, o compilador procura pelo módulo e, se encontrado, cria um objeto desse mesmo módulo inicializando-o. A linguagem Python possui apenas um tipo de objeto "módulo" e todos os módulos são desse tipo, independentemente do módulo ser implementado em Python, C ou qualquer outra linguagem. Para ajudar a organizar os módulos e fornecer uma hierarquia de nomenclatura, a linguagem Python usa o conceito de pacotes. No *script* criado foram utilizados os seguintes *packages*:

```
from itertools import count
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import numpy
from numpy.linalg import norm
import MDAnalysis as mda
import csv
import math
import logging
import datetime
import calendar
import os
import shutil
from pandas import array
import collections
import glob
from PyPDF2 import PdfMerger
```

- **Itertools**

Este módulo implementa vários blocos de construção das iterações inspiradas em APL, Haskell e SML, todas linguagens de programação. Cada um foi reformulado em uma forma adequada para Python. O módulo padroniza um conjunto básico de ferramentas rápidas e eficientes em memória que são úteis sozinhas ou em combinação, criando assim uma espécie de “álgebra de iteração”, tornando possível construir ferramentas especializadas de forma sucinta e eficiente em Python.

- **Matplotlib**

Matplotlib é uma biblioteca abrangente para criar visualizações estáticas, animadas e interativas em Python. Permite criar gráficos de qualidade, e figuras interativas que podem ser ampliadas, movidas e atualizadas. Destaca-se pela sua fácil personalização, que permite alterar facilmente o estilo visual e o *layout*. Os gráficos criados podem ser exportados para vários formatos. Por fim este pode ser integrado com vários outros pacotes,

- **Numpy**

NumPy (Python numérico) é uma biblioteca Python usada maioritariamente para trabalhar com *arrays*, mas também possui funções matemáticas muito completas para trabalhar no domínio da álgebra linear e matrizes. A NumPy é escrita parcialmente em Python, mas a maioria das partes que requerem computação rápida são escritas em C ou C++.

A linguagem Python apresenta a hipótese de trabalhar com *arrays* de forma nativa, porém o seu processamento é bastante lento. A NumPy fornece um objeto *array* que é até 50x mais rápido que os *arrays* tradicionais de Python. Os *arrays* são usados com muita frequência em análise de dados onde a velocidade e os recursos são muito importantes.

Os *arrays* em NumPy são armazenados num local contínuo de memória, diferente das listas, o que permite que sejam acedidos e manipulados com muito maior rapidez. Esse comportamento é chamado de *locality of reference*/ localização da referência, e é também otimizado para trabalhar com as arquiteturas de CPU mais atuais.

- **csv**

O formato csv (*comma separated values*) é o formato de importação e exportação mais comum para bases de dados. A falta de um padrão bem definido significa que podem existir pequenas diferenças nos dados produzidos e consumidos por diferentes aplicações. Ainda assim, o formato geral é semelhante o suficiente para que seja possível escrever um único módulo que possa manipular eficientemente esses dados.

O módulo **csv** implementa classes para ler e gravar dados no formato csv. Permite escrever dados no formato preferido pelo Excel ou ler dados de um arquivo, sem conhecer os detalhes precisos do formato csv usado pelo Excel. Este pacote é ainda versátil ao ponto de poder descrever os formatos csv compreendidos por outras aplicações ou definir o seu próprio formato csv.

- **math**

Este módulo fornece acesso às funções matemáticas definidas em C. Por exemplo, "math.sqrt()" devolve a raiz quadrada de qualquer operação matemática ou valor colocado no interior da função.

- **logging**

Este módulo define funções e classes que implementam um sistema de *logs*/registo de eventos flexível para aplicativos e bibliotecas. O principal benefício de ter a API (*Application Programming Interface*) de *log* fornecida por um módulo é o facto de todos os módulos Python poderem utilizar as funções do *log*. As mensagens podem ser definidas por um parâmetro previamente definido, como por exemplo *log* de alerta, *log* de alarme, ou *log* de informação, e face a isto só será escrito para o ficheiro a informação mediante o nível de *log* definido.

Esta funcionalidade pode ser usada também como método de *debug*. Os filtros fornecem um recurso mais refinado para determinar quais registos de *log* devem ser gerados, uma vez que é possível especificar o *layout* do registo de *log* na saída final devido aos diferentes tipos que podem ser passados como parâmetro no *input*.

- **datetime**

O módulo **datetime** fornece classes para fácil manipulação de datas e horas. Embora a aritmética de data e hora seja suportada pela linguagem Python, o foco desta implementação está na extração eficiente de atributos para formatação e manipulação de *output* relacionados com datas e horas.

- **calendar**

Este módulo permite utilizar informações sobre calendários no formato Unix, o mais utilizado para a execução de *scripts* uma vez que a implementação de *scripts* é feita muitas vezes em Linux. Este fornece também funções úteis adicionais relacionadas ao calendário.

- **os**

Este módulo fornece uma maneira fácil e intuitiva de utilizar funcionalidades do sistema operacional através da consola *bash*. Por exemplo, é possível ler ou escrever um arquivo com a função **open()** . Todas as funções que aceitam *file paths* ou arquivo aceitam *bytes* e *strings* como *input* e devolvem um objeto do mesmo tipo.

- **shutil**

O módulo **shutil** oferece várias operações de alto nível em arquivos e coleções de arquivos. Em particular, são fornecidas funções que suportam a cópia, movimentação e remoção de arquivos. Este é complementado pelo modulo **os,** anteriormente referido, que é mais indicado para operações individuais em arquivos.

- **collections**

Este módulo implementa tipos de dados especializados, fornecendo alternativas aos tipos "core" internos de uso geral no Python, tais como *dict*, *list*, *set* e *tuple*. Em alguns cenários, este módulo facilita a manipulação de *arrays* e listas, como por exemplo na remoção de duplicados.

- **glob**

O módulo **glob** encontra todos os caminhos que correspondem a um padrão específico de acordo com as regras usadas pela *shell* do Unix. É preciso ter atenção ao *output* uma vez que os resultados são retornados numa ordem aleatória.

- **PdfMerger**

O módulo **PdfMerger** permite realizar manipulação de ficheiros do tipo *.pdf*, como por exemplo junção de ficheiros especificados num *array*.

A função de junção de ficheiros pode ser ativada pela função *PdfMerger()*. A função*.write()* escolhe o nome do output e a função *.close()* grava o *output* num ficheiro com todos os ficheiros descritos no *input* e com o nome previamente definido.

2.3. SELEÇÃO DO UNIVERSO

A classe Universe é classe mais geral e abrangente, unindo uma topologia e uma trajetória. Todos os códigos de MDAnalysis começam com uma definição de “Universe". Normalmente, um universo é criado a partir de ficheiros. No caso do projeto realizado, a topologia era o ficheiro em formato *.gro* e a trajetória encontrava-se em formato *.xtc*.

```
gro = mda.Universe('md.gro')  # sem trajetória
xtc = mda.Universe('md.gro','md.xtc') # com trajetória
```

2.4. SELEÇÃO DE GRUPOS DE ÁTOMOS

Os diferentes grupos de átomos relevantes para cada interação a analisar foram selecionados como descrito de seguida.

2.4.1. SELEÇÃO DE SISTEMAS AROMÁTICOS

No caso dos resíduos com anéis aromáticos, foram selecionados conforme a sua geometria, uma vez que os cálculos a efetuar dependem parcialmente dessa geometria. No caso do triptofano, foram consideradas três possibilidades, como indicadas na Tabela 2.

Tabela 2 - Critérios para identificação de anéis aromáticos

Seleção de resíduos	Nome dos átomos a selecionar	Notas
TYR OR TYO[a] OR PHE	CG, CD1, CD2, CE1, CE2, CZ	Hexágonos
HI*	CG, CD2, NE2, CE1, ND1	HI* = HIT, HIS, HISA, HISB, HISH [b]; Pentágonos
TRP	CG, CD1, NE1, CD2, CE2	Pentágonos
TRP	CD2, CE2, CZ2, CH2, CZ3, CE3	Hexágonos
TRP	CG, CD1, NE1, CD2, CE2, CZ2, CH2, CZ3, CE3	-

a) Os resíduos TYO correspondem a resíduos de tirosina (TYR) desprotonados à forma de fenolato (por saída do protão hidroxílico).

b) Os diferentes tipos de resíduos de histidina correspondem a diferentes estados de protonação do grupo imidazol (ImH) – em HISH, ambos os azotos do grupo imidazol estão protonados, encontrando-se o resíduo na forma de imidazol protonado (ImH_2^+); na forma neutra (ImH), a protonação ocorre no átomo ND1 ou no átomo NE2, em HISA ou HISB, respectivamente; HIT corresponde à forma desprotonada (imidazolato, Im^-) verificada na presença de, por exemplo, metais que baixam acentuadamente o pKa da cadeia lateral do resíduo.

Para extração de um *array* do tipo “AtomGroup” (ou “ResidueGroup”), com todos os átomos (ou resíduos) envolvidos no estabelecimento de interações aromáticas (π-π), foi usado um código de seleção como.

```
A_aromatic (A_aromatic_rs) = gro.select_atoms("(resname TYR and (name CG CD1 CD2 CE1 CE2 CZ)) or (resname TYO and (name CG CD1 CD2 CE1 CE2 CZ)) or (resname PHE and (name CG CD1 CD2 CE1 CE2 CZ)) or (resname HIT and (name CG CD2 NE2 CE1 ND1)) or (resname HIS and (name CG CD2 NE2 CE1 ND1)) or (resname HISA and (name CG CD2 NE2 CE1 ND1)) or (resname HISB and (name CG CD2 NE2 CE1 ND1)) or (resname HISH and (name CG CD2 NE2 CE1 ND1)) or (resname TRP and (name CG CD1 NE1 CD2 CE2)) or (resname TRP and (name CD2 CE2 CZ2 CH2 CZ3 CE3)) or (resname TRP and (name CG CD1 NE1 CD2 CE2 CZ2 CH2 CZ3 CE3))")(.residues)
```

2.4.2. SELEÇÃO DE CATIÕES, ANIÕES E METAIS

A seleção de átomos com carga positiva (indicados na Tabela 3) seguiu a mesma lógica aplicada anteriormente, tal como a seleção de aniões (Tabela 4) e de metais (Tabela 5).

Tabela 3 - Critérios para identificação de catiões e átomos com carga parcial positiva

Nome Resíduo	Nome Átomo	Notas
ARG	NH1	Se e só se existirem simultaneamente os átomos HH11 e HH12
ARG	NH2	Se e só se existirem os átomos HH21 E HH22 simultaneamente
LYS	NZ	Têm de existir simultaneamente os átomos HZ1, HZ2 e HZ3 nesse resíduo; Nome: LYS, LYSH, LYSX, ou LYSY, correspondendo a diferentes estados de protonação
HISH	ND1	Têm de existir simultaneamente os átomos HD1 e HE2
FE3	FE3	Também metal
V3	V3	Também metal
VO	VA	Também metal
VO2	VB	Também metal

```
B_cations (B_cations_res) = gro.select_atoms("(resname ARG and name NH1 and name HH11 or name HH12) or (resname ARG and name NH2 and name HH21 or name HH22) or (resname LYS and name NZ and name HZ1 or name HZ2 or name HZ3) or (resname LYSH and name NZ and name HZ1 or name HZ2 or name HZ3) or (resname LYSX and name NZ and name HZ1 or name HZ2 or name HZ3) or (resname LYSY and name NZ and name HZ1 or name HZ2 or name HZ3) or (resname HISH and name ND1 and name HD1 or name HE2) or (name FE3) or (name V3) or (name VA) or (name VB)") (.residues)
```

Tabela 4 - Critérios para identificação de átomos com carga parcial negativa

Nome Resíduo	Nome Átomo		Notas
ASP	OD1 OD2	OR	Nome do resíduo tem de ser exatamente ASP. Não pode ser ASPH
CYS	SG		Não pode existir átomo HG nesse resíduo
GLU	OE1 OE2	OR	Nome do resíduo tem de ser exatamente GLU. Não pode ser GLUH
TYR OR TYO	OH		Não pode existir átomo HH nesse resíduo
HIT	NE2 ND1	OU	Não pode existir átomo HD1 nem HE2 nesse resíduo
VO	OVA		Também metal
VO2	OVB		Também metal

```
C_anions (C_anions_res) = gro.select_atoms("(resname ASP and name OD1 or name OD2) or (resname CYS and name SG and not name H) or (resname GLU and name OE1 or name OE2) or (resname TYR and name OH and not name HH) or (resname TYO and name OH and not name HH) or (resname HIT and name NE2 and not name HD1 and not name HE2) or (resname HIT and name ND1 and not name HD1 and not name HE2) or (resname VO and name OVA) or (resname VO2 and name OVB)") (.residues)
```

Tabela 5 - Critérios para identificação de metais

Nome Resíduo	Nome Átomo	Notas
FE3	FE3	Também catião/ anião
V3	V3	Também catião/ anião
VO	VA	Também catião/ anião
VO2	VB	Também catião/ anião

```
D_metals (D_metals_res) = gro.select_atoms("(name FE3) or (name V3) or (name VA) or (name
VB)") (.residues)
```

2.4.3. SELEÇÃO DE GRUPOS HX PARA INTERAÇÕES π-HX

Para o estabelecimento de interações do tipo ligação de hidrogénio entre sistemas π e grupos XH foram considerados os grupos tiol, amina e hidroxilo de cadeias laterais de resíduos de cisteína (Tabela 6), arginina, asparagina, glutamina, histidina, lisina e triptofano (Tabela 7), ou aspartato, glutamato, serina, treonina e tirosina (Tabela 8), respetivamente.

Nestes casos, só pode ocorrer esta interação se ambos o átomo de hidrogénio e o átomo dador (indicados nas tabelas seguintes) estiverem simultaneamente presentes no mesmo resíduo.

Tabela 6 - Critérios para identificação de grupos tiol

Nome Resíduo	Hidrogénio	Dador
CYS	HG	SG

```
E_thiol = gro.select_atoms("(resname CYS and (name H SG))").atoms
```

Tabela 7 - Critérios para identificação de grupos amina

Nome Resíduo	Dador	Hidrogénio
ARG	NH1	HH11
		HH12
	NH2	HH21
		HH22
ARGN / ARG	NH1	HH1
	NH2	HH21
		HH22
ASN	ND2	HD21
		HD22

Nome Resíduo	Dador	Hidrogénio
GLN	NE2	HE21
		HE22
HIS	HD1	ND1
	HE2	NE2
LYS	HZ1	NZ
	HZ2	NZ
LYSH / LYS	HZ1	NZ
	HZ2	NZ
	HZ3	NZ
TRP	HE1	NE1

```
F_amine = gro.select_atoms("(resname ARG and (name NH1 HH11)) or (resname ARG and (name NH1 HH12)) or (resname ARG and (name NH2 HH21)) or (resname ARG and (name NH2 HH22)) or (resname ARGN and (name NH1 HH1)) or (resname ARGN and (name NH2 HH21)) or (resname ARGN and (name NH2 HH22)) or (resname ARG and (name NH1 HH1)) or (resname ARG and (name NH2 HH21)) or (resname ARG and (name NH2 HH22)) or (resname ASN and (name ND2 HD21)) or (resname ASN and (name ND2 HD22)) or (resname GLN and (name NE2 HE21)) or (resname GLN and (name NE2 HE22)) or (resname HIS and (name HD1 ND1)) or (resname HIS and (name HE2 NE2)) or (resname LYS and (name HZ1 NZ)) or (resname LYS and (name HZ2 NZ)) or (resname LYSH and (name HZ1 NZ)) or (resname LYSH and (name HZ2 NZ)) or (resname LYSH and (name HZ3 NZ)) or (resname LYS and (name HZ1 NZ)) or (resname LYS and (name HZ2 NZ)) or (resname LYS and (name HZ3 NZ)) or (resname TRP and (name HE1 NE1))").atoms
```

Tabela 8 - Critérios para identificação de grupos hidroxilo

Nome Resíduo	Hidrogénio	Dador
ASPH	HD2	OD2
GLUH	HE2	OE2
SER	HG	OG
THR	HG1	OG1
TYR	HH	OH

```
G_hydroxyl = gro.select_atoms("(resname ASPH and (name HD2 OD2)) or (resname GLUH and (name HE2 OE2)) or (resname SER and (name HG OG)) or (resname THR and (name HG1 OG1)) or (resname TYR and (name HH OH))").atoms
```

2.5. FUNÇÕES AUXILIARES

De acordo as boas práticas de programação em que se deve ao máximo evitar a repetição do código, esta abordagem foi tida em conta e foi criado um conjunto de funções auxiliar. Estas remetem para funções que são constantemente necessárias ao longo do código e que definindo *inputs* da forma mais genérica possível é possível fazer chamadas às funções com os *inputs* diferentes, desde que do mesmo tipo definido na função. Apenas é necessário garantir que a função "base" está criada e definida antes das chamadas, para não haver erros de compilação.

- **unit_vector()**

Um exemplo é a implementação da função **unit_vector ()**, que devolve o vetor unitário de um conjunto de coordenadas – tem apenas um campo de argumento de entrada e espera receber um conjunto de coordenadas.

```
#Returns the unit vector of the vector.

def unit_vector(vector):

    return vector / np.linalg.norm(vector)
```

Esta função pode ser usada individualmente ou como subfunção da função **angle_between ()**.

- **angle_between()**

Esta função retorna o ângulo, em radianos, entre 2 conjuntos de coordenadas. Tem dois campos de argumento de entrada e é esperado receber 2 conjuntos de coordenadas. Remete para a subfunção **unit_vector()** uma vez que qualquer dos parâmetros de entrada são enviados para esta função no sentido de obter o vetor unitário de cada um deles. Após ambos os vetores unitários estarem disponíveis é calculado o ângulo, em radianos, entre as duas coordenadas, utilizando funções complementares da biblioteca do numPy:

```
#Returns the angle in radians between vectors 'v1' and 'v2'.
def angle_between(v1, v2):

    """ Exemples:
    >>> angle_between((1, 0, 0), (0, 1, 0))
    1.5707963267948966
    >>> angle_between((1, 0, 0), (1, 0, 0))
    0.0
    >>> angle_between((1, 0, 0), (-1, 0, 0))
    3.141592653589793
    """

    v1_u = unit_vector(v1)
    v2_u = unit_vector(v2)

    return np.arccos(np.clip(np.dot(v1_u, v2_u), -1.0, 1.0))
```

Esta função pode ser usada individualmente ou como subfunção da função angle_between_degree ().

- **angle_between_degree()**

Esta função retorna o ângulo, em graus, entre 2 conjuntos de coordenadas. Tem dois campos de argumento de entrada e é esperado receber 2 conjuntos de coordenadas. Remete para a subfunção angle_between () uma vez que precisa de saber o ângulo previamente calculado em radianos para poder converter para graus. Após ambos os vetores unitários estarem disponíveis e calculado o ângulo em radianos entre as duas coordenadas, esta função utiliza o valor retornado pela função angle_between() e realiza a conversão.

```
#Returns the angle in degrees between vectors ‘v1’ and ‘v2’.

Def angle_between_degree(v1, v2):

    return angle_between(v1,v2)*57.2957795
```

- **findVec ()**

Função que retorna o vetor de qualquer conjunto de coordenadas que seja passado como *input*. Tem dois campos de argumento de entrada e é esperado receber 2 conjuntos de coordenadas. Esta função é utilizada individualmente e remete a uma subfunção **multiDimenDist**().

```
#findVec sub function

def multiDimenDist(point1,point2):

   #find the difference between the two points, its really the same as below
   deltaVals = [point2[dimension]-point1[dimension] for dimension in range(len(point1))]
   runningSquared = 0

#because the pythagarom theorm works for any dimension we can just use that
   for 20iff in deltaVals:
       runningSquared += 20iff**2

   return runningSquared**(1/2)

#findVec function

def findVec(point1,point2,unitSphere = False):

  #setting unitSphere to True will make the vector scaled down to a sphere with a radius one,
instead of it’s orginal length
  finalVector = [0 for 20iff in point1]

  for dimension, 20iff in enumerate(point1):
      #finding total 20ifference for that co-ordinate(x,y,z…)
      deltaCoOrd = point2[dimension]-coOrd
      #adding total difference
      finalVector[dimension] = deltaCoOrd

  if unitSphere:
      totalDist = multiDimenDist(point1,point2)
      unitVector =[]
      for dimen in finalVector:
          unitVector.append( dimen/totalDist)

      return unitVector

  else:

      return finalVector
```

2.6. ORGANIZAÇÃO DO CÓDIGO

A organização do código permite ter as funções de forma modular em que apenas é necessário chamar um **mainMenu()** principal no início de cada execução. Mediante as seleções feitas pelo utilizador, o *script* processa a execução necessária. Desta maneira é possível poupar recursos da máquina uma vez que apenas está a correr o necessário. Esta poupança de recursos tem influência direta nos tempos de processamento, uma vez que, evitando processar funcionalidades desnecessárias, reduzimos o tempo das corridas.

```
##main
checkD“r("\L”gs")
checkD“r("\Arch”ve")
checkD“r("\R”ns")
checkD“r("\P”Fs")
moveFil“s("\*.”og“, "\L”gs")
moveFil“s("\*.”ip“, "\Arch”ve")
moveFil“s("/”6*“, "\R”ns")
moveFil“s("\*.”df“, "\P”Fs")
mainMenu()
```

Quando é lançado o *script* após correr todas as validações de diretorias e estabilização do ambiente onde está colocado o *script* pela função **checkDir(),** e é feita toda a movimentação de ficheiros antigos para as pastas de arquivo pela função **moveFiles()**, é chamada a função **mainMenu()**. Esta função é responsável pela primeira interface apresentada ao utilizador e permite que este faça a seleção do que pretende realizar.

A função **mainMenu()** apresenta uma lista de opções que o utilizador pode selecionar, e fica a aguardar pelo *input* do mesmo.:

- a opção 1 remete para a análise das interações com anéis aromáticos.
- a opção 2 remete para a análise das interações relacionadas com aniões, catiões e metais. É chamado um submenu (**option2Menu()**) que dá ao utilizador novamente a possibilidade da escolha mais concreta das interações a analisar.
- a opção 3 remete para a análise de interações π-HX. É chamado um submenu (**option3Menu()**) que dá ao utilizador novamente a possibilidade da escolha mais concreta das interações a analisar.
- a opção 9 corresponde ao código de “exit” terminado a execução.

Qualquer outro *input* que não esteja contido na listagem de opções possíveis será tratado como erro, enviando uma mensagem informativa ao utilizador e terminando a execução do *script* de seguida.

```
#Main Menu function
#Menu customization, options definition and function callbacks
def mainMenu():
    pri“t("------------------------------------------------------------------------------")
    pri“t("---------------------------------- MENU ----------------------------------")
    pri“t("------------------------------------------------------------------------------")
    pri“t("--                                                                          --")
    pri“—- 1 --> Interactions Between Metals and Aromatic Compounds              --")
    pri“—- 2 --> Interactions Between Metals-Cation / Metals-Anion               --")
    pri“—- 3 --> Interactions Between XH-Metals                                  --")
    pri“—- 9 --> Quit                                                            --")
    pri“t("--                                                                    --")
    pri“t("------------------------------------------------------------------------------")
    pri“t("---------------------------------- MENU ----------------------------------")
    pri“t("------------------------------------------------------------------------------")
    pri“”("")
    option = int(inp“t("Please choose your option”.."))
```

```
    if option == 1:
        pri“t("Option 1 Selected. Interactions Between Metals and Aromatic Compoun”s.")
        functionMetalAromatic(A_aromatic_res,metal_res)
        pri“t("Script Execution Start”d.")
    elif option == 2:
        pri“t("Option 2 Selected. Interactions Between Metals-Cation / Metals-Ani”n.")
        option2Menu()
    elif option == 3:
        pri“t("Option 3 Selected. Interactions Between XH-Meta”s.")
        option3Menu()
    elif option == 9:
        pri“t("Option 9 Selected. Script Excecution Stopp”d.")
        quit()
    else:
        pri“t("Option unavailable. Please Run the script again with an option contained in
the me”u.")
        unavailableOption(option)
```

Os menus secundários referidos, correspondentes às funções **option2Menu()** e **option3Menu()** apresentam uma lógica e organização semelhantes, apresentando uma lista de opções e aguardando *input* do utilizador. No caso da função **option2Menu(),** as opções 1 e 2 correspondem, respetivamente, ao estudo de interações entre catiões (opção 1) ou aniões (opção 2), tendo a opção 4 a função de regresso ao menu inicial e a opção 9 a mesma finalidade que anteriormente.

```
#Interactions Between Metals-Cation / Metals-Anion Menu function
#Option 2 Selected.
#Menu customization, options definition and function callbacks.
def option2Menu():
    print("------------------------------------------------------------------------------")
    print("------------------------------ OPTION 2 MENU -------------------------------")
    print("------------------------------------------------------------------------------")
    print("--                                                                          --")
    print("-- 1 --> Interactions Between Metals-Cation                                 --")
    print("-- 2 --> Interactions Between Metals-Anion                                  --")
    print("-- 4 --> Back to Main Menu                                                  --")
    print("-- 9 --> Quit                                                               --")
    print("--                                                                          --")
    print("------------------------------------------------------------------------------")
    print("------------------------------ OPTION 2 MENU -------------------------------")
    print("------------------------------------------------------------------------------")
    print("")
    option2 = int(input("Please choose your option..."))
    if option2 == 1:
        print("Option 1 Selected. Interactions Between Metals-Cation.")
        print("Script Execution Started.")
        functionMetalCation(metal_res,B_cations_res)
        print("Script Execution Ended.")
    elif option2 == 2:
        print("Option 2 Selected. Interactions Between Metals-Anion.")
        print("Script Execution Started.")
        functionMetalAnion(metal_res,C_anions_res)
        print("Script Execution Ended.")
    elif option2 == 4:
        print("Option 4 Selected. Returned to Main Menu.")
        mainMenu()
```

```
    elif option2 == 9:
        print("Option 9 Selected. Script Excecution Stopped.")
        quit()
    else:
        print("Option unavailable. Please Run the script again with an option contained in
the menu.")
        unavailableOption(option2)
```

No caso da função **option3Menu()**, as opções 1, 2 e 3 correspondem ao estudo de interações envolvendo grupos tiol, amina, ou hidroxilo, respetivamente, e as opções 4 e 9 têm a mesma finalidade que em **option2Menu()**.

```
#Interactions Between Metals-Cation / Metals-Anion /Menu function
#Option 3 Selected.
#Menu customization, options definition and function callbacks.
def option3Menu():
    print("------------------------------------------------------------------------------")
    print("------------------------------ OPTION 3 MENU -------------------------------")
    print("------------------------------------------------------------------------------")
    print("--                                                                          --")
    print("-- 1 --> Interactions Between Metals-HS                                     --")
    print("-- 2 --> Interactions Between Metals-HN                                     --")
    print("-- 3 --> Interactions Between Metals-HO                                     --")
    print("-- 4 --> Back to Main Menu                                                  --")
    print("-- 9 --> Quit                                                               --")
    print("--                                                                          --")
    print("------------------------------------------------------------------------------")
    print("------------------------------ OPTION 3 MENU -------------------------------")
    print("------------------------------------------------------------------------------")
    print("")
    option3 = int(input("Please choose your option..."))
    if option3 == 1:
        print("Option 1 Selected. Interactions Between Metals-HS.")
        print("Script Execution Started.")
        functionEList(metal_res,E_thiol)
        print("Script Execution Ended.")
    elif option3 == 2:
        print("Option 2 Selected. Interactions Between Metals-HN.")
        print("Script Execution Started.")
        functionFList(metal_res,F_amine)
        print("Script Execution Ended.")
    elif option3 == 3:
        print("Option 3 Selected. Interactions Between Metals-HO.")
        print("Script Execution Started.")
        functionGList(metal_res,G_hydroxyl)
        print("Script Execution Ended.")
    #elif option3 == 4:
        #print("Option 4 Selected. Interactions Between Metals-HC.")
        #print("Script Execution Started.")
        #functionHList(A_aromatic_res,G_methyl)
        #print("Script Execution Ended.")
    elif option3 == 4:
        print("Option 4 Selected. Returned to Main Menu.")
        mainMenu()
    elif option3 == 9:
        print("Option 9 Selected. Script Excecution Stopped.")
        quit()
```

```
    else:
        print("Option unavailable. Please Run the script again with an option contained in
the menu.")
        unavailableOption(option3)
```

Qualquer outro *input* que não esteja contido na listagem de opções possíveis será tratado como erro, enviando uma mensagem informativa ao utilizador e terminando a execução do *script* de seguida.

Adicionalmente foi criada a função **unavailableOption()** que recebe o input do utilizador como argumento, e é invocado nos casos em que o *input* no menu é um valor não aceite. Esta função regista a tentativa de execução como se de uma execução normal se tratasse, registando ao nível do *log* o erro que originou bem como as datas em que ocorreu.

```
#Unavailable Option Function
#Will add log information about the execution attempt.
def unavailableOption(option) :
    currentDate=datetime.datetime.now()
    dateEpoch = calendar.timegm(currentDate.timetuple())
    logging.basicConfig(filename  =  "logfile_"  +  str(dateEpoch)  +  ".log",  level  =
logging.INFO)
    logging.info(str(datetime.datetime.now()) + ": Execution Started.")
    logging.info(str(datetime.datetime.now()) + ": Option '" + str(option) + "' unavailable.
")
    logging.info(str(datetime.datetime.now()) + ": Unable to run script.")
    logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
```

2.7. ORGANIZAÇÃO DO *SCRIPT*

Antes de ser iniciadas as iterações de todas as interações possíveis, existem 2 valores muito importante que devem ser tidos em conta mediante os resultados desejados. Existem 2 variáveis definidas no início do código que influenciam de forma direta a volumetria de dados e consecutivamente a discrepância dos valores/ resultados obtidos ao nível dos gráficos dos ângulos e das distâncias. Uma vez que estão definidos no início do código como variável global, qualquer alteração ao valor necessária é bastante fácil uma vez que, substituindo somente na variável, esta alteração fica transversal a todo o código e consecutivamente a todas as funções das interações. A maxMeanDistanceTest e a minMeanAnglesTest são responsáveis por definir os intervalos de valores entre os quais as médias dos ângulos e das distâncias ao longo de todos os *frames* são considerados aceitáveis e com algum interesse para a análise a realizar. Posteriormente só serão considerados e criados os gráficos e ficheiros de valores dos valores contidos nestes intervalos.

Durante a seleção de um determinado tipo de interação para ser analisado, é feita a chamada de uma função diferente. A demonstração da lógica criada será feita mediante da apresentação de um caso específico. Foi selecionada a função responsável para as interações entre metais e o grupo amina (x-HN) (restante código na íntegra contido no capítulo 7 dos anexos). A **functionFList(input1,input2)** foi selecionada por ser a mais complexa e assim poder ser dado uma *overview* da lógica. Esta função tem 2 argumentos de entrada e durante a execução recebe as variáveis metal_res e F_amine, respetivamente.

A função inicia com a preparação do ambiente para realização da execução e criação dos ficheiros de *log* onde ficará registado tudo o que é realizado por parte do *script.* O tipo de *log* é definido como informativo de modo que todas as *strings* passadas como argumento sejam passadas para o *log.*

```
#Metals–HN Function
def functionFList(input1,input2):
    count = 1
    currentDate=datetime.datetime.now()
    dateEpoch = calendar.timegm(currentDate.timetuple())
    logFilename = "logfile_" + str(dateEpoch) + ".log"
    newDirectoryName = str(dateEpoch)
    os.mkdir(newDirectoryName)
    logging.basicConfig(filename = logFilename , level = logging.INFO)
    logging.info(str(datetime.datetime.now()) + ": Execution Started.")
    logging.info(str(datetime.datetime.now()) + ": Option 2 Selected. Interactions Between
Metal-HN.")
    logging.info(str(datetime.datetime.now())  +  ":  Execution  Directory  Created  '"  +
str(newDirectoryName) +"'.")
    print  ("Execution  log  file  can  be  found  in  ('"  +  str(newDirectoryName)  +  "/"  +
str(logFilename) + "').")
```

Seguidamente é feita a preparação de todas as interações possíveis, são organizados dois *arrays* de dados baseados nos *inputs*, um com os IDs dos metais, mais precisamente do seu resíduo, e outro para os IDs dos resíduos das aminas. Este último sofre um tratamento de dados ligeiramente diferente, pois é feita a remoção de duplicados do *array* para evitar que sejam repetida iterações desnecessárias. Tendo por base a informação da tabela 7, em que são apresentadas as possíveis combinações de dadores e hidrogénios a partir de um determinado resíduo e sendo que na lista de valores idAmine_NoDups temos os tipos de resíduos, conseguimos facilmente identificar os hidrogénios (h_list) e os dadores (donor_list) a utilizar na iteração.

```
idMetal = []
    for i in input1:
        resid = i.resid
        idMetal.append(resid)

idAmine = []
    for i in input2:
        resid = i.resid
        idAmine.append(resid)
    idAmine_NoDups = ([item for item, count in collections.Counter(idAmine).items() if count
> 1])
```

```
    for i in idMetal:
        for j in idAmine_NoDups:
            getResid = gro.select_atoms("resid " + str(j)).residues
            getResid = str(getResid)
            getResid = getResid.split('[<Residue ')[1]
            getResid = getResid.split(',')[0]

            donor_list = []
            h_list =  []

            if (getResid == "ARG"):
                donor_list = ["NH1", "NH2"]
                h_list =  ["HH11","HH12","HH21","HH22","HH1"]
            if (getResid == "ARGN"):
                donor_list = ["NH1", "NH2"]
                h_list =  ["HH1","HH21","HH22"]
            if (getResid == "ASN"):
                donor_list = ["ND2"]
                h_list =  ["HD21","HD22"]
            if (getResid == "GLN"):
                donor_list = ["NE2"]
                h_list =  ["HE21","HE22"]
            if (getResid == "HIS"):
                donor_list = ["HD1","HE2"]
                h_list =  ["ND1","NE2"]
            if (getResid == "LYS"):
                donor_list = ["HZ1","HZ2","HZ3"]
                h_list =  ["NZ"]
            if (getResid == "LYSH"):
                donor_list = ["HZ1","HZ2","HZ3"]
                h_list =  ["NZ"]
            if (getResid == "TRP"):
                donor_list = ["HE1"]
                h_list =  ["NE1"]
```

Sendo que agora estão disponíveis os IDs dos metais, dos hidrogénios e dos dadores das aminas, é agora necessário preparar as seleções diretamente no universo xtc, que contém as trajetórias e permite fazer a análise ao longo de todos os *frames.*

```
for d in donor_list:
    for n in h_list:
        donor_select = gro.select_atoms("name " + str(d) + " and resid " + str(j))
        h_select = gro.select_atoms("name " + str(n) + " and resid " + str(j))

        if len(donor_select) > 0 and len(h_select) > 0 :
            donor = donor_select[0]
            h = h_select[0]

            xtc_metal = xtc.select_atoms("(resid " + str(i) + ")")
            xtc_metal_name = xtc_metal[0]
            xtc_metal_name = xtc_metal_name.resname

            xtc_H = xtc.select_atoms("name " + str(h.name) + " and type " + str(h.type) + "
and resid " + str(h.resid))
            xtc_H_name = xtc_H[0]
            xtc_H_name = xtc_H_name.name
```

```
            xtc_donor = xtc.select_atoms("name " + str(donor.name) + " and type " +
str(donor.type) + " and resid " + str(donor.resid))
            xtc_donor_name = xtc_donor[0]
            xtc_donor_name = xtc_donor_name.name

            logging.info(str(datetime.datetime.now()) + ": --> CURRENT INTERACTION")
            logging.info(str(datetime.datetime.now()) + ": Metal --> " + str(i) + " " +
str(xtc_metal_name) )
            logging.info(str(datetime.datetime.now()) + ": " + str(xtc_metal))
            logging.info(str(datetime.datetime.now()) + ": Thiol --> " + str(j) + " " +
str(xtc_H_name) + "_" + str(xtc_donor_name) )
            logging.info(str(datetime.datetime.now()) + ": " + str(xtc_H) + " " + "   +
str(xtc_donor))

            arrayFramesLoop = []
            arrayDistancesLoop = []
            arrayAnglesLoop = []
            arrayInteractionLoop = []
            arrayInteractionResidLoop = []
            arrayFrameDistanceAngles = []
```

Tendo as seleções todas previamente feitas é possível utilizar a *trajectory()* do ficheiro de universo .xtc para extrair a informação relativa a um determinado *frame.* Dito isto foi feita a iteração e a análise das posições dos metais, hidrogénios e dadores, sendo consecutivamente calculada as distâncias e os ângulos em cada frame. Todos os valores foram guardados em *arrays* temporários para no final do ciclo dos *frames* poder ser feita a média de ambos e validar se apresentam valores que sejam importantes para o estudo em curso, caso contrário serão descartados, os *arrays* limpos e o ciclo inicia novamente com a interação seguinte.

```
for ts in xtc.trajectory:
    positionMetal = xtc_metal.positions
    positionDonor = xtc_donor.positions
    positionH = xtc_H.positions
    centroidMetal = getCenteroid(positionMetal)
    avgsDonor = [sum(vals)/len(positionDonor) for vals in zip(*positionDonor)]
    avgsH = [sum(vals)/len(positionH) for vals in zip(*positionH)]
    vector_H_Donor = findVec(avgsH,avgsDonor)
    vector_H_Centroid = findVec(avgsH,centroidMetal)
    angle=angle_between_degree(vector_H_Donor,vector_H_Centroid)
    dist = math.sqrt((centroidMetal[0]-avgsDonor[0])**2 + (centroidMetal[1]-avgsDonor[1])**2
+ (centroidMetal[2]-avgsDonor[2])**2 )
    arrayFramesLoop.append(ts.frame)
    arrayDistancesLoop.append(dist)
    arrayAnglesLoop.append(angle)
    arrayFrameDistanceAngles.append(str(xtc.trajectory.time)+   ','   +   str(dist)+   ','   +
str(angle))
arrayInteractionResidLoop.append(str(i) + "VS" + str(j))
arrayInteractionLoop.append(str(xtc_metal_name)   +   "VS"   +   str(xtc_H_name)   +   "_"   +
str(xtc_donor_name))
mean = numpy.mean(arrayDistancesLoop)
meanAngles = numpy.mean(arrayAnglesLoop)
```

Feitos os cálculos da média das distâncias e dos ângulos para uma determinada interação, é feita a validação do mesmo contra os valores pré-estabelecidos e aceites como padrão para considerar importante ou não a interação. Caso se verifique que ambos os valores cumprem os requisitos, este são adicionados ao *array* que será responsável por criar os gráficos no final da verificação de todas as interações. Adicionalmente, para todos os casos que cumpram as condições, será escrito um ficheiro .csv com todos os valores de distâncias e ângulos para todos os *frames*, bem como as respetivas médias. Estes ficheiros estão identificados com um ID único e com a o nome da interação para facilitar a sua análise e consulta.

```
if (mean < maxMeanDistanceTest) and (meanAngles > minMeanAnglesTest) and (mean != 0.0) :
    logging.info(str(datetime.datetime.now()) + ": Distance Mean (" + str(mean) + ")")
    logging.info(str(datetime.datetime.now()) + ": Angles Mean (" + str(meanAngles) + ")")
    logging.info(str(datetime.datetime.now()) + ": Values will be added to the plot.")
    arrayFrames.append(arrayFramesLoop)
    arrayDistances.append(arrayDistancesLoop)
    arrayAngles.append(arrayAnglesLoop)
    arrayInteraction.append(arrayInteractionLoop)
    arrayInteractionResid.append(arrayInteractionResidLoop)
    csvfile = str(count)+ "_" + str(arrayInteractionResidLoop[0]) + "_" + str(dateEpoch)
+".csv"
    interaction = str(i) + str(xtc_metal_name) + "_" + str(j) + str(xtc_H_name) + "_" +
str(xtc_donor_name)
    arrayAllResults.append(str(interaction) + ";" + str(mean) + ";" + str(meanAngles))
    csvresults = "results.csv"
    arrayPlotNameDistance.append(str(i)   +   str(xtc_metal_name)   +   "_"   +   str(j)   +
str(xtc_H_name) + "_" + str(xtc_donor_name))
    arrayPlotNameAngle.append(str(i) + str(xtc_metal_name) + "_" + str(j) + str(xtc_H_name)
+ "_" + str(xtc_donor_name))
    with open(csvfile, "w") as fp:
        wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
        wr.writerow(['Avarage Distance', mean])
        wr.writerow(['Interaction', arrayInteractionLoop[0]])
        wr.writerow(['Frame','Distance','Angle'])
        for x in arrayFrameDistanceAngles :
            wr.writerow ([x])
    shutil.move(csvfile, newDirectoryName)
    count += 1
    #logging.info(str(datetime.datetime.now()) + ": File Created -> " + str(csvfile))
    logging.info(str(datetime.datetime.now()) + ": File '" + str(csvfile) +"' Created on
Execution Directory (" + str(newDirectoryName) +").")
else:
    #print("MEAN >= 10 --> Will be ignored")
    logging.info(str(datetime.datetime.now()) + ": Distance Mean (" + str(mean) + ")")
    logging.info(str(datetime.datetime.now()) + ": Angles Mean (" + str(meanAngles) + ")")
    logging.info(str(datetime.datetime.now()) + ": Values Will be ignored.")
    #arrayFramesLoop.clear()
    #arrayDistancesLoop.clear()
    #arrayAnglesLoop.clear()
logging.info(str(datetime.datetime.now()) + ": All interactions checked.")
```

O trecho de código seguinte antecede o processo de criação dos gráficos, e é responsável por criar um ficheiro que é colocado na pasta de execução com um resumo dos resultados, ou seja, escreve somente as interações que cumpriram os requisitos, caso existam, e as suas respetivas médias de ângulos e de distâncias. Este ficheiro serve apenas para consulta e

validação do *output* da execução. Uma vez que só tem as médias, qualquer informação mais detalhada deve ser consultada no ficheiro individual. De forma geral este ficheiro permite ter uma *overview* da execução.

```
if arrayAllResults:
        with open(csvresults, "w") as fp:
            wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
            wr.writerow(['Interaction;Distance;Angle'])
            for x in arrayAllResults :
                wr.writerow ([x])
        shutil.move(csvresults, newDirectoryName)
```

De seguida inicia-se o processo de criação dos ficheiros pdf com os gráficos dos ângulos e das distâncias caso existam interações que cumpram os requisitos. Inicialmente o processo cria um ficheiro pdf por cada execução mas, no final, são agrupados num único ficheiro de distâncias e de ângulos contendo todos os pdfs inicialmente gerados.

```
if(count > 1 ):
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Started.")

        #Plot Distance/Frames
        for index,x in enumerate(arrayDistances):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Distance (nm)")
            #plt.title("Metal_Amine - Distance")
            plt.title(str(arrayPlotNameDistance[0]))
            arrayPlotNameDistance.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            distancesPlotName = "Distance_Metal_Amine_" + labelString + "_" + str(dateEpoch)+
".pdf"
            plt.savefig(distancesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Distance_Metal_Amine*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameDistances = "All_Distances_Metal_Amine_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameDistances)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #distancesPlotName = "Distances_SmallUniverse_SinglePlot_Test_" + labelString + "_"
+ str(dateEpoch)+ ".pdf"
        #plt.savefig(distancesPlotName)
        logging.info(str(datetime.datetime.now())     +     ":     Plot     File     '"     +
str(mergeFilenameDistances) +"' Created on Execution Directory (" + str(newDirectoryName)
+").")
        shutil.move(mergeFilenameDistances, newDirectoryName)

        #Clear Current Plot
```

```
        plt.clf()

        #Plot Angles/Frames
        for index,x in enumerate(arrayAngles):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Angles Amplitude (º)")
            #plt.title("Metal_Amine - Angles")
            plt.title(str(arrayPlotNameAngle[0]))
            arrayPlotNameAngle.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            anglesPlotName ="Angles_Metal_Amine_" + labelString + "_" + str(dateEpoch)+
".pdf"
            plt.savefig(anglesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Angles_Metal_Amine*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameAngles = "All_Angles_Metal_Amine_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameAngles)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #logging.info(str(datetime.datetime.now()) + ": Plot File Created -> " +
str(anglesPlotName))
        logging.info(str(datetime.datetime.now()) + ": Plot File '" +
str(mergeFilenameAngles) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
        shutil.move(mergeFilenameAngles, newDirectoryName)

        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Ended.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
        #os.remove(logFilename)
else:
    logging.info(str(datetime.datetime.now()) + ": No interactions matched the requisites.")
    logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process will be
ignored.")
    logging.info(str(datetime.datetime.now()) + ": No plots will be created.")
    logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
    shutil.copy(logFilename, newDirectoryName)
    shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
```

2.8. MANUTENÇÃO/ *HOUSEKEEPING*

O tema de "*housekeeping*" / manutenção foi também tido em conta no desenvolvimento do *script*. Uma vez que podem ser obtidos resultados com grande volumetria a longo prazo, o acumular de resultados pode começar a afetar o bom funcionamento da máquina. Foram implementadas funções de manutenção e verificação que correm sempre no início de cada

execução. Durante cada execução são criados vários ficheiros e pastas. Mais concretamente, uma pasta com o ID único da execução, um ficheiro *log* com o mesmo ID, e um ficheiro *zip* com tudo o que possa dizer respeito a esta mesma execução. Dentro da pasta ficarão disponíveis os ficheiros *pdf* correspondentes aos resultados obtidos, o ficheiro de registo *log* correspondente e todos os ficheiros *csv* das interações representadas com um ID de interação único e para fácil identificação, quer nos gráficos, quer no ficheiro de registo.

A função **checkDir()** recebe um "*path*" como *input* e faz a validação se a diretoria existe . Caso exista, a função continua e não faz nada, caso contrário a diretoria é criada.

```
##Check if a DIR exist
def checkDir(inputDir):
    pwd=os.getcwd()
    newDir=str(pwd) + inputDir
    exists= os.path.exists(newDir)
    if(exists==False):
        #print("Created!")
        os.mkdir(newDir)
```

A função **moveFiles()** recebe 2 *inputs*, um "*pattern*" como primeiro argumento e um "path" como segundo *input*. Quando é feita a chamada da função, o *script* valida a existência de ficheiros com o "*pattern*" indicado, agrupa-os num *array* e depois move os ficheiros selecionados e contidos no *array* para a diretoria indicada no segundo argumento.

```
#Move files to input DIR
def moveFiles(input1, input2):
    pwd=os.getcwd()
    #print(pwd)
    pattern = input1
    files = glob.glob(pwd + pattern)
    logsDir=str(pwd) + input2
    for file in files:
        shutil.move(file, logsDir)
```

Antes do início de cada execução é feita sempre a validação da existência de 3 diretorias, nomeadamente:

- *Logs*
- *Archive*
- *Runs*

Caso estas diretorias não existam são criadas, garantindo assim que o ambiente está pronto para correr o *script* e tem tudo o que seja necessário criado. Assim sendo é possível garantir

o bom funcionamento do *script* mesmo mudando o local do mesmo, pois ele próprio cria o ambiente e as condições necessárias para executar. Estas funções de *housekeeping* são invocadas no início de cada execução, antes de ser invocada a função **mainMenu()**.

3. RESULTADOS E DISCUSSÃO

3.1. DESEMPENHO DO CÓDIGO

O *script* desenvolvido foi aplicado a 3 estruturas diferentes da transferrina. Enquanto apo-proteína, esta proteína assume uma conformação aberta (PDB ID 1BP5) [13] que, ao ligar Fe(III), na presença de carbonato, se altera para uma conformação fechada (PDB ID 1A8E) [14], com um local de ligação bem definido. No entanto, a hTf pode também ligar outros aniões sinergísticos mais volumosos, como o oxalato, estabilizando numa conformação relaxada (PDB ID 1RYO) [15], com um local de ligação de volume intermédio entre o observado nas conformações fechada e aberta.

O código desenvolvido foi aplicado a simulações por dinâmica molecular, sem restrições de cada uma das conformações da hTf complexada com Fe(III) na presença carbonato (em cada um dos três estados de protonação possíveis), bem como de cada conformação de hTf complexada com V(III), igualmente na presença de carbonato. Esta aplicação permite não só verificar a funcionalidade do código, mas também permitirá analisar a geometria de coordenação para cada um dos metais. Estas simulações foram desenvolvidas previamente no grupo de investigação, não sendo parte integrante do trabalho aqui apresentado.

Espera-se que numa conformação mais fechada da proteína seja mais fácil estabelecer interações com o metal e, consequentemente, devendo ser possível observar maior número de interações, uma vez que é feita a triagem das distâncias e dos ângulos de modo que só sejam considerados os mais importantes. Consequentemente, conformações mais abertas terão o efeito contrário, tornando mais difícil estabelecer interações entre a proteína e o metal.

Numa primeira abordagem, analisou-se o número de interações identificadas automaticamente para cada uma das simulações (Tabela 9; todas as interações identificadas são apresentadas em anexo, na secção 6.8).

Analisando as interações obtidas, é possível perceber que estão em conformidade com o esperado (todas as interações e resultados contidos no capítulo 8 dos anexos). Para conformação fechada foram identificadas um total de 14 interações por cada sistema simulado, com distâncias entre átomos inferiores a 4,5 Å.

No caso da conformação relaxada, foram consideradas válidas 45 interações, valor este inferior aos encontrados na conformação fechada, suportando o facto que devido à sua conformação mais relaxada, existe mais dificuldade em estabelecer interações com a proteína.

Tabela 9 - Número de interações identificadas identificadas em cada simulação

Interação	Fe(III) CO_3^{2-}	Fe(III) HCO_3^-	Fe(III) H_2CO_3	V(III) CO_3^{2-}	V(III) HCO_3^-	V(III) H_2CO_3	
M - π	NA	0	0	0	0	0	Conformação fechada (*iron-bound)*
M – Catião	NA	0	0	2	0	0	
M – Anião	NA	14	14	14	14	14	
M ··· H-S	NA	0	0	0	0	0	
M ··· H-N	NA	0	0	0	0	0	
M ··· H-O	NA	0	0	0	0	0	
M - π	0	0	0	0	0	0	Conformação relaxada (*oxalate-bound)*
M – Catião	2	1	0	0	0	0	
M – Anião	14	14	14	0	0	0	
M ··· H-S	0	0	0	0	0	0	
M ··· H-N	0	0	0	0	0	0	
M ··· H-O	0	0	0	0	0	0	
M - π	0	0	0	0	0	0	Conformação aberta (apo-proteína)
M – Catião	0	0	0	0	0	0	
M – Anião	0	2	0	0	0	5	
M ··· H-S	0	2	0	0	0	0	
M ··· H-N	0	0	0	0	0	0	
M ··· H-O	0	0	0	0	0	1	

No caso destas duas conformações, as interações predominantes são do tipo metal-anião, como esperado, com identificação esporádica de algumas interações repulsivas metal – catião, envolvendo o resíduo Lys296, um dos resíduos responsáveis pela alteração da conformação proteica em resposta ao pH do meio. Por fim, como seria de esperar, na conformação aberta, foram encontradas apenas 10 interações possíveis, uma vez que nesta conformação o local de ligação se encontra aberto.

É também importante frisar que não foram identificadas pelo *script* produzido inteirações relevantes do tipo M···H-X, sendo que este tipo de interações não se encontra descrito na literatura para a hTf.

Todos estes resultados foram obtidos utilizando um limite de 4,5 Å para a média de cada distância entre átomos ao longo da simulação. Foram também realizados alguns ensaios preliminares com um limite de 10 Å, tendo-se obtido um total de 58, 63, e 35 interações nas conformações fechada, relaxada e aberta, respetivamente. O aumento do número de interações está de acordo com o esperado pela aplicação de um *cut-off* superior, e a variação desse número de interações conforme a conformação da proteína está de acordo com o esperado. Neste caso (com um *cut-off* superior) verifica-se que os resíduos Lys206 e Lys296 aparecem na vizinhança do ambiente de coordenação (Figuras 7,8 e 9). Estes dois resíduos

participam na alteração conformacional da proteína em resposta à alteração do pH, sendo que a sua protonação favorece a abertura da proteína (e libertação do metal).

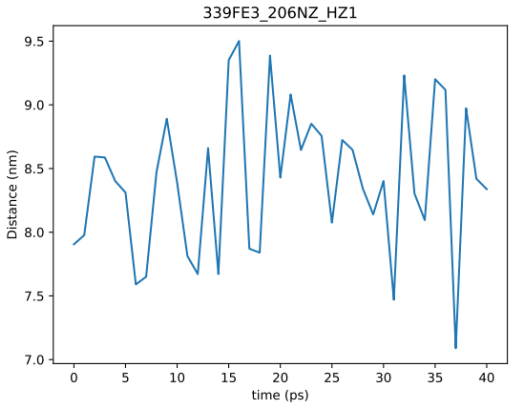

Figura 7 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Lys206NZ – H1(NZ) conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

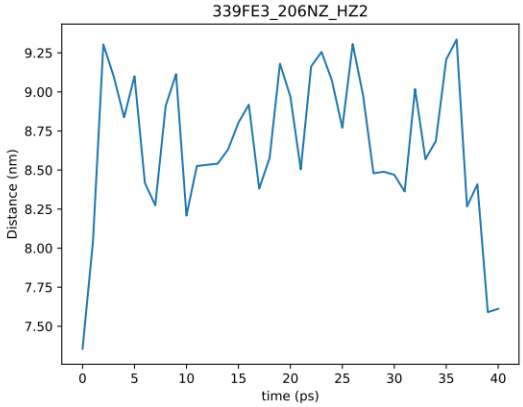

Figura 8 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Lys206NZ – H2(NZ) conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

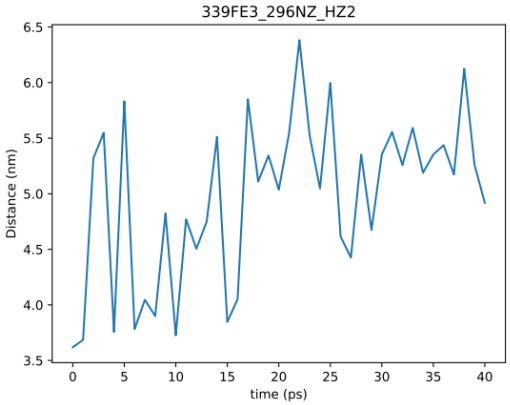

Figura 9 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Lys296NZ – H2(NZ) conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

3.2. AMBIENTE DE COORDENAÇÃO

De modo a fazer a comparação para um conjunto de dados semelhantes, de modo que se possa ter um termo de comparação, analisou-se a interação de cada uma das conformações da proteína com o metal na presença decarbonato nas diferentes formas de protonação (todas as interações e resultados contidos no capítulo 8 dos anexos). Os valores médios medidos para as distâncias entre átomos e para ângulos de ligação são apresentados na Tabela 10.

Tabela 10 - Ambiente de coordenação do metal na hTf. São apresentadas as médias das distâncias e ângulos para as interações do tipo metal-anião relevantes nas diferentes conformações da proteína.

	Conformação fechada (*iron-bound*)		**Conformação relaxada (*oxalate-bound)***		**Conformação aberta (apo-proteína)**	
	Distância média (Å)	Ângulo médio (º)	Distância média (Å)	Ângulo médio (º)	Distância média (Å)	Ângulo médio (º)
Fe(III) + CO_3^{2-}	-	-	2,3	21,9	NA	NA
Fe(III) + HCO_3^-	2,3	19,9	2,3	20,9	3,8	45,7
Fe(III) + H_2CO_3	2,1	17,6	2,2	17,6	NA	NA
V(III) + CO_3^{2-}	2,3	21,1	NA	NA	NA	NA
V(III) + HCO_3^-	2,2	19,6	NA	NA	NA	NA
V(III) + H_2CO_3	2,1	17,1	NA	NA	4,4	57,2

Os resultados obtidos utilizando o código desenvolvido indicam que, no caso da conformação fechada, responsável pelo transporte dos metais em circulação, o metal participa em interações do tipo metal-anião com distâncias médias entre 2,0 e 2,2 Å, com um valor médio de ângulo entre 17º e 22º. Estes resultados são compatíveis com um ambiente de coordenação típico, constituído por átomos coordenadores N e O. O mesmo ambiente de coordenação, medido assim, é mantido para V(III), de acordo com os dados experimentais existentes, que sustentam o transporte deste metal pela hTf.

Já no caso da conformação relaxada, em que o local de ligação é capaz de acomodar uma molécula de oxalato, mais volumosa que o carbonato, verifica-se no caso da ligação ao Fe o ambiente de coordenação se mantém, mas que a proteína é incapaz de ligar V.

A apoproteína (conformação aberta) está descrita na literatura como não transportando metais, sendo necessária uma alteração do pH do meio para que, através da desprotonação de resíduos específicos, tornando-a capaz de ligar metais. Os resultados obtidos indicam claramente que a proteína nesta conformação não liga eficazmente metais, uma vez que as distâncias médias medidas (apenas para dois casos) são superiores a 3,5 Å, sem importância biológica.

No caso do sistema $hTf^{relaxada}$-Fe(III)-HCO_3^-, foram, globalmente, encontradas interações do tipo M – anião, com os resíduos Asp63, Tyr95, Tyr188 e His249, e uma interação do tipo M – catião, com o resíduo Lys296 (Figura 10 a Figura 13). Todas estas interações estão descritas como estando envolvidas na coordenação do metal na hTf.

No mesmo sistema, utilizando a conformação relaxada, verifica-se que o metal interage com os mesmo resíduos (Asp63, Tyr95, Tyr188 e His249, Figura 14 a Figura 17), podendo também interagir, embora a uma distância superior, com o resíduo Lys296 (Figura 18)

No mesmo sistema, mas na conformação aberta, foram encontradas interações do tipo metal-anião/metal-SH com os resíduos Cys227 e Cys241 (Figura 19 e Figura 20), que não estão descritas como relevantes, e poderá ser apenas interações aleatórias à superfície da proteína.

Na presença de hidrogenocarbonato, verifica-se que a substituição de Fe(III) por V(III) não altera significativamente o ambiente de coordenação (dados apresentados em anexo, secção 6.8), reforçando a ideia que a proteína consegue transportar os dois metais.

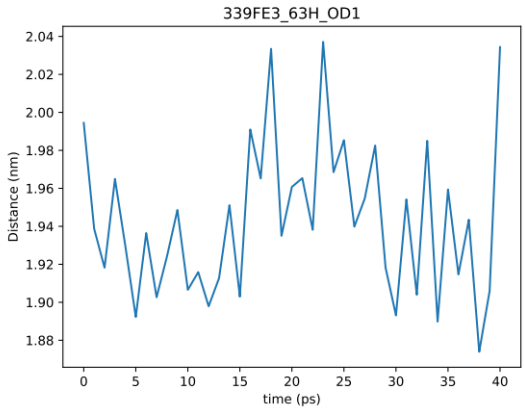

Figura 10 - Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD1 na conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

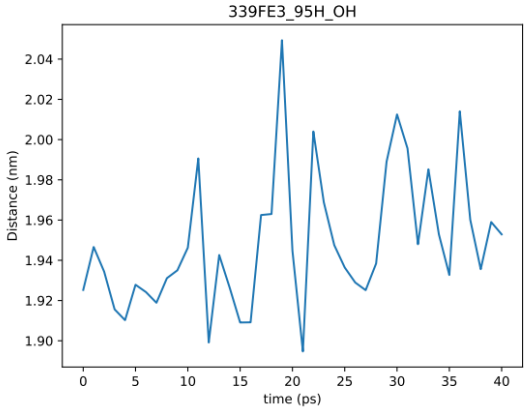

Figura 11 - Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr95O(fenol) na conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

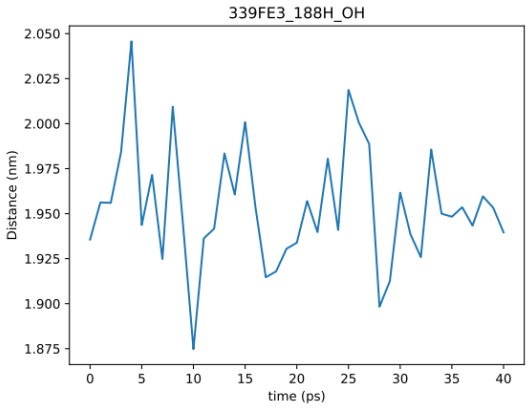

Figura 12 - Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr188O(fenol) na conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

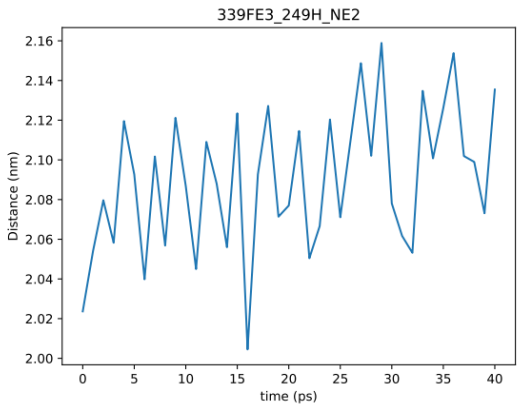

Figura 13 - Variação ao longo do tempo das distâncias para a interação Fe(III) – His249NE na conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

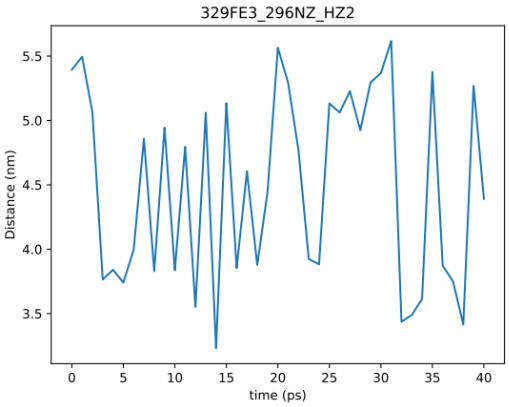

Figura 14 - Variação ao longo do tempo das distâncias para a interação Fe(III) – Lys296NZ na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

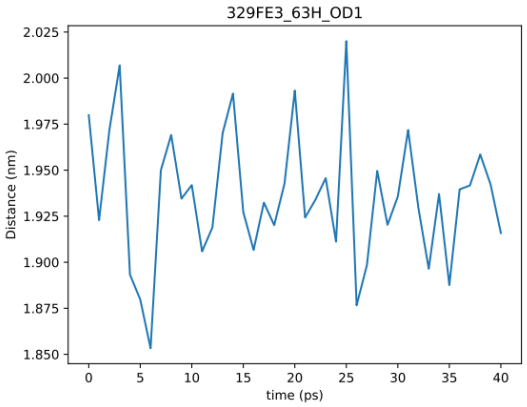

Figura 15 - Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD1 na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

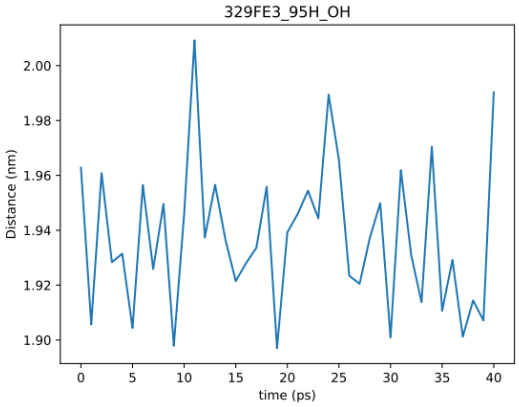

Figura 16 - Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr95O(fenol) na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

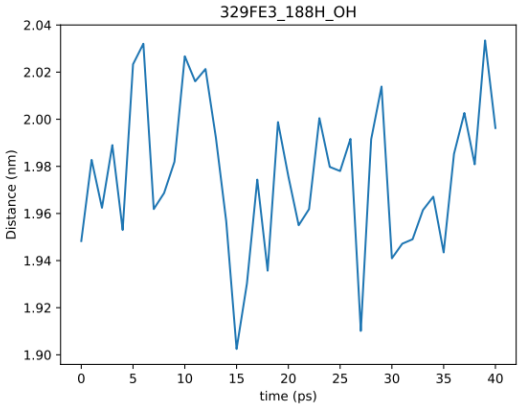

Figura 17 - Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr188O(fenol) na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

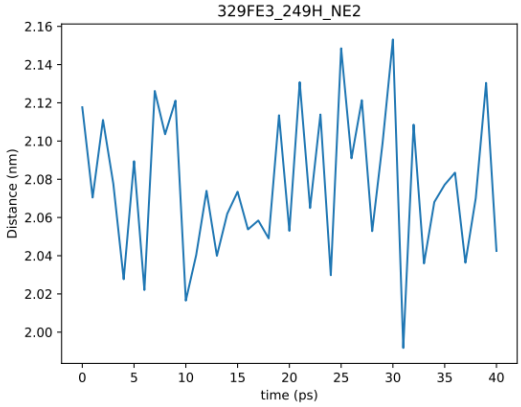

Figura 18 - Variação ao longo do tempo das distâncias para a interação Fe(III) – His249NE na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

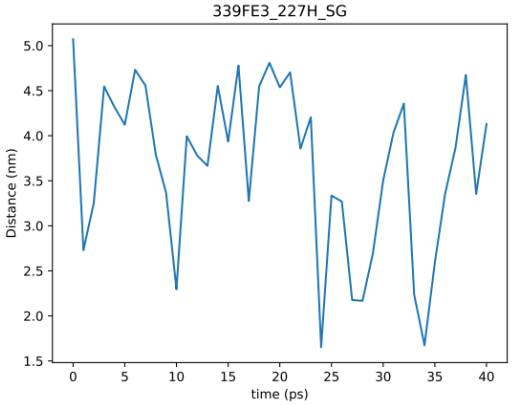

Figura 19 - Variação ao longo do tempo das distâncias para a interação Fe(III) – Cys227S na conformação aberta da proteína – sistema hTf + Fe(III) + HCO_3^-.

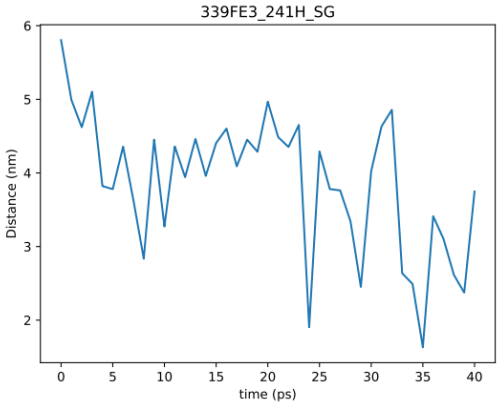

Figura 20 - Variação ao longo do tempo das distâncias para a interação Fe(III) – Cys241S na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

3.3. ANÁLISE DO ANIÃO FAVORECIDO

Na tentativa de perceber qual o anião favorecido durante as interações foi feita a análise das interações entre metal e aniões para as 3 conformações da proteína. Foi feita a escolha deste universo pois apresentava resultados de interações possivelmente úteis para estudo mediante os parâmetros de filtragem (médias de distâncias menores que 4,50 Å), para todos os metais Fe(III) e V(III) e com diferentes estados de protonação: resíduos COA (ião carbonato, CO_3^{2-}), COB (ião hidrogenocarbonato, $HCO3^-$) e COC (ácido carbónico, H_2CO_3). Foi realizada a análise dos valores das médias das distâncias e dos ângulos para cada interação singular e de seguida a média total para cada caso, sendo os resultados apresentados na tabela 10.

Face aos valores obtidos, e na tentativa de perceber qual o anião favorecido durante as interações dos metais e os aniões para a proteína, é possível indicar que existe um padrão entre as médias de cada metal, Fe(III) e V(III). Analisando em detalhe é possível perceber que os metais estão sempre coordenados pelos aminoácidos Asp63, Tyo95, Tyo188, Hit249.

Tendo em conta a média da distância é possível perceber que de forma padrão estas são inferiores nas interações dos metais com o estado de protonação do ácido carbónico (H_2CO_3), apresentando valores de médias das distâncias na ordem dos 2,05 a 2,09 Å para o caso da proteína fechada e 2,15 Å para a proteína relaxada. No que a ângulos diz respeito também foi possível verificar um padrão nestas interações, tendo tido um valor baixo comparativamente aos restantes na ordem entre 17º e os 21º.

Adicionalmente e tendo em conta que os valores se manifestaram mais baixos no estado de protonação do ácido carbónico, e usando a proteína na conformação fechada como caso de estudo, é possível afirmar que, apesar das interações serem as mesmas, elas foram ligeiramente mais baixas nas interações cujo metal era o Fe(III), com valores de médias das distâncias na ordem dos 2,05 Å enquanto no caso do metal V(III) as médias das distâncias foram de 2,09 Å . Pode-se assim afirmar que face aos valores observados a proteína 1a8e de conformação mais fechada poderá coordenar melhor Fe(III) vs. V(III) na presença de carbonato na forma protonada (H_2CO_3).

Por fim, numa tentativa de identificar se havia algum padrão entre os 4 *resíduos* para os quais estavam sempre contidos os aniões para as interações destas duas conformações (fechada e relaxada), foi feita a análise dos valores das médias das distâncias e dos ângulos para cada interação singular e de seguida a média total para cada caso (Tabelas 11 a 14).

Analisando os resultados obtidos foi possível verificar que tanto na proteína na conformação fechada como na relaxada existe coerência entre os valores e as interações. As médias foram mais baixas nas interações com o resíduo Tyr95, com valores de distâncias na ordem de 1,93 Å e 1,95 Å, e as médias dos ângulos foram também mais baixas. Em sentido inverso os valores mais altos foram verificados nas interações com o resíduo His249, com valores de distâncias na ordem de 3,02 Å e 3,1 Å. Estes dados estão de acordo com a literatura que sugere uma maior distância Fe-N na histidina do que Fe-O nos restantes aminoácidos.

Tabela 11 – Distância média (Å) para as Interações entre metais e aniões na conformação fechada da hTf

	Fe(III) + HCO_3^-	Fe(III) + H_2CO_3	V(III) + CO_3^{2-}	V(III) + HCO_3^-	V(III) + H_2CO_3	Média Distâncias
Asp63	2,9	1,9	2,9	2,9	1,9	2,6
Tyr95	1,9	1,9	1,9	1,9	1,9	1,9
Tyr188	1,9	1,9	1,9	1,9	1,9	1,9
His249	3,1	2,9	3,1	3,1	2,9	3,0

Tabela 12 – Ângulo médio para as Interações entre metais e aniões na conformação fechada da hTf

	Fe(III) + HCO_3^-	Fe(III) + H_2CO_3	V(III) + CO_3^{2-}	V(III) + HCO_3^-	V(III) + H_2CO_3	Média Ângulos
Asp63	29,4	15,8	30,7	28,2	17,2	24,3
Tyr95	14,4	14,3	15,4	14,6	13,5	14,4
Tyr188	16,7	15,2	18,6	16,9	15,1	16,5
His249	32,6	33,6	31,7	30,5	30,8	31,8

Tabela 13 - Distância média (Å) para as Interações entre metais e aniões na conformação relaxada da hTf

	Fe(III) + CO_3^{2-}	Fe(III) + HCO_3^-	Fe(III) + H_2CO_3	V(III) + CO_3^{2-}	V(III) + HCO_3^-	V(III) + H_2CO_3	Média Distâncias
Asp63	3,0	2,9	2,0	NA	NA	NA	2,5
Tyr95	1,9	1,9	1,9	NA	NA	NA	1,9
Tyr188	1,9	1,9	1,9	NA	NA	NA	1,9
His249	3,1	3,1	3,1	NA	NA	NA	3,1

Tabela 14 - Ângulo médio para as Interações entre metais e aniões na conformação relaxada da hTf

	Fe(III) + CO_3^{2-}	Fe(III) + HCO_3^-	Fe(III) + H_2CO_3	V(III) + CO_3^{2-}	V(III) + HCO_3^-	V(III) + H_2CO_3	Média Ângulos
Asp63	30,4	29,8	13,6	NA	NA	NA	21,7
Tyr95	17,4	16,0	15,9	NA	NA	NA	16,0
Tyr188	19,6	18,4	16,1	NA	NA	NA	17,3
His249	30,1	30,5	29,0	NA	NA	NA	29,7

4. CONCLUSÃO

Em conclusão, a utilização de uma ferramenta poderosa como a biblioteca *open source* orientada a objetos para análise estrutural e temporal de resultados de dinâmica molecular, MDAnalysis, torna a interpretação e análise dos resultados mais intuitiva. Uma vez que a estrutura de uma proteína não é estática, a possibilidade de se recorrer a esta ferramenta de Python bem como a conjugação com outras bibliotecas úteis, característica da própria linguagem Python, permite que sejam analisados de forma rápida e eficaz ficheiros de grandes dimensões provenientes de simulações de dinâmica molecular, que de outra forma seriam impossíveis de analisar sem recurso computacional com tanto detalhe, rapidez e eficácia.

O *script* desenvolvido foi aplicado ao estudo da capacidade de uma proteína envolvida no transporte de ferro, a transferrina, associada ao anião carbonato, transportar também vanádio, bem como à sua capacidade de utilizar outros aniões sinergísticos.

Foram analisadas as ligações para todos os tipos de interações possíveis alternando os metais entre Fe(III) e V(III) e com diferentes estados de protonação: COA (Ião carbonato, CO_3^{2-}), COB (Ião hidrogenocarbonato, HCO_3^-) e COC (Ácido carbónico, H_2CO_3).

Foi observado que conformações mais fechadas das proteínas (conformação ligada ao ferro e conformação ligada ao oxalato) tinham mais facilidade em estabelecer ligações, com médias de distância na ordem de 2,06 Å e 17º e 2,15 Å e 18º respetivamente, do que a conformação mais relaxada/aberta de todas 1bp5. Nos casos em que se verificou interações coincidentes, a distância média na conformação mais fechada é inferior à observada na conformação ligada ao oxalato (mais relaxada).

Na tentativa de perceber qual o anão favorecido, pela análise das interações entre os metais e os aniões para todas as proteínas, os resultados demostraram, que as distâncias médias eram mais baixas de forma padrão nas interações com o estado de protonação COC (Ácido carbónico, H_2CO_3) e cujo metal era Fe(III).

Entre as conformações mais fechadas verificou se que as interações com média de distância inferior a 4,5 Å foram sempre entre 4 resíduos, Asp63, Tyr95, Tyr188 e His249, do qual o resíduo Tyr95 apresentou a média das distâncias mais baixas na ordem dos 1,95 Å.

Por fim numa tentativa de identificar se havia algum padrão entre os resíduos para os quais estavam sempre contidos os aniões para as interações da conformação aberta foi feita a análise dos e concluiu-se que os valores mais baixos estavam nos resíduos Cys227 e Cys241, cujo resíduo é uma cisteína, com médias de distâncias entre os 3,7 e os 3,9 Å.

5. REFERÊNCIAS BIBLIOGRÁFICAS

1. Hospital, A., Goñi, J.R., Orozco, M., Gelpí, J.L.: Molecular dynamics simulations: advances and applications. Adv Appl Bioinform Chem. 8, 37–47 (2015). https://doi.org/10.2147/AABC.S70333

2. Durrant, J.D., McCammon, J.A.: Molecular dynamics simulations and drug discovery. BMC Biol. 9, 1–9 (2011). https://doi.org/10.1186/1741-7007-9-71/FIGURES/4

3. Polanski, J.: Chemoinformatics. Comprehensive Chemometrics. 4, 459–506 (2009). https://doi.org/10.1016/B978-044452701-1.00006-5

4. Han, X., Shin, W.H., Christoffer, C.W., Terashi, G., Monroe, L., Kihara, D.: Study of the Variability of the Native Protein Structure. Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics. 1–3, 606–619 (2019). https://doi.org/10.1016/b978-0-12-809633-8.20148-9

5. Pitman, M.R., Menz, R.I.: Methods for Protein Homology Modelling. Applied Mycology and Biotechnology. 6, 37–59 (2006). https://doi.org/10.1016/S1874-5334(06)80005-5

6. Krokhotin, A., Dokholyan, N. v.: Computational methods toward accurate RNA structure prediction using coarse-grained and all-atom models. Methods Enzymol. 553, 65–89 (2015). https://doi.org/10.1016/BS.MIE.2014.10.052

7. Konteatis, Z.D., Klon, A.E., Zou, J., Meshkat, S.: Computational Approach to De Novo Discovery of Fragment Binding for Novel Protein States. Methods Enzymol. 493, 357–380 (2011). https://doi.org/10.1016/B978-0-12-381274-2.00014-5

8. Loschwitz, J., Olubiyi, O.O., Hub, J.S., Strodel, B., Poojari, C.S.: Computer simulations of protein–membrane systems. Prog Mol Biol Transl Sci. 170, 273–403 (2020). https://doi.org/10.1016/BS.PMBTS.2020.01.001

9. Khalatur, P.G.: Molecular Dynamics Simulations in Polymer Science: Methods and Main Results. Polymer Science: A Comprehensive Reference, 10 Volume Set. 1, 417–460 (2012). https://doi.org/10.1016/B978-0-444-53349-4.00016-9

10. Michaud-Agrawal, N., Denning, E.J., Woolf, T.B., Beckstein, O.: MDAnalysis: A toolkit for the analysis of molecular dynamics simulations. J Comput Chem. 32, 2319–2327 (2011). https://doi.org/10.1002/JCC.21787

11. Gowers, R.J., Linke, M., Barnoud, J., Reddy, T.J.E., Melo, M.N., Seyler, S.L., Domanski, J., Dotson, D.L., Buchoux, S., Kenney, I.M., Beckstein, O.: MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations. Proceedings of the 15th Python in Science Conference. 98–105 (2019). https://doi.org/10.25080/MAJORA-629E541A-00E

12. Mathies, G., Gast, P., Chasteen, N.D., Luck, A.N., Mason, A.B., Groenen, E.J.J.: Exploring the Fe(III) binding sites of human serum transferrin with EPR at 275 GHz. Journal of Biological Inorganic Chemistry. 20, 487–496 (2015). https://doi.org/10.1007/S00775-014-1229-Z/FIGURES/7

13. Jeffrey, P.D., Bewley, M.C., MacGillivray, R.T.A., Mason, A.B., Woodworth, R.C., Baker, E.N.: Ligand-induced conformational change in transferrins: Crystal structure of the open form of the N-terminal half-molecule of human transferrin. Biochemistry. 37, 13978–13986 (1998). https://doi.org/10.1021/BI9812064/ASSET/IMAGES/LARGE/BI9812064F00005.JPEG

14. MacGillivray, R.T.A., Moore, S.A., Chen, J., Anderson, B.F., Baker, H., Luo, Y., Bewley, M., Smith, C.A., Murphy, M.E.P., Wang, Y., Mason, A.B., Woodworth, R.C., Brayer, G.D., Baker, E.N.: Two high-resolution crytal sructures of the recombinant N-lobe of human tranferrin reveal a structure change implicated in iron release. Biochemistry. 37, 7919–7928 (1998). https://doi.org/10.1021/BI980355J/ASSET/IMAGES/LARGE/BI980355JF00004.JPEG

15. Halbrooks, P.J., Mason, A.B., Adams, T.E., Briggs, S.K., Everse, S.J.: The Oxalate Effect on Release of Iron from Human Serum Transferrin Explained. J Mol Biol. 339, 217–226 (2004). https://doi.org/10.1016/J.JMB.2004.03.049

16. Jensen Frank: Introduction to Computational Chemistry, 3rd Edition | Wiley. WILEY. 660 (2017)

6. ANEXOS

6.1. ANEXO 1 - INSTALAÇÃO MDANALYSIS

Sendo o MDAnalysis uma ferramenta orientada a objetos escrita e compilada em Python, a maneira mais fácil de instalar e começar a utilizar esta ferramenta passa pela instalação do pacote utilizando “conda” ou “pip”.

Instalação com **conda**:

```
conda config --add channels conda-forge
conda install mdanalysis
```

Após instalar pode ser necessário actualizar o *software*:

```
conda update mdanalysis
```

Se for preciso realizar testes, a ferramenta possui um pacote complementar que facilita o processo:

```
conda install mdanalysistests
```

Instalação com **pip**:

```
pip install --upgrade MDAnalysis
```

Para actualizar:

```
pip install --upgrade MDAnalysis[analysis]
```

Para instalar o pacote de testes:

```
pip install --upgrade MDAnalysisTests
```

6.2. ANEXO 2 - MENUS DAS INTERAÇÕES COM ANÉIS AROMÁTICOS

```
------------------------------------------------------------------------------
---------------------------------- MENU --------------------------------------
------------------------------------------------------------------------------
--                                                                          --
-- 1 --> Interactions Between Aromatic Compounds                            --
-- 2 --> Aromatic-Cation / Aromatic-Anion / Aromatic-Metal Interactions      --
-- 3 --> XH-Aromatic Interactions                                           --
-- 9 --> Quit                                                               --
--                                                                          --
------------------------------------------------------------------------------
---------------------------------- MENU --------------------------------------
------------------------------------------------------------------------------

Please choose your option...
```

Figura A.1 - Menu Inicial das Interações com Anéis Aromáticos

```
Please choose your option...2
Option 2 Selected. Interactions Between Aromatic-Cation / Aromatic-Anion / Aromatic-Metal.
------------------------------------------------------------------------------
----------------------------- OPTION 2 MENU ----------------------------------
------------------------------------------------------------------------------
--                                                                        --
-- 1 --> Interactions Between Aromatic-Cation                             --
-- 2 --> Interactions Between Aromatic-Anion                              --
-- 3 --> Interactions Between Aromatic-Metal                              --
-- 4 --> Back to Main Menu                                                --
-- 9 --> Quit                                                             --
--                                                                        --
------------------------------------------------------------------------------
----------------------------- OPTION 2 MENU ----------------------------------
------------------------------------------------------------------------------

Please choose your option...
```

Figura A.2 - Menu das Interações entre Anéis Aromáticos e Catiões/Aniões/Metais

```
Please choose your option...3
Option 3 Selected. Interactions Between XH-Aromatic.
------------------------------------------------------------------------------
----------------------------- OPTION 3 MENU ----------------------------------
------------------------------------------------------------------------------
--                                                                          --
-- 1 --> Interactions Between Aromatic-HS                                   --
-- 2 --> Interactions Between Aromatic-HN                                   --
-- 3 --> Interactions Between Aromatic-HO                                   --
-- 4 --> Back to Main Menu                                                  --
-- 9 --> Quit                                                               --
--                                                                          --
------------------------------------------------------------------------------
----------------------------- OPTION 3 MENU ----------------------------------
------------------------------------------------------------------------------

Please choose your option...
```

Figura A.3 - Menu das Interações entre Anéis Aromáticos e Radicais XH

6.3. ANEXO 3 - MENUS DAS INTERAÇÕES COM METAIS

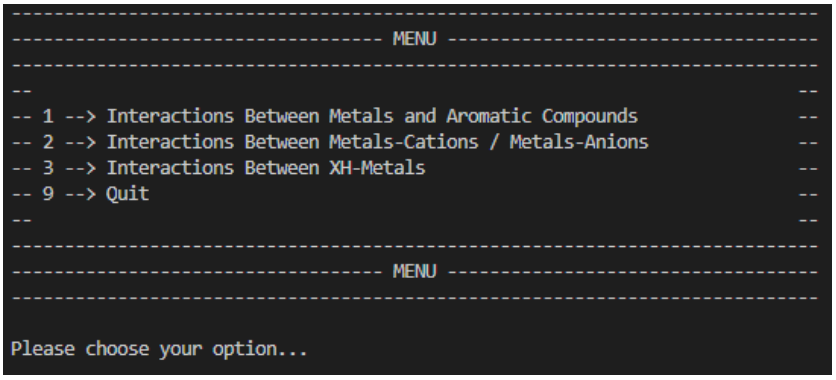

Figura A.4 - Menu Inicial das Interações com Metais

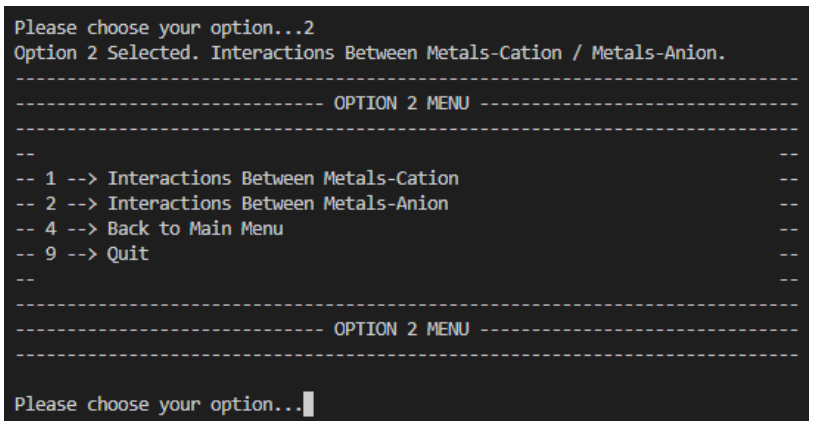

Figura A.5 - Menu das Interações entre Metais e Catiões/Aniões

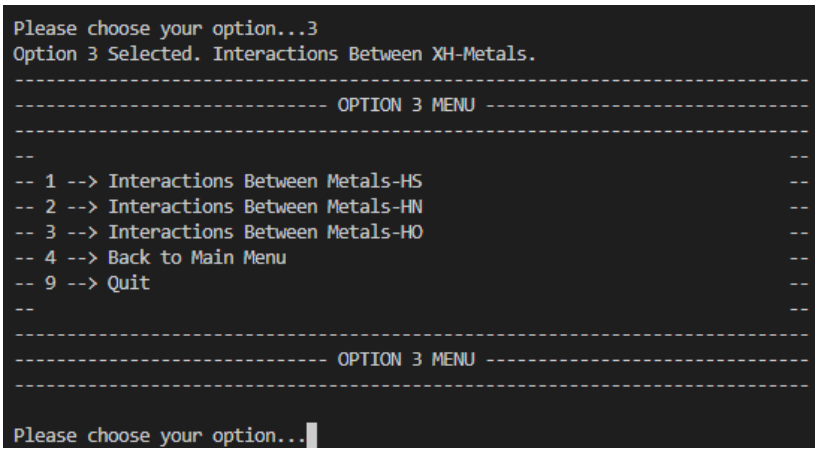

Figura A.6 - Menu das Interações entre Metais e Radicais XH

6.4. ANEXO 4 - MENU INICIAL

```python
#Main Menu function
#Menu customization, options definition and function callbacks
def mainMenu():
    print("------------------------------------------------------------------------------")
    print("---------------------------------- MENU ----------------------------------")
    print("------------------------------------------------------------------------------")
    print("--                                                                          --")
    print("-- 1 --> Interactions Between Metals and Aromatic Compounds                 --")
    print("-- 2 --> Interactions Between Metals-Cation / Metals-Anion                  --")
    print("-- 3 --> Interactions Between XH-Metals                                    --")
    print("-- 9 --> Quit                                                              --")
    print("--                                                                          --")
    print("------------------------------------------------------------------------------")
    print("---------------------------------- MENU ----------------------------------")
    print("------------------------------------------------------------------------------")
    print("")
    option = int(input("Please choose your option..."))
    if option == 1:
        print("Option 1 Selected. Interactions Between Metals and Aromatic Compounds.")
        functionMetalAromatic(A_aromatic_res,metal_res)
        print("Script Execution Started.")
    elif option == 2:
        print("Option 2 Selected. Interactions Between Metals-Cation / Metals-Anion.")
        option2Menu()
    elif option == 3:
        print("Option 3 Selected. Interactions Between XH-Metals.")
        option3Menu()
    elif option == 9:
        print("Option 9 Selected. Script Excecution Stopped.")
        quit()
    else:
        print("Option unavailable. Please Run the script again with an option contained in the menu.")
        unavailableOption(option)
```

Figura A.7 - Lógica do Menu Inicial

6.5. ANEXO 5 - MENU DE INTERAÇÕES COM CATIÕES E ANIÕES

```
#Interactions Between Metals-Cation / Metals-Anion Menu function
#Option 2 Selected.
#Menu customization, options definition and function callbacks.
def option2Menu():
    print("------------------------------------------------------------------------------")
    print("----------------------------- OPTION 2 MENU ----------------------------------")
    print("------------------------------------------------------------------------------")
    print("--                                                                          --")
    print("-- 1 --> Interactions Between Metals-Cation                                 --")
    print("-- 2 --> Interactions Between Metals-Anion                                  --")
    print("-- 4 --> Back to Main Menu                                                  --")
    print("-- 9 --> Quit                                                               --")
    print("--                                                                          --")
    print("------------------------------------------------------------------------------")
    print("----------------------------- OPTION 2 MENU ----------------------------------")
    print("------------------------------------------------------------------------------")
    print("")
    option2 = int(input("Please choose your option..."))
    if option2 == 1:
        print("Option 1 Selected. Interactions Between Metals-Cation.")
        print("Script Execution Started.")
        functionMetalCation(metal_res,B_cations_res)
        print("Script Execution Ended.")
    elif option2 == 2:
        print("Option 2 Selected. Interactions Between Metals-Anion.")
        print("Script Execution Started.")
        functionMetalAnion(metal_res,C_anions_res)
        print("Script Execution Ended.")
    elif option2 == 4:
        print("Option 4 Selected. Returned to Main Menu.")
        mainMenu()
    elif option2 == 9:
        print("Option 9 Selected. Script Excecution Stopped.")
        quit()
    else:
        print("Option unavailable. Please Run the script again with an option contained in the menu.")
        unavailableOption(option2)
```

Figura A.8 - Lógica do Menu de Interações com Catiões e Aniões

6.6. ANEXO 6 - MENU DE INTERAÇÕES COM TIOIS, AMINAS E HIDROXILO

```
#Interactions Between Metals-Cation / Metals-Anion /Menu function
#Option 3 Selected.
#Menu customization, options definition and function callbacks.
def option3Menu():
    print("------------------------------------------------------------------------------")
    print("----------------------------- OPTION 3 MENU ----------------------------------")
    print("------------------------------------------------------------------------------")
    print("--                                                                          --")
    print("-- 1 --> Interactions Between Metals-HS                                     --")
    print("-- 2 --> Interactions Between Metals-HN                                     --")
    print("-- 3 --> Interactions Between Metals-HO                                     --")
    print("-- 4 --> Back to Main Menu                                                  --")
    print("-- 9 --> Quit                                                               --")
    print("--                                                                          --")
    print("------------------------------------------------------------------------------")
    print("----------------------------- OPTION 3 MENU ----------------------------------")
    print("------------------------------------------------------------------------------")
    print("")
    option3 = int(input("Please choose your option..."))
    if option3 == 1:
        print("Option 1 Selected. Interactions Between Metals-HS.")
        print("Script Execution Started.")
        functionEList(metal_res,E_thiol)
        print("Script Execution Ended.")
    elif option3 == 2:
        print("Option 2 Selected. Interactions Between Metals-HN.")
        print("Script Execution Started.")
        functionFList(metal_res,F_amine)
        print("Script Execution Ended.")
    elif option3 == 3:
        print("Option 3 Selected. Interactions Between Metals-HO.")
        print("Script Execution Started.")
        functionGList(metal_res,G_hydroxyl)
        print("Script Execution Ended.")
    elif option3 == 4:
        print("Option 4 Selected. Returned to Main Menu.")
        mainMenu()
    elif option3 == 9:
        print("Option 9 Selected. Script Excecution Stopped.")
        quit()
    else:
        print("Option unavailable. Please Run the script again with an option contained in the menu.")
        unavailableOption(option3)
```

Figura A.9 - Lógica do Menu de Interações com Tiois, Aminas e Hidroxilo

6.7. ANEXO 7 - CÓDIGO DAS FUNÇÕES PRINCIPAIS

6.7.1. FUNÇÃO DAS INTERAÇÕES COM ANEIS AROMÁTICOS

```
#Interactions Between Aromatic Compounds and Metals
def functionMetalAromatic(input1,input2):
    count = 1
    currentDate=datetime.datetime.now()
    dateEpoch = calendar.timegm(currentDate.timetuple())
    logFilename = "logfile_" + str(dateEpoch) + ".log"
    newDirectoryName = str(dateEpoch)
    os.mkdir(newDirectoryName)
    logging.basicConfig(filename = logFilename , level = logging.INFO)
    logging.info(str(datetime.datetime.now()) + ": Execution Started.")
    logging.info(str(datetime.datetime.now()) + ": Option 1 Selected. Interactions Between
Aromatic Compounds and Metals.")
    logging.info(str(datetime.datetime.now())  +  ":  Execution  Directory  Created  '"  +
str(newDirectoryName) +"'.")
    print  ("Execution  log  file  can  be  found  in  ('"  +  str(newDirectoryName)  +  "/"  +
str(logFilename) + "').")

    idAromatic = []
    for i in input1:
        resid = i.resid
        idAromatic.append(resid)

    #339 FE3
    j=339
    k=338

    for i in idAromatic:
        xtc_aromatic = xtc.select_atoms("(resid " + str(i) + ")")
        xtc_aromatic_name = xtc_aromatic[0]
        xtc_aromatic_name = xtc_aromatic_name.resname

        xtc_metal = xtc.select_atoms("resid 339")
        xtc_metal_name = xtc_metal[0]
        xtc_metal_name = xtc_metal_name.name

        xtc_anion_metal = xtc.select_atoms("resid 338")
        xtc_anion_metal_name = xtc_anion_metal[0]
        xtc_anion_metal_name = xtc_anion_metal_name.resname

        logging.info(str(datetime.datetime.now()) + ": --> CURRENT INTERACTION")
        logging.info(str(datetime.datetime.now())  +  ":  Aromatic  -->  "  +  str(i)  +  "  "  +
str(xtc_aromatic_name) )
        logging.info(str(datetime.datetime.now()) + ": " + str(xtc_aromatic))
        logging.info(str(datetime.datetime.now())  +  ":  METAL  -->  "  +  str(j)  +  "  "  +
str(xtc_metal_name) + "_" + str(xtc_anion_metal_name) )
        logging.info(str(datetime.datetime.now())  +  ":  "  +  str(xtc_metal)  +  "  +  "   +
str(xtc_anion_metal))

        arrayFramesLoop = []
        arrayDistancesLoop = []
        arrayAnglesLoop = []
        arrayInteractionLoop = []
        arrayInteractionResidLoop = []
        arrayFrameDistanceAngles = []
```

```
        for ts in xtc.trajectory:
            positionAromatic = xtc_aromatic.positions
            positionMetal = xtc_metal.positions
            positionAnionMetal = xtc_anion_metal.positions
            centroidAromatic = getCenteroid(positionAromatic)
            centroidMetal = getCenteroid(positionMetal)
            centroidAnionMetal = getCenteroid(positionAnionMetal)
            #avgsMetal = [sum(vals)/len(positionMetal) for vals in zip(*positionMetal)]
            vector_Metal_AnionMetal = findVec(centroidMetal,centroidAnionMetal)
            vector_Metal_Aromatic = findVec(centroidMetal,centroidAromatic)
            angle=angle_between_degree(vector_Metal_AnionMetal,vector_Metal_Aromatic)
            dist      =      math.sqrt((centroidAromatic[0]-centroidAnionMetal[0])**2      +
(centroidAromatic[1]-centroidAnionMetal[1])**2           +           (centroidAromatic[2]-
centroidAnionMetal[2])**2 )
            arrayFramesLoop.append(ts.frame)
            arrayDistancesLoop.append(dist)
            arrayAnglesLoop.append(angle)
            arrayFrameDistanceAngles.append(str(xtc.trajectory.time)+ ',' + str(dist)+ ',' +
str(angle))
        arrayInteractionResidLoop.append(str(i) + "VS" + str(j))
        arrayInteractionLoop.append(str(xtc_aromatic_name) + "VS" + str(xtc_metal) + "_" +
str(xtc_anion_metal))
        mean = numpy.mean(arrayDistancesLoop)
        meanAngles = numpy.mean(arrayAnglesLoop)
        if (mean < maxMeanDistanceTest) and (meanAngles > minMeanAnglesTest) and (mean !=
0.0) :
            logging.info(str(datetime.datetime.now()) + ": Distance Mean (" + str(mean) +
")")
            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" + str(meanAngles)
+ ")")
            logging.info(str(datetime.datetime.now()) + ": Values will be added to the
plot.")
            arrayFrames.append(arrayFramesLoop)
            arrayDistances.append(arrayDistancesLoop)
            arrayAngles.append(arrayAnglesLoop)
            arrayInteraction.append(arrayInteractionLoop)
            arrayInteractionResid.append(arrayInteractionResidLoop)
            csvfile  =  str(count)+  "_"  +  str(arrayInteractionResidLoop[0])  +  "_"  +
str(dateEpoch) +".csv"
            interaction = str(i) + str(xtc_aromatic_name) + "_" + str(j) + str(xtc_metal_name)
+ "_" + str(k) + str(xtc_anion_metal_name)
            arrayAllResults.append(str(interaction)   +   ";"   +   str(mean)   +   ";"   +
str(meanAngles))
            csvresults = "results.csv"
            arrayPlotNameDistance.append(str(i) + str(xtc_aromatic_name) + "_" + str(j) +
str(xtc_metal_name) + "_" + str(k) + str(xtc_anion_metal_name))
            arrayPlotNameAngle.append(str(i)  +  str(xtc_aromatic_name)  +  "_"  +  str(j)  +
str(xtc_metal_name) + "_" + str(k) + str(xtc_anion_metal_name))
            with open(csvfile, "w") as fp:
                wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
                wr.writerow(['Avarage Distance', mean])
                wr.writerow(['Interaction', arrayInteractionLoop[0]])
                wr.writerow(['Frame','Distance','Angle'])
                for x in arrayFrameDistanceAngles :
                    wr.writerow ([x])
            shutil.move(csvfile, newDirectoryName)
            count += 1
            #logging.info(str(datetime.datetime.now()) + ": File Created -> " + str(csvfile))
            logging.info(str(datetime.datetime.now()) + ": File '" + str(csvfile) +"' Created
on Execution Directory (" + str(newDirectoryName) +").")
        else:
```

```
            #print("MEAN >= 10 --> Will be ignored")
            logging.info(str(datetime.datetime.now()) + ": Distance Mean (" + str(mean) +
")")
            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" + str(meanAngles)
+ ")")
            logging.info(str(datetime.datetime.now()) + ": Values Will be ignored.")
            #arrayFramesLoop.clear()
            #arrayDistancesLoop.clear()
            #arrayAnglesLoop.clear()
    logging.info(str(datetime.datetime.now()) + ": All interactions checked.")

    if arrayAllResults:
        with open(csvresults, "w") as fp:
            wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
            wr.writerow(['Interaction;Distance;Angle'])
            for x in arrayAllResults :
                wr.writerow ([x])
        shutil.move(csvresults, newDirectoryName)

    if(count > 1 ):
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Started.")

        #Plot Distance/Frames
        for index,x in enumerate(arrayDistances):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Distance (nm)")
            #plt.title("Metal_Aromatic - Distance")
            plt.title(str(arrayPlotNameDistance[0]))
            arrayPlotNameDistance.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            distancesPlotName   =   "Distance_Metal_Aromatic_"   +   labelString   +   "_"   +
str(dateEpoch)+ ".pdf"
            plt.savefig(distancesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Distance_Metal_Aromatic*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameDistances = "All_Distances_Metal_Aromatic_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameDistances)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #distancesPlotName = "Distances_SmallUniverse_SinglePlot_Test_" + labelString + "_"
+ str(dateEpoch)+ ".pdf"
        #plt.savefig(distancesPlotName)
        logging.info(str(datetime.datetime.now())     +     ":     Plot     File     '"     +
str(mergeFilenameDistances) +"' Created on Execution Directory (" + str(newDirectoryName)
+").")
        shutil.move(mergeFilenameDistances, newDirectoryName)

        #Clear Current Plot
        plt.clf()

        #Plot Angles/Frames
```

```
        for index,x in enumerate(arrayAngles):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Angles Amplitude (º)")
            #plt.title("Metal_Aromatic - Angles")
            plt.title(str(arrayPlotNameAngle[0]))
            arrayPlotNameAngle.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            anglesPlotName ="Angles_Metal_Aromatic_" + labelString + "_" + str(dateEpoch)+
".pdf"
            plt.savefig(anglesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Angles_Metal_Aromatic*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameAngles = "All_Angles_Metal_Aromatic_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameAngles)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #logging.info(str(datetime.datetime.now())  +  ":  Plot  File  Created  ->  "  +
str(anglesPlotName))
        logging.info(str(datetime.datetime.now())    +    ":    Plot    File    '"    +
str(mergeFilenameAngles) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
        shutil.move(mergeFilenameAngles, newDirectoryName)

        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Ended.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
        #os.remove(logFilename)
    else:
        logging.info(str(datetime.datetime.now())  +  ":  No  interactions  matched  the
requisites.")
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process will be
ignored.")
        logging.info(str(datetime.datetime.now()) + ": No plots will be created.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
```

6.7.2. FUNÇÃO DAS INTERAÇÕES COM CATIÕES

```
#Interactions Between Metals-Cation Function
def functionMetalCation(input1,input2):
    count = 1
    currentDate=datetime.datetime.now()
    dateEpoch = calendar.timegm(currentDate.timetuple())
    logFilename = "logfile_" + str(dateEpoch) + ".log"
    newDirectoryName = str(dateEpoch)
    os.mkdir(newDirectoryName)
    logging.basicConfig(filename = logFilename , level = logging.INFO)
    logging.info(str(datetime.datetime.now()) + ": Execution Started.")
    logging.info(str(datetime.datetime.now()) + ": Option 1 Selected. Interactions Between
Metal-Cation.")
    logging.info(str(datetime.datetime.now())  +  ":  Execution  Directory  Created  '"  +
str(newDirectoryName) +"'.")
    print ("Execution log file can be found in ('" + str(newDirectoryName) + "/" +
str(logFilename) + "').")

    idMetal = []
    for i in input1:
        resid = i.resid
        idMetal.append(resid)

    idCation = []
    for i in input2:
        resid = i.resid
        idCation.append(resid)

    for i in idMetal:
        for j in idCation:
            getResid = gro.select_atoms("resid " + str(j)).residues
            getResid = str(getResid)
            getResid = getResid.split('[<Residue ')[1]
            getResid = getResid.split(',')[0]

            donor_list = []
            h_list =  []

            if (getResid == "ARG"):
                donor_list = ["NH1", "NH2"]
                h_list =  ["HH11","HH12","HH21","HH22"]
            if (getResid == "LYS"):
                donor_list = ["HZ1","HZ2","HZ3"]
                h_list =  ["NZ"]
            if (getResid == "LYSH"):
                donor_list = ["HZ1","HZ2","HZ3"]
                h_list =  ["NZ"]
            if (getResid == "LYSX"):
                donor_list = ["HZ1","HZ2","HZ3"]
                h_list =  ["NZ"]
            if (getResid == "LYSY"):
                donor_list = ["HZ1","HZ2","HZ3"]
                h_list =  ["NZ"]
            if (getResid == "LYSX"):
                donor_list = ["HZ1","HZ2","HZ3"]
                h_list =  ["NZ"]
            if (getResid == "HISH"):
                donor_list = ["ND1"]
                h_list =  ["HD1","HE2"]
```

```
            for d in donor_list:
                for n in h_list:
                    donor_select = gro.select_atoms("name " + str(d) + " and resid " + str(j))
                    h_select = gro.select_atoms("name " + str(n) + " and resid " + str(j))

                    if len(donor_select) > 0 and len(h_select) > 0 :
                        donor = donor_select[0]
                        h = h_select[0]

                        xtc_metal = xtc.select_atoms("(resid " + str(i) + ")")
                        xtc_metal_name = xtc_metal[0]
                        xtc_metal_name = xtc_metal_name.resname

                        xtc_H = xtc.select_atoms("name " + str(h.name) + " and type " +
str(h.type) + " and resid " + str(h.resid))
                        xtc_H_name = xtc_H[0]
                        xtc_H_name = xtc_H_name.name

                        xtc_donor = xtc.select_atoms("name " + str(donor.name) + " and type
" + str(donor.type) + " and resid " + str(donor.resid))
                        xtc_donor_name = xtc_donor[0]
                        xtc_donor_name = xtc_donor_name.name

                        logging.info(str(datetime.datetime.now())   +   ":   -->   CURRENT
INTERACTION")
                        logging.info(str(datetime.datetime.now()) + ": Metal --> " + str(i)
+ " " + str(xtc_metal_name) )
                        logging.info(str(datetime.datetime.now()) + ": " + str(xtc_metal))
                        logging.info(str(datetime.datetime.now()) + ": Cation --> " + str(j)
+ " " + str(xtc_H_name) + "_" + str(xtc_donor_name) )
                        logging.info(str(datetime.datetime.now()) + ": " + str(xtc_H) + " +
"  + str(xtc_donor))

                        arrayFramesLoop = []
                        arrayDistancesLoop = []
                        arrayAnglesLoop = []
                        arrayInteractionLoop = []
                        arrayInteractionResidLoop = []
                        arrayFrameDistanceAngles = []
                        for ts in xtc.trajectory:
                            positionMetal = xtc_metal.positions
                            positionDonor = xtc_donor.positions
                            positionH = xtc_H.positions
                            centroidMetal = getCenteroid(positionMetal)
                            avgsDonor   =   [sum(vals)/len(positionDonor)   for   vals   in
zip(*positionDonor)]
                            avgsH = [sum(vals)/len(positionH) for vals in zip(*positionH)]
                            vector_H_Donor = findVec(avgsH,avgsDonor)
                            vector_H_Centroid = findVec(avgsH,centroidMetal)
                            angle=angle_between_degree(vector_H_Donor,vector_H_Centroid)
                            dist    =    math.sqrt((centroidMetal[0]-avgsDonor[0])**2    +
(centroidMetal[1]-avgsDonor[1])**2 + (centroidMetal[2]-avgsDonor[2])**2 )
                            arrayFramesLoop.append(ts.frame)
                            arrayDistancesLoop.append(dist)
                            arrayAnglesLoop.append(angle)
                            arrayFrameDistanceAngles.append(str(xtc.trajectory.time)+ ',' +
str(dist)+ ',' + str(angle))
                        arrayInteractionResidLoop.append(str(i) + "VS" + str(j))
                        arrayInteractionLoop.append(str(xtc_metal_name)    +    "VS"    +
str(xtc_H_name) + "_" + str(xtc_donor_name) )
                        mean = numpy.mean(arrayDistancesLoop)
```

```
                        meanAngles = numpy.mean(arrayAnglesLoop)
                        if (mean < maxMeanDistanceTest) and (meanAngles > minMeanAnglesTest)
and (mean != 0.0) :
                            logging.info(str(datetime.datetime.now()) + ": Distance Mean ("
+ str(mean) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" +
str(meanAngles) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Values will be
added to the plot.")
                            arrayFrames.append(arrayFramesLoop)
                            arrayDistances.append(arrayDistancesLoop)
                            arrayAngles.append(arrayAnglesLoop)
                            arrayInteraction.append(arrayInteractionLoop)
                            arrayInteractionResid.append(arrayInteractionResidLoop)
                            csvfile = str(count)+ "_" + str(arrayInteractionResidLoop[0]) +
"_" + str(dateEpoch) +".csv"
                            interaction = str(i) + str(xtc_metal_name) + "_" + str(j) +
str(xtc_H_name) + "_" + str(xtc_donor_name)
                            arrayAllResults.append(str(interaction) + ";" + str(mean) + ";"
+ str(meanAngles))
                            csvresults = "results.csv"
                            arrayPlotNameDistance.append(str(i) + str(xtc_metal_name) + "_"
+ str(j) + str(xtc_H_name) + "_" + str(xtc_donor_name))
                            arrayPlotNameAngle.append(str(i) + str(xtc_metal_name) + "_" +
str(j) + str(xtc_H_name) + "_" + str(xtc_donor_name))
                            with open(csvfile, "w") as fp:
                                wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
                                wr.writerow(['Avarage Distance', mean])
                                wr.writerow(['Interaction', arrayInteractionLoop[0]])
                                wr.writerow(['Frame','Distance','Angle'])
                                for x in arrayFrameDistanceAngles :
                                    wr.writerow ([x])
                            shutil.move(csvfile, newDirectoryName)
                            count += 1
                            #logging.info(str(datetime.datetime.now()) + ": File Created ->
" + str(csvfile))
                            logging.info(str(datetime.datetime.now())  +  ":  File  '"  +
str(csvfile) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
                        else:
                            #print("MEAN >= 10 --> Will be ignored")
                            logging.info(str(datetime.datetime.now()) + ": Distance Mean ("
+ str(mean) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" +
str(meanAngles) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Values Will be
ignored.")
                            #arrayFramesLoop.clear()
                            #arrayDistancesLoop.clear()
                            #arrayAnglesLoop.clear()
    logging.info(str(datetime.datetime.now()) + ": All interactions checked.")

    if arrayAllResults:
        with open(csvresults, "w") as fp:
            wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
            wr.writerow(['Interaction;Distance;Angle'])
            for x in arrayAllResults :
                wr.writerow ([x])
        shutil.move(csvresults, newDirectoryName)

    if(count > 1 ):
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Started.")
```

```
        #Plot Distance/Frames
        for index,x in enumerate(arrayDistances):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Distance (nm)")
            #plt.title("Metal_Cation - Distance")
            plt.title(str(arrayPlotNameDistance[0]))
            arrayPlotNameDistance.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            distancesPlotName   =   "Distance_Metal_Cation_"   +   labelString   +   "_"   +
str(dateEpoch)+ ".pdf"
            plt.savefig(distancesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Distance_Metal_Cation*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameDistances = "All_Distances_Metal_Cation_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameDistances)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #distancesPlotName = "Distances_SmallUniverse_SinglePlot_Test_" + labelString + "_"
+ str(dateEpoch)+ ".pdf"
        #plt.savefig(distancesPlotName)
        logging.info(str(datetime.datetime.now())     +     ":     Plot     File     '"     +
str(mergeFilenameDistances) +"' Created on Execution Directory (" + str(newDirectoryName)
+").")
        shutil.move(mergeFilenameDistances, newDirectoryName)

        #Clear Current Plot
        plt.clf()

        #Plot Angles/Frames

        for index,x in enumerate(arrayAngles):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Angles Amplitude (º)")
            #plt.title("Metal_Cation - Angles")
            plt.title(str(arrayPlotNameAngle[0]))
            arrayPlotNameAngle.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            anglesPlotName  ="Angles_Metal_Cation_"  +  labelString  +  "_"  +  str(dateEpoch)+
".pdf"
            plt.savefig(anglesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Angles_Metal_Cation*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
```

```
        mergeFilenameAngles = "All_Angles_Metal_Cation_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameAngles)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #logging.info(str(datetime.datetime.now())  +  ":  Plot  File  Created  ->  "  +
str(anglesPlotName))
        logging.info(str(datetime.datetime.now())    +    ":    Plot    File    '"    +
str(mergeFilenameAngles) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
        shutil.move(mergeFilenameAngles, newDirectoryName)

        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Ended.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
        #os.remove(logFilename)
    else:
        logging.info(str(datetime.datetime.now()) + ": No interactions matched the requisites
((mean < 4.5) and (meanAngles > 120) and (mean != 0.0)).")
        logging.info(str(datetime.datetime.now())  +  ":  Graphics  Creation  Process  will  be
ignored.")
        logging.info(str(datetime.datetime.now()) + ": No plots will be created.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
```

6.7.3. FUNÇÃO DAS INTERAÇÕES COM ANIÕES

```
#Interactions Between Metals-Anion Function
def functionMetalAnion(input1,input2):
    count = 1
    currentDate=datetime.datetime.now()
    dateEpoch = calendar.timegm(currentDate.timetuple())
    logFilename = "logfile_" + str(dateEpoch) + ".log"
    newDirectoryName = str(dateEpoch)
    os.mkdir(newDirectoryName)
    logging.basicConfig(filename = logFilename , level = logging.INFO)
    logging.info(str(datetime.datetime.now()) + ": Execution Started.")
    logging.info(str(datetime.datetime.now()) + ": Option 2 Selected. Interactions Between
Metal-Anion.")
    logging.info(str(datetime.datetime.now())  +  ":  Execution  Directory  Created  '"  +
str(newDirectoryName) +"'.")
    print ("Execution log file can be found in ('" + str(newDirectoryName) + "/" +
str(logFilename) + "').")

    idMetal = []
    for i in input1:
        resid = i.resid
        idMetal.append(resid)

    idAnion = []
    for i in input2:
        resid = i.resid
        idAnion.append(resid)


    for i in idMetal:
        for j in idAnion:
            getResid = gro.select_atoms("resid " + str(j)).residues
            getResid = str(getResid)
            getResid = getResid.split('[<Residue ')[1]
            getResid = getResid.split(',')[0]

            donor_list = []
            h_list =  []

            ######################################
            #NEEDS VALIDATION
            ######################################

            if (getResid == "ASP"):
                donor_list = ["OD1", "OD2"]
                h_list =  ["H"]
            if (getResid == "CYS"):
                donor_list = ["SG"]
                h_list =  ["H"]
            if (getResid == "GLU"):
                donor_list = ["OE1","OE2"]
                h_list =  ["H"]
            if (getResid == "TYR"):
                donor_list = ["OH"]
                h_list =  ["H", "HD1", "HD2", "HE1", "HE2"]
            if (getResid == "TYO"):
                donor_list = ["OH"]
                h_list =  ["H", "HD1", "HD2", "HE1", "HE2"]
            if (getResid == "HIT"):
                donor_list = ["NE2","ND1"]
```

```
                h_list =  ["H"]

            for d in donor_list:
                for n in h_list:
                    donor_select = gro.select_atoms("name " + str(d) + " and resid " + str(j))
                    h_select = gro.select_atoms("name " + str(n) + " and resid " + str(j))

                    if len(donor_select) > 0 and len(h_select) > 0 :
                        donor = donor_select[0]
                        h = h_select[0]

                        xtc_metal = xtc.select_atoms("(resid " + str(i) + ")")
                        xtc_metal_name = xtc_metal[0]
                        xtc_metal_name = xtc_metal_name.resname

                        xtc_H = xtc.select_atoms("name " + str(h.name) + " and type " +
str(h.type) + " and resid " + str(h.resid))
                        xtc_H_name = xtc_H[0]
                        xtc_H_name = xtc_H_name.name

                        xtc_donor = xtc.select_atoms("name " + str(donor.name) + " and type
" + str(donor.type) + " and resid " + str(donor.resid))
                        xtc_donor_name = xtc_donor[0]
                        xtc_donor_name = xtc_donor_name.name

                        logging.info(str(datetime.datetime.now())   +   ":   -->   CURRENT
INTERACTION")
                        logging.info(str(datetime.datetime.now()) + ": Metal --> " + str(i)
+ " " + str(xtc_metal_name) )
                        logging.info(str(datetime.datetime.now()) + ": " + str(xtc_metal))
                        logging.info(str(datetime.datetime.now()) + ": Anion --> " + str(j)
+ " " + str(xtc_H_name) + "_" + str(xtc_donor_name) )
                        logging.info(str(datetime.datetime.now()) + ": " + str(xtc_H) + " +
"  + str(xtc_donor))

                        arrayFramesLoop = []
                        arrayDistancesLoop = []
                        arrayAnglesLoop = []
                        arrayInteractionLoop = []
                        arrayInteractionResidLoop = []
                        arrayFrameDistanceAngles = []
                        for ts in xtc.trajectory:
                            positionMetal = xtc_metal.positions
                            positionDonor = xtc_donor.positions
                            positionH = xtc_H.positions
                            centroidMetal = getCenteroid(positionMetal)
                            avgsDonor   =   [sum(vals)/len(positionDonor)   for   vals   in
zip(*positionDonor)]
                            avgsH = [sum(vals)/len(positionH) for vals in zip(*positionH)]
                            vector_H_Donor = findVec(avgsH,avgsDonor)
                            vector_H_Centroid = findVec(avgsH,centroidMetal)
                            angle=angle_between_degree(vector_H_Donor,vector_H_Centroid)
                            dist    =    math.sqrt((centroidMetal[0]-avgsDonor[0])**2    +
(centroidMetal[1]-avgsDonor[1])**2 + (centroidMetal[2]-avgsDonor[2])**2 )
                            arrayFramesLoop.append(ts.frame)
                            arrayDistancesLoop.append(dist)
                            arrayAnglesLoop.append(angle)
                            arrayFrameDistanceAngles.append(str(xtc.trajectory.time)+ ',' +
str(dist)+ ',' + str(angle))
                        arrayInteractionResidLoop.append(str(i) + "VS" + str(j))
```

```
                        arrayInteractionLoop.append(str(xtc_metal_name)     +     "VS"     +
str(xtc_H_name) + "_" + str(xtc_donor_name) )
                        mean = numpy.mean(arrayDistancesLoop)
                        meanAngles = numpy.mean(arrayAnglesLoop)
                        #if (mean < maxMeanDistance) and (meanAngles > minMeanAngles) and
(mean != 0.0) : # ----------------> Valor falado na reunião
                        #if (mean < 15) and (meanAngles > 120) and (mean != 0.0) : # ------
----------> Forçar Valores
                        if (mean < maxMeanDistanceTest) and (meanAngles > minMeanAnglesTest)
and (mean != 0.0) :
                            logging.info(str(datetime.datetime.now()) + ": Distance Mean ("
+ str(mean) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" +
str(meanAngles) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Values will be
added to the plot.")
                            arrayFrames.append(arrayFramesLoop)
                            arrayDistances.append(arrayDistancesLoop)
                            arrayAngles.append(arrayAnglesLoop)
                            arrayInteraction.append(arrayInteractionLoop)
                            arrayInteractionResid.append(arrayInteractionResidLoop)
                            csvfile = str(count)+ "_" + str(arrayInteractionResidLoop[0]) +
"_" + str(dateEpoch) +".csv"
                            interaction = str(i) + str(xtc_metal_name) + "_" + str(j) +
str(xtc_H_name) + "_" + str(xtc_donor_name)
                            arrayAllResults.append(str(interaction) + ";" + str(mean) + ";"
+ str(meanAngles))
                            csvresults = "results.csv"
                            arrayPlotNameDistance.append(str(i) + str(xtc_metal_name) + "_"
+ str(j) + str(xtc_H_name) + "_" + str(xtc_donor_name))
                            arrayPlotNameAngle.append(str(i) + str(xtc_metal_name) + "_" +
str(j) + str(xtc_H_name) + "_" + str(xtc_donor_name))
                            with open(csvfile, "w") as fp:
                                wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
                                wr.writerow(['Avarage Distance', mean])
                                wr.writerow(['Interaction', arrayInteractionLoop[0]])
                                wr.writerow(['Frame','Distance','Angle'])
                                for x in arrayFrameDistanceAngles :
                                    wr.writerow ([x])
                            shutil.move(csvfile, newDirectoryName)
                            count += 1
                            #logging.info(str(datetime.datetime.now()) + ": File Created ->
" + str(csvfile))
                            logging.info(str(datetime.datetime.now())  +  ":  File  '"  +
str(csvfile) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
                        else:
                            #print("MEAN >= 10 --> Will be ignored")
                            logging.info(str(datetime.datetime.now()) + ": Distance Mean ("
+ str(mean) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" +
str(meanAngles) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Values Will be
ignored.")
                            #arrayFramesLoop.clear()
                            #arrayDistancesLoop.clear()
                            #arrayAnglesLoop.clear()
    logging.info(str(datetime.datetime.now()) + ": All interactions checked.")

    if arrayAllResults:
        with open(csvresults, "w") as fp:
            wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
```

```
            wr.writerow(['Interaction;Distance;Angle'])
            for x in arrayAllResults :
                wr.writerow ([x])
        shutil.move(csvresults, newDirectoryName)

    if(count > 1 ):
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Started.")

        #Plot Distance/Frames
        for index,x in enumerate(arrayDistances):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Distance (nm)")
            #plt.title("Metal_Anion - Distance")
            plt.title(str(arrayPlotNameDistance[0]))
            arrayPlotNameDistance.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            distancesPlotName = "Distance_Metal_Anion_" + labelString + "_" + str(dateEpoch)+
".pdf"
            plt.savefig(distancesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Distance_Metal_Anion*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameDistances = "All_Distances_Metal_Anion_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameDistances)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #distancesPlotName = "Distances_SmallUniverse_SinglePlot_Test_" + labelString + "_"
+ str(dateEpoch)+ ".pdf"
        #plt.savefig(distancesPlotName)
        logging.info(str(datetime.datetime.now())    +    ":    Plot    File    '"    +
str(mergeFilenameDistances) +"' Created on Execution Directory (" + str(newDirectoryName)
+").")
        shutil.move(mergeFilenameDistances, newDirectoryName)

        #Clear Current Plot
        plt.clf()

        #Plot Angles/Frames

        for index,x in enumerate(arrayAngles):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Angles Amplitude (º)")
            #plt.title("Metal_Anion - Angles")
            plt.title(str(arrayPlotNameAngle[0]))
            arrayPlotNameAngle.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            anglesPlotName ="Angles_Metal_Anion_" + labelString + "_" + str(dateEpoch)+
".pdf"
            plt.savefig(anglesPlotName)
            plt.clf()
```

```
        pwd=os.getcwd()
        pattern = "/Angles_Metal_Anion*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameAngles = "All_Angles_Metal_Anion_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameAngles)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #logging.info(str(datetime.datetime.now())  +  ":  Plot  File  Created  ->  "  +
str(anglesPlotName))
        logging.info(str(datetime.datetime.now())    +    ":    Plot    File    '"    +
str(mergeFilenameAngles) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
        shutil.move(mergeFilenameAngles, newDirectoryName)

        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Ended.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
        #os.remove(logFilename)
    else:
        logging.info(str(datetime.datetime.now())  +  ":  No  interactions  matched  the
requisites.")
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process will be
ignored.")
        logging.info(str(datetime.datetime.now()) + ": No plots will be created.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
```

6.7.4. FUNÇÃO DAS INTERAÇÕES COM INTERAÇÕES X-SH

```
#Metal-SH Function
def functionEList(input1,input2):
    count = 1
    currentDate=datetime.datetime.now()
    dateEpoch = calendar.timegm(currentDate.timetuple())
    logFilename = "logfile_" + str(dateEpoch) + ".log"
    newDirectoryName = str(dateEpoch)
    os.mkdir(newDirectoryName)
    logging.basicConfig(filename = logFilename , level = logging.INFO)
    logging.info(str(datetime.datetime.now()) + ": Execution Started.")
    logging.info(str(datetime.datetime.now()) + ": Option 1 Selected. Interactions Between
Metal-HS.")
    logging.info(str(datetime.datetime.now())  +  ":  Execution  Directory  Created  '"  +
str(newDirectoryName) +"'.")
    print ("Execution log file can be found in ('" + str(newDirectoryName) + "/" +
str(logFilename) + "').")

    idMetal = []
    for i in input1:
        resid = i.resid
        idMetal.append(resid)

    idThiol = []
    for i in input2:
        resid = i.resid
        idThiol.append(resid)
    idThiol_NoDups = ([item for item, count in collections.Counter(idThiol).items() if count
> 1])

    for i in idMetal:
        for j in idThiol_NoDups:
            getResid = gro.select_atoms("resid " + str(j)).residues
            getResid = str(getResid)
            getResid = getResid.split('[<Residue ')[1]
            getResid = getResid.split(',')[0]

            donor_list = []
            h_list =  []

            if (getResid == "CYS"):
                donor_list = ["SG"]
                h_list =  ["H"]

            for d in donor_list:
                for n in h_list:
                    donor_select = gro.select_atoms("name " + str(d) + " and resid " + str(j))
                    h_select = gro.select_atoms("name " + str(n) + " and resid " + str(j))

                    if len(donor_select) > 0 and len(h_select) > 0 :
                        donor = donor_select[0]
                        h = h_select[0]

                        xtc_metal = xtc.select_atoms("(resid " + str(i) + ")")
                        xtc_metal_name = xtc_metal[0]
                        xtc_metal_name = xtc_metal_name.resname

                        xtc_H = xtc.select_atoms("name " + str(h.name) + " and type " +
str(h.type) + " and resid " + str(h.resid))
                        xtc_H_name = xtc_H[0]
```

```
                        xtc_H_name = xtc_H_name.name

                        xtc_donor = xtc.select_atoms("name " + str(donor.name) + " and type
" + str(donor.type) + " and resid " + str(donor.resid))
                        xtc_donor_name = xtc_donor[0]
                        xtc_donor_name = xtc_donor_name.name

                        logging.info(str(datetime.datetime.now())   +   ":   -->   CURRENT
INTERACTION")
                        logging.info(str(datetime.datetime.now()) + ": Metal --> " + str(i)
+ " " + str(xtc_metal_name) )
                        logging.info(str(datetime.datetime.now()) + ": " + str(xtc_metal))
                        logging.info(str(datetime.datetime.now()) + ": Thiol --> " + str(j)
+ " " + str(xtc_H_name) + "_" + str(xtc_donor_name) )
                        logging.info(str(datetime.datetime.now()) + ": " + str(xtc_H) + " +
"  + str(xtc_donor))

                        arrayFramesLoop = []
                        arrayDistancesLoop = []
                        arrayAnglesLoop = []
                        arrayInteractionLoop = []
                        arrayInteractionResidLoop = []
                        arrayFrameDistanceAngles = []
                        for ts in xtc.trajectory:
                            positionMetal = xtc_metal.positions
                            positionDonor = xtc_donor.positions
                            positionH = xtc_H.positions
                            centroidMetal = getCenteroid(positionMetal)
                            avgsDonor   =   [sum(vals)/len(positionDonor)   for   vals   in
zip(*positionDonor)]
                            avgsH = [sum(vals)/len(positionH) for vals in zip(*positionH)]
                            vector_H_Donor = findVec(avgsH,avgsDonor)
                            vector_H_Centroid = findVec(avgsH,centroidMetal)
                            angle=angle_between_degree(vector_H_Donor,vector_H_Centroid)
                            dist    =    math.sqrt((centroidMetal[0]-avgsDonor[0])**2    +
(centroidMetal[1]-avgsDonor[1])**2 + (centroidMetal[2]-avgsDonor[2])**2 )
                            arrayFramesLoop.append(ts.frame)
                            arrayDistancesLoop.append(dist)
                            arrayAnglesLoop.append(angle)
                            arrayFrameDistanceAngles.append(str(xtc.trajectory.time)+ ',' +
str(dist)+ ',' + str(angle))
                        arrayInteractionResidLoop.append(str(i) + "VS" + str(j))
                        arrayInteractionLoop.append(str(xtc_metal_name)    +    "VS"    +
str(xtc_H_name) + "_" + str(xtc_donor_name) )
                        mean = numpy.mean(arrayDistancesLoop)
                        meanAngles = numpy.mean(arrayAnglesLoop)
                        #if (mean < maxMeanDistance) and (meanAngles > minMeanAngles) and
(mean != 0.0) : # ----------------> Valor falado na reunião
                        #if (mean < 15) and (meanAngles > 120) and (mean != 0.0) : # ------
----------> Forçar Valores
                        if (mean < maxMeanDistanceTest) and (meanAngles > minMeanAnglesTest)
and (mean != 0.0) :
                            logging.info(str(datetime.datetime.now()) + ": Distance Mean ("
+ str(mean) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" +
str(meanAngles) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Values will be
added to the plot.")
                            arrayFrames.append(arrayFramesLoop)
                            arrayDistances.append(arrayDistancesLoop)
                            arrayAngles.append(arrayAnglesLoop)
```

```
                            arrayInteraction.append(arrayInteractionLoop)
                            arrayInteractionResid.append(arrayInteractionResidLoop)
                            csvfile = str(count)+ "_" + str(arrayInteractionResidLoop[0]) +
"_" + str(dateEpoch) +".csv"
                            interaction = str(i) + str(xtc_metal_name) + "_" + str(j) +
str(xtc_H_name) + "_" + str(xtc_donor_name)
                            arrayAllResults.append(str(interaction) + ";" + str(mean) + ";"
+ str(meanAngles))
                            csvresults = "results.csv"
                            arrayPlotNameDistance.append(str(i) + str(xtc_metal_name) + "_"
+ str(j) + str(xtc_H_name) + "_" + str(xtc_donor_name))
                            arrayPlotNameAngle.append(str(i) + str(xtc_metal_name) + "_" +
str(j) + str(xtc_H_name) + "_" + str(xtc_donor_name))
                            with open(csvfile, "w") as fp:
                                wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
                                wr.writerow(['Avarage Distance', mean])
                                wr.writerow(['Interaction', arrayInteractionLoop[0]])
                                wr.writerow(['Frame','Distance','Angle'])
                                for x in arrayFrameDistanceAngles :
                                    wr.writerow ([x])
                            shutil.move(csvfile, newDirectoryName)
                            count += 1
                            #logging.info(str(datetime.datetime.now()) + ": File Created ->
" + str(csvfile))
                            logging.info(str(datetime.datetime.now())  +  ":  File  '"  +
str(csvfile) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
                        else:
                            #print("MEAN >= 10 --> Will be ignored")
                            logging.info(str(datetime.datetime.now()) + ": Distance Mean ("
+ str(mean) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" +
str(meanAngles) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Values Will be
ignored.")
                            #arrayFramesLoop.clear()
                            #arrayDistancesLoop.clear()
                            #arrayAnglesLoop.clear()
    logging.info(str(datetime.datetime.now()) + ": All interactions checked.")

    if arrayAllResults:
        with open(csvresults, "w") as fp:
            wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
            wr.writerow(['Interaction;Distance;Angle'])
            for x in arrayAllResults :
                wr.writerow ([x])
        shutil.move(csvresults, newDirectoryName)

    if(count > 1 ):
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Started.")

        #Plot Distance/Frames
        for index,x in enumerate(arrayDistances):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Distance (nm)")
            #plt.title("Metal_Thiol - Distance")
            plt.title(str(arrayPlotNameDistance[0]))
            arrayPlotNameDistance.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
```

```
            distancesPlotName = "Distance_Metal_Thiol_" + labelString + "_" + str(dateEpoch)+
".pdf"
            plt.savefig(distancesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Distance_Metal_Thiol*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameDistances = "All_Distances_Metal_Thiol_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameDistances)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #distancesPlotName = "Distances_SmallUniverse_SinglePlot_Test_" + labelString + "_"
+ str(dateEpoch)+ ".pdf"
        #plt.savefig(distancesPlotName)
        logging.info(str(datetime.datetime.now())     +     ":     Plot     File     '"     +
str(mergeFilenameDistances) +"' Created on Execution Directory (" + str(newDirectoryName)
+").")
        shutil.move(mergeFilenameDistances, newDirectoryName)

        #Clear Current Plot
        plt.clf()

        #Plot Angles/Frames

        for index,x in enumerate(arrayAngles):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Angles Amplitude (º)")
            #plt.title("Metal_Thiol - Angles")
            plt.title(str(arrayPlotNameAngle[0]))
            arrayPlotNameAngle.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            anglesPlotName  ="Angles_Metal_Thiol_"  +  labelString  +  "_"  +  str(dateEpoch)+
".pdf"
            plt.savefig(anglesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Angles_Metal_Thiol*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameAngles = "All_Angles_Metal_Thiol_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameAngles)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #logging.info(str(datetime.datetime.now())  +  ":  Plot  File  Created  ->  "  +
str(anglesPlotName))
        logging.info(str(datetime.datetime.now())     +     ":     Plot     File     '"     +
str(mergeFilenameAngles) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
        shutil.move(mergeFilenameAngles, newDirectoryName)
```

```
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Ended.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
        #os.remove(logFilename)
    else:
        logging.info(str(datetime.datetime.now())  +  ":  No  interactions  matched  the
requisites.")
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process will be
ignored.")
        logging.info(str(datetime.datetime.now()) + ": No plots will be created.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
```

6.7.5. FUNÇÃO DAS INTERAÇÕES COM INTERAÇÕES X-HN

```
#Metals-HN Function
def functionFList(input1,input2):
    count = 1
    currentDate=datetime.datetime.now()
    dateEpoch = calendar.timegm(currentDate.timetuple())
    logFilename = "logfile_" + str(dateEpoch) + ".log"
    newDirectoryName = str(dateEpoch)
    os.mkdir(newDirectoryName)
    logging.basicConfig(filename = logFilename , level = logging.INFO)
    logging.info(str(datetime.datetime.now()) + ": Execution Started.")
    logging.info(str(datetime.datetime.now()) + ": Option 2 Selected. Interactions Between
Metal-HN.")
    logging.info(str(datetime.datetime.now())  +  ":  Execution  Directory  Created  '"  +
str(newDirectoryName) +"'.")
    print ("Execution log file can be found in ('" + str(newDirectoryName) + "/" +
str(logFilename) + "').")

    idMetal = []
    for i in input1:
        resid = i.resid
        idMetal.append(resid)

    idAmine = []
    for i in input2:
        resid = i.resid
        idAmine.append(resid)
    idAmine_NoDups = ([item for item, count in collections.Counter(idAmine).items() if count
> 1])

    for i in idMetal:
        for j in idAmine_NoDups:
            getResid = gro.select_atoms("resid " + str(j)).residues
            getResid = str(getResid)
            getResid = getResid.split('[<Residue ')[1]
            getResid = getResid.split(',')[0]

            donor_list = []
            h_list =  []

            if (getResid == "ARG"):
                donor_list = ["NH1", "NH2"]
                h_list =  ["HH11","HH12","HH21","HH22","HH1"]
            if (getResid == "ARGN"):
                donor_list = ["NH1", "NH2"]
                h_list =  ["HH1","HH21","HH22"]
            if (getResid == "ASN"):
                donor_list = ["ND2"]
                h_list =  ["HD21","HD22"]
            if (getResid == "GLN"):
                donor_list = ["NE2"]
                h_list =  ["HE21","HE22"]
            if (getResid == "HIS"):
                donor_list = ["HD1","HE2"]
                h_list =  ["ND1","NE2"]
            if (getResid == "LYS"):
                donor_list = ["HZ1","HZ2","HZ3"]
                h_list =  ["NZ"]
            if (getResid == "LYSH"):
                donor_list = ["HZ1","HZ2","HZ3"]
```

```
                h_list =  ["NZ"]
            if (getResid == "TRP"):
                donor_list = ["HE1"]
                h_list =  ["NE1"]

            for d in donor_list:
                for n in h_list:
                    donor_select = gro.select_atoms("name " + str(d) + " and resid " + str(j))
                    h_select = gro.select_atoms("name " + str(n) + " and resid " + str(j))

                    if len(donor_select) > 0 and len(h_select) > 0 :
                        donor = donor_select[0]
                        h = h_select[0]

                        xtc_metal = xtc.select_atoms("(resid " + str(i) + ")")
                        xtc_metal_name = xtc_metal[0]
                        xtc_metal_name = xtc_metal_name.resname

                        xtc_H = xtc.select_atoms("name " + str(h.name) + " and type " +
str(h.type) + " and resid " + str(h.resid))
                        xtc_H_name = xtc_H[0]
                        xtc_H_name = xtc_H_name.name

                        xtc_donor = xtc.select_atoms("name " + str(donor.name) + " and type
" + str(donor.type) + " and resid " + str(donor.resid))
                        xtc_donor_name = xtc_donor[0]
                        xtc_donor_name = xtc_donor_name.name

                        logging.info(str(datetime.datetime.now())   +   ":   -->   CURRENT
INTERACTION")
                        logging.info(str(datetime.datetime.now()) + ": Metal --> " + str(i)
+ " " + str(xtc_metal_name) )
                        logging.info(str(datetime.datetime.now()) + ": " + str(xtc_metal))
                        logging.info(str(datetime.datetime.now()) + ": Thiol --> " + str(j)
+ " " + str(xtc_H_name) + "_" + str(xtc_donor_name) )
                        logging.info(str(datetime.datetime.now()) + ": " + str(xtc_H) + " +
"  + str(xtc_donor))

                        arrayFramesLoop = []
                        arrayDistancesLoop = []
                        arrayAnglesLoop = []
                        arrayInteractionLoop = []
                        arrayInteractionResidLoop = []
                        arrayFrameDistanceAngles = []
                        for ts in xtc.trajectory:
                            positionMetal = xtc_metal.positions
                            positionDonor = xtc_donor.positions
                            positionH = xtc_H.positions
                            centroidMetal = getCenteroid(positionMetal)
                            avgsDonor   =   [sum(vals)/len(positionDonor)   for   vals   in
zip(*positionDonor)]
                            avgsH = [sum(vals)/len(positionH) for vals in zip(*positionH)]
                            vector_H_Donor = findVec(avgsH,avgsDonor)
                            vector_H_Centroid = findVec(avgsH,centroidMetal)
                            angle=angle_between_degree(vector_H_Donor,vector_H_Centroid)
                            dist    =    math.sqrt((centroidMetal[0]-avgsDonor[0])**2    +
(centroidMetal[1]-avgsDonor[1])**2 + (centroidMetal[2]-avgsDonor[2])**2 )
                            arrayFramesLoop.append(ts.frame)
                            arrayDistancesLoop.append(dist)
                            arrayAnglesLoop.append(angle)
```

```
                            arrayFrameDistanceAngles.append(str(xtc.trajectory.time)+ ',' +
str(dist)+ ',' + str(angle))
                        arrayInteractionResidLoop.append(str(i) + "VS" + str(j))
                        arrayInteractionLoop.append(str(xtc_metal_name)     +     "VS"     +
str(xtc_H_name) + "_" + str(xtc_donor_name) )
                        mean = numpy.mean(arrayDistancesLoop)
                        meanAngles = numpy.mean(arrayAnglesLoop)
                        #if (mean < maxMeanDistance) and (meanAngles > minMeanAngles) and
(mean != 0.0) : # ----------------> Valor falado na reunião
                        #if (mean < 15) and (meanAngles > 120) and (mean != 0.0) : # ------
----------> Forçar Valores
                        if (mean < maxMeanDistanceTest) and (meanAngles > minMeanAnglesTest)
and (mean != 0.0) :
                            logging.info(str(datetime.datetime.now()) + ": Distance Mean ("
+ str(mean) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" +
str(meanAngles) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Values will be
added to the plot.")
                            arrayFrames.append(arrayFramesLoop)
                            arrayDistances.append(arrayDistancesLoop)
                            arrayAngles.append(arrayAnglesLoop)
                            arrayInteraction.append(arrayInteractionLoop)
                            arrayInteractionResid.append(arrayInteractionResidLoop)
                            csvfile = str(count)+ "_" + str(arrayInteractionResidLoop[0]) +
"_" + str(dateEpoch) +".csv"
                            interaction = str(i) + str(xtc_metal_name) + "_" + str(j) +
str(xtc_H_name) + "_" + str(xtc_donor_name)
                            arrayAllResults.append(str(interaction) + ";" + str(mean) + ";"
+ str(meanAngles))
                            csvresults = "results.csv"
                            arrayPlotNameDistance.append(str(i) + str(xtc_metal_name) + "_"
+ str(j) + str(xtc_H_name) + "_" + str(xtc_donor_name))
                            arrayPlotNameAngle.append(str(i) + str(xtc_metal_name) + "_" +
str(j) + str(xtc_H_name) + "_" + str(xtc_donor_name))
                            with open(csvfile, "w") as fp:
                                wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
                                wr.writerow(['Avarage Distance', mean])
                                wr.writerow(['Interaction', arrayInteractionLoop[0]])
                                wr.writerow(['Frame','Distance','Angle'])
                                for x in arrayFrameDistanceAngles :
                                    wr.writerow ([x])
                            shutil.move(csvfile, newDirectoryName)
                            count += 1
                            #logging.info(str(datetime.datetime.now()) + ": File Created ->
" + str(csvfile))
                            logging.info(str(datetime.datetime.now())  +  ":  File  '"  +
str(csvfile) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
                        else:
                            #print("MEAN >= 10 --> Will be ignored")
                            logging.info(str(datetime.datetime.now()) + ": Distance Mean ("
+ str(mean) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" +
str(meanAngles) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Values Will be
ignored.")
                            #arrayFramesLoop.clear()
                            #arrayDistancesLoop.clear()
                            #arrayAnglesLoop.clear()
    logging.info(str(datetime.datetime.now()) + ": All interactions checked.")
```

```
    if arrayAllResults:
        with open(csvresults, "w") as fp:
            wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
            wr.writerow(['Interaction;Distance;Angle'])
            for x in arrayAllResults :
                wr.writerow ([x])
        shutil.move(csvresults, newDirectoryName)

    if(count > 1 ):
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Started.")

        #Plot Distance/Frames
        for index,x in enumerate(arrayDistances):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Distance (nm)")
            #plt.title("Metal_Amine - Distance")
            plt.title(str(arrayPlotNameDistance[0]))
            arrayPlotNameDistance.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            distancesPlotName = "Distance_Metal_Amine_" + labelString + "_" + str(dateEpoch)+
".pdf"
            plt.savefig(distancesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Distance_Metal_Amine*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameDistances = "All_Distances_Metal_Amine_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameDistances)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #distancesPlotName = "Distances_SmallUniverse_SinglePlot_Test_" + labelString + "_"
+ str(dateEpoch)+ ".pdf"
        #plt.savefig(distancesPlotName)
        logging.info(str(datetime.datetime.now())    +    ":    Plot    File    '"    +
str(mergeFilenameDistances) +"' Created on Execution Directory (" + str(newDirectoryName)
+").")
        shutil.move(mergeFilenameDistances, newDirectoryName)

        #Clear Current Plot
        plt.clf()

        #Plot Angles/Frames

        for index,x in enumerate(arrayAngles):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Angles Amplitude (º)")
            #plt.title("Metal_Amine - Angles")
            plt.title(str(arrayPlotNameAngle[0]))
            arrayPlotNameAngle.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
```

```
            anglesPlotName ="Angles_Metal_Amine_" + labelString + "_" + str(dateEpoch)+
".pdf"
            plt.savefig(anglesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Angles_Metal_Amine*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameAngles = "All_Angles_Metal_Amine_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameAngles)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #logging.info(str(datetime.datetime.now())  +  ":  Plot  File  Created  ->  "  +
str(anglesPlotName))
        logging.info(str(datetime.datetime.now())    +    ":    Plot    File    '"    +
str(mergeFilenameAngles) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
        shutil.move(mergeFilenameAngles, newDirectoryName)

        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Ended.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
        #os.remove(logFilename)
    else:
        logging.info(str(datetime.datetime.now())  +  ":  No  interactions  matched  the
requisites.")
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process will be
ignored.")
        logging.info(str(datetime.datetime.now()) + ": No plots will be created.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
```

6.7.6. FUNÇÃO DAS INTERAÇÕES COM INTERAÇÕES X-OH

```
##Hydroxyl group (Metals – OH)
def functionGList(input1,input2):
    count = 1
    currentDate=datetime.datetime.now()
    dateEpoch = calendar.timegm(currentDate.timetuple())
    logFilename = "logfile_" + str(dateEpoch) + ".log"
    newDirectoryName = str(dateEpoch)
    os.mkdir(newDirectoryName)
    logging.basicConfig(filename = logFilename , level = logging.INFO)
    logging.info(str(datetime.datetime.now()) + ": Execution Started.")
    logging.info(str(datetime.datetime.now()) + ": Option 3 Selected. Interactions Between
Metal-OH.")
    logging.info(str(datetime.datetime.now())  +  ":  Execution  Directory  Created  '"  +
str(newDirectoryName) +"'.")
    print ("Execution log file can be found in ('" + str(newDirectoryName) + "/" +
str(logFilename) + "').")

    idMetal = []
    for i in input1:
        resid = i.resid
        idMetal.append(resid)

    idHydroxyl = []
    for i in input2:
        resid = i.resid
        idHydroxyl.append(resid)
    idHydroxyl_NoDups = ([item for item, count in collections.Counter(idHydroxyl).items() if
count > 1])

    for i in idMetal:
        for j in idHydroxyl_NoDups:
            getResid = gro.select_atoms("resid " + str(j)).residues
            getResid = str(getResid)
            getResid = getResid.split('[<Residue ')[1]
            getResid = getResid.split(',')[0]

            donor_list = []
            h_list =  []
            #print(getResid)
            #query = "(resid " + str(j) + " and (name HD2 or name OD2)) or (resid " + str(j)
+ " and (name HE2 or name OE2)) or (resid " + str(j) + " and (name HG or name OG)) or (resid
" + str(j) + " and (name HG1 or name OG1)) or (resid " + str(j) + " and (name HH or name
OH))"
            if (getResid == "ASPH"):
                #query1 = "resid " + str(j) + " and (name HD2)"
                #query2 = "resid " + str(j) + " and (name OD2)"
                donor_list = ["OD2"]
                h_list =  ["HD2"]
            if (getResid == "GLUH"):
                #query1 = "resid " + str(j) + " and (name HE2)"
                #query2 = "resid " + str(j) + " and (name OE2)"
                donor_list = ["OE2"]
                h_list =  ["HE2"]
            if (getResid == "SER"):
                #query1 = "resid " + str(j) + " and (name HG)"
                #query2 = "resid " + str(j) + " and (name OG)"
                donor_list = ["OG"]
                h_list =  ["HG"]
            if (getResid == "THR"):
```

```
                #query1 = "resid " + str(j) + " and (name HG1)"
                #query2 = "resid " + str(j) + " and (name OG1)"
                donor_list = ["OG1"]
                h_list =  ["HG1"]
            if (getResid == "TYR"):
                #query1 = "resid " + str(j) + " and (name HH)"
                #query2 = "resid " + str(j) + " and (name OH)"
                donor_list = ["OH"]
                h_list =  ["HH"]

            for d in donor_list:
                for n in h_list:
                    donor_select = gro.select_atoms("name " + str(d) + " and resid " + str(j))
                    h_select = gro.select_atoms("name " + str(n) + " and resid " + str(j))

                    if len(donor_select) > 0 and len(h_select) > 0 :
                        donor = donor_select[0]
                        h = h_select[0]

                        xtc_metal = xtc.select_atoms("(resid " + str(i) + ")")
                        xtc_metal_name = xtc_metal[0]
                        xtc_metal_name = xtc_metal_name.resname

                        xtc_H = xtc.select_atoms("name " + str(h.name) + " and type " +
str(h.type) + " and resid " + str(h.resid))
                        xtc_H_name = xtc_H[0]
                        xtc_H_name = xtc_H_name.name

                        xtc_donor = xtc.select_atoms("name " + str(donor.name) + " and type
" + str(donor.type) + " and resid " + str(donor.resid))
                        xtc_donor_name = xtc_donor[0]
                        xtc_donor_name = xtc_donor_name.name

                        logging.info(str(datetime.datetime.now())   +   ":   -->   CURRENT
INTERACTION")
                        logging.info(str(datetime.datetime.now()) + ": Metal --> " + str(i)
+ " " + str(xtc_metal_name) )
                        logging.info(str(datetime.datetime.now()) + ": " + str(xtc_metal))
                        logging.info(str(datetime.datetime.now())  +  ":  Hydroxyl  -->  "  +
str(j) + " " + str(xtc_H_name) + "_" + str(xtc_donor_name) )
                        logging.info(str(datetime.datetime.now()) + ": " + str(xtc_H) + " +
"  + str(xtc_donor))

                        arrayFramesLoop = []
                        arrayDistancesLoop = []
                        arrayAnglesLoop = []
                        arrayInteractionLoop = []
                        arrayInteractionResidLoop = []
                        arrayFrameDistanceAngles = []

                        for ts in xtc.trajectory:
                            positionMetal = xtc_metal.positions
                            positionDonor = xtc_donor.positions
                            positionH = xtc_H.positions
                            centroidMetal = getCenteroid(positionMetal)
                            avgsDonor   =   [sum(vals)/len(positionDonor)   for   vals   in
zip(*positionDonor)]
                            avgsH = [sum(vals)/len(positionH) for vals in zip(*positionH)]
                            vector_H_Donor = findVec(avgsH,avgsDonor)
                            vector_H_Centroid = findVec(avgsH,centroidMetal)
                            angle=angle_between_degree(vector_H_Donor,vector_H_Centroid)
```

```
                            dist    =    math.sqrt((centroidMetal[0]-avgsDonor[0])**2    +
(centroidMetal[1]-avgsDonor[1])**2 + (centroidMetal[2]-avgsDonor[2])**2 )
                            arrayFramesLoop.append(ts.frame)
                            arrayDistancesLoop.append(dist)
                            arrayAnglesLoop.append(angle)
                            arrayFrameDistanceAngles.append(str(xtc.trajectory.time)+ ',' +
str(dist)+ ',' + str(angle))
                        arrayInteractionResidLoop.append(str(i) + "VS" + str(j))
                        arrayInteractionLoop.append(str(xtc_metal_name)    +    "VS"    +
str(xtc_H_name) + "_" + str(xtc_donor_name) )
                        mean = numpy.mean(arrayDistancesLoop)
                        meanAngles = numpy.mean(arrayAnglesLoop)
                        #if (mean < maxMeanDistance) and (meanAngles > minMeanAngles) and
(mean != 0.0) : # ----------------> Valor falado na reunião
                        #if (mean < 15) and (meanAngles > 120) and (mean != 0.0) : # ------
----------> Forçar Valores
                        if (mean < maxMeanDistanceTest) and (meanAngles > minMeanAnglesTest)
and (mean != 0.0) :
                            logging.info(str(datetime.datetime.now()) + ": Distance Mean ("
+ str(mean) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" +
str(meanAngles) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Values will be
added to the plot.")
                            arrayFrames.append(arrayFramesLoop)
                            arrayDistances.append(arrayDistancesLoop)
                            arrayAngles.append(arrayAnglesLoop)
                            arrayInteraction.append(arrayInteractionLoop)
                            arrayInteractionResid.append(arrayInteractionResidLoop)
                            csvfile = str(count)+ "_" + str(arrayInteractionResidLoop[0]) +
"_" + str(dateEpoch) +".csv"
                            interaction = str(i) + str(xtc_metal_name) + "_" + str(j) +
str(xtc_H_name) + "_" + str(xtc_donor_name)
                            arrayAllResults.append(str(interaction) + ";" + str(mean) + ";"
+ str(meanAngles))
                            csvresults = "results.csv"
                            arrayPlotNameDistance.append(str(i) + str(xtc_metal_name) + "_"
+ str(j) + str(xtc_H_name) + "_" + str(xtc_donor_name))
                            arrayPlotNameAngle.append(str(i) + str(xtc_metal_name) + "_" +
str(j) + str(xtc_H_name) + "_" + str(xtc_donor_name))
                            with open(csvfile, "w") as fp:
                                wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
                                wr.writerow(['Avarage Distance', mean])
                                wr.writerow(['Interaction', arrayInteractionLoop[0]])
                                wr.writerow(['Frame','Distance','Angle'])
                                for x in arrayFrameDistanceAngles :
                                    wr.writerow ([x])
                            shutil.move(csvfile, newDirectoryName)
                            count += 1
                            #logging.info(str(datetime.datetime.now()) + ": File Created ->
" + str(csvfile))
                            logging.info(str(datetime.datetime.now())  +  ":  File  '"  +
str(csvfile) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
                        else:
                            #print("MEAN >= 10 --> Will be ignored")
                            logging.info(str(datetime.datetime.now()) + ": Distance Mean ("
+ str(mean) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Angles Mean (" +
str(meanAngles) + ")")
                            logging.info(str(datetime.datetime.now()) + ": Values Will be
ignored.")
```

```
                            #arrayFramesLoop.clear()
                            #arrayDistancesLoop.clear()
                            #arrayAnglesLoop.clear()

    logging.info(str(datetime.datetime.now()) + ": All interactions checked.")

    if arrayAllResults:
        with open(csvresults, "w") as fp:
            wr = csv.writer(fp,  delimiter='\t',lineterminator='\n')
            wr.writerow(['Interaction;Distance;Angle'])
            for x in arrayAllResults :
                wr.writerow ([x])
        shutil.move(csvresults, newDirectoryName)

    if(count > 1 ):
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Started.")

        #Plot Distance/Frames
        for index,x in enumerate(arrayDistances):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Distance (nm)")
            #plt.title("Metal_Hydroxyl - Distance")
            plt.title(str(arrayPlotNameDistance[0]))
            arrayPlotNameDistance.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            distancesPlotName   =   "Distance_Metal_Hydroxyl_"   +   labelString   +   "_"   +
str(dateEpoch)+ ".pdf"
            plt.savefig(distancesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Distance_Metal_Hydroxyl*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameDistances = "All_Distances_Metal_Hydroxyl_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameDistances)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #distancesPlotName = "Distances_SmallUniverse_SinglePlot_Test_" + labelString + "_"
+ str(dateEpoch)+ ".pdf"
        #plt.savefig(distancesPlotName)
        logging.info(str(datetime.datetime.now())     +     ":     Plot     File     '"     +
str(mergeFilenameDistances) +"' Created on Execution Directory (" + str(newDirectoryName)
+").")
        shutil.move(mergeFilenameDistances, newDirectoryName)

        #Clear Current Plot
        plt.clf()

        #Plot Angles/Frames

        for index,x in enumerate(arrayAngles):
            labelString = str(index+1)
            plt.xlabel("time (ps)")
            plt.ylabel("Angles Amplitude (º)")
```

```
            #plt.title("Metal_Hydroxyl - Angles")
            plt.title(str(arrayPlotNameAngle[0]))
            arrayPlotNameAngle.pop(0)
            plt.plot(arrayFrames[0],x, label=labelString)
            #plt.legend()
            anglesPlotName ="Angles_Metal_Hydroxyl_" + labelString + "_" + str(dateEpoch)+
".pdf"
            plt.savefig(anglesPlotName)
            plt.clf()

        pwd=os.getcwd()
        pattern = "/Angles_Metal_Hydroxyl*"
        pdfs = glob.glob(pwd + pattern)
        merger = PdfMerger()
        for pdf in pdfs:
            merger.append(pdf)
        mergeFilenameAngles = "All_Angles_Metal_Hydroxyl_" + str(dateEpoch)+ ".pdf"
        merger.write(mergeFilenameAngles)
        merger.close()
        for pdf in pdfs:
            os.remove(pdf)

        #logging.info(str(datetime.datetime.now())  +  ":  Plot  File  Created  ->  "  +
str(anglesPlotName))
        logging.info(str(datetime.datetime.now())    +    ":    Plot    File    '"    +
str(mergeFilenameAngles) +"' Created on Execution Directory (" + str(newDirectoryName) +").")
        shutil.move(mergeFilenameAngles, newDirectoryName)

        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process Ended.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
        #os.remove(logFilename)
    else:
        logging.info(str(datetime.datetime.now())  +  ":  No  interactions  matched  the
requisites.")
        logging.info(str(datetime.datetime.now()) + ": Graphics Creation Process will be
ignored.")
        logging.info(str(datetime.datetime.now()) + ": No plots will be created.")
        logging.info(str(datetime.datetime.now()) + ": Execution Ended.")
        shutil.copy(logFilename, newDirectoryName)
        shutil.make_archive(newDirectoryName, 'zip', newDirectoryName)
```

6.8. ANEXO 8 - RESULTADOS GLOBAIS OBTIDOS COM TODAS AS CONFORMAÇÕES DA PROTEÍNA

6.8.1. PROTEÍNA NA CONFORMAÇÃO FECHADA

Tabela A 1 – Valor médio das distâncias para as interacções de metais com aniões na conformação fechada – Fe(III) + HCO_3^-

Interação	Média Distâncias (Å)
339FE3_63H_OD1	1,9
339FE3_63H_OD2	3,9
339FE3_95H_OH	1,9
339FE3_95HD1_OH	1,9
339FE3_95HD2_OH	1,9
339FE3_95HE1_OH	1,9
339FE3_95HE2_OH	1,9
339FE3_188H_OH	1,9
339FE3_188HD1_OH	1,9
339FE3_188HD2_OH	1,9
339FE3_188HE1_OH	1,9
339FE3_188HE2_OH	1,9
339FE3_249H_NE2	2,0
339FE3_249H_ND1	4,1

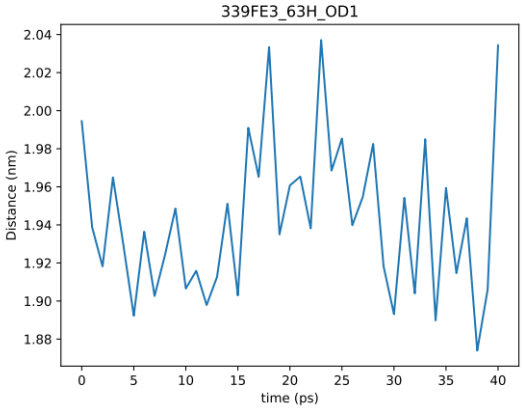

Figura A 1- Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD1(carboxilato) na conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

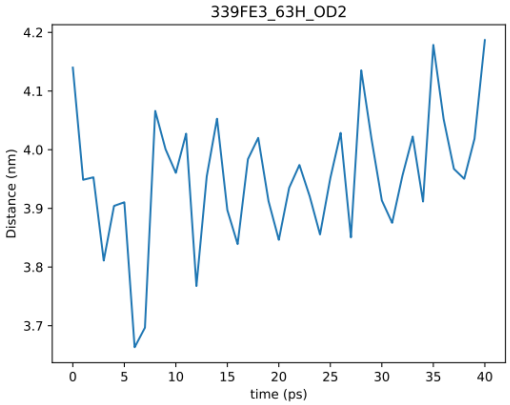

Figura A 2- Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD2(carboxilato) na conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

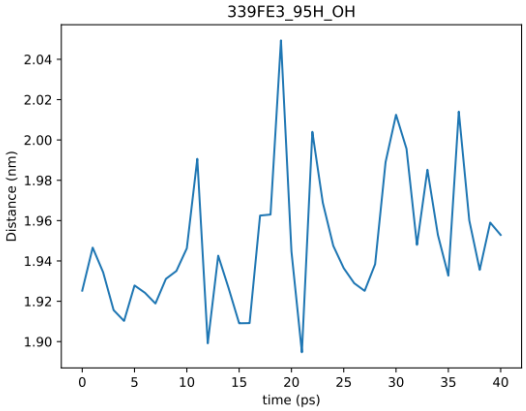

Figura A 3- Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr95O(fenol) na conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

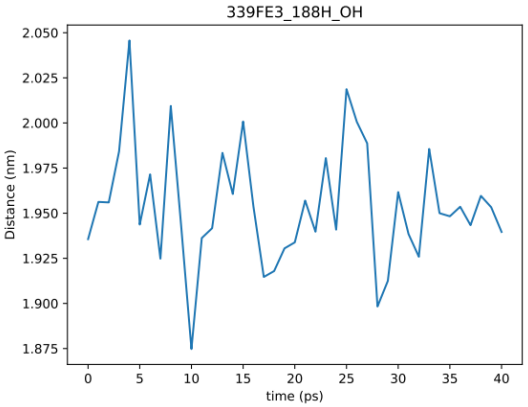

Figura A 4- Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr188(fenol) na conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

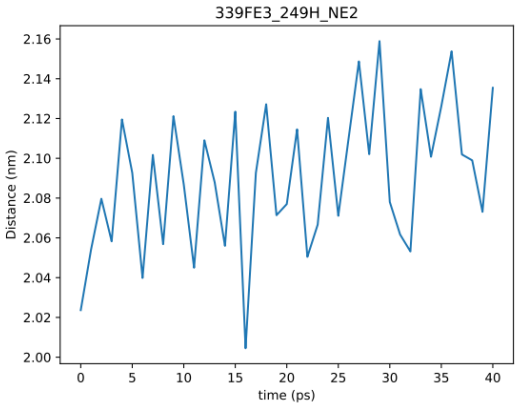

Figura A 5- Variação ao longo do tempo das distâncias para a interação Fe(III) – His249NE na conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

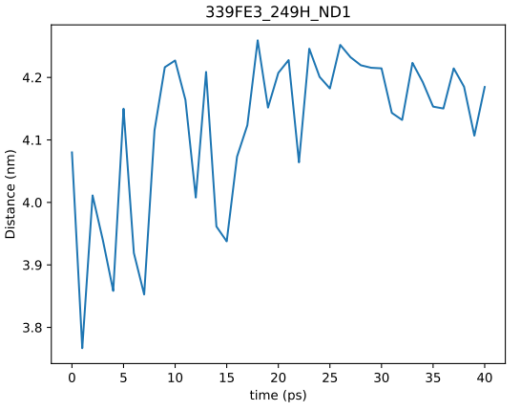

Figura A 6 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249ND na conformação fechada da proteína – sistema hTf + Fe(III) + HCO_3^-.

Tabela A 2 – Valor médio das distâncias para as interacções de metais com aniões na conformação fechada – Fe(III) + H_2CO_3

Interação	Média Distâncias (Å)
339FE3_63H_OD1	1,9
339FE3_63H_OD2	1,9
339FE3_95H_OH	1,9
339FE3_95HD1_OH	1,9
339FE3_95HD2_OH	1,9
339FE3_95HE1_OH	1,9
339FE3_95HE2_OH	1,9
339FE3_188H_OH	1,9
339FE3_188HD1_OH	1,9
339FE3_188HD2_OH	1,9
339FE3_188HE1_OH	1,9
339FE3_188HE2_OH	1,9
339FE3_249H_NE2	2,0
339FE3_249H_ND1	3,6

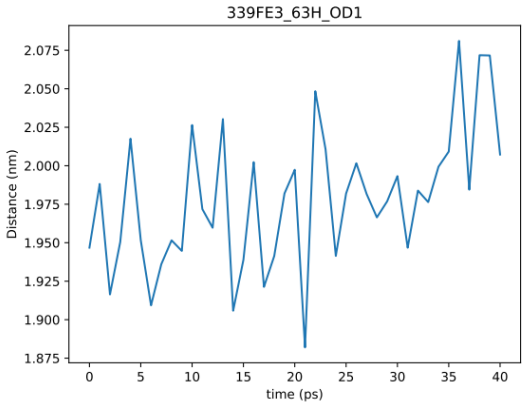

Figura A 7 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD1 na conformação fechada da proteína – sistema hTf + Fe(III) + H_2CO_3.

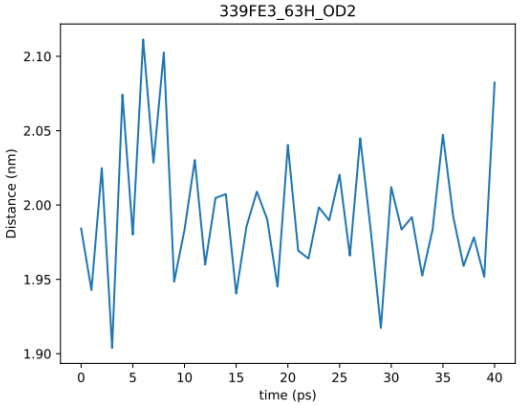

Figura A 8 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD2 na conformação fechada da proteína – sistema hTf + Fe(III) + H_2CO_3.

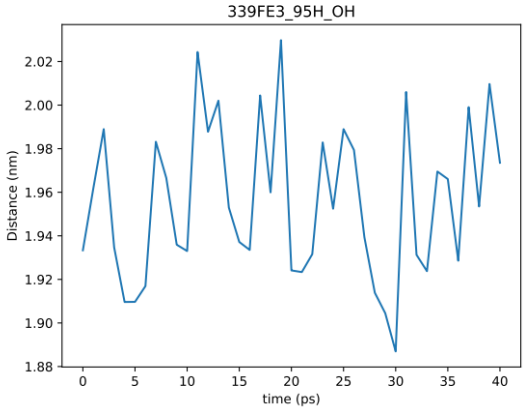

Figura A 9 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr95O(fenol) na conformação fechada da proteína – sistema hTf + Fe(III) + H_2CO_3.

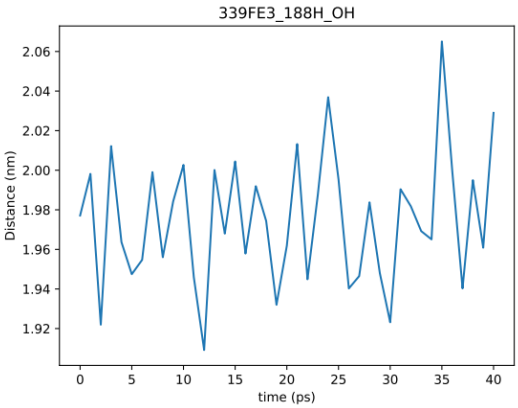

Figura A 10 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr188O(fenol) na conformação fechada da proteína – sistema hTf + Fe(III) + H_2CO_3.

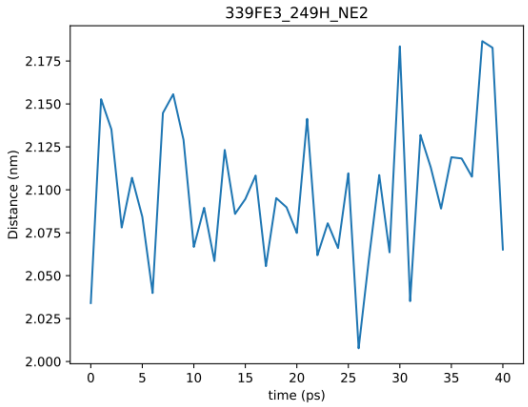

Figura A 11- Variação ao longo do tempo das distâncias para a interação Fe(III) – His249NE na conformação fechada da proteína – sistema hTf + Fe(III) + H_2CO_3.

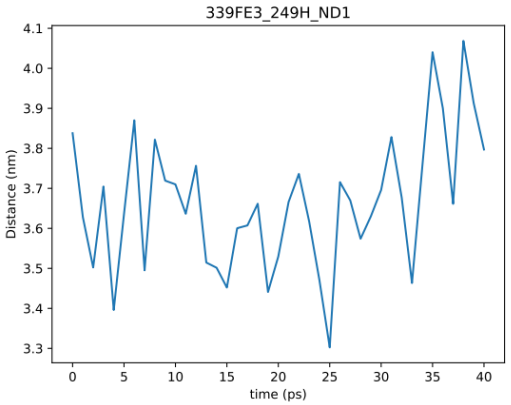

Figura A 12- Variação ao longo do tempo das distâncias para a interação Fe(III) – His249ND na conformação fechada da proteína – sistema hTf + Fe(III) + H_2CO_3.

Tabela A 3 – Valor médio das distâncias para as interacções de metais com aniões na conformação fechada – V(III) + CO_3^{2-}

Interação	Média Distâncias (Å)
339V3_63H_OD1	1,9
339V3_63H_OD2	4,0
339V3_95H_OH	1,9
339V3_95HD1_OH	1,9
339V3_95HD2_OH	1,9
339V3_95HE1_OH	1,9
339V3_95HE2_OH	1,9
339V3_188H_OH	1,9
339V3_188HD1_OH	1,9
339V3_188HD2_OH	1,9
339V3_188HE1_OH	1,9
339V3_188HE2_OH	1,9
339V3_249H_NE2	2,0
339V3_249H_ND1	4,1

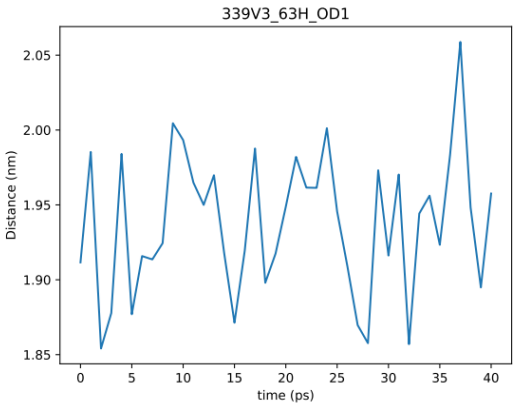

Figura A 13 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD1 na conformação fechada da proteína – sistema hTf + V(III) + CO_3^2.

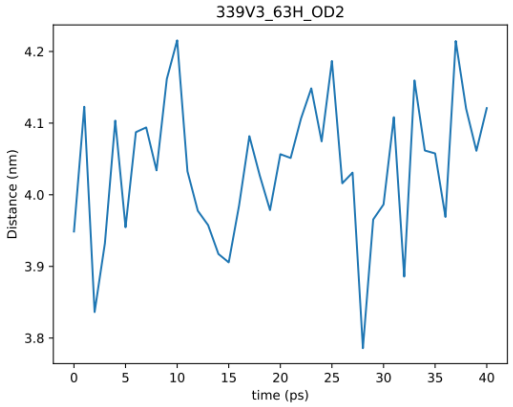

Figura A 14 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD2 na conformação fechada da proteína – sistema hTf + V(III) + CO_3^2.

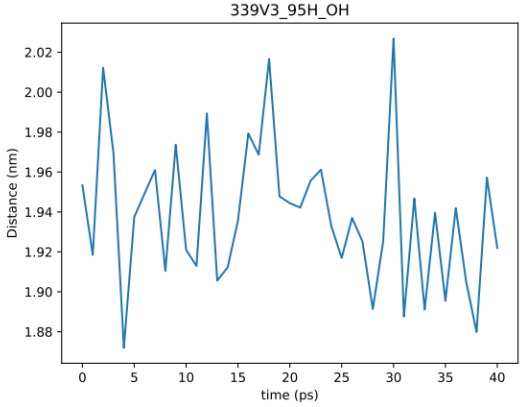

Figura A 15 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr95O(fenol) na conformação fechada da proteína – sistema hTf + V(III) + CO_3^2.

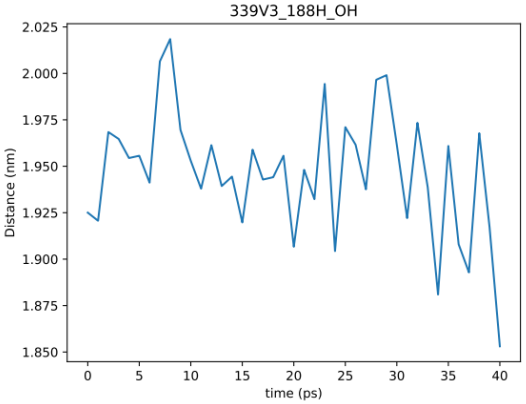

Figura A 16 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr188O(fenol) na conformação fechada da proteína – sistema hTf + V(III) + CO_3^2.

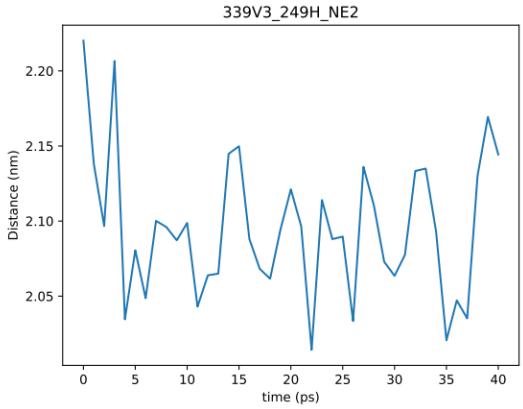

Figura A 17 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249NE na conformação fechada da proteína – sistema hTf + V(III) + CO_3^2.

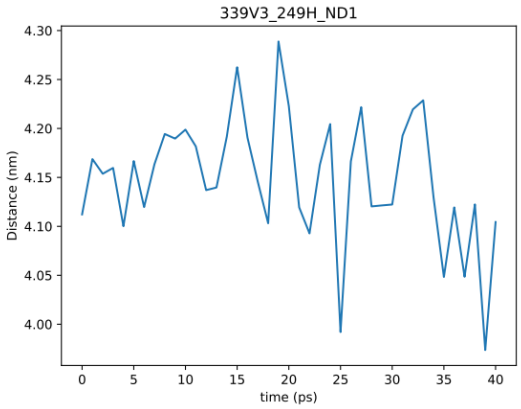

Figura A 18 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249ND na conformação fechada da proteína – sistema hTf + V(III) + CO_3^2.

Tabela A 4 – Valor médio das distâncias para as interacções de metais com aniões na conformação fechada – V(III) + HCO_3^-

Interação	Média Distâncias (Å)
339V3_63H_OD1	1,9
339V3_63H_OD2	3,8
339V3_95H_OH	1,9
339V3_95HD1_OH	1,9
339V3_95HD2_OH	1,9
339V3_95HE1_OH	1,9
339V3_95HE2_OH	1,9
339V3_188H_OH	1,9
339V3_188HD1_OH	1,9
339V3_188HD2_OH	1,9
339V3_188HE1_OH	1,9
339V3_188HE2_OH	1,9
339V3_249H_NE2	2,0
339V3_249H_ND1	4,1

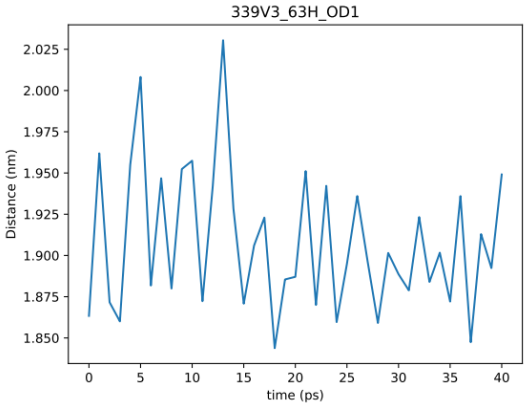

Figura A 19 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD1 na conformação fechada da proteína – sistema hTf + V(III) + HCO_3^-.

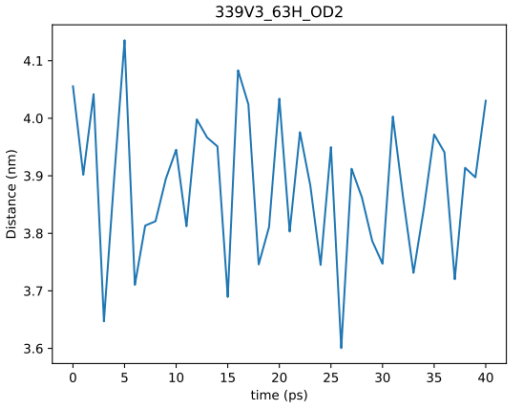

Figura A 20 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD2 na conformação fechada da proteína – sistema hTf + V(III) + HCO_3^-.

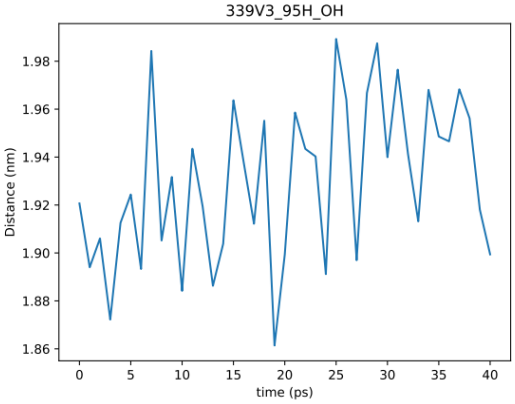

Figura A 21 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr95O(fenol) na conformação fechada da proteína – sistema hTf + V(III) + HCO_3^-.

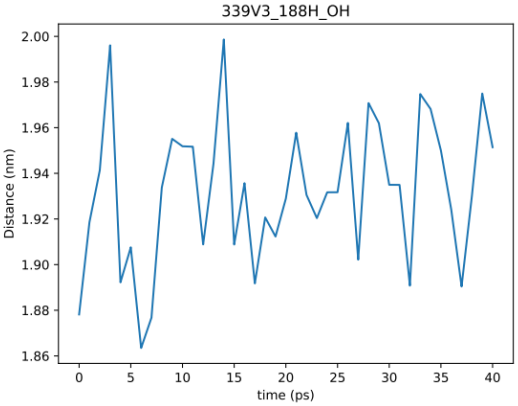

Figura A 22 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr188(fenol) na conformação fechada da proteína – sistema hTf + V(III) + HCO_3^-.

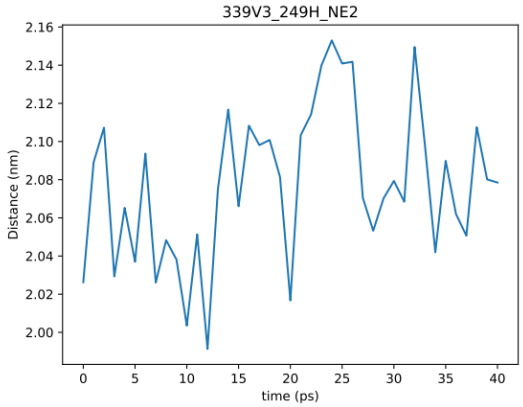

Figura A 23 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249NE na conformação fechada da proteína – sistema hTf + V(III) + HCO_3^-.

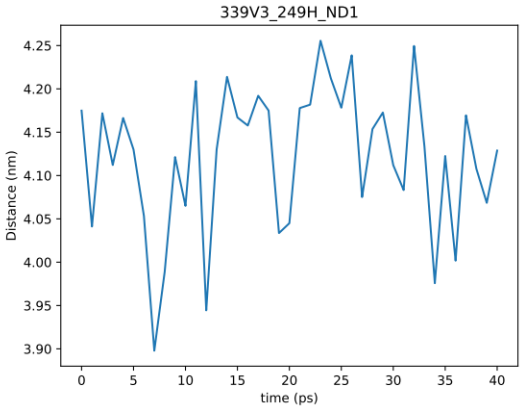

Figura A 24 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249ND na conformação fechada da proteína – sistema hTf + V(III) + HCO_3^-.

Tabela A 5 – Valor médio das distâncias para as interacções de metais com aniões na conformação fechada – V(III) + H_2CO_3

Interação	Média Distâncias (Å)	Média Ângulos (º)
339V3_63H_OD1	1,9	16,3
339V3_63H_OD2	1,9	18,0
339V3_95H_OH	1,9	3,2
339V3_95HD1_OH	1,9	8,1
339V3_95HD2_OH	1,9	10,2
339V3_95HE1_OH	1,9	21,2
339V3_95HE2_OH	1,9	24,9
339V3_188H_OH	1,9	5,9
339V3_188HD1_OH	1,9	9,4
339V3_188HD2_OH	1,9	12,0
339V3_188HE1_OH	1,9	21,5
339V3_188HE2_OH	1,9	26,1
339V3_249H_NE2	2,0	19,1
339V3_249H_ND1	3,7	42,4

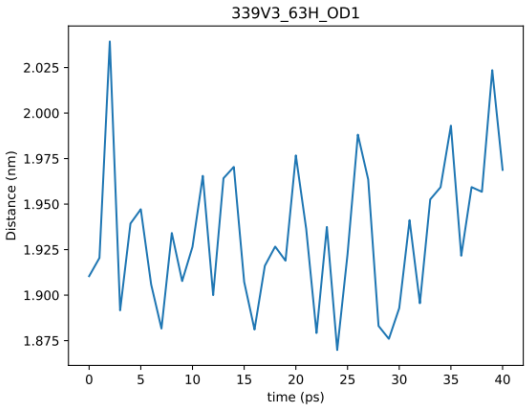

Figura A 25 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD1 na conformação fechada da proteína – sistema hTf + V(III) + H_2CO_3.

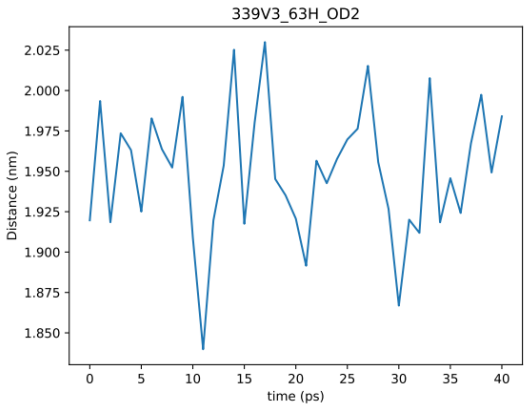

Figura A 26 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD2 na conformação fechada da proteína – sistema hTf + V(III) + H_2CO_3.

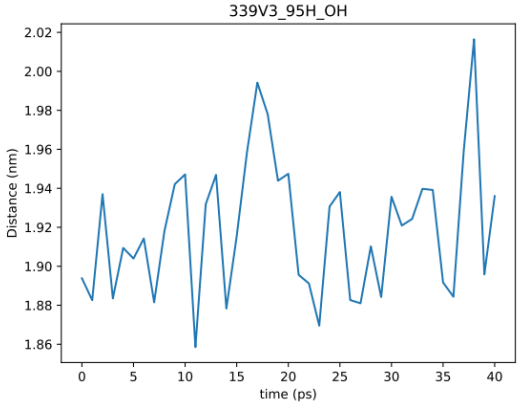

Figura A 27 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr95O(fenol) na conformação fechada da proteína – sistema hTf + V(III) + H_2CO_3.

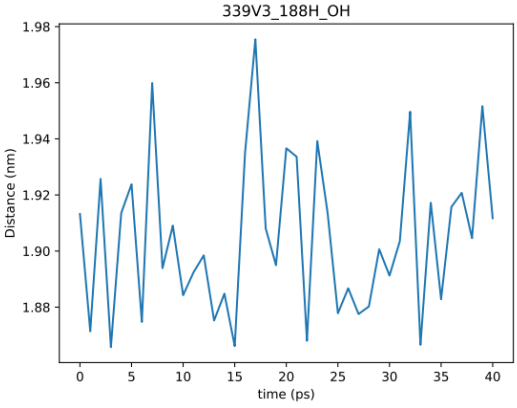

Figura A 28 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr188O(fenol) na conformação fechada da proteína – sistema hTf + V(III) + H_2CO_3.

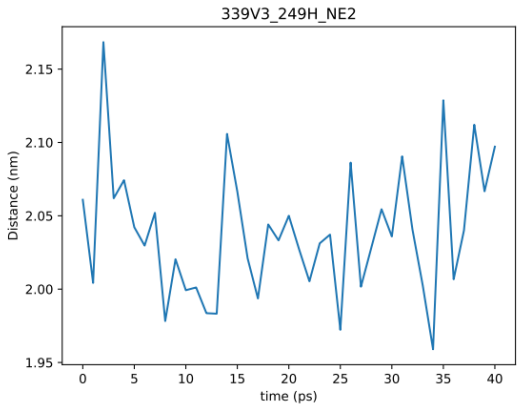

Figura A 29 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249NE na conformação fechada da proteína – sistema hTf + V(III) + H_2CO_3.

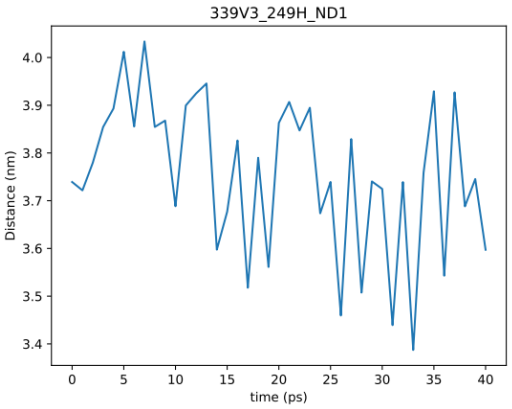

Figura A 30 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249ND na conformação fechada da proteína – sistema hTf + V(III) + H_2CO_3.

6.8.2. PROTEÍNA NA CONFORMAÇÃO RELAXADA

Tabela A 6 – Valor médio das distâncias para as interacções de metais com aniões na conformação relaxada – Fe(III) + CO_3^{2-}

Interação	Média Distâncias (Å)
329FE3_63H_OD1	1,9
329FE3_63H_OD2	4,0
329FE3_95H_OH	1,9
329FE3_95HD1_OH	1,9
329FE3_95HD2_OH	1,9
329FE3_95HE1_OH	1,9
329FE3_95HE2_OH	1,9
329FE3_188H_OH	1,9
329FE3_188HD1_OH	1,9
329FE3_188HD2_OH	1,9
329FE3_188HE1_OH	1,9
329FE3_188HE2_OH	1,9
329FE3_249H_NE2	2,1
329FE3_249H_ND1	4,1

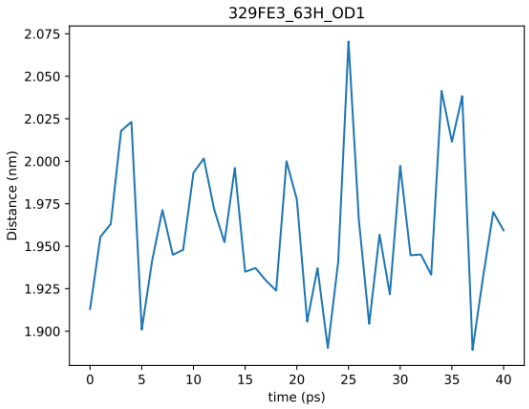

Figura A 31 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD1 na conformação relaxada da proteína – sistema hTf + Fe(III) + CO_3^{2-}.

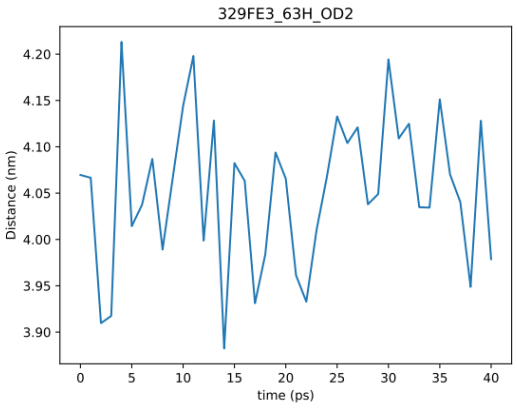

Figura A 32 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Asp63OD2 na conformação relaxada da proteína – sistema hTf + Fe(III) + CO_3^{2-}.

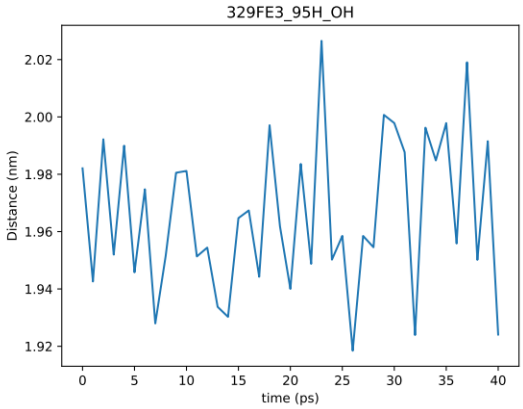

Figura A 33 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr95O(fenol) na conformação relaxada da proteína – sistema hTf + Fe(III) + CO_3^{2-}.

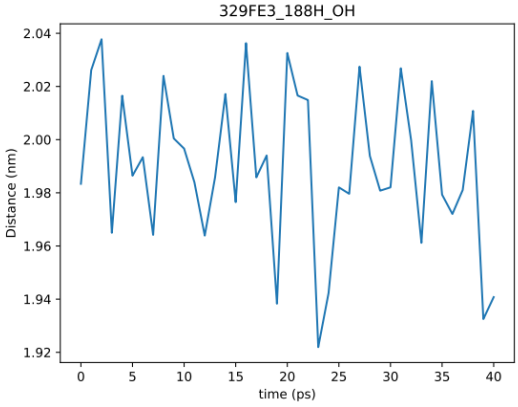

Figura A 34 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr188(fenol) na conformação relaxada da proteína – sistema hTf + Fe(III) + CO_3^{2-}.

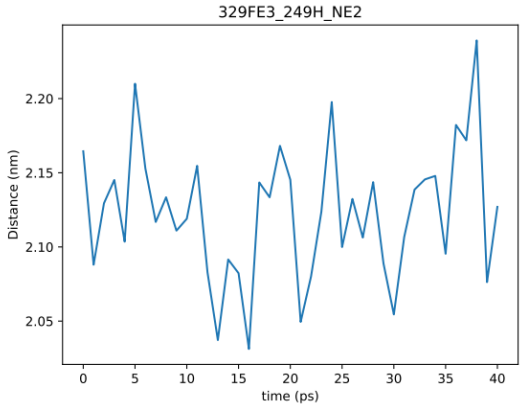

Figura A 35 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249NE na conformação relaxada da proteína – sistema hTf + Fe(III) + CO_3^{2-}.

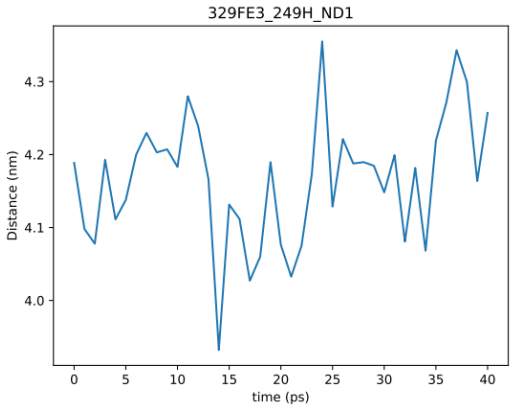

Figura A 36 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249ND na conformação relaxada da proteína – sistema hTf + Fe(III) + CO_3^{2-}.

Tabela A 7 – Valor médio das distâncias para as interacções de metais com aniões na conformação relaxada – Fe(III) + HCO_3^-

Interação	Média Distâncias (Å)	Média Ângulos (º)
329FE3_63H_OD1	1,9	16,7
329FE3_63H_OD2	3,9	42,9
329FE3_95H_OH	1,9	4,6
329FE3_95HD1_OH	1,9	11,4
329FE3_95HD2_OH	1,9	16,0
329FE3_95HE1_OH	1,9	19,5
329FE3_95HE2_OH	1,9	28,8
329FE3_188H_OH	1,9	12,7
329FE3_188HD1_OH	1,9	14,8
329FE3_188HD2_OH	1,9	13,0
329FE3_188HE1_OH	1,9	26,6
329FE3_188HE2_OH	1,9	25,1
329FE3_249H_NE2	2,0	16,9
329FE3_249H_ND1	4,1	44,1

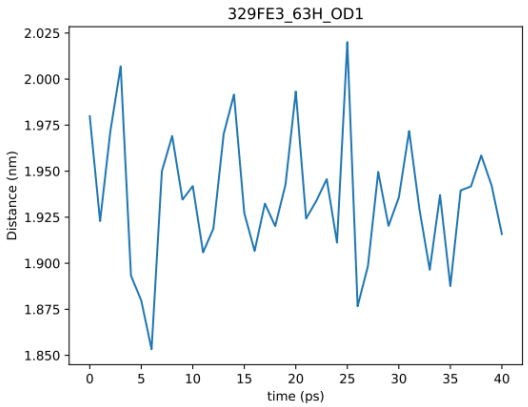

Figura A 37 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr63OD1 na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

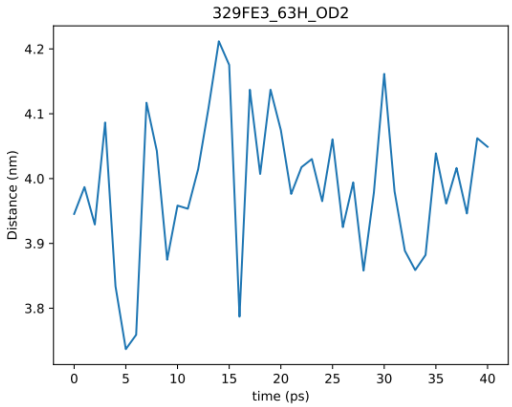

Figura A 38 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr63OD2 na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

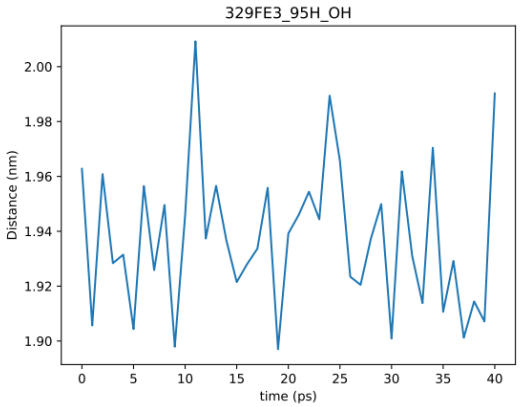

Figura A 39 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr95O(fenol) na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

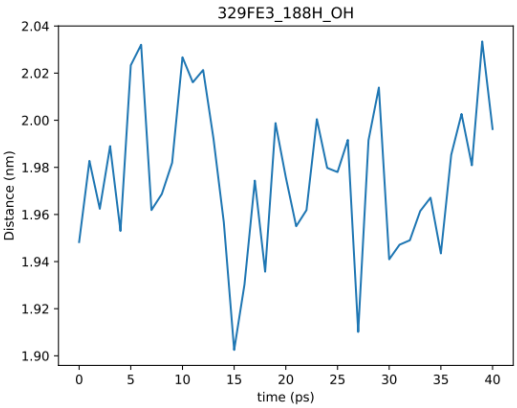

Figura A 40 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr188O(fenol) na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

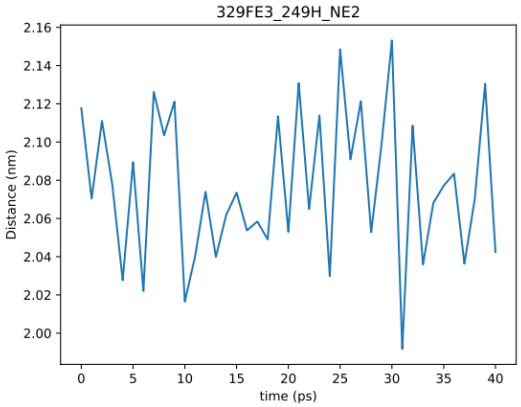

Figura A 41 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249NE2 na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

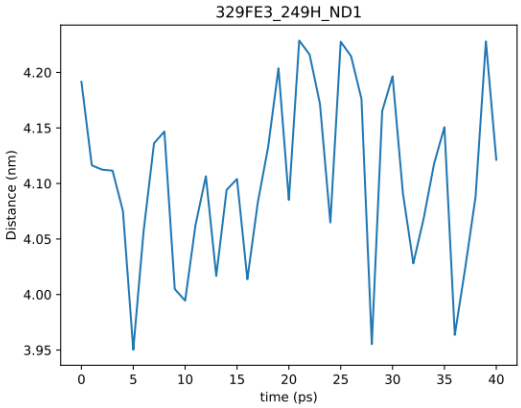

Figura A 42 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249ND1 na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

Tabela A 8 – Valor médio das distâncias das interacções de metais com aniões na conformação relaxada – Fe(III) + H_2CO_3

Interação	Média Distâncias (Å)	Média Ângulos (º)
329FE3_63H_OD1	2,0	5,3
329FE3_63H_OD2	2,0	21,8
329FE3_95H_OH	1,9	4,4
329FE3_95HD1_OH	1,9	11,8
329FE3_95HD2_OH	1,9	14,0
329FE3_95HE1_OH	1,9	21,7
329FE3_95HE2_OH	1,9	27,6
329FE3_188H_OH	1,9	11,9
329FE3_188HD1_OH	1,9	11,2
329FE3_188HD2_OH	1,9	12,0
329FE3_188HE1_OH	1,9	21,4
329FE3_188HE2_OH	1,9	24,2
329FE3_249H_NE2	2,1	15,7
329FE3_249H_ND1	4,0	42,3

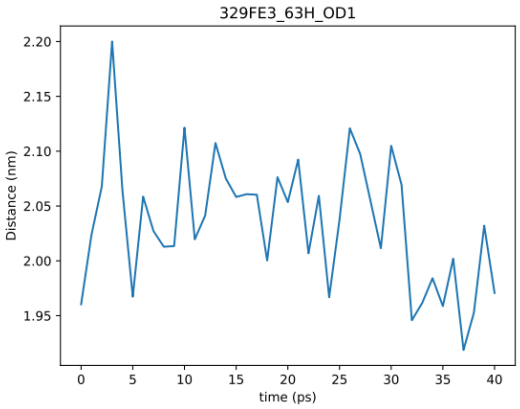

Figura A 43 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr63OD1 na conformação relaxada da proteína – sistema hTf + Fe(III) + H_2CO_3.

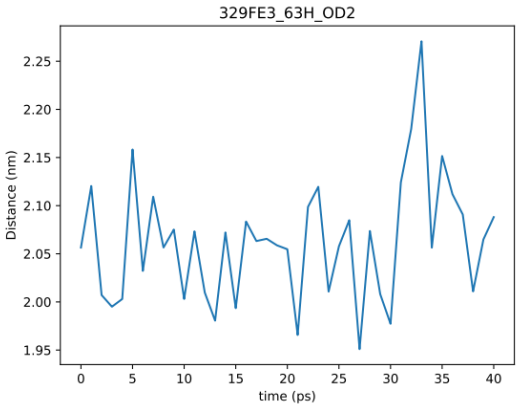

Figura A 44 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr63OD2 na conformação relaxada da proteína – sistema hTf + Fe(III) + H_2CO_3.

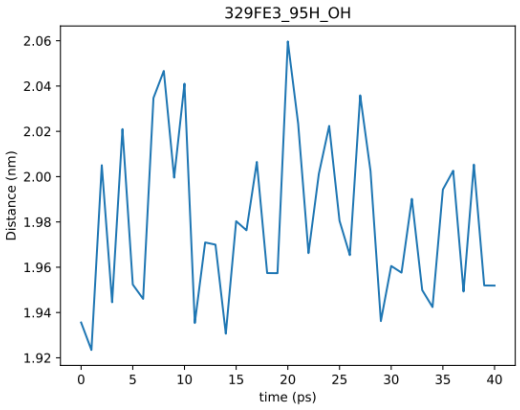

Figura A 45 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr95O(fenol) na conformação relaxada da proteína – sistema hTf + Fe(III) + H_2CO_3.

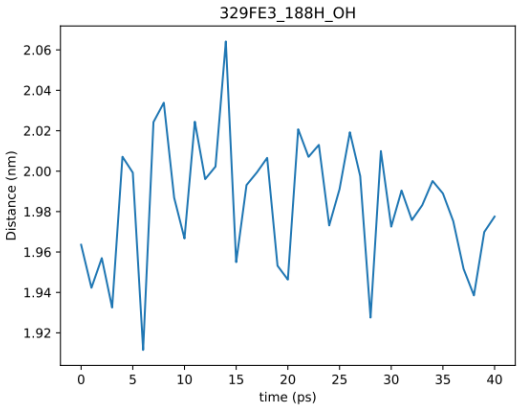

Figura A 46 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Tyr188O(fenol) na conformação relaxada da proteína – sistema hTf + Fe(III) + H_2CO_3.

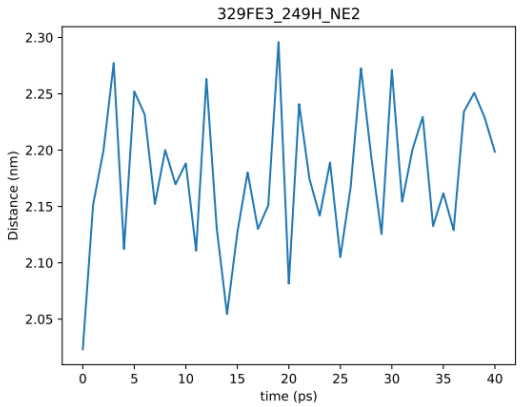

Figura A 47 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249NE na conformação relaxada da proteína – sistema hTf + Fe(III) + H_2CO_3.

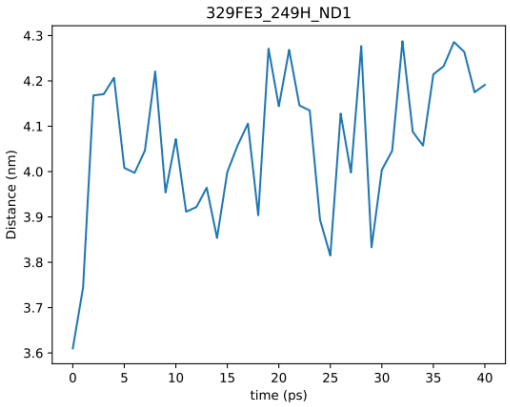

Figura A 48 – Variação ao longo do tempo das distâncias para a interação Fe(III) – His249ND na conformação relaxada da proteína – sistema hTf + Fe(III) + H_2CO_3.

6.8.3. PROTEÍNA NA CONFORMAÇÃO ABERTA

Tabela A 9 – Valor médio das distâncias das interacções de metais com tióis na conformação aberta – Fe(III) + HCO_3^-

Interação	Média Distâncias (Å)	Média Ângulos (º)
339FE3_227H_SG	3,6	48,0
339FE3_241H_SG	3,8	43,3

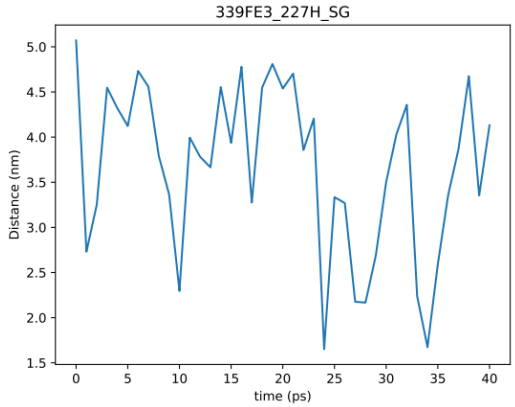

Figura A 49 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Cys227S na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

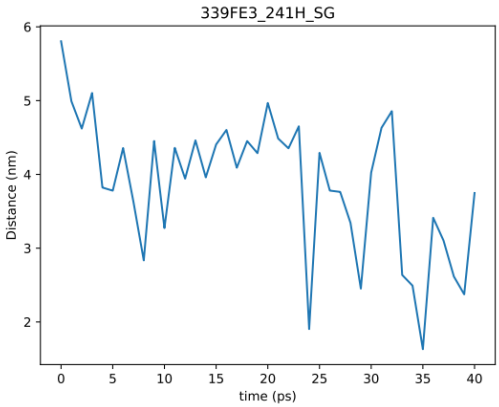

Figura A 50 – Variação ao longo do tempo das distâncias para a interação Fe(III) – Cys241S na conformação relaxada da proteína – sistema hTf + Fe(III) + HCO_3^-.

Tabela A 10 – Valor médio das distâncias das interacções de metais com aniões na conformação aberta – V(III) + HCO_3^-

Interação	Média Distâncias (Å)	Média Ângulos (º)
339V3_317H_OH	4,3	30,8

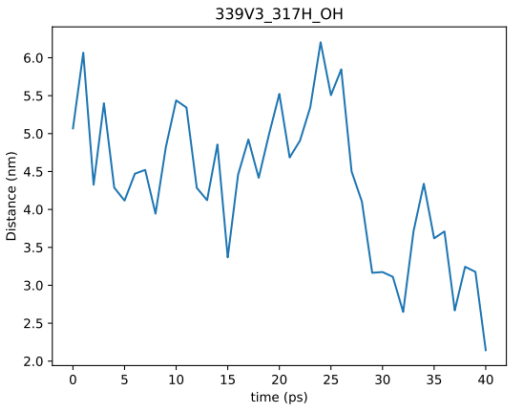

Figura A 51 – Variação ao longo do tempo das distâncias para a interação V(III) – Cys227S na conformação relaxada da proteína – sistema hTf + V(III) + HCO_3^-.