April 2016

Simulating HIV-1 Protease Mutations for Conferred Drug Resistance

Edward Thomas Caputo
Worcester Polytechnic Institute

Paige Regan Waechter
Worcester Polytechnic Institute

Sydney Elizabeth Tucker
Worcester Polytechnic Institute

SIMULATING HIV-1 PROTEASE MUTATIONS FOR CONFERRED DRUG RESISTANCE

A MAJOR QUALIFYING PROJECT

Submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree in Bachelor of Science
in
Biomedical Engineering

By

____
Edward Caputo

____
Sydney Tucker

____
Paige Waechter

Date: April 28, 2016
Sponsoring Organization: University of Massachusetts Medical School

Project Advisors:

____
Professor Celia Schiffer, Advisor

____
Professor Anjana Jain, Co-Advisor

Abstract

A major challenge in the long-term management of HIV is drug resistance caused from high rate and error prone viral replication. To examine mechanisms of drug resistance within HIV-1 protease complexed with Darunavir, specific point mutations were placed in the protease amino acid sequence and molecular dynamic simulations were run. Darunavir was chosen as the modeled ligand as it is the most potent protease inhibitor commercially available. MATLAB and python scripts were developed to efficiently and consistently analyze simulation data. The team hypothesized that there would be a difference in inhibitor interactions and protein dynamic behavior in mutant variants compared to wild type. Although some aspects of increased resistance were seen with compounded mutations, overall this trend was not observed across every facet of our analysis.

Acknowledgment

The team would like to thank Celia Schiffer PhD, Nese Kurt Yilmaz, PhD, the rest of the Schiffer Laboratory at the University of Massachusetts Medical School, and Professor Anjana Jain for their help in completing this project.

Authorship

Edward Caputo, Sydney Tucker and Paige Waechter contributed equally in the completion of this project and report.

Table of Contents

Table of Figures

Table of Tables

Executive Summary

Human immunodeficiency virus (HIV) is a global epidemic, negatively affecting the quality of human life since it’s peak of public awareness during the 1980’s [3]. HIV is a retrovirus that has evolved to target and destroy the cells that make up the human immune system, specifically CD4 T-cells. After HIV has destroyed the majority of the body's immune system, the disease is reclassified as acquired immunodeficiency syndrome (AIDS), where the risk of co-infection with other viruses is high and often fatal.

Currently, HIV-1 remains a non-curable disease that requires a strict regimen of antiviral drugs to prevent disease progression. With nearly 37 million infected and 2 million new cases annually world wide, the need for more potent treatment or a cure is escalating [4]. Further there is an estimated 1.2 million Americans living with HIV-1, the viral serotype found in the Western world [5]. The FDA has approved 25 different antiviral drugs, with 7 classes that each target different stages of HIV-1’s life cycle [6]. Protease inhibitors are a class of potent antivirals normally used in late stage, severe cases of HIV-1 infection. These are competitive inhibitors that prevent HIV-1 protease from cleaving viral polyproteins, inhibiting virion maturation. However, HIV-1’s high rate of replication, lack of error-proof mechanisms, and selective pressures in response to drug presence promotes drug resistance [7, 8]. Drug resistance is a result of amino acid mutations that alter the shape of the proteins binding pocket such that a drug cannot bind properly and maintain functionality. Normally, mutations causing drug resistance are located within the active site, as well as non-active site regions of the protease. Active site mutations directly impact inhibitor binding and non-active site mutations affect the tertiary structure [7]. The team hypothesized that active mutations involved in drug resistance follow an additive trend.

Through discussion with the client, the team chose three active site mutations for the team to analyze. These mutations are resistant variants of HIV-1 protease found in HIV-1 infected mouse models. The first resistant variant was a single amino acid mutation of residue 84 from an isoleucine (I) to a valine (V), notated I84V. The second resistant variant was a double amino acid mutation of I84V and the amino acid mutation of the residue 82 from a valine (V) to a phenylalanine (F), notated V82F. The third resistant variant was the triple amino acid mutation involving mutations I84V, V82F, and an amino acid mutation of the residue 46 from methionine (M) to isoleucine (I), notated M46I.

Computational analysis, specifically molecular dynamics (MD), was selected to simulate the protein's dynamic behavior. The data from MD simulations can be used to analyze the effects of these mutations on the ability of the inhibitor to bind to the active site. First, the team modified crystal structures of the protein using molecular modeling software (Schrodinger's Maestro) to generate point mutations. A water system (TIP3) and force field system (OPLS_2005) were added, and the protein system was minimized to the lowest energy state ("native" folded conformation). The modified model is then imported into molecular dynamic software (Desmond) to simulate the protein system for 100 ns, which gives enough time for the protein system to equilibrate to 300K and provide an accurate analysis of its dynamic behavior. A Wild Type crystal structure without an altered protein sequence was also generated and simulated under the same conditions to serve as a control.

Three 100- nanosecond simulations were conducted for each mutation and Wild Type models. The simulations provide atom-coordinates and energies of the protein system over the 100 nanoseconds. The data from the simulations is further analyzed by other programs, such as scripts for VMD and Schrodinger's Maestro, to gather the protein's C-alpha root-mean square deviations (RMSD), protein residues' and ligand atoms' root-mean square fluctuations (RMSF), van der Waal energies and hydrogen bonds between the ligand and protein.

The process of compiling and comparing these analyses across mutations was inefficient and difficult. Therefore, the team developed scripts in MATLAB and Python to increase the efficiency and effectiveness of analysis. The scripts standardized the data across mutations, generated visualizations of the data and allowed the adaptability for future analyses.

Average Protein RMSF was compared to the Wild Type average and the absolute difference was summed. V82F+I84V had the greatest absolute difference, followed by I82V and M46I+V82F+I84V. Additionally heat maps were generated showing differences compared to Wild Type with a color gradient ranging from red as a maximum and blue as a minimum. Similar to Protein RMSF, Ligand RMSF was compared to the Wild Type average and the absolute difference was summed. In the case of Ligand RMSF, an additive trend was observed with resistance increasing from the single mutant variant to the double mutant variant with an increase in fluctuation of 0.07 Angstroms. Additionally the triple mutant showed greater resistance compared to the double mutant with a fluctuation increases of 0.23 Angstroms. Hydrogen Bond percentages of each mutation were subtracted from the Wild Type percentages and summed. Hydrogen Bond percentages followed a different trend than the previous analyses with M46I+V82F+I84V retaining the most bonds. In the case of double mutation variant, the Wild Type had a significantly greater bond character. As expected, the mutated variants had less van der Waals interactions with the inhibitor. I84V had the fewest attractions with a summed difference of -3.63 kcal/mol compared to WT. V82F+I84V and M46I+V82F+I84V had similar van der Waal interactions with summed changes of -2.031 and -2.032 kcal/mol, respectively.

Ligand RMSF data supports our hypothesis; however, further analysis displayed inconsistent trends with drug resistance. M46I+V82F+I84V showed the most inhibitor fluctuation, followed by V82F+I84V, and I84V demonstrated the least amount of inhibitor movement. However, in the case of Protein RMSF, van der Waal interactions and hydrogen bond percentages, additive behavior is not supported. Van der Waal interactions provided inconclusive data, with I84V and V82F+I84V having a similar amount of interactions. With respect to Protein RMSF, V82F+I84V had the greatest fluctuation compared to Wild Type, followed by M46I+V82F+I84V, then I84V. Similar to Protein RMSF, V82F+I84V demonstrated the highest percentage of hydrogen bonds, followed by M46I+V82F+I84V, then I84V. This however, partially supports the hypothesis, as V82F+I84V and M46I+V82F+I84V appear to be additive.

To further support the conclusions of this project, the team suggests MD analysis of additional compounded mutations within the active site. Also, examining different environmental backgrounds can strengthen future correlations. Finally, the data generated from this project can be used in the development of protease inhibitors that are designed to retain potency across compounded mutations that may confer additive resistance.

1.0 Introduction

Human immunodeficiency virus (HIV) has been a major epidemic affecting roughly 34 million people worldwide [4]. Out of the 34 million, there are approximately 1.2 million people in the United States currently living with HIV [5]. Up to $19.1 billion dollars has been spent annually to support HIV treatments in underdeveloped countries, where HIV infections are most prevalent [9]. Developed countries, such as the United States, have been able to limit the spread of HIV-1, the most common serotype, through education, awareness and readily available antiretroviral therapies.

HIV is categorized as a retrovirus, which has the ability of integrating its viral genome into the genome of the host cell [10]. Once the viral genome is incorporated into the host cell's genome, the virus uses the host cell’s internal components to replicate, propagating the viral genome. A common method to treat retroviruses is inhibiting certain steps of the HIV viral life cycle, such as inhibiting HIV-1 protease for HIV. HIV-1 protease is a protein that cleaves synthesized polyproteins essential to the maturation of HIV [11]. Protease inhibitors are small molecules that can bind themselves in the active site of protease and cause protease to lose its functionality. Currently there are six different classes of drugs to treat HIV and the course of treatment normally consists of a “cocktail” of two or more drugs [9].

The major challenge in effectively treating HIV is the rate at which the virus mutates [7]. The two factors that cause high mutation rates throughout HIV replication are the short life cycles of the virus and the use of RNA as the genetic makeup. When mutations occur within the 198 amino acids that protease is composed of, the protein's three-dimensional structure will be altered. Current protease inhibitors are designed to inhibit HIV protease that do not have an altered structure. This results in resistance to the protease inhibitor over the course of HIV replication cycles.

To combat this issue of drug resistance, antiretrovirals are currently administered in combination known as highly active antiretroviral therapy (HAART) [12]. HAART therapy is the current gold standard for HIV management. However, there may be systemic side effects such as hepatotoxicity, kidney stones, and increased cholesterol levels [10]. When a patient is on HAART therapy, they reach a level of clinical latency. Over time therapy can become less effective, HIV levels in the blood stream will rise and CD4 cell counts will begin to fall.

Decreases in therapy effectiveness can potentially be improved by better understanding the mechanisms of drug resistance. This project aims to analyze drug resistance variants of the HIV-1 protease and develop patterns of drug resistance based on mutations. These mutations will be established from studying simulations of protease mutations and wild type HIV-1. Specifically, comparing mutated variants to the wild type using molecular dynamic principles are within the scope of this project. The potency of inhibitor therapies of each mutation will be quantified by analyzing mutated variants and wild type interactions with the inhibitor.

This project developed analysis software to effectively and consistently analyze the effect of mutations within HIV-1 protease on inhibitor interactions. Scripts were developed to determine the fluctuation within the protein and ligand, the alpha-carbon distance to track movement and changes in the active site. Additionally, hydrogen bonds and van der Waals interactions in the presence of Darunavir, the most potent protease inhibitor commercially available, were examined. By considering all of the above aspects of drug resistance, conclusions on the effect of each mutation on inhibitor effectiveness were drawn.

Future implications of the conclusions drawn through this project allow for the reverse engineering of novel protease inhibitors. Improved inhibitors can be developed through protein engineering, which is the design of a new protein or enzyme that has novel or desirable functions [13]. Protein engineering will be applied to this project by computationally developing new mutated HIV-1 protease structures by homology modeling. The mutated HIV-1 protease will have one amino acid replacement compared to the wild type. Mutations will be modeled using molecular dynamic software, and movement of the protein will be compared to the wild type.

By focusing on future mutations, drug designs should have a higher efficacy than existing market therapies and multiple inhibitors are no longer needed. From the potency data gained comparing the effectiveness of existing therapies, the type and rate of release required will be considered. Simulating HIV-1 protease mutations and applying molecular dynamic principles will provide an understanding of the functionality of the protease, as well as allow future mutations to be predicted. An adaptive analysis model may provide further drug resistance correlations leading to an improved approach to protease inhibitors.

2.0 Literature Review

2.1 Human Immunodeficiency Virus

HIV, which stands for human immunodeficiency virus, is a virus that infects the host's immune cells [3]. These immune cells are typically T cells that have CD4 receptors, of which HIV binds to, on their surfaces. HIV can eventually lead to AIDS, Auto Immune Deficiency Syndrome, and eventually death from co-infection. The world is currently experiencing an HIV/AIDS epidemic with about 37 million people infected, 2 million newly infected, and a mortality rate of 1.2 million in the year 2014 [4]. With a large population infected with HIV and continuous new cases of infection, the need for more potent treatment or a cure is escalating.

HIV is a retrovirus and there is no cure to eradicate the virus permanently. Instead, there are several antiretroviral drugs that repress the virus by slowing down its replication and infection rate into the target cells, which are the immune cells for HIV. Currently, there are 28 FDA-approved antiretroviral drugs available to patients for the repression of HIV [6]. Unfortunately, there are two main problems associated with these drugs, resistance and administration. There is a need to further understand these resistance mechanisms and develop a new drug or method to make the current antiretroviral drugs effective towards the resistant variants. The second problem is that the delivery of this drug to the body is not ideal. This includes the lack of convenience and adherence to taking the drugs, the toxic side effects, and the limited entry options of the drug to the body [10].

2.1.1 Clinical Relevance

There have been large financial investments made in treatment research. The therapy is required to be taken for the entirety of a patient's life and several people, especially in under developed countries, are unable to afford or obtain it [6]. Research so far has developed the 28 FDA-approved antiretroviral drugs and has increased the knowledge about the virus and its mechanisms. However, research still needs to continue to better understand the interactions between the virus and the host, in order to provide alternative drug delivery methods.

Recognizing these needs, our project involves researching drug resistance patterns to achieve a better understanding of the virus and its mechanism to resist current drug therapies. From this research, a predictive model of mutations is to be designed that can be used in alternative drug design.

2.1.2 Viral Structure and Life Cycle

HIV is an enveloped virus with several types of proteins embedded on the surface [14]. Two glycoproteins, gp120 and gp41 exist connected on the cell surface and are vital to viral entry [15]. Viral gp120 binds to the host protein receptor CD4, found on leukocytes, namely T lymphocytes. In addition to binding to the CD4, the virus must also bind to a chemokine co-receptor, CCR5. Binding to the CD4 and CCR5 receptor and co-receptor triggers fusion of the viral and host membranes through gp41. Gp41 consists of three primary domains: the intra-envelope domain, trans-envelop anchor and the extra-envelop domain. The extra-envelop domain is directly involved in the fusion process and is made of two hydrophobic heptad repeats, HR1 and HR2, and a hinge region. Upon binding and inserting fusion peptides into the host membrane, gp41 dissociates from gp120, causing gp41 to fold into a hairpin structure. The heptad repeats lie anti-parallel forming a 6-helix bundle, promoting the fusion and entry of HIV. The figure below displays the mechanism of how HIV attaches to the host cell and fuses together for the delivery of its RNA (Fig. 2.1).

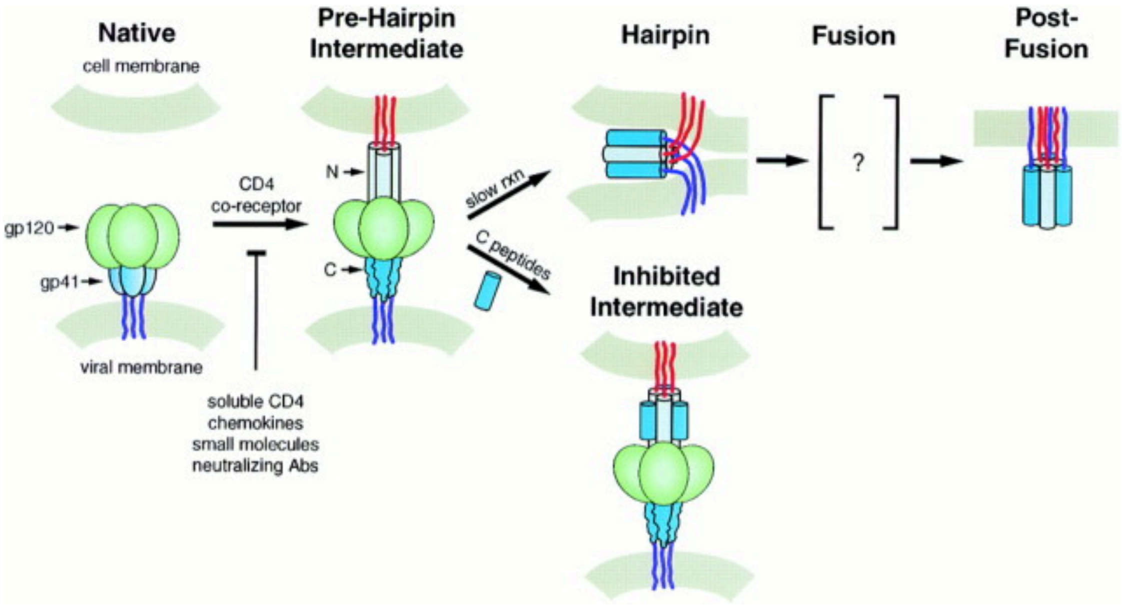


Figure 2. 1: Attachment/Entry and Fusion of HIV to the host cell [14]

Two proteins, gp41 and gp120, from the surface of the HIV virus stretch and bind to the host cell's membrane (pre-hairpin Intermediate) [14]. The connection then brings the membranes closer together (hairpin) to allow for fusion and entry of the HIV's RNA into the inside of the host cell (post-fusion). There are possible interactions at the pre-hairpin intermediate to inhibit the cell membranes to draw closer together (hairpin). These would stop the fusion of HIV and the host cell, thus a possible therapeutic treatment of HIV.

The next step of the cycle is reverse transcription that entails HIV RNA converting to DNA [16]. The enzyme reverse transcriptase (RT) is a heterodimer with a p51 and p66 subunit. The p66 subunit has catalytically active DNA polymerase and RNase H domains that both are responsible for converting the single-stranded RNA into double-stranded DNA. The first process involves using the viral RNA genome as a template for the host-cell transfer RNA to make a minus-strand DNA. This results to a RNA/DNA hybrid that the RNAse H domain cuts into several short RNA segments. Two of these RNA segments are polypurine tracts (PPTs) that start the synthesis of plus-strand DNA, which comes together with the minus-strand DNA to form a double-stranded DNA viral genome. The RNase H removes the PPTs from the DNA and exposes the integration sequence, which will be used for the integration step of HIV's life cycle. These processes of RT have to be followed precisely or integration will be prevented.

For HIV DNA integration, the DNA needs to integrate itself into a chromosome of the host cell [17, 18]. The enzyme integrase (IN) catalyzes the process of the viral DNA to inserting itself into the host chromosome. The first step of the process is IN trimming two nucleosides from the DNA. Next the IN stays bound onto the DNA and other viral proteins come together with IN to form a complex known as pre-integration complex (PIC). These viral proteins are reverse transcriptase (RT), MA, CA, and other accessory proteins. The PIC connects both ends of the DNA and travels through the cytoplasm to the nuclear membrane. It easily goes through the nuclear membrane because of its karyophilic properties due to the protein importin 7 and TNPO3. Another protein NUP153, which regulates nucleocytoplasmic trafficking, is a cellular protein that also helps the PIC to cross through the nucleus. Once the complex is in the nucleus, the integration of the DNA into the host's DNA starts. A large number of proteins are involved in this process that include HMGA1, BAF, Ku, LEDGF, HAT P300, HAT GCN5, LAP2-alpha, Emerin, JNK/Pin1, RAD51, and KAP1. After integration, there are some post-integration steps that allow gene expression and virion production. The proteins involved in those processes are INI1, VBP1, Daxx,transcription regulators/chromatin binding factors, and Huwe1. Any of these proteins are possible targets for therapeutic inhibition, but mechanisms of some proteins are better understood than others and thus are more likely a target for inhibition.

After integration, the host cell goes into a resting period known as resting peripheral blood lymphocytes (PBL) [19]. PBL is a state at which HIV-infected host cells have the HIV genome in the DNA but is not expressed yet. The PBL state is also known as the state of latency for HIV-infected patients. PBL state continues until a set of cellular factors interacts with amino acid sequences at the HIV long terminal repeat (LTR). The main cellular factor that initiates transcription is NF-κB, which is a protein with p50 and p65 subunits. P65 leads the transcriptional activity of the HIV genome in the immune cells' nuclei.

Transcription involves splicing the produced RNA into 46 sections. The spliced RNAs include the fully spliced mRNAs, which include Tat, Rev, and Nef, and the single spliced mRNAS, which include Vpu, Vpr, Vif, and Env [20]. Tat enhances the expression of the HIV genome via elongation of viral transcriptions with TAR, SP1, NF-κB, and other cellular factors [19]. Tat works with other proteins, cyclin T1 and PCAF, to increase HIV transcription and its quality [21]. Additionally, an enzyme human sirtuin 1 (SIRT1) recycles Tat in order for transcription to continue. The other mRNAs are responsible for other functions later in the HIV life cycle.

Another step of the HIV-life cycle is the assembly of the retrovirus particles, which is also known as packaging or encapsidation. In charge of this step are two strands of genomic RNA. These strands of RNA interact with the polyprotein Gag, which is the main structural polyprotein of HIV-1 capsids, and the ψ (packing signal) portion of the RNA [22]. Gag is 55 kDa and consists of four major subdomains. These subdomains include matrix (MA), capsid (CA), nucleocapsid (NC), and p6. The Gag protein is illustrated in Figure 2. The NC region of Gag bridges together the individual Gag monomers by gRNA (genomic RNA). The NC portion of Gag specifically binds to the ψ portion of the RNA to facilitate RNA packaging into virus particles [1, 20]. Further, the ψ portion of the RNA has four stem-loops (SL1-SL4). SL1 mediates RNA dimerization, SL2 and SL3 bind to the NC, and SL3 directs packing of heterologous RNAs [1].

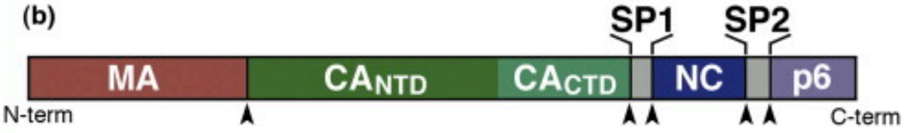


Figure 2. 2: Gag Polyprotein Domain Structure. Shows the MA, CA, NC, and p6 protein sections [1]

Once these RNAs, glycoproteins of the virion envelope (Env), viral structure enzymes (Gag), and viral enzymatic proteins (Pol) all assemble, the budding of the HIV viral particle is initiated [23]. The Gag proteins organize these proteins and protect them in an inner viral membrane. MA is the matrix layer of the inner viral membrane, the NC provides a nucleocapsid layer around the viral RNA genome, and

CA is the conical capsid surrounding the nucleocapsid, RT, and IN. P6 gathers the cellular components and Vpu support virus release needed for viral budding.

After budding, CA proteins reassemble to form a mature virus [24]. The capsid adopts a cone shape with the help of the envelope proteins. Also the NV/RNA complex condenses to the center of the core and genomic RNA dimer becomes more stable. A 5% ribosome shift of the C-terminal of gag results to a Gag-Pol protein when translated. The Gag-Pol protein encodes the information for producing a viral protease (PR). The PR slices the CA proteins so they can assemble into a mature virus. Once the virus is mature, the virus is able to infect other immune host cells and create more viruses.

2.2 Current Therapies

HIV antiviral medication targets the specific stages of the viral life cycle: fusion into the host, integration of viral genome into the host, translation of viral RNA and maturation of the virus following budding. Each inhibitor works in an independent fashion by various mechanisms to achieve the common goal of rendering the HIV virus nonfunctional and prevent CD4 cell destruction.

2.2.1 Fusion Inhibitors

The first class of HIV antiviral medication is fusion inhibitors, which prevent the entry of the virus into the host cell. There are five classes that block various stages of the fusion process: binding peptides to the heptad repeats 1 (HR1), binding peptides to the heptad repeats 2 (HR2), peptide-mimetic inhibitors, non-peptide inhibitors and CCR5 antagonists [15, 25]. Fusion inhibitors are infrequently prescribed, especially in early stage antiviral therapy. Among the fusion inhibitor classes, the most commonly prescribed and most effective are the peptide sequences that bind to HR1 and HR2. This review of available fusion inhibitors focuses primarily on the successful HR1 and HR2 binding inhibitors.

The first class is made of inhibitors that are successful by binding peptides to the HR1 [15]. However, only one fusion inhibitor, T20, is FDA approved and available on the market, while the remaining are still under development. T20 or Enfuvirtide, is a peptide sequence that is derived from the HR2 amino acids that binds to the HR1. This binding prevents interactions with the HR2 that are necessary to create the 6-helix bundle. T20 has a half maximal inhibitory concentration (IC_{50}) of 1.0 nM. Second generation peptide sequences that bind to HR1 have been explored. One example, C34, utilizes the same mechanism as T20 but only uses non-overlapping targets, where T20 targets may overlap. Further, the C34 sequence has a higher potency than T20, with an IC_{50} = 3.0 nM. An additional second generation HR1 targeted peptide that provides promising inhibition is T1249. This peptide offers high potential as it has been proven to be effective in both HIV-1 and HIV-2 types. Additionally, T1249 is more potent than T20, likely due to the fact that it binds to more targets, and has shown to be effective against HIV strains that are resistant to T20. Resistance often arises to this class of fusion inhibitors, as T20 and C34 only target 8-10 amino acids, and any mutations to the HR1 often render the inhibitors ineffective.

The second class of fusion inhibitors works in a similar fashion and is composed of peptide sequences that instead bind to the HR2 [15]. T21 is a synthetic 38 sequence N-peptide that is derived from HR2 amino acids. The first 25 amino acids in the peptide bind to the HR2 and is more potent than HR1 targeting peptides, with IC_{50} = 2.7 uM. Another HR2 targeted peptide is N36, which is a 36 amino acid sequence that is derived from HR1 amino acids. N36 has an IC_{50} = 308 nM, but also has two derivatives with 9 amino acid substitutions. These derivatives have been proven to be more effective than the parent N36 with IC_{50} = 16 uM.

Peptide-mimetic fusion inhibitors are large hydrophobic protein-like chains that have exhibited anti-HIV potential with an IC_{50} = 10.4 uM [15]. However, due to the hydrophobicity of peptide-mimetic molecules, their binding capacity is enhanced with trapping agents. Additional development with bind mechanisms and trapper agents are needed to promote the use of peptide-mimetic inhibitors. Non-peptide fusion inhibitors are another class of fusion inhibitors that have been developed, yet are not widely used. These inhibitors are a non-binding approach target at the heptad repeats to block the 6-helix bundle formation. Lastly, CCR5 antagonists bind to the CCR5 co-receptor to block CCR5 signaling.

2.2.2 Integrase Inhibitors

Integrase inhibitors are a smaller class of inhibitors that prevent the insertion of viral DNA into the host cell's regular genome [26, 27]. Host cells do not contain an integrase equivalent, therefore there is an increase in targeting efficiency and no interference with normal cellular function. Integrase inhibitors work to prevent the insertion of viral DNA through a two-step process. First, the 3' endonucleolytic processing of viral DNA is blocked. The second portion of integrase inhibition is preventing strand transfer, which is the joining of host and viral DNA. Divalent metals, such as magnesium ions, are required for both 3' processing and strand transfer, and is often the mechanism of integrase inhibitors.

The primary class and first true class of integrase inhibitors with no entry inhibitor mechanism is 4-Aryl-2,4-diketobutanoic acids (DKA) [27]. DKAs interact with the divalent metals in the active site, resulting in chelation of critical metals, rendering viral integrase nonfunctional. The most competitive and first FDA approved integrase inhibitor on the market is Raltegravir, which resulted from optimizing original DKA formulations. Raltegravir improved the metabolic stability, drug release profile and potency drawbacks of original DKAs with an IC_{50} of more than 50 uM, making it a more potent therapy than entry inhibitors.

2.2.3 Nucleoside Reverse Transcriptase Inhibitors

Nucleoside reverse transcriptase inhibitors (NRTIs) are the third class of inhibitors that work to prevent transcription and replication of the viral genome [28]. NRTIs competitively incorporate into nascent viral DNA through substrate binding. Lacking a 3' OH group, NRTIs successfully bind in place of viral reverse transcriptase and terminate the chain. However, the major challenge with NRTIs is the competition with natural dNTPs for recognition and catalysis in order to prevent DNA synthesis.

One example of a NRTI is Zidovudine, marketed under the brand name Retrovir. It was the first HIV antiviral to be approved by the FDA in 1987 [29]. However, Retrovir is primarily used in the treatment of maternal to infant transmission of HIV or in combination treatments.

2.2.4 Non-Nucleoside Reverse Transcriptase Inhibitors

Non-nucleoside reverse transcriptase inhibitors (NNRTIs) are similar to NRTIs, however, they are noncompetitive and target HIV-1 reverse transcriptase at a non-substrate binding site [12]. They are highly active against HIV-1, but cannot target HIV-2 or other retroviruses. Additionally, the major drawback of NNRTIs is they are notorious for triggering drug resistant variants.

There are two main classes of NNRTIs, which target either an allosteric or TIBO binding site [12]. 1-(2-Hydroxyethoxymethyl)-6-(phenylthio)thymine (HEPT) targets the allosteric site to disrupt enzyme activity and is functionally related to the binding site of HIV reverse transcriptase. The primary disadvantage of HEPT analogue NNRTIs is there is enhanced likelihood of resistance mutations. The second class, 4,5,6,7-Tetrahydroimidazo[4,5,1-jk][1,4]benzodiazepin-2(1H)-one or TIBO, targets the TIBO binding site.

2.2.5 Protease Inhibitors

The final class of HIV inhibitors is protease inhibitors. They prevent the cleavage of the viral polyprotein chain, which blocks the maturation of the virus following budding from the host cell. There are a total of eight protease inhibitors on the market, however the two newest and most frequently used in combination therapy are atazanavir and darunavir.

Original protease inhibitors became FDA approved in the 1990s, including saquinavir, ritonavir, nelfinavir and indinavir [30]. These first generation were successful but had major limitations. These included low potency, several severe side effects and complications.

Approved in 2003, atazanavir was one of the first second generation protease inhibitors [11]. In a study conducted by Molina *et al*., atazanavir was proven to be a potent inhibitor, especially when boosted with ritonavir. The study utilized patients who had not yet received atazanavir or ritonavir antiviral treatment. 82% treated with a combination of atazanavir and ritonavir responded to treatment and had plasma counts less than 50 HIV RNA copies/ml.

Darunavir is one of the recommended antivirals by the NIH Office of AIDS Research (OAR) for the treatment of HIV-1 [31]. It was originally patented in 2001 and approved by the FDA in 2006 as a second generation protease inhibitor, improving the potency and adverse side effects of first generation inhibitors [32]. Darunavir, [(1*R*,5*S*,6*R*)-2,8-dioxabicyclo[3.3.0]oct-6-yl] *N*-[(2*S*,3*R*)-4- [(4-aminophenyl)sulfonyl- (2-methylpropyl)amino]-3-hydroxy-1-phenyl- butan-2-yl] carbamate), is an effective, highly potent nonpeptidic inhibitor that embeds itself into the protease active site via hydrogen bonds (Fig. 2.3). The structure allows more hydrogen bonds within the active site to be formed compared to other protease inhibitors [31].

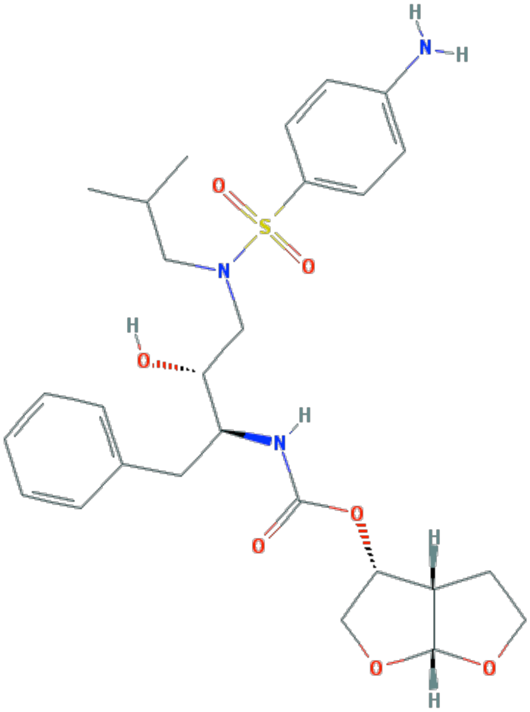


Figure 2. 3: Darunavir Structure [2]

Darunavir binds through hydrogen bonds to Asp 29 and Asp 30, successfully inhibiting HIV gag and gag-pol polyprotein cleavage. Binding at these sites attributes to its high potency [33]. In laboratory-synthesized strains, darunavir had 50% effective concentration of 1-5 nmol/L. The addition of α_1-acid glycoprotein (AAG) and human serum more accurately mimics *in vivo* and increased the IC_{50} 20-fold. Darunavir is often administered with a boosting dose of ritonavir to increase drug effectiveness. Single

600 mg doses of darunavir have a bioavailability of about 37%, compared to 82% when administered with a 100 mg ritonavir.

2.2.6 Combination Therapy

HIV antiviral therapies are most commonly prescribed in combination in a highly active antiretroviral therapy (HAART) approach, referred to as the "Anti-HIV Cocktail" [12, 34]. Conceptualized in 1996, HAART has since significantly improved the prognosis and quality of life of HIV-infected patients. Modeled after tuberculosis treatment, three or more therapies are taken at once in order to produce a synergistic effect between different molecular targets, to lower the dose of each individual drug and decrease the probability of drug resistance. Tenofovir is the most commonly used antiviral in the United States, prescribed to 65% of patients on an antiviral regimen. The World Health Organization (WHO) launched the "3 by 5 Program", which was an initiative to get 3 million HIV infected patients on antiviral therapy by 2005, in response the effectiveness of HAART.

In a study conducted by Finzi *et al.*, 22 patients treated with HAART were successfully treated over the course of 30 months [34]. The patients involved in this study had T cell counts ranging from 0.2 to 16 per 10^6 cells at the time combination therapy began. HIV-1 RNA levels were approximately undetectable 2 months after the HAART regimen was implemented (Fig. 2.4). In the same time frame, the majority of patients exhibited a significant increase in CD4 cell counts.

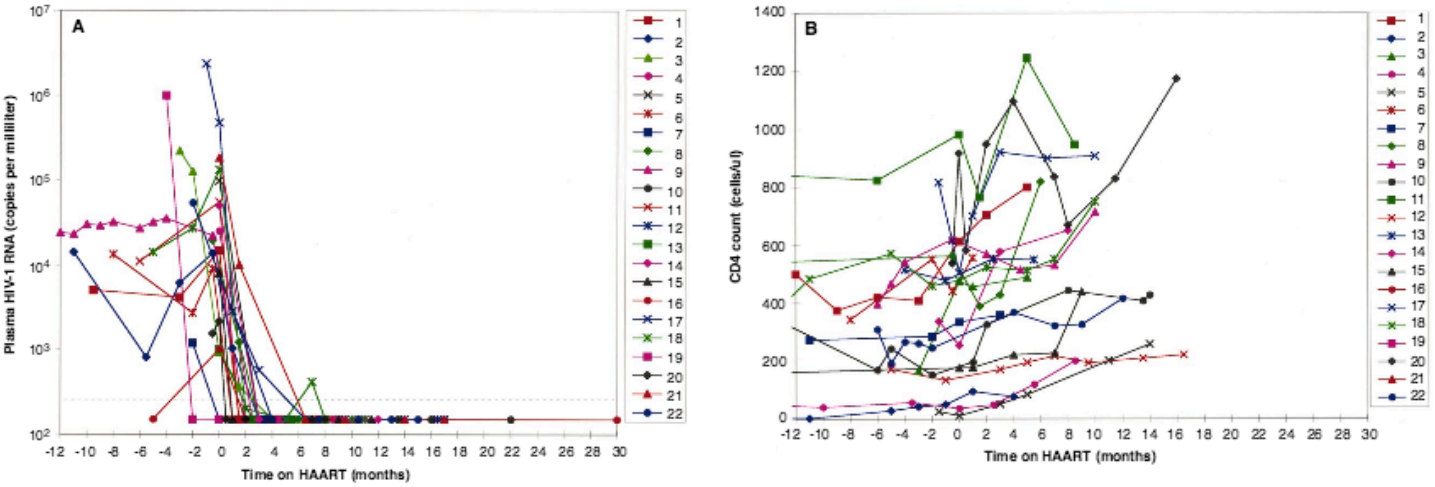


Figure 2. 4: Effect of HAART Therapy on Plasma HIV-1 RNA and CD4 Cell Counts [34]

Administered HAART therapies usually consist of three of four NRTIs and/or NNRTIs, one or more of three major drug classes, or a single PI with a boosting inhibitor [30]. Since HAART therapy is the future to HIV management, there are several drugs available on the market that are combinations of different inhibitors as one medication. One combination of three NRTIs is Trizivir, composed of abacavir, lamivudine and zidovudine. Approved in 2011, Complera is composed of 2 NRTIs and 1 NNRTI: emtricitabine, tenofovir disproxyl fumarate and rilpirivirine. The two newest FDA approved combination capsules are Evotaz and Prezcobiz, approved in January 2015 as protease inhibitors with a boosting inhibitor. Both use cobicistat to enhance the effectiveness of the inhibitor. Evotaz contains atazanavir and Prezcobiz is made with darunavir.

2.2.7 Therapy Limitations

The primary limitation to HIV antiviral treatment is the many systemic and adverse side effects [35]. Since HAART therapy is widely used, common effects of several frequent high doses of medication is kidney stones and hepatotoxicity. The high doses of toxic drugs can often damage the liver and excess drug accumulates in the kidneys. Another side effect is a gastrointestinal reaction, such as nausea, vomiting and diarrhea, which is consistent with several medications. High levels of triglycerides, cholesterol and blood glucose levels is another common systemic effect that can lead to fat redistribution, decreased bone density and bone marrow suppression.

Medical adherence is another drawback to current HIV antiviral cocktails [36]. Patients fail to take their medication as prescribed for a variety of reasons. Cost of medication is one of the most common reasons, as HAART therapy can cost $2000-$5000 monthly, and amount to over $500,000 in a lifetime. Because of cost and/or availability, lower doses of medication are sometimes taken in attempt to stretch supplies or are shared amongst families.

Another factor that significantly affects therapy effectiveness is the resource availability and socioeconomic status of patients. A study conducted by Dabis *et al.* found that patients in low-income settings had lower CD4 cell counts and a higher instance of mortality when using HAART therapy than higher income countries [37]. One of the primary factors concluded to contributing to the higher mortality rate is the lack of follow up throughout the progression of treatment. Additionally, co-infections with other infectious pathogens, specifically mycobacteria, can have a significant impact on therapy effectiveness in poorer countries. Lastly, another main factor in therapy success in low-incomes is the free access to drugs and use of generic brands. Several countries in south and east Africa, including Botswana, Malawi and Uganda have restricted access to therapies.

2.3 HIV-1 Protease

HIV-1 protease is vital for the maturation of HIV after the virus buds from the host cell. This is one component of the life cycle that can be targeted and inhibiting therapies to limit the propagation of the virus throughout the body.

2.3.1 Structure

HIV-1 viral protease is a relatively small protein that accounts for a pivotal role in the life cycle of HIV-1 [7]. In nature there are many forms of protease that utilize several types of catalytic mechanisms to cleave specific peptide bonds within a protein structure. Retroviral HIV-1 protease is a considered to be a symmetrical homodimer that is composed of two identical dimers. Each of these small protease dimer subunits are made up of 99 amino acids. Modeling of HIV-1 protease mutants is much simpler than larger complex proteins since these dimers only consist of 99 amino acids. Each dimer has 18 β-strands and 2 α-helixes that are oriented to create a structurally stable and highly functional enzyme.

HIV-1 protease has six major regions that play roles in the structural stability and catalytic properties of the enzyme [38]. The first region of importance for HIV-1 protease is known as the flap domains located on the most superior region of the enzyme. The flap domains for each dimer are formed by two anti-parallel β-strands composed of viral amino acids 44 through 57 [38]. The flap domains are highly mobile domains within HIV-1 protease that interact with viral gag-pol polyproteins to assist in peptide bond cleavage. These flap domains are able to move easily due to the elbow regions consisting of viral amino acids 49-52 that directly allow for the anti-parallel flaps to move about the elbow axis. The reason for the high amounts of mobility of the viral protease flap domains is the glycine rich regions with the elbow region. HIV-1 protease flap domains and elbow regions work together to ensure proper binding of the gag-pol substrate to the enzymatic active site. Scott et al. found that the relative static charges of the viral flaps and active site domains to the charge of the gag-pol substrate cause conformational changes in HIV-1 protease for proper substrate binding [39]. The active site within this enzyme is highly hydrophilic as a result of the catalytic properties present within this region. The flap domains of the enzyme have a hydrophobic surface that remains in a closed conformation when in contact with its external environment to maintain a neutral charge. However, the gag-pol polyprotein has a net positive charge, which wants to repel the positively charged flap domains of the viral protease causing the flaps to pivot about their glycine rich elbow regions. While the flap domains undergo the conformational change the net negative charge of the active site attracts the positively charged gag-pol substrate for proper binding. Through the use of molecular dynamic simulations of wild type and mutant variants of HIV-1 protease it was discovered that glycine-51 in the elbow region found of the flap domain significantly affects viral protease activity. A mutation to glycine-51 can cause the flap domains to remain in a closed conformation when the gag-pol substrate tries to interact with the enzyme.

The next two major domains of HIV-1 protease are the highly hydrophobic regions. These domains are located in the medial and inferior regions of viral proteases [40]. The first of these hydrophobic domains is the medial region composed of antiparallel β-strands, which play an important role in the stability of the homodimer. The inferior region of protease is composed of both β-strands and α-helixes that also contribute to the overall stability of the enzyme. These two hydrophobic regions are differentiated by what are known as the 10's and 60's loops. The 10's loop is specific to the medial hydrophobic domain that consists of viral amino acids 15 through 18. Similarly, the 60's loop is specific to the inferior hydrophobic domain, which consists of viral amino acids 66 through 69. Until recently it was thought that the hydrophobic domains of HIV-1 protease had little to no effect on the proper binding of the gag-pol substrate to the enzymatic active site. Hydrophobic sliding was observed when HIV-1 protease wild type and hydrophobic mutants were simulated using molecular dynamics [41]. The Schiffer lab was able to determine that protease undergoes conformational changes within the hydrophobic domains of the protease through the "sliding" movement of the loop regions.

The final domains of HIV-1 proteases molecular structure responsible for functionality are the dimerization and active site regions [42, 43]. HIV-1 protease is an enzyme that is formed when the two identical dimer subunits fold into the proper form. The most important site of interaction between the two dimers during the protein folding process is the dimerization region. The dimerization region is where the four amino acid long N and C termini of both dimers orient themselves tightly together to form a functional enzyme. Arguably the most important region within HIV-1 protease is the active site. The active site is where HIV-1 protease performs the catalytic properties that the enzyme is known for. HIV-1 proteases catalytic activity is due to two aspartic acid residues (Asp-25 and Asp-25') which cleaves specific peptide bonds while creating by products typically seen in hydrolysis.

2.3.2 HIV Virion Maturation

As described earlier, the HIV-1 viral maturation process occurs after virions have budded out of the host cell. The immature virion consists of gag and gag-pol polyproteins that are inactive proteins.

These viral proteins become functional when the gag and gag-pol polyprotein complexes are cleaved at ten specific peptide bonds [44]. HIV-1 protease is also initially in an inactive form during virion budding since it is associated with the gag-pro-pol polyprotein. Once virion budding occurs, HIV-1 protease undergoes self-maturation where it cleaves itself from the gag-pro-pol polyprotein. The mechanism that HIV-1 protease performs to undergo self-maturation is still not understood.

However, the functional form of the viral protease cleaves the polyproteins found within the virion to produce proteins required for HIV-1 survival known as MA, CA, NC, RT, and IN. The matrix (MA) protein plays a major role in pre-budding since it is responsible for aligning the gag and gag-pol polyproteins to the plasma membrane of the host cell [44, 45]. Once matured, MA proteins orient themselves directly under the virion plasma membrane. Capsid (CA) proteins are responsible for forming a stable shell within the virion to protect essential viral proteins for viral replication including NC, PR, RT, and IN. The capsid protein is thought to be another important protein associated with the HIV-1 life cycle since this particular protein has been observed to have the lowest amount mutations [45]. The nucleocapsid (NC) proteins are known to bind to viral RNA in order to successfully bring the viral RNA towards the center of the capsid during maturation. Reverse transcriptase (RT) protein is the protein that gives retroviruses the capability of converting viral RNA into viral DNA. Finally, integrase (IN) protein is an enzyme that HIV-1 used to integrate the viral DNA produced by RT into the host's genome.

2.4 Mechanisms of Drug Resistance

One of the most important factors leading to antiviral therapy failure is viral resistance [46]. The key to improving viral inhibitors is understanding the mechanisms and factors that lead to drug resistant variants of HIV-1.

2.4.1 Mutation Based Drug Resistance

Of the roughly 25 different drugs used to suppress HIV-1 approved by the FDA, protease inhibitors are the most effective choice of inhibitors [7]. The efficiency of HIV-1 protease inhibitors is partly due to the fact that the inhibition of the viral protease causes the essential proteins for viral replication to remain inactive. HIV-1 has been able form drug resistant variants for each type of inhibitor, however, protease inhibitors have the most mutants.

HIV-1 drug resistance is not a static process, conversely, many factors contribute to viral drug resistance [8]. One important factor that contributes to HIV-1 drug resistance is the short life cycle of the retrovirus. Since the virus is short lived it has an extremely high rate of replication such that virions are produced daily. Also, HIV-1 does not have a proofreading mechanism for the RT protein so there is a high amount of random genetic variation between successive replications of the virus. The combination of high replication rates and error-prone transcription results in the evolution of HIV-1 strains *in vivo*. HIV-1 can make random mutations that are useful for survival in the presence of an inhibiting drug and this particular strain will be selected for. Drug resistance can also be reached by means that are not related to the nature of HIV-1 [7]. The most common means of drug resistance if the failure to consistently take the anti-viral treatments prescribed.

Successful drug resistant variants of HIV-1 protease are capable of exhibiting the catalytic functions of the enzyme through mechanisms slightly different from the wild type [7]. The tertiary structure of HIV-1 protease is directly dependent on the 99 amino acids found in each dimer and the way in which they fold. The inhibition of protease through protease inhibitors depends on the hydrophilic/hydrophobic interactions between the substrate and active site as well as the three dimensional orientation of both entities. Protease inhibitors are small molecules that mimic the three dimensional orientation of the substrate region that binds to the active site [47]. These protease inhibitors alter the functionality of HIV-1 protease by competing with the substrate for binding in the active site. The advantage of protease inhibitors is that they have a much higher affinity for the active site since they were designed to mimic the transitional state of the substrate. Drug resistant strains of HIV-1 protease can be developed through mutations within the active site. Functional active site variants can reduce the affinity of an inhibitor towards the active site and give the substrate a better chance at binding.

2.4.2 Substrate Shape Dependent Resistance

Protease inhibitors are peptidomimetics, relying on structure based design targeting, opposed to site specific protein binding within the active site [46]. HIV protease is a resilient and adaptable homodimer that can bind to its substrate even without having properly oriented ligands. Prabu-Jeyabalan *et al.* identified crystal structures for six complexes that correspond to six of the ten gag and pol cleavage sites. It was concluded that viral protease recognizes the asymmetric shape of the substrate peptides instead of the specific amino acid sequence or set of hydrogen bonds. Protease inhibitors are successfully structured based designs. However, HIV mutates frequently and is error-prone to frameshift mutations, altering the active site. Therefore, marginal crystal structure changes of the protease substrate may confer drug resistance.

2.5 Drug Resistance Simulation and Molecular Dynamics

Drug resistance often occurs in sites of HIV-1 protease that experience the most movement, such as the flap regions [39]. Crystallization and sometimes NMR are unable to provide enough details on the role of individual atoms. These atomic details are often necessary to understand the substrate recognition process and why mutations on and around high movement regions lead to drug resistance.

Hypotheses of atomic level movement of HIV-1 protease are often based on molecular dynamic simulations [39]. Simulations provide a dynamic viewpoint of several biomolecules and interactions between biomolecules [48] These simulations have become a standard tool for analyzing biomolecules and have been developed to use more realistic boundary conditions and longer simulation times. Molecular dynamics bring the dynamic movement data for biomolecules in solution, time-average molecular properties that are comparable to the experimental properties, and thermally accessible conformations. With that information, scientists can gather the movement of the biomolecule, free energy differences (ligand binding), and ligand-docking applications.

2.5.1 Homology Modeling

Homology modeling is essential for creating structurally accurate models for proteins [49]. The model uses a protein structure prediction process via a computer program that includes template identification, alignment, and model building [50]. Template identification is finding a PDB (Protein Data Bank) file of a biomolecule from an amino acid sequence that the user created or edited. This edit could refer to a mutation of one of the amino acids of the sequence. The computer program provides several possible template identifications and the user chooses which PDB or structure best represents the desired biomolecule. The alignment stage pertains to aligning the entered amino acid sequence of the user to the amino acid sequence of the chosen template. Next, the 3-dimensional model of the user's amino acid is built based off of the 3-D structure from the template [49]. Creating these models use the computer programs' structure prediction workflow, called a run, which uses particular templates, paths, and settings. The built structure can be manually modified if the prediction software did not provide the user's desired structure. These "mistakes" are essential to get fixed because the misrepresentations can greatly affect the chemical/physical properties and therefore the movement of the protein.

2.5.2 Molecular Dynamic Preparations

After the model of the protein is developed, the protein must be refined, which is part of the preparation process for experimentally resolved protein structures used in calculations [49]. Refinement includes loop refinement, side-chain prediction, minimization, rigid-body minimization, hybrid Monte Carlo conformational searching, binding site refinement, and energy analysis based off of the structure's geometry [50]. The refinement stage double-checks if the physical and chemical properties of certain aspects of the biomolecule are realistic. Specifically, the refinement tools can fix many structural problems, such as formal charges, bond orders, and disparities between the sequence and the structure. This type of correction was mentioned before in the homology modeling, but the software checks in case the user did not notice every structure problem.

Many computer programs also provide solvation models for the biomolecule. Solvation models provide an environment for the biomolecule [50]. There are three common options for the solvation model. The first is VSGB that provides an aqueous model/ water environment. The vacuum option turns off the solvation model (no water) chloroform that uses the SGB method. The model also has boundary sizes to determine how large the user would like the environment for the biomolecule to be. Standard size for HIV-1 protease is a boundary size of 10 x 10 x 10 angstroms [39]. The refining loops capability checks the loop structure using different algorithms for various loop lengths [50]. The prediction of side chains capability estimates the conformation of the biomolecules' side chains by sampling multiple orientations to obtain the one with the smallest energy (minimized energy). The side chains that are chosen for the sidechain minimization step are the mutated amino acids/residues.

Minimization involves sampling different orientations of the biomolecule or parts of the biomolecule to find the lowest energy orientation [50]. There is an option to treat part of the system as a rigid body with freely moving atoms and freezing the rest of the system, which is known as rigid-body minimization. Rigid-body minimization is sometimes preferred because it is less time-consuming than minimization. The hybrid Monte Carlo process simulates the molecular dynamics of the biomolecule using high-temperature. This explores the conformational space of the biomolecule to obtain the lowest energy structure/orientation. The protein-ligand complexes can be used to refine the interactions between the protein and ligand by sampling positions and conformations of the ligand. This process is also known as binding site refinement. The last step of refinement is analyzing the molecular mechanics energy. The program uses an all-atom force field to predict the energies. These energies can be broken down to covalent, Coulombic, van der Waals, and solvation energy contributions. The energies can be broken down to certain sections of the biomolecule, such as residues, and visualized on the structure.

2.5.3 Molecular Dynamic Simulation

Once the preparation of the model is complete, it can run through a molecular dynamic simulation [39]. This simulation is run through a separate computer program with the necessary calculations to provide accurate movements of the protein. Specifically, the program takes in all the parameters, which include several constraints on how the atoms move, and the original atom coordinates. A time is selected on how long the simulation of the protein is ran, which 100 nanoseconds is a common choice. The atom coordinates, based off of the constraints, are recorded after each time frame of the simulation.

After the simulation is done, the computer program gives the information on the atom coordinates and their energy levels for each time frame, which is commonly 10,000 [51]. This information can be further analyzed to summarize the movement of the protein and/or to predict the interactive forces between or within the protein and inhibitor.

2.5.4 Molecular Dynamic Analysis

One of these analyses is known as RMSF, or root-mean squared fluctuation. The equation for RMSF is shown below [52]:

$$RMSF_i = [\frac{1}{T}\sum_{t_j=1}^{T}\left|r_i(t_{j)}) - r_i^{ref}\right|^2]^{1/2}$$

The RMSF is the deviation between the position of atom I and some reference position [52]. The time point for the molecular dynamic simulation is denoted as j, the T is the total period of the molecular dynamic simulation, $r_i(t)$ is the position of the atom in that trajectory time point, and r_i^{ref} is the position of the atom from the reference time point, which is usually time zero. RMSF values represent the average movement of the atom over the entire simulation time. When comparing the RMSF values of certain atoms or sections of the biomolecule, the values can show which sections are more flexible or experience movement more than others.

There are several other analysis tools that VMD can calculate. For this project, the protein-ligand RMSD, protein RMSF, ligand RMSF, hydrogen bonding, and van der Waals values between the ligand and biomolecule. The equation for the RMSD is shown below [52]:

$$RMSD(t) = \left[\frac{1}{M}\sum_{i=1}^{N} m_i\left|r_{\square}(t) - r_i^{ref}\right|^2\right]^{1/2}$$

M is the sum of masses of the molecules being investigated; N is the number of atoms, m and the mass of the atom that is being investigated. The RMSD values for a biomolecule or usually calculated for the C-alpha atoms because they are the most stable /backbone of the biomolecule [52]. The values of RMSD are compared among the C-alpha atoms (one pertaining to each residue of HIV-1 protease) and represent the total movement of the pertaining atom.

Hydrogen bonding is whether the ligand and protein have potential to make hydrogen bonds during their interaction. This type of bonding gives an idea on whether the ligand will successfully attach and/or stay in the active site. The hydrogen bonding is defined by a certain distance between the two atoms, which is usually around 3 angstroms [52]. The van der Waals values are the attractive forces that keep the ligand attached to the active site.

2.5.5 Molecular Dynamic Software

Maestro:

Maestro is a molecular visualization interactive computer program produced by the company Schrodinger [53]. When coupled with Prime, another Schrodinger program, the program can build a computer-generated model of a biomolecule with realistic environmental properties. Prime includes refinement, solvation models, and minimization. This model will later be ran through a molecular dynamic simulation to provide information on the atomic-level movements of the biomolecule. The program is user friendly and used mainly for building, visualizing, and sharing 3-dimensional chemical models. For more detail of the homology modeling and refinement criteria, see the *Prime User Manual* provided by Schrodinger [50].

Desmond:

Desmond is the computer program that carries out molecular dynamic simulation [51]. This computer program is highly used among researchers because of its high accuracy and its effective computational abilities. Desmond is often considered the most accurate molecular dynamic simulation compared to other similar programs.

VMD:

VMD is an interactive 3-dimensional molecular graphics program that is commonly used to analyze a biomolecule and its molecular dynamics simulation information, such as trajectories [54]. The program is able to take atomic coordinates, chemical and physical properties, and other information given by a PDB to provide an accurate graphical view of the biomolecule. The user can directly interact with the biomolecule by rotation, zooming in and out, and selecting certain atoms or sections of the biomolecule. Several features are included in VMD, such as coloring options, selecting options, graphical view options, and the ability to run additional analysis via external programming scripts. The ability to use external programming scripts allows the user to achieve further information about the biomolecule. For example, the trajectories of the atomic coordinates and their properties at several time points across the span of the molecular dynamic simulation can be used to display a 3-D graphical view of the movements of the proteins. The atomic coordinates of the different time points of the molecular dynamic simulation also gives the VMD to analyze the movements and interactions of sections of the biomolecule. These movements and interactions include RMSF (biomolecule and ligand), RMSD (biomolecule and ligand), hydrogen bonding, and van der Waals.

3.0 Project Strategy

The client Celia Schiffer, PhD, has an interest in "understanding the molecular basis of drug resistance and the ways the natural substrate specificity is maintained by the resistance viral variants [of HIV-1 protease]" [55]. However, HIV has been proven to be a highly error-prone virus that also has a high replication rate, leading to drug resistant variants. This project aims to understand the molecular basis of drug resistance common among affected patients. Further, the team will develop software programs to efficiently analyze aspects of drug resistance.

3.1 Initial Client Statement

Initially, the client provided a statement for our team:

Using Molecular Dynamic principles to analyze the effectiveness of the inhibitor on mutated variants of HIV-1 protease. Project goals are to include prediction of potential mutations of amino acid residues.

After receiving the client statement, the team researched the current benchmarks of HIV-1 protease therapeutics to gain a better understanding of the project. Through research, it was found that protease inhibitors are used as final stage antiretroviral therapy, where drug resistance has limited the effectiveness of initial therapies. Instead of focusing solely on protease inhibition, the team considered alternative mechanisms to prevent viral propagation within the body. These alternate considerations led to the development the following need statement:

Develop a process to understand the mechanisms of drug resistance in HIV-1 protease, as it has a high degree of error-prone replication, in order to design more potent therapies to render the virus non functional regardless of present mutations.

The need statement entails the project's main goal and also allows the design space to be broadened. The engineering design processes can establish the best mechanism on how to address the need statement.

3.2 Technical Requirements

Lists of research and design objectives were developed based on the client need and after gaining a better understanding of the project. From ranked objectives, project constraints and functional needs were identified.

3.2.1 Research Objectives

These research objectives outline the necessary characteristics of an effective process to determine HIV-1 drug resistance mechanisms. Primary objectives include this research to be accurate, measurable, repeatable and reproducible. Research objectives were compiled into an objectives tree, and are described in further detail below (Fig. 3.1).

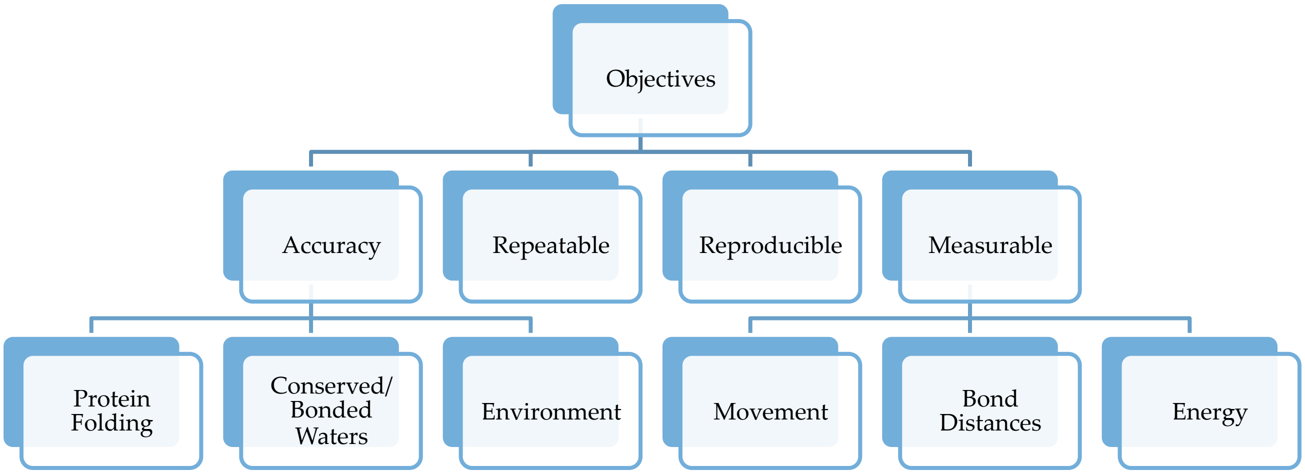


Figure 3. 1: Research Objectives Tree

The first objective identified is accuracy. It is critical that models accurately reflect the actual molecular structure and behavior. If the viral models are not indicative of actual behaviors, data is invalid, as it would not properly depict the real time virus mechanisms. Protein folding, conserved/bonded waters and environmental factors further characterize accuracy. Protein folding is dependent upon the consistency of the amino acid sequences. Different sequences will result in different folding patterns that can lead to various quaternary structures. When modeling mutations, the folded protein structure must mimic in vivo conditions. Conserved/bonded waters play a role in inhibitor binding and must be accurately portrayed to simulate in vivo settings. Lastly, environmental factors including temperature, pressure and pH can affect inhibitor binding and functionality.

Another objective stipulated is measurability. Various data must be measured from the research in order to analyze inhibitor effectiveness. Dynamic protein behavior, including movement, bond distances and energies must be able to be measured. Movement is an important factor in determining whether the substrate is available for binding. This movement must be able to be detected in order to quantify binding and inhibitor uptake abilities. Bond distances are directly related to the presence of hydrogen bonds. This is an important measurement to determine if the inhibitor is experiencing hydrogen bonding with the protease. Energy levels are directly reflective of the interactions between the inhibitor and the substrate. These interactions, van der Waals forces, are determined by measuring the change in energy of the system when an inhibitor is bound and unbound. Bond distances and energy levels contribute to the overall strength of inhibitor to protein binding.

Repeatability is another important objective that must be satisfied through this process. When conducting research to develop conclusions and correlations, multiple trials must be executed. Therefore, the process must be able to be repeated multiple times, garnering consistent results across replicates.

The final primary objective identified is reproducibility. This entire experiment must be able to be conducted again, even in a different laboratory setting. Results obtained through additional experiments following this process should still conclude consistent results.

In order to rank objectives in order of importance, the team created a pairwise comparison chart (Table 3.1). Objectives were ranked against each other using a number system of 0, 0.5 and 1. Objectives that are determined to be of greater importance was given a 1 and the other objective received a 0. Each objective received a 0.5 if they were determined to be of equal importance.

Table 3. 1: Research Objectives Pairwise Comparison Chart

Objective	Accuracy	Measurable	Repeatability	Reproducibility	Score
Accuracy	—	0.5	1	0.5	2
Measurable	0.5	—	1	1	2.5
Repeatability	0	0	—	0.5	0.5
Reproducibility	0.5	0.5	0.5	—	1.5

Accuracy and measurable were deemed as equally important since accurate measurements a vital to this project. The measurements obtained will be used to quantify inhibitor effectiveness, and this must accurately portray in vivo conditions. Comparing accuracy and repeatability, accuracy was deemed more important as there will always be variation among replicates because of the dynamic protein behavior. The team determined accuracy and reproducibility are of equal importance, since both are required to draw correlations of the effect of mutations on inhibitor effectiveness. Measurability is considered more significant than repeatability since conclusions will be based off of measured data. Measurability is of greater importance than reproducibility since measurements over multiple experiments determine validity. Repeatability and reproducibility are considered equally important, as both are required in developing a valid process.

Using the pairwise comparison results, the team ranked objectives from most significant to least significant (Table 3.2).

Table 3. 2: Ranked Research Objectives

Ranking	Primary Objectives
1	Measurable
2	Accuracy
3	Reproducibility
4	Repeatability

3.2.2 Design Objectives

In order to properly analyze research simulation results, a software program is to be designed primary and secondary objectives were defined. Primary design objectives include, accuracy, adaptability, and efficiency. Design objectives were compiled into an objectives tree, and are described in further detail below (Fig. 3.2).

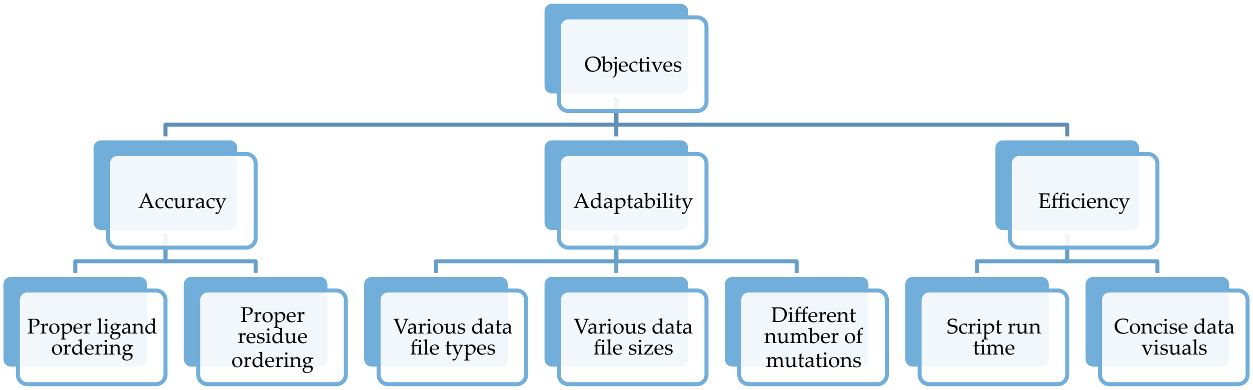


Figure 3. 2: Design Objectives Tree

The first objective defined is accuracy, as it is crucial the program accurately analyzes the simulation data. Included in accuracy is proper atom ordering during analysis, as raw data is exported with a non-ordered ligand sequence and the first residue of chain B is at the end of the data set.

The second design objective is adaptability. The modification of the programs to account for different mutations and data files is critical. Analysis of several point mutations and replicates are required to understand the mechanisms of the drug resistance. Further, the program should be used for additional experiments to maintain consistent analysis and must be able to be adapted accordingly. Therefore, the program needs to account for multiple data files and expected to provide consistent results.

The last design objective of the software programs is efficiency. Efficiency is vital to the analysis of the research outcomes and the ability to draw conclusions from the analysis. Efficiency is further characterized by the time for the programs to run and the capability of presenting the data representation in a clear and concise fashion.

To rank objectives in order of importance, the team created a pairwise comparison chart (Table 3.3). Objectives were ranked against each other in the same fashion as research objectives.

Table 3. 3: Design Objectives Pairwise Comparison Chart

Objective	Accuracy	Adaptability	Efficiency	Score
Accuracy	—	1	1	2
Adaptability	0	—	0	0
Efficiency	0	1	—	1

Accuracy was ranked above adaptability, as the program must accurately account for discrepancies among data files. For similar reasons, efficiency was determined to be less importance than accuracy. In the context of our project, efficiency is more important than adaptability, as data must be presented in a comprehensive manner with minimal script processing time.

Using the pairwise comparison results, the team ranked objectives from most significant to least significant (Table 3.4).

Table 3. 4: Ranked Design Objectives

Ranking	Primary Objectives
1	Accuracy
2	Efficiency
3	Adaptability

3.2.3 Project Constraints

The team had to take into account multiple constraints based upon our client statement and design meetings with our project advisor, Celia Schiffer, PhD. The most important constraints that the team must take into account during the course of the project are:

- In vivo environment
- Minimum replicates
- Time

The first constraint of this project is the design must mimic the in vivo environment, including pH, temperature and pressure. The pH must be regulated, such that it remains near body pH, 7.4. In order to account for fluctuations within the modeling, the pH range is set to be 4.0 to 10.0 but is monitored throughout experimentation. Temperature is also an important factor in determining inhibitor effectiveness, since increased temperature increases molecular movement. In order to ensure accurate results, temperature must be 37°C and remain consistent throughout. Pressure must also remain consistent throughout experimentation, being conducted at one atmospheric pressure.

The second constraint of this project is the minimum number of replicates for each mutation. This project encompasses analysis from three protease mutations. Drug resistant patterns would be insufficient based off less than three replicates of each variant.

Our final constraint is the timeline in place of the completion of the project. The project must be completed before Project Presentation Day during the 2016 school year.

3.3 Industry Standards

Industry standards for Molecular Dynamic (MD) simulations are fairly new and still developing. So far, there are requirements for the equilibration, boundary conditions, and the need for several MD runs [56]. For each MD simulation performed, equilibration must be reached. Equilibration can be detected by several factors, including root-mean squared displacement, or deviation, and steady temperature. The steady behavior of the root-mean square displacement of the protein and the steady temperature throughout the simulation will be observed. If neither are obtained in the particular simulation, the simulation is no longer valid for analysis. Boundary conditions are essential to an accurate simulation, as too large or too small directly impact the results. The boundaries are set to values of previous literature pertaining to our project, as they have tested different boundary sizes and found the optimal size. Lastly, the number of MD simulations for the same conditions are vital to an accurate analysis. MD runs are not quite accurate, but they are precise. Thus, several runs will give us enough values (mean value) to obtain accurate results.

Another industry standard this project must adhere to are the standards and guidelines surrounding computer programming. MATLAB is one language with a published guide that specifies variable naming convention, the proper formulation and development of functions, organization of files, statements, and formatting [57]. This guideline will be considered when writing code for this project, as the team is designing a program that can be adapted for further analysis and potentially other proteins. Therefore, the code must be logical, well documented and commented, and formatted clearly for future edits to be made quickly.

The final industry standard considered throughout the course of this project is ISO 18458:2015, regulating biomimetics [58]. This standard specifies regulations and definitions for computational analysis or systems that mimic a biochemical process. Recently developed, it provides a framework for biomimetic technologies and applications including scripting languages and programming resources.

3.4 Revised Client Statement

After the team gained a better understanding of the background and the broader need of the project, the team proposed a revised client statement. Discussions with the client further modified the team's revised client statement. Thus, the final revision was developed that both the team and client contributed to.

Using molecular dynamic principles to analyze the effectiveness of the inhibitor on mutated HIV-1 protease variants. Goals are to include finding potential resistance patterns to specific mutations, and discovering attributes for a drug that would be most effective to several mutations and possibly different classes of HIV.

The revised statement added additional desired goals of the project. From literature and discussion, the team discovered that the most effective way to assess the need statement is to continue with the initial proposal but with further analysis.

3.5 Project Approach

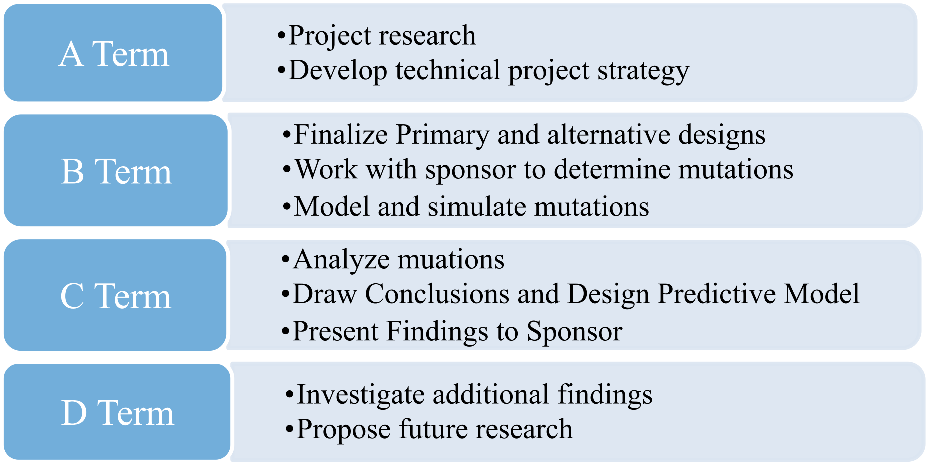
In order to ensure project objectives are met within the given constraints, the team developed a project strategy broken down into a technical, management and financial approach.

3.5.1 Technical Approach

The team is considering using molecular dynamics to analyze inhibitor interactions, however will also develop alternative designs presented in chapter 4 using traditional “wet lab” analysis. After determining the best form of analysis, mutations will be decided and data collected. From the data gained through experimentation, a software program will be written to effectively analyze and display results. Conclusions will be drawn to reach the project goal.

3.5.2 Management Approach

In order to complete the project within the specified time constraint, the team developed a Gantt chart to direct a course of action over the project timeline (refer to Appendix A). This chart breaks down each major project component into smaller tasks, to ensure milestones and deliverables are met. Task times were generously estimated to ensure the team adheres to the proposed timeline. Additionally, tasks and project work were front loaded to provide flexibility, should a research obstacle arise. To use time most effectively, certain team members were allocated and assigned to complete specific tasks. Tasks were assigned based upon the individual strengths each member added to the project group. In addition to the Gantt chart, the team created a project plan summary for each term (Fig. 3.2).



A Term
- Project research
- Develop technical project strategy

B Term
- Finalize Primary and alternative designs
- Work with sponsor to determine mutations
- Model and simulate mutations

C Term
- Analyze muations
- Draw Conclusions and Design Predictive Model
- Present Findings to Sponsor

D Term
- Investigate additional findings
- Propose future research

Figure 3. 3: Project Management Plan by Term

3.5.3 Financial Approach

As with any design project, financial constraints must be taken into consideration. This will not be a limiting factor in our project, as the team will be conducting research and theoretical design. Should the project require any financial support, the team will mostly rely on resources available in the Schiffer Laboratory at the University of Massachusetts Medical School.

4.0 Design Process

The preliminary developments of the project design are presented in this chapter. This includes a summary of design needs, functions and specifications. Additionally, the conceptual and alternative designs are explored and evaluated.

4.1 Needs Analysis

As the project evolved through meetings with the team and project sponsors, needs were identified and evaluated. We classified objectives to distinguish between functional needs that are required for the project and desirable needs that are not crucial to the design (Table 4.1).

Table 4. 1: Project Needs Classification

Functional Needs	Desirable Needs
Accuracy of simulation	Repeatability of simulation replicates
Adaptable analysis for multiple mutations	Measurable Alpha Carbon Distances
Measurable RMSF, van der Waals interactions, and hydrogen bonding	Comparison of different background environments

The three most important objectives determined by the team are accuracy, adaptability, and measurability. Based off these objectives, the functional needs deemed necessary to the project design are that the simulation is an accurate portrayal of *in vivo* behavior, the software analysis program is adaptable for various mutations, and that Root-Mean-Squared Fluctuation (RMSF), van der Waal interactions and hydrogen bonds can be measured.

The first functional requirement of the project design is the simulation must accurately depict *in vivo* conditions, including environmental parameters, protein conformation, and conserved and bonded waters. Accuracy of the simulation is imperative in analyzing the inhibitor effectiveness. Secondly, the analysis program must be designed to be adapted to various mutations beyond the three mutations the team is evaluating. Variants the program can analyze must range from single point mutations to a combination of n-point mutations and must be accurate and consistent across all scenarios. The final functional requirement requires the analysis program to be designed to include accurate measures of protein movement and energy. This can specifically be defined as RMSD, RMSF for the protein and ligand, van der Waals interactions and hydrogen bond percentages.

Repeatability, C-Alpha distance measurements, as well as comparing different background environments were categorized as desirable needs. Due to the dynamic nature of HIV-1 protease and flap movement, slight variations in replicate data, especially in protein and ligand RMSF data, are to be expected. However, accurate simulation setup can mitigate any large discrepancies between replicates making our design more consistent.

Secondly, C-Alpha Distance measurements provide additional information about the protein and ligand behavior, although are not essential for the scope of our analysis. More pertinent information can be gained through examining RMSF, van der Waals and hydrogen bonds, which are of interest to our sponsor. Similar to C-Alpha Distance data, comparing mutation data across different backgrounds would be beneficial to observe any different inhibitor interactions.

4.2 Conceptual Designs

To meet the project's functional and desirable needs, the team will follow a set of steps to achieve the analysis of the HIV-1 protease mutations' affect on the binding of the inhibitor. These steps are shown in Figure 4.1.

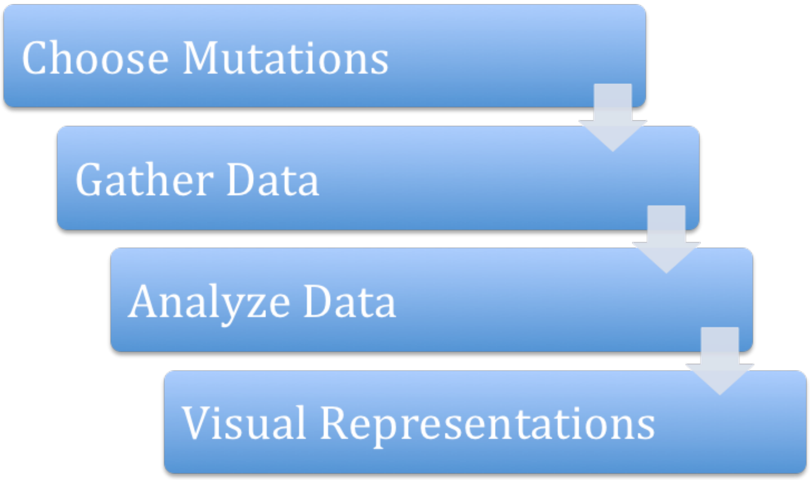


Figure 4. 1: Conceptual Project Design

Firstly, the team will choose at least three different mutations to analyze and compare to wild type. To test the hypothesis of whether there is increased resistance conferred with increased mutations, we wanted to choose mutations that are compounded (following the pattern x, x+y, and x+y+z). Once the mutations are chosen, the team will determine a process to effectively gather large amounts of data. These data include time-dependent protein dynamic behavior and inhibitor interactions. An important consideration in developing this process is the number of replicates to achieve accurate and sufficient data. Lastly, the team will team will develop analysis programs to process the information collected. This is comprised of calculating certain measurements, including RMSD, RMSF, van der Waals, hydrogen bonds, and alpha carbon distances across the active site, and comparing those measurements to wild type and other mutation variants. These scripts will make this process more efficient and effective, and provide visual representations.

4.3 Alternative Designs

Since there are a variety of techniques to analyze protein functionality and interactions, there are several approaches the team is considering to approach our project problem. Conventional methods of protein analysis are facilitated through "wet" lab practices that give either direct or indirect information about the protein of interest. *In vitro* analyses that can detect protein functionality include protein purification and protein detecting. However, these practices are not specific enough to determine the finest details involved in drug resistance development since they are limited to detecting only the highest level of protein complexity (tertiary and quaternary structures).

The recent growth of technology allowed for the use of computational analysis in protein analysis. Information can span from the quaternary structure down to individual atoms. Currently, computational analysis methods include protein structure prediction, protein sequence/structural alignment, and molecular dynamics. Each of these techniques provides valuable information about protein functionality, but molecular dynamics is the only method that provides real-time analysis of protein-ligand interactions.

The team brainstormed a list of attributes that are necessary to our project and determined which method, computational or wet lab analysis would suit the project needs. These attributes are listed in Table 4.2 and the methods more suitable to meet each attribute are marked with an x.

Table 4. 2: Comparison of Computational and Wet Lab Analysis

Attributes	Computational Analysis	"Wet" Lab Analysis
Subtle Specificity	X	
Ideal Environmental Assumptions	X	
Cost-Effectiveness	X	
Amount of Data	X	

There are many open-source and private molecular dynamic analysis software available for use. However, private software is normally the recommended since this analysis requires super-computers to conduct the massive amounts of computational analysis. Although, the team had accessibility to the Desmond molecular dynamic software additional software were researched such as Abalone, ADUN, Amber, COSMOS, CP2K, and Culgi.

Within Schroedinger's Maestro software are many selection parameters that need to be properly accounted for to obtain accurate data for our protein analysis. Firstly, the proper force field must be selected in order to subject our protein-ligand complex to mimic in vivo environments. These force fields can be broken up into classical, polarizable, reactive, and coarsed-grained. For the contexts of this analysis, the classical family of force fields was best suited due to its specific effects related to proteins. This family is further broken down into MMFF, CHARMM, Amber, and OPLS, which are all force fields specific to proteins and protein compositions. OPLS is only force field that incorporates the proper environment that accounts for proteins, small molecules, nucleic acid, and lipids. Another important parameter to consider is the time span of simulation. Typically molecular dynamic simulations are conducted in the nanosecond range. A simulation that is too short can result in incomplete data collection while a simulation that is too long can provide a less accurate representation of events occurring during the simulation. The Schiffer Lab has found through trial and error, that 100 nanosecond simulation provides an adequate amount of time to provide accurate data. Additionally, the team had to decide on whether to focus on active vs. non-active site mutations. It is well documented that mutations made within the protein's binding pocket have a higher probability of affecting inhibitor binding dynamics. However, mutations found outside of the active side can alter the overall tertiary structure of the protein during protein folding. Finally, these simulations can be run on either a CPU or a GPU. The major difference between the two is that simulations run on a GPU are 30-80% faster than a CPU. The use of a GPU for simulations is the most logical choice since time was a major constraint for this project's completion.

In order to analyze the data from the molecular dynamic simulations, the team will write software scripts. Common languages utilized for biological analysis are python and C. Additionally, the team considered the use of MATLAB for the analysis process. A list of strengths and weakness for each programing language is shown below in table 4.3.

Table 4. 3: Comparison of Programming Languages

Programming Language	Pros	Cons
Python	• Easy to to Use • Modular (Biopython) • Able to Read PDBs	• Modules are separate downloads • Difficulty handling large files
C	• C Library • Built in functions	• No namespace • No run time checking • Team Unfamiliarity
Matlab	• Easy to to Use • Nice Visuals • Statistical Built-In Models • Visual Workspace	• Difficulty importing PBD’s

4.4 Final Design

After evaluating design alternatives and comparing the benefits and limitations of each option, the team decided on the final design. The first design decision was to use molecular dynamics simulation software to observe inhibitor behavior and conferred drug resistance. Molecular dynamics (MD) was chosen, as it is the industry standard to compute time dependent behavior of molecular systems, and is frequently used in the Schiffer laboratory. The key benefit of MD is it allows for the modeling of complex dynamic processes including protein stability and conformational changes. This will ensure the functional need of accuracy is met. From molecular dynamics, a study on drug interactions can be completed, meeting the project client statement of determining DRV effectiveness in the presence of various mutations.

To conduct the MD simulations, the team chose to utilize Desmond. Desmond software analyzes models created by Maestro, an all-purpose molecular modeling environment manufactured by Schroedinger. In addition to having access to the Maestro software through the Schiffer laboratory, Schroedinger is a leading MD software providing advanced algorithms and customizable features. This allows for the protease amino acid residues to be replaced with the mutation of interest.

When determining mutations to analyze, the team considered both active and non-active site mutations. Although shown to affect drug resistance, the mechanism by which non-active site mutations confer drug resistance is unknown. For this reason, the team chose to analyze active site mutations, which affects the conformation of the enzymatic binding pocket and impacts inhibitor binding. The chosen point mutations are: I84V, where residue 84, isoleucine, is replaced with valine for both chains, V82F+I84V, where valine is replaced with phenylalanine, and M46I+V82F+I84V, with methionine replaced with isoleucine. These mutations were found through previous *in vitro* viral passaging conducted by laboratories in collaboration with the Schiffer Lab.

The final design decision the team made was the coding language used for the analysis program. For the majority of scripts, MATLAB was chosen because of its superior visual and graphing capabilities as well as its ability to import large data files. The resulting data files range in size from hundreds to thousands of data points, and the team decided MATLAB was the most capable and versatile program. The versatility of MATLAB would lend well to importing and reading various file types and sizes, while running at a fast speed. Further, the lab has MATLAB access and our design can be incorporated and supplement existing scripts that are across several coding languages. However, the team recognized the limitation with MATLAB and decided to use python for one script to modify beta values in a PDB file in order to generate heat maps for visual analysis.

Once all design decisions were made, the team conducted a preliminary test of a single mutant variant that had previously been analyzed by the lab. The molecular dynamic simulation provided data consistent with past data, so the team continued with simulations and analysis of the I84V, V82F+I84V, and M46I+V82F+I84V variants.

5.0 Design Verification

This chapter presents the results of each experiment performed modeling the HIV-1 protease wild type, mutant variants I84V, V82F+I84V, and M46I+ V82F+I84V. The results include protein RMSD, protein RMSF, ligand RMSF, alpha carbon distances, van der Waals interactions, and hydrogen bonds.

5.1 Protein RMSD

The protein root-mean-squared deviation (RMSD) was calculated for the 100 nanosecond simulation that was conducted for each replicate of each variant. RMSD analysis determines the extent of protein equilibration through molecular dynamic simulation. This information is used to verify proper protein behavior during the simulation. Additionally, the moving average of the RMSD was calculated for each variant. Figure 5.1 shows the protein RMSD and their moving average for each replicate of each variant. The replicate data is shown in blue and the moving average in red.

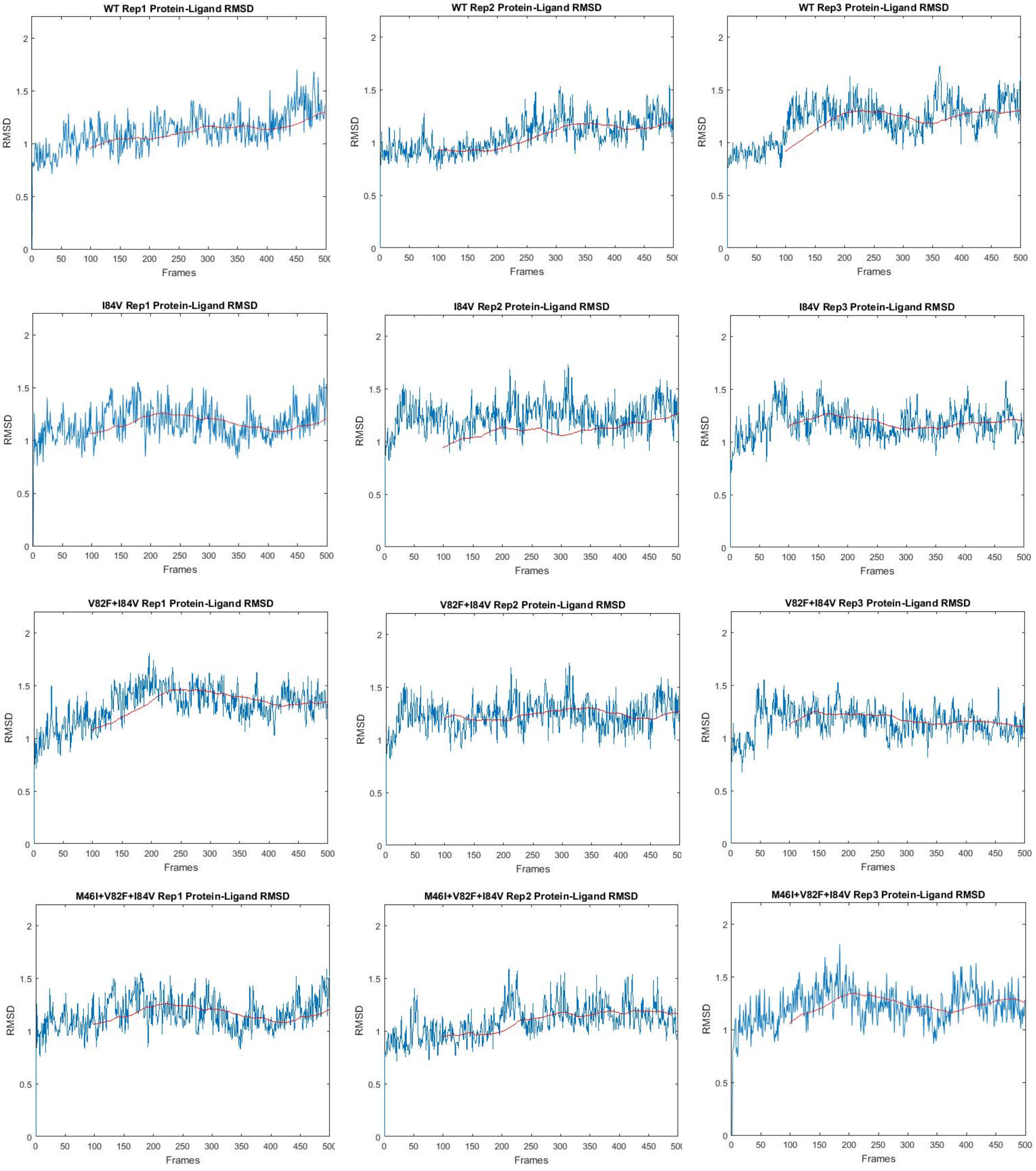


Figure 5. 1: Protein RMSD replicates 1-3 of WT, I84V, V82F+I84V, and M46I+V82F+I84V

The average RMSD of the three replicates were calculated for each variant. The averages of the RMSD were then compared to wild type to observe any possible differences from the wild type simulations (Fig. 5.2). The replicate RMSD data is shown in blue and the moving average in red.

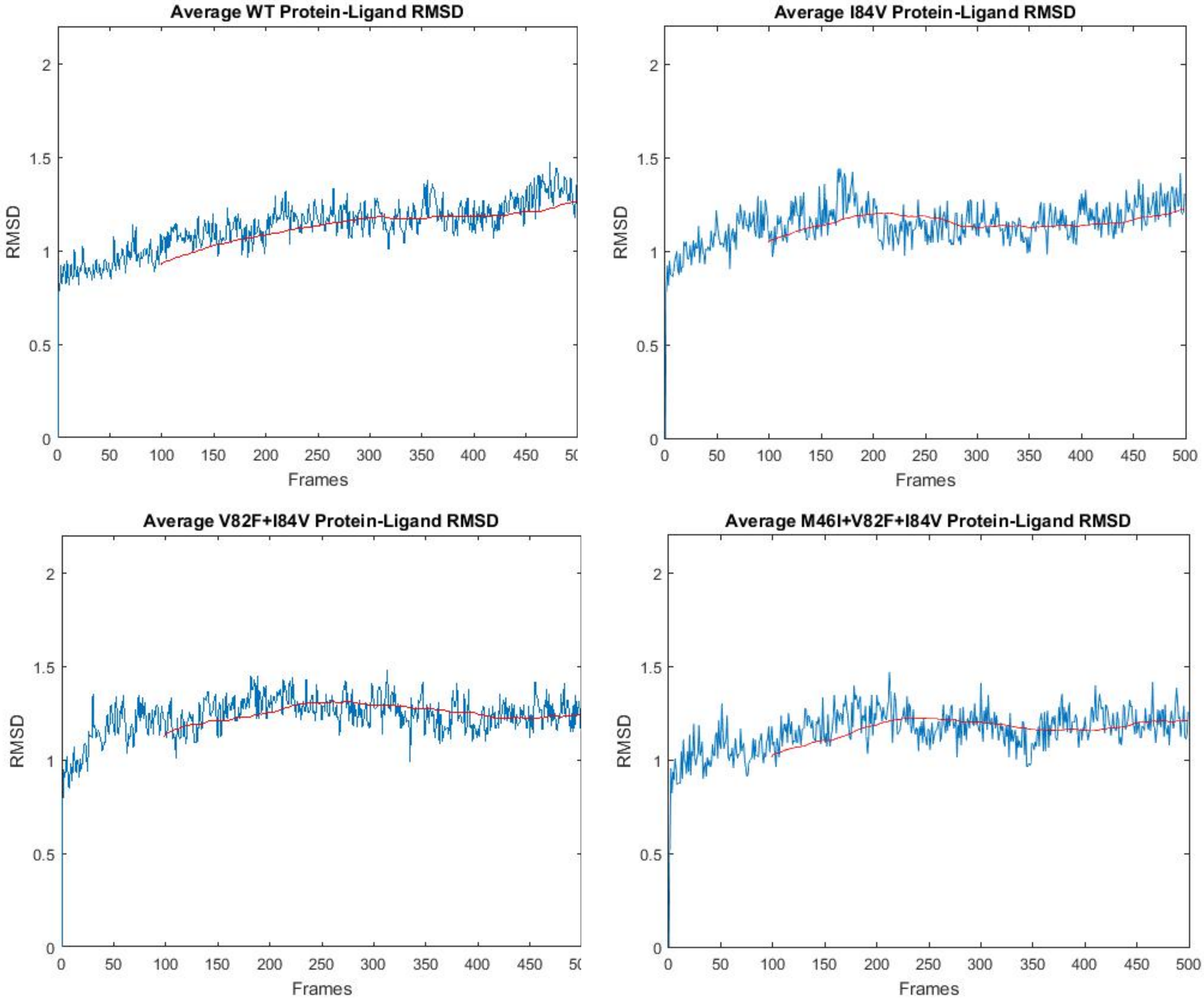


Figure 5. 2: Average protein RMSD of WT, I84V, V82F+I84V, and M46I+V82F+I84V

A constant RMSD suggests an equilibrated (temperature, pressure, etc.) model of the protein. The jump to the constant position represents the equilibration process to the required temperature and environmental conditions. Overall, the average protein-ligand RMSD plots showed proper equilibration to roughly 1.25 angstroms. Further protein analysis could be conducted since the protein's behavior appropriate.

5.2 Protein RMSF

The protein root-mean-squared fluctuation (RMSF) of each residue was calculated for each variant during their 100 nanosecond simulations. Protein RMSF describes the amount of fluctuation, or movement, of protein residues. Three replicates were conducted for each variant and the protein RMSF was calculated.

The following plots show the protein RMSF for each mutation and wild type simulations (Fig 5.3). The line graphs contain the replicate data with the red, green, and blue lines corresponding to

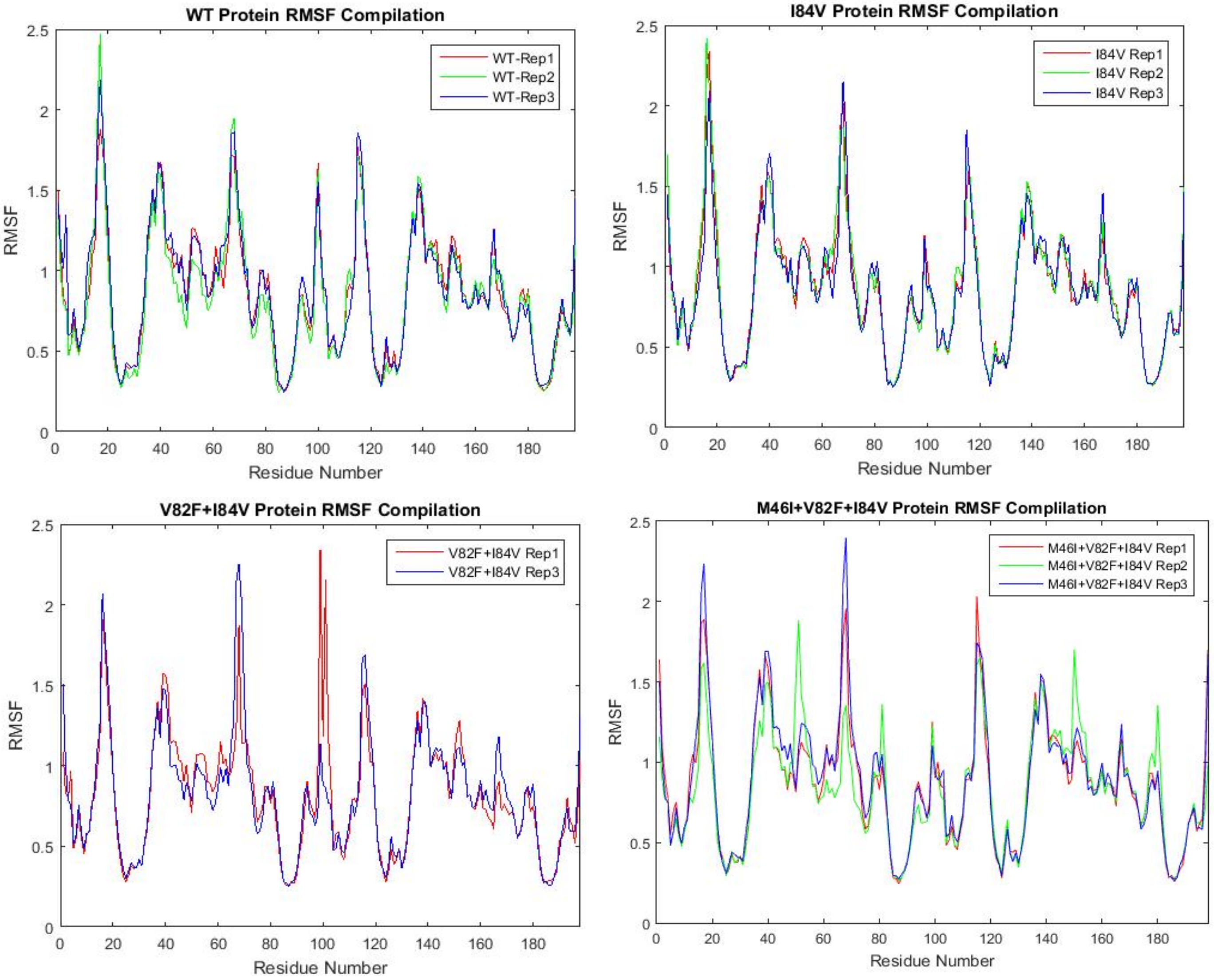


Figure 5. 3: Protein RMSF Compilation of WT, I84V, V82F+I84V, and M46I+V82F+I84V

replicates 1, 2, and 3, respectively.

The average RMSF values of the three replicates were calculated and graphed on a single graph. This was done to highlight any differences from the average mutant RMSF's to wild type. HIV-1 wild type protease is depicted by the solid black line (Fig 5.4). Average protease variants I84V, V82F+I84V, and M46I+V82F+I84V are identified by red, blue, and green, respectively.

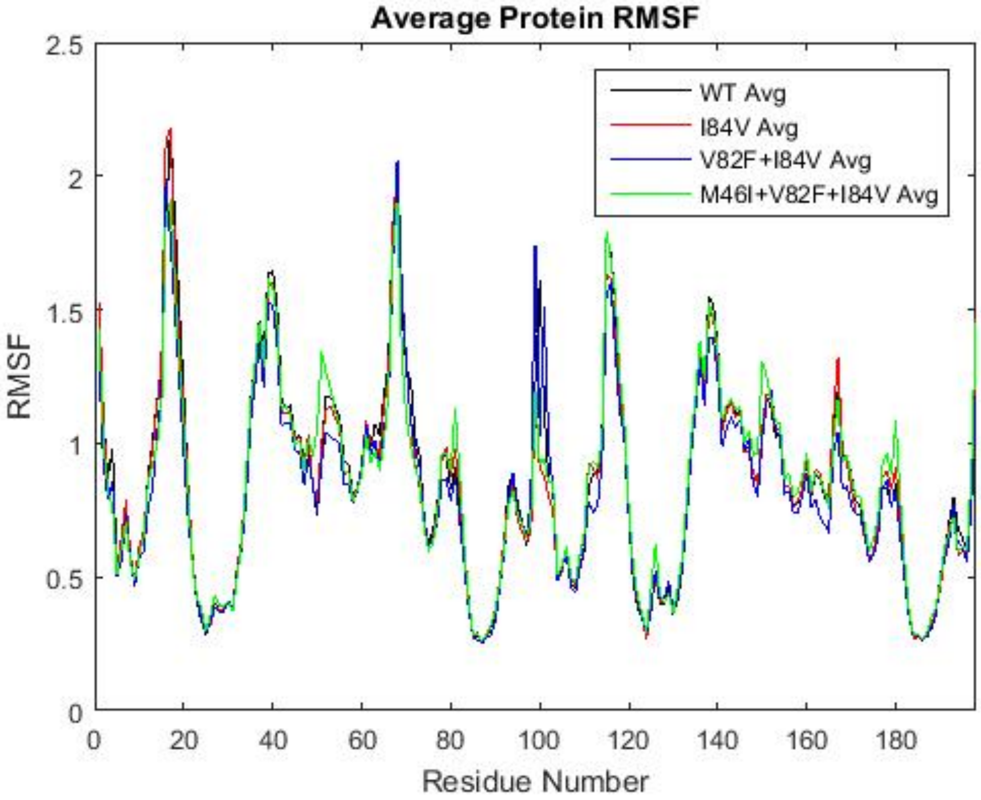


Figure 5. 4: Average Protein RMSF values for WT, I84V, V82F+I84V, and M46I+V82F+I84V

The average protein RMSF compilation of each mutant and wild type showed notable differences. Firstly, the there were notable differences in fluctuation across each variant around residues 1, 99, and 199. This is to be expected since this is the protein's dimerization region which is extremely motile. Overall RMSF values remained similar except for key regions such as the active site, flaps, and elbow regions. This line graph representation of protein RMSF makes it difficult to detect subtle changes between variants.

The average protein RMSF values of each mutation were subtracted from the wild type averages to observe fluctuations caused by each mutation. This data was displayed in a bar plot format to improve visualization of the data fluctuation. Additionally, the average protein RMSF differences were split into separate plots for chain A and B to increase the visual size allowing for easier detection of significant changes. Protease variant differences of I84V, V82F+I84V, and M46I+V82F+I84V are identified by red, blue, and green, respectively (Fig. 5.5).

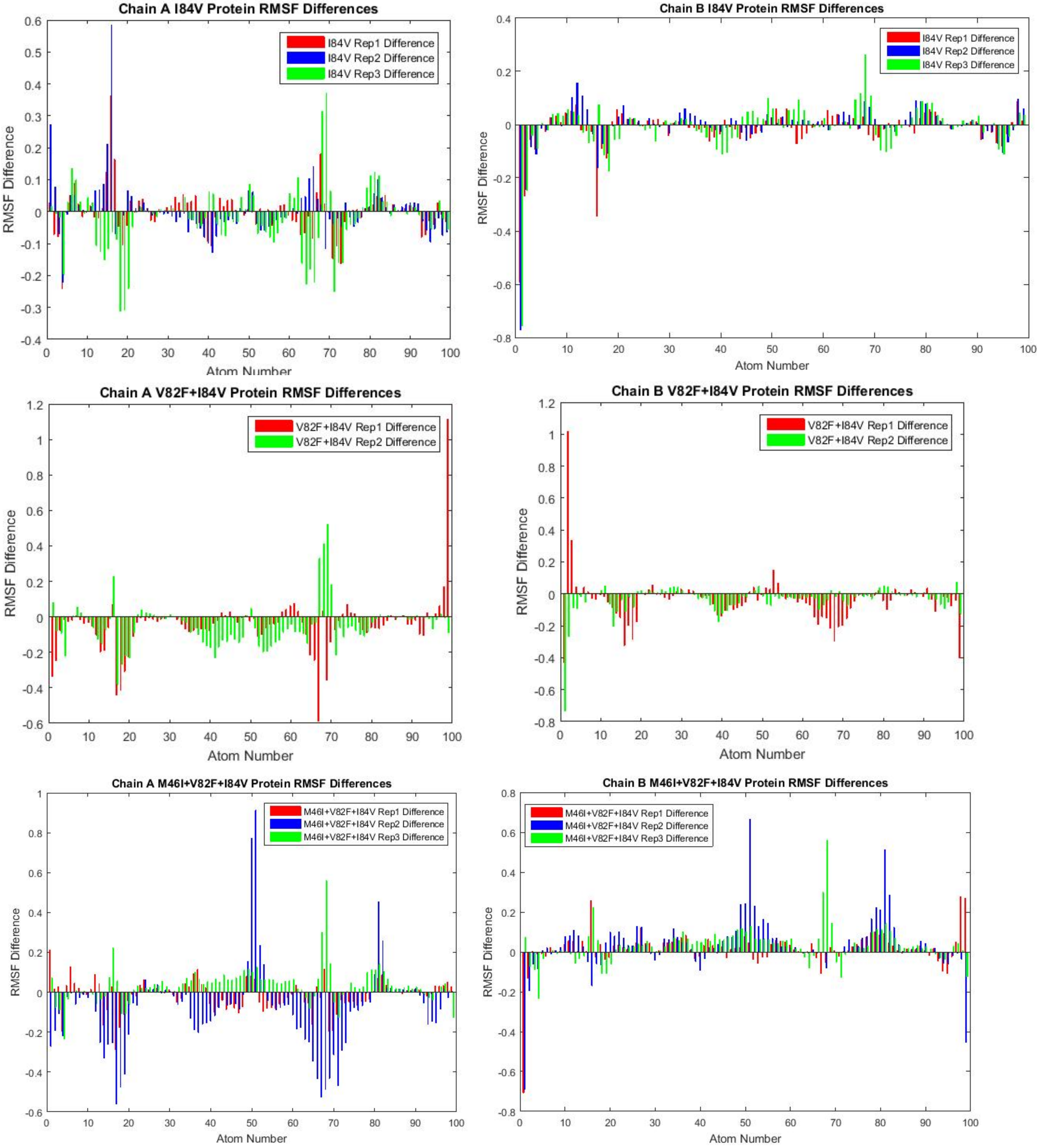


Figure 5. 5: Differences Compared to WT protein RMSF

In the case of protein RMSF differences, several residues exhibit subtle differences to the wild type that can be considered negligible. To account for this, an additional plot showing the significant differences was created. Significant differences are shown below, defined as the difference from the wild type value at each residue is larger than the standard deviation of three wild type values at that residue (Fig. 5.6).

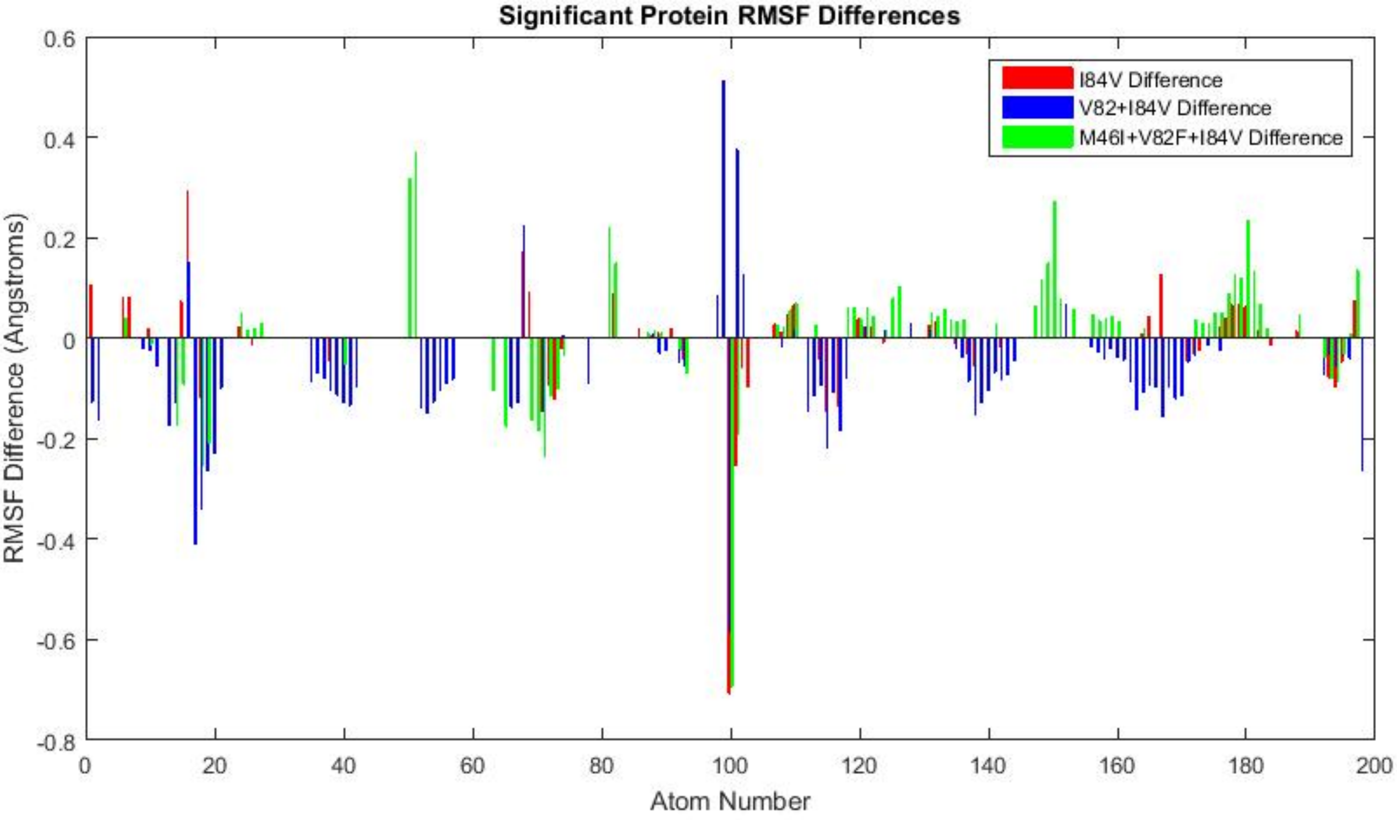


Figure 5. 6: Significant Differences Compared to WT Protein RMSF

The most significant fluctuations for each variant occurred around the dimerization region (residue 99). V82F+I84V also displayed significant differences around residues 19, 40, 55,118,142,and 175. This variant had the greatest amount of variation when compared to wild type.

Additionally, protein RMSF differences to wild type were also displayed in the form of heat maps, generated using PyMol. These heat maps display the protein structure as a cartoon ribbon. Dark blue shading displays residues with the least fluctuation, greatest stability, and span to red representing residues of highest fluctuations, least stability. Figure 5.7 depicts the protease protein colored by RMSF values ranging from -0.3643 to 0.7060 compared to Figure 5.8, which shows the RMSF difference to wild type with difference values ranging from 0.2506 to 2.1776.

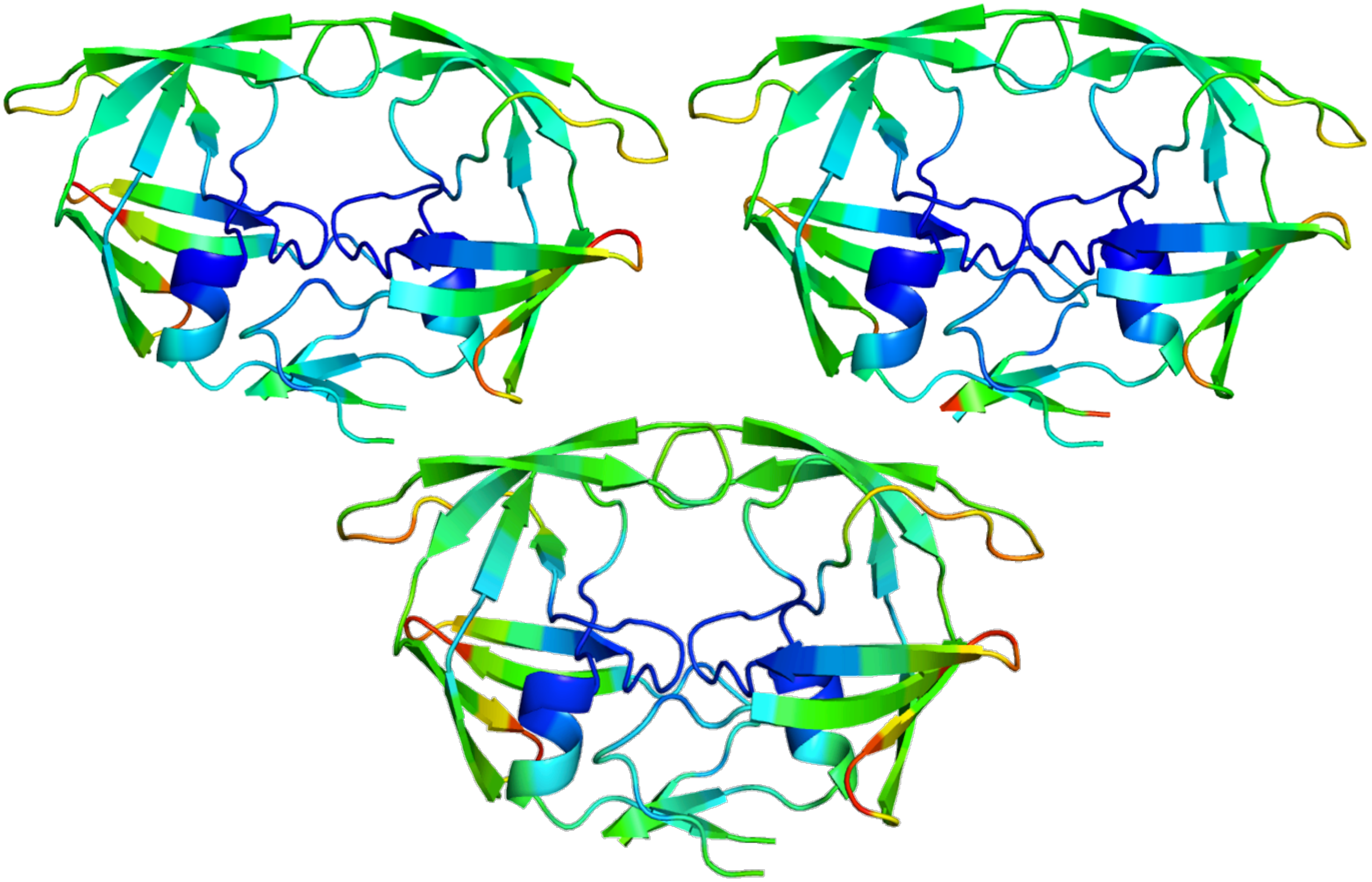


Figure 5. 7: Protein RMSF Heat Maps for I84V (top left), V82F+I84V (top right), and M46I+V82F+I84V (bottom)

Protein RMSF heat maps in figure 5.7 displays the information shown in Figure 5.4 with respect to the protein's structure. RMSF values of each residue were visually displayed in their correct protein primary structure. Additionally, a color scale was correlated to these RMSF values to display fluctuation in terms of a color. This display allows for a 3-D representation of the protein capable of visualizing regions of fluctuation.

Key structural and functional regions of the protease were colored blue, indicating the most stable (Fig. 5.7). Specifically the alpha helices and the active site for each variant exhibited the highest stability. Regions of moderate to high fluctuation throughout the simulation included the flaps and 60’s loop regions. I84V and M46I+V82F+I84V variants displayed small amounts of green and a significant amount of yellow and red. However, V82F+I84V just showed moderate fluctuation in the 60’s loop being strictly green. I84V and V82F+I84V variants displayed moderate fluctuation within the elbow regions, shown as yellow. The M46I+V82F+I84V variant showed an increase in fluctuation in its

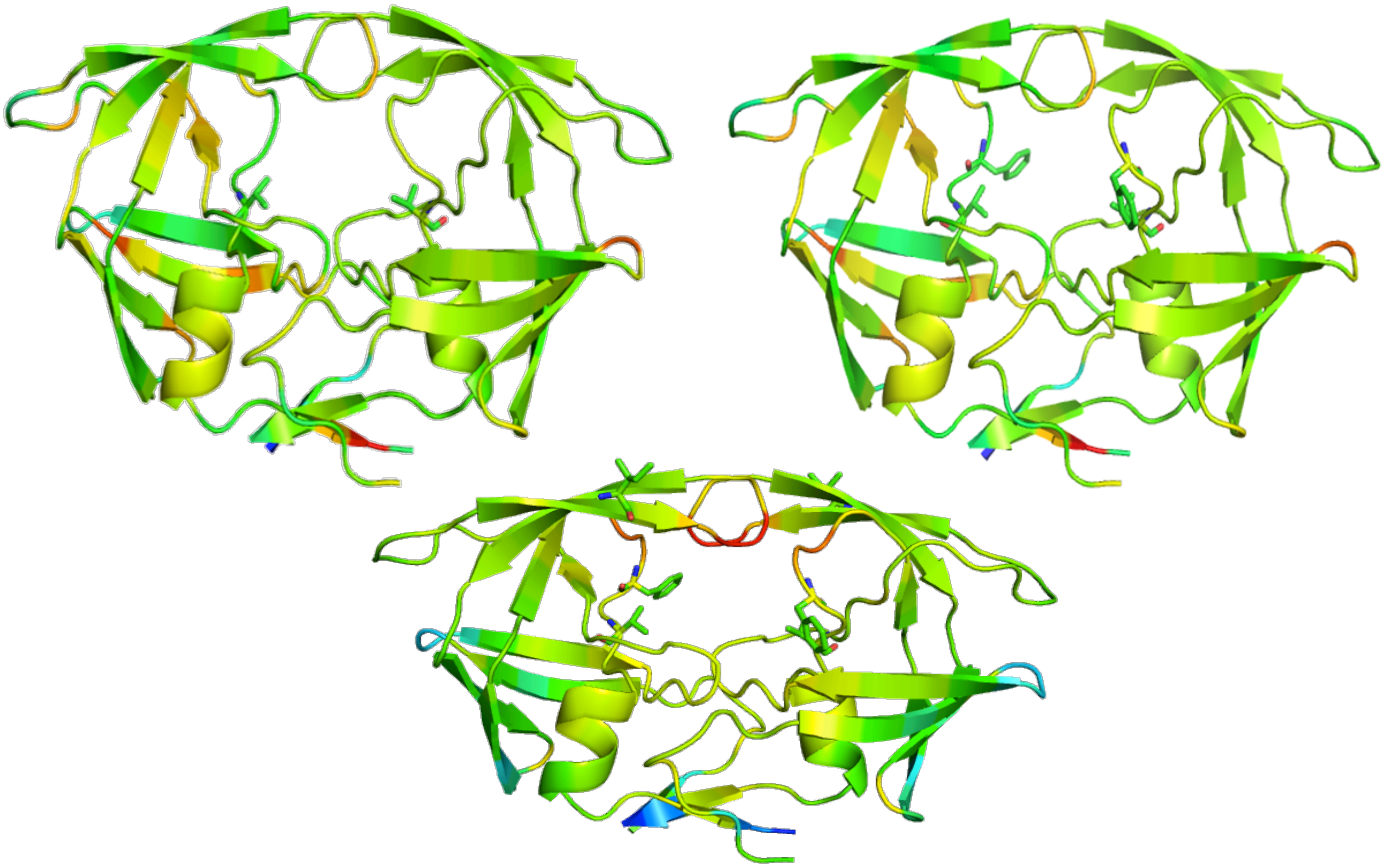


Figure 5. 8: Protein RMSF Differences Compared to WT Heat Maps for I84V (top left), V82F+I84V (top right), and M46I+V82F+I84V (bottom)

Similarly, Figure 5.8 displays protein RMSF differences compared to wild type on the protein structure. These representations allow for easier differentiation of fluctuation significance between variants to be observed. Regions in green represent the most conserved areas with fluctuation consistent with wild type. I84V and V82F+I84V variants had very similar color gradients with the V82F+I84V variant having slightly more yellow identifying slight increases in fluctuation compared to wild type. However, the M46I+V82F+I84V variant displayed a great amount of fluctuation compared to wild type.

The active site showed moderate (yellow) fluctuation compared to both I84V and V82F+I84V (green). Also, the M46I+V82F+I84V variant displayed less fluctuation (light blue) compared to both I84V and V82F+I84V (green) in the 60's loop region. Finally, the most significant fluctuation occurred in the tip of the flaps (red) compared to both I84V and V82F+I84V (green/yellow).

5.3 Ligand RMSF

The ligand root-mean-squared fluctuation (RMSF) of each residue was calculated for each variant during 100 nanosecond simulations. Ligand RMSF describes the amount of fluctuation, or movement, of ligand atoms. Three replicates were executed for each variant and the ligand RMSF was calculated for each one. The following plots show the ligand RMSF for each mutation and wild type simulations. The line graphs contain the replicate data with the red, green, and blue lines corresponding to replicates 1, 2, and 3, respectively (Fig. 9).

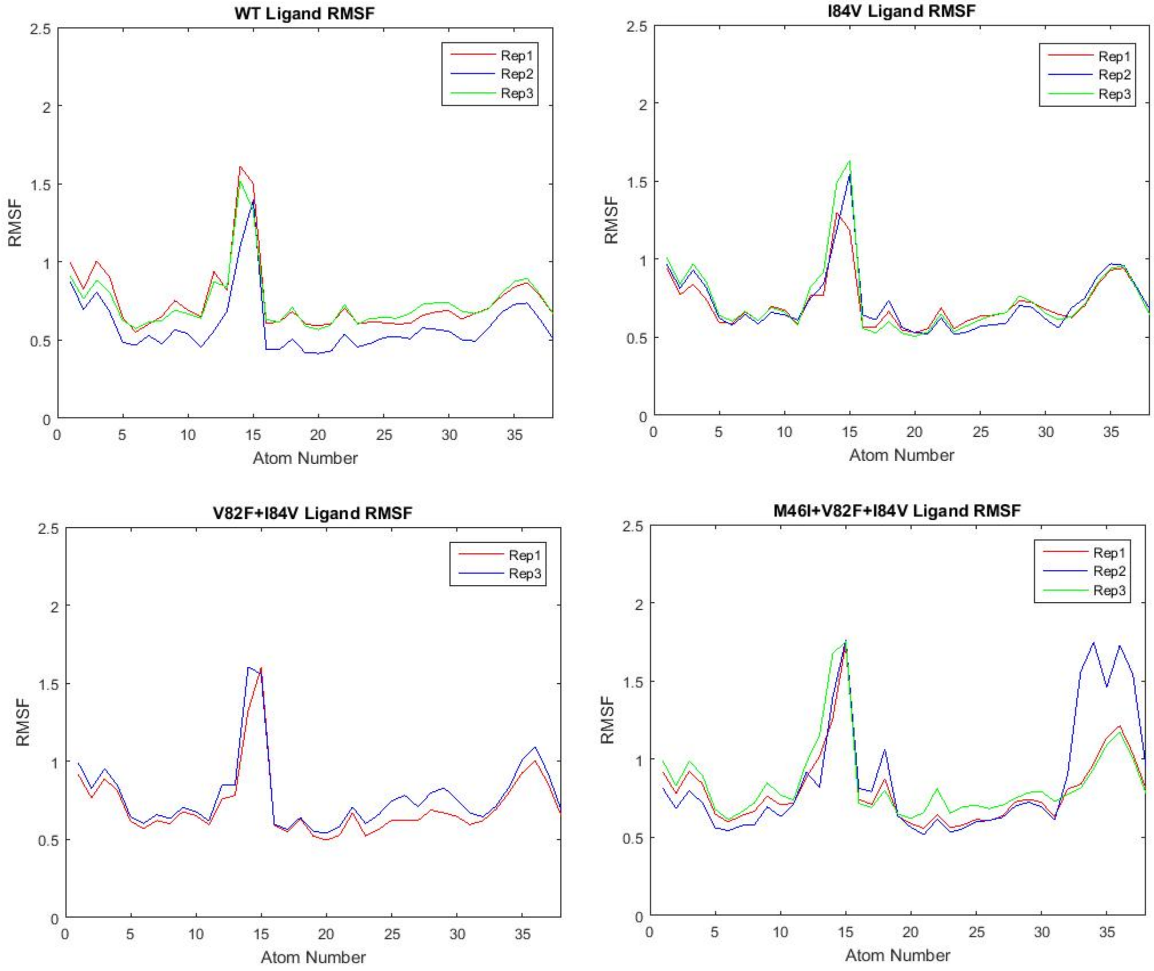


Figure 5. 9: Ligand RMSF Compilation of WT, I84V, V82F+I84V, and M46I+V82F+I84V

The average RMSF values of the three replicates were calculated and graphed on a single graph. This was done to highlight any differences from the average mutant RMSF's to wild type. The ligand wild type is depicted by the solid black line (Fig 5.10). Average ligand variants I84V, V82F+I84V, and M46I+V82F+I84V are identified by red, blue, and green, respectively.

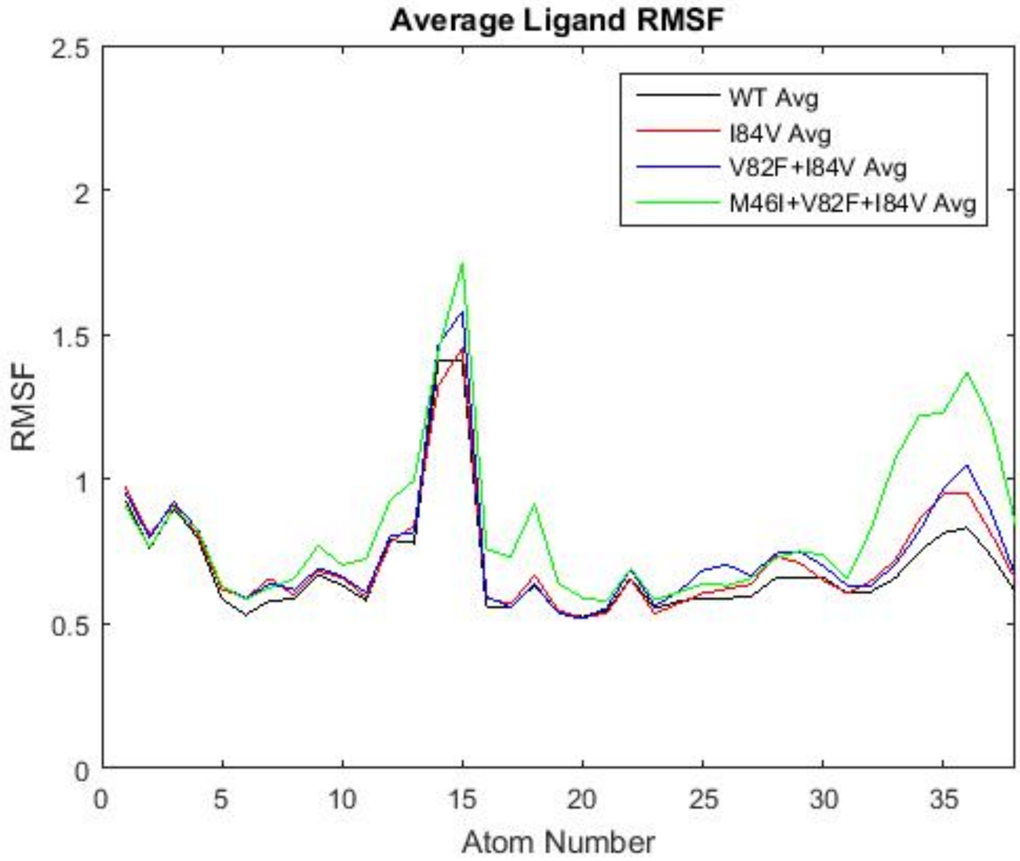
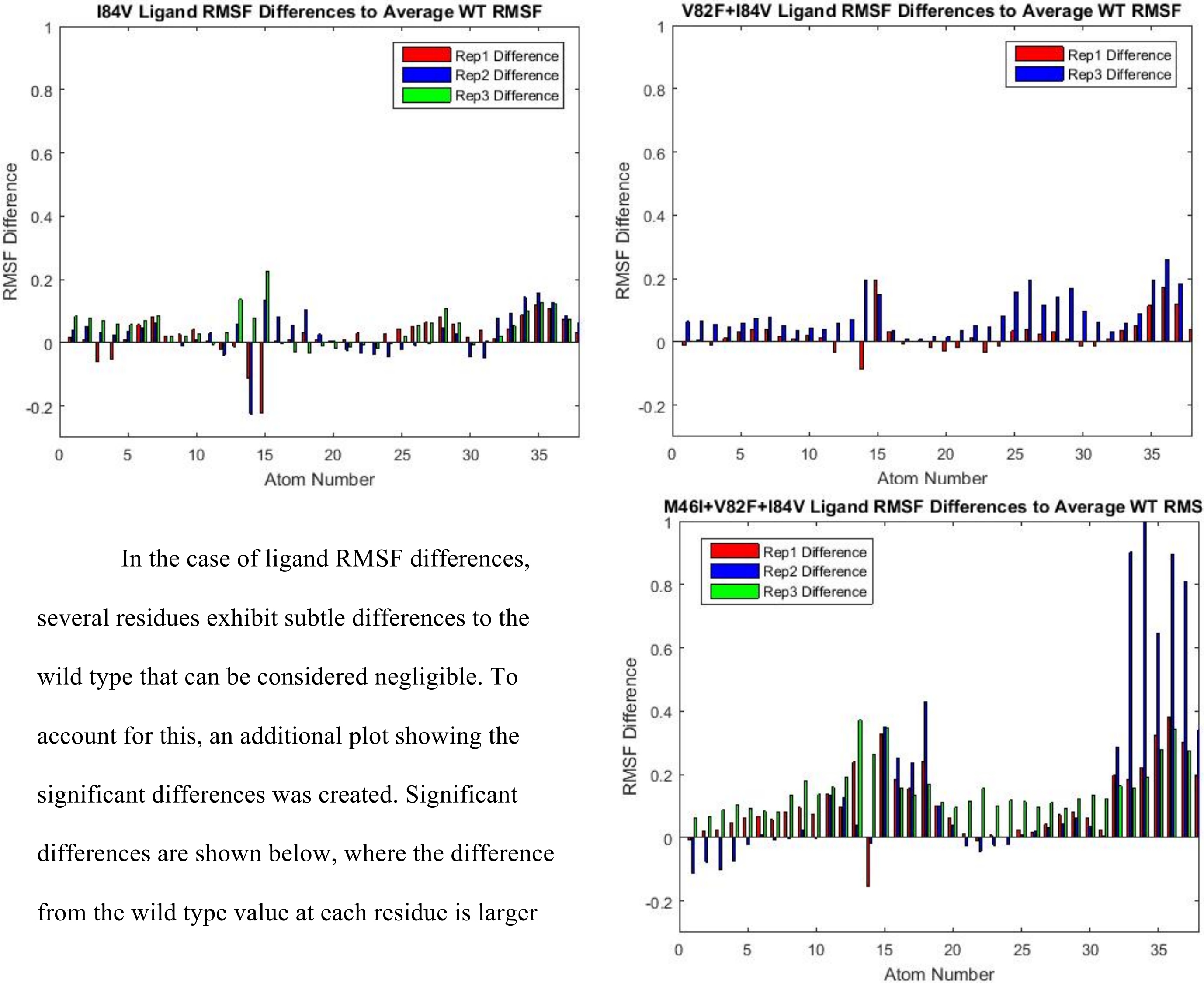


Figure 5. 10: Ligand RMSF Compilation of WT, I84V, V82F+I84V, and M46I+V82F+I84V

The average ligand RMSF compilation of each mutant and wild type showed limited differences. The line graphs for average wild type, I84V, and V82F+I84V ligand RMSF were similar. Each of these graphs followed the same trends with slight variations from wild type near residues 26, 28, and 35. However, the M46I+V82F+I84V variant follows a similar trend with substantially more significant differences around resides 7, 11, 13, 17, and 31-38.

The average ligand RMSF values of each mutation were subtracted from the wild type averages to observe fluctuations caused by each mutation. This data was displayed in a bar plot format to improve visualization the fluctuation data. Ligand mutant variant differences I84V, V82F+I84V, and M46I+V82F+I84V are identified by red, blue, and green, respectively (Fig. 5.11).



In the case of ligand RMSF differences, several residues exhibit subtle differences to the wild type that can be considered negligible. To account for this, an additional plot showing the significant differences was created. Significant differences are shown below, where the difference from the wild type value at each residue is larger than the standard deviation of three wild type values at that residue (Fig. 5.12).

Figure 5. 11: Ligand RMSF Differences Compared to WT for I84V, V82F+I84V, and M46I+V82F+I84V

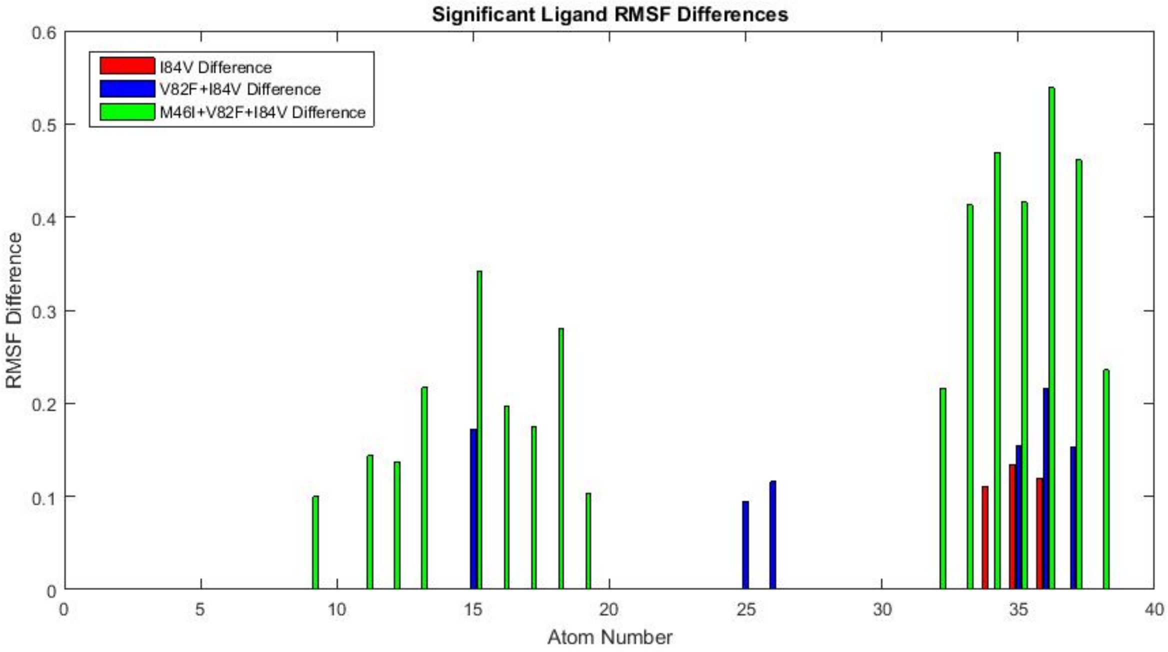


Figure 5. 12: Significant Ligand RMSF Differences Compared to WT for I84V, V82F+I84V, and M46I+V82F+I84V

The significant ligand RMSF differences to wild type bar plot provided a better visualization of this analysis. Similar to Figure 5.11, the M46I+V82F+I84V variant clearly displays the most fluctuation. This is observed mostly in atoms 32 through 38, corresponding to the benzene ring. There was minimal fluctuation observed in the cyclopentadiene region, with the only significant change in RMSF occurring in atoms 25 and 26 when complexed to V82F+I84V.

5.4 Alpha Carbon Distances

Distances between the alpha-carbons (C-Alpha) of residues across the active site were recorded during the 100 nanosecond simulation for each HIV-1 protease variant. These distances determine the relative size of the active site and give insight on the dynamic movement of the protease. The residues of interest were 25-25’, 84-84’, 25-50, 25-50’, 25’-50, and 25’-50’, shown as red lines in the Figure 5.13 below. The apostrophe following residue values signifies chain B residues and the red lines indicate the initial position of these alpha carbon pairings.

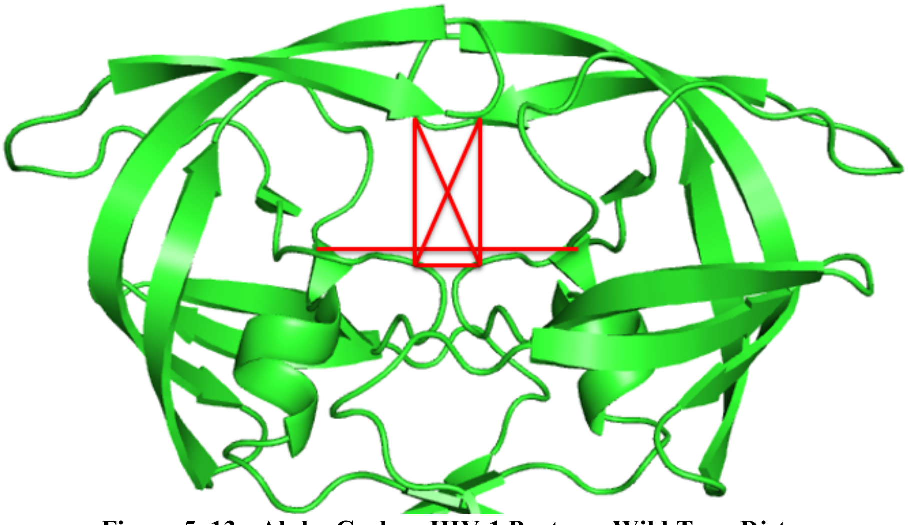


Figure 5. 13: Alpha Carbon HIV-1 Protease Wild Type Distances

The C-alpha distances of WT for each replicate are shown in Figure 5.14. The distances of the first row are between 25-25’, 84-84’, 25-50 (left to right) and the second row are between 25-50’, 25’-50’, and 25’-50 (left to right). Replicate 1 is shown in blue, Replicate 2 is shown in red, and Replicate 3 is shown in yellow.

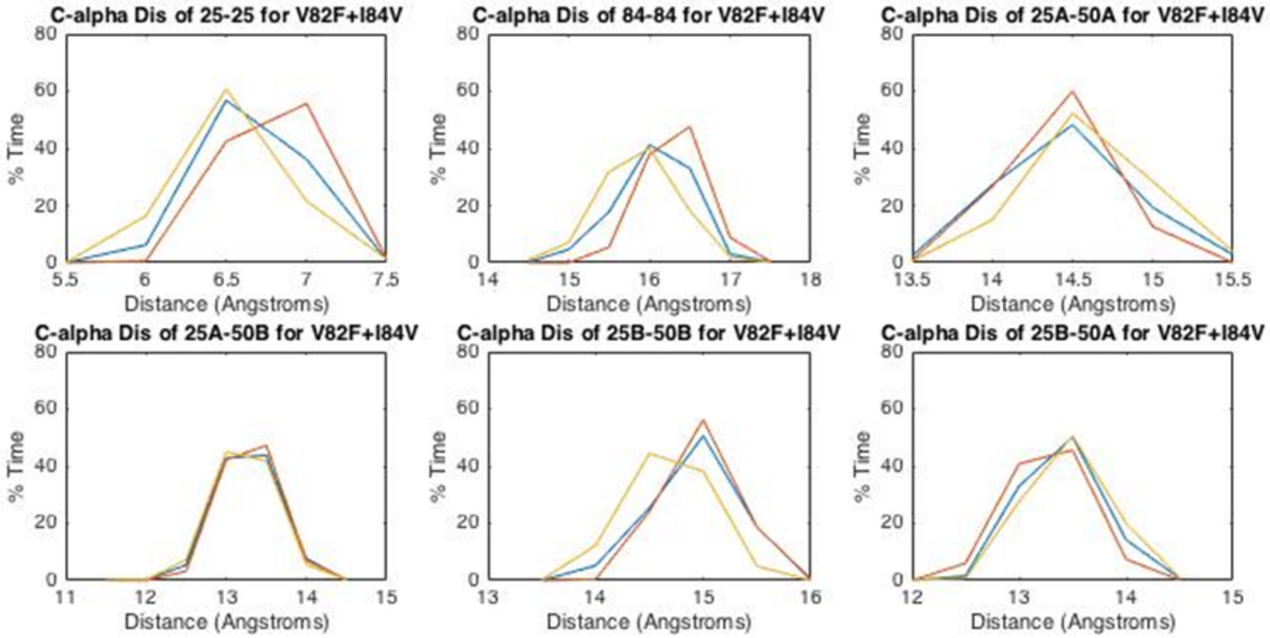


Figure 5. 14: Wild Type Alpha Carbon Distances

The C-alpha distances for I84V, V82F+I84V, and M46I+V82F+I84V were also calculated for each replicate (Fig. 5.15, Fig. 5.16, and Fig 5.17).

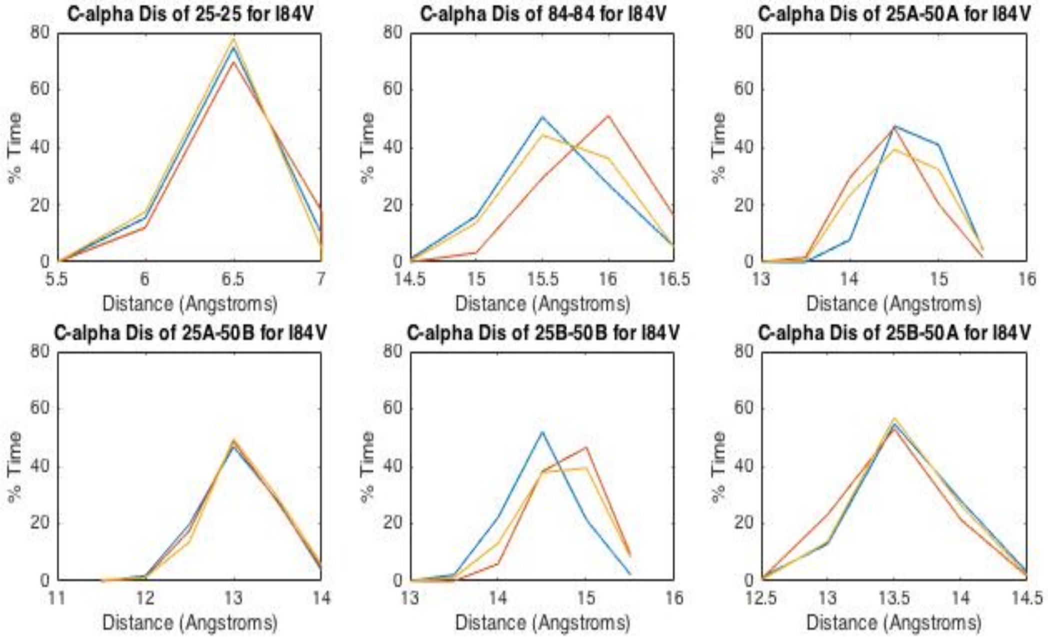


Figure 5. 15: I84V Alpha Carbon Distances

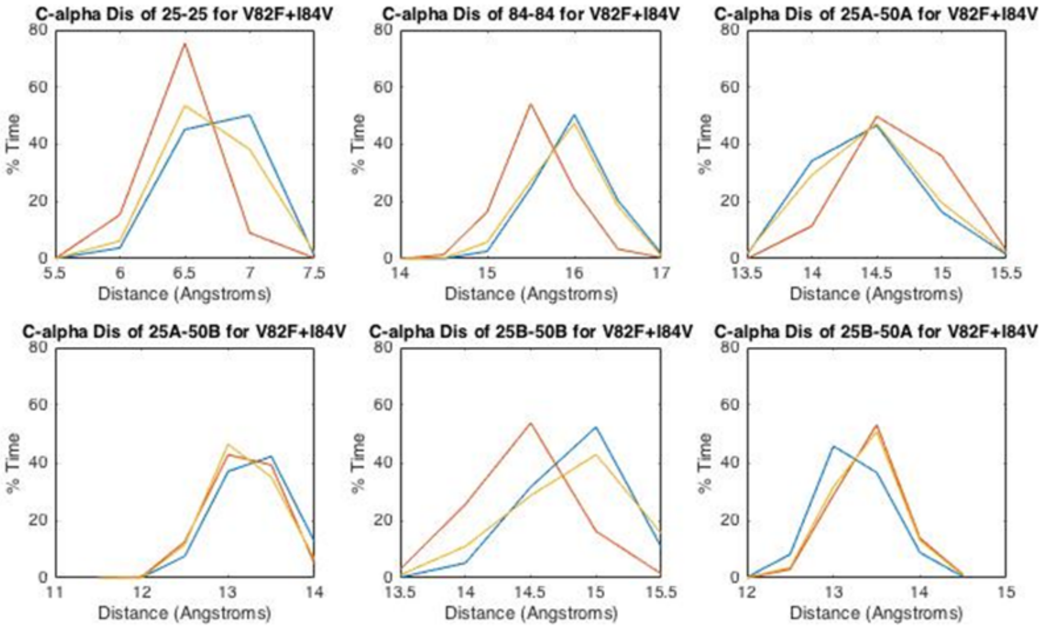


Figure 5. 16: V82F+I84V Alpha Carbon Distances

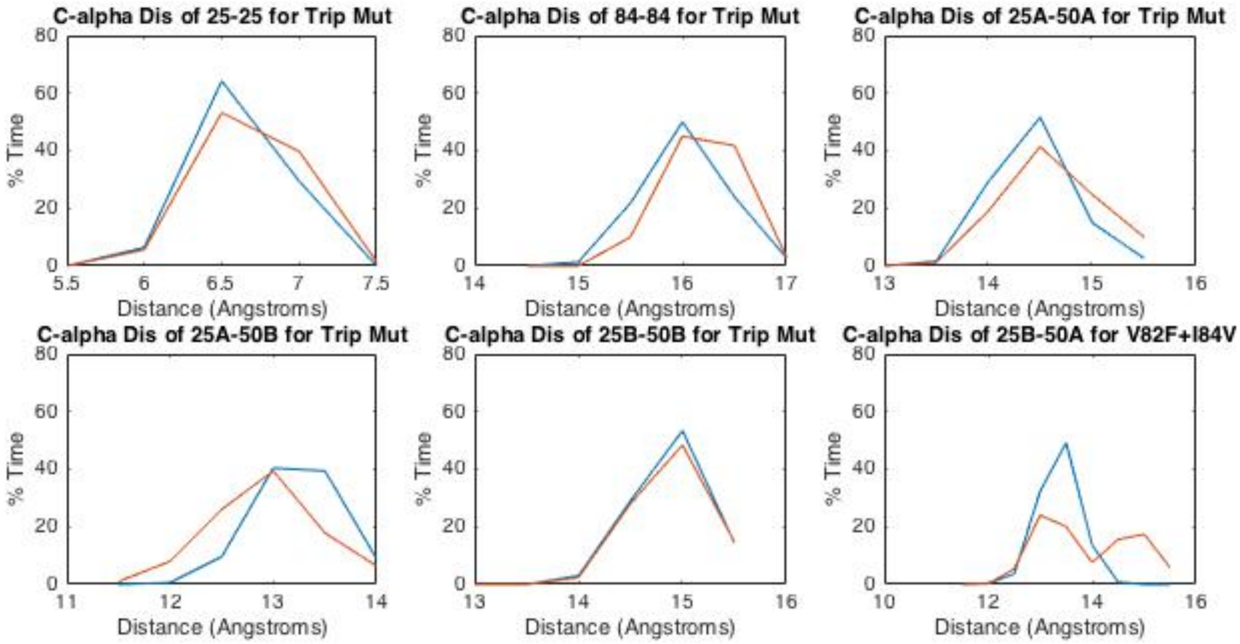


Figure 5. 17: M46I+V82F+I84V Alpha Carbon Distances

For each variant, the C-alpha distances were consistent between each of the replicates. The average of each variant was calculated and put compared to Wild Type to see any differences in patterns. The averages of each variant are shown in Figure 5.18. Wild Type is shown in blue, I84V is shown in red, V82F+I84V is shown in yellow, and M46I+V82F+I84V is shown in purple.

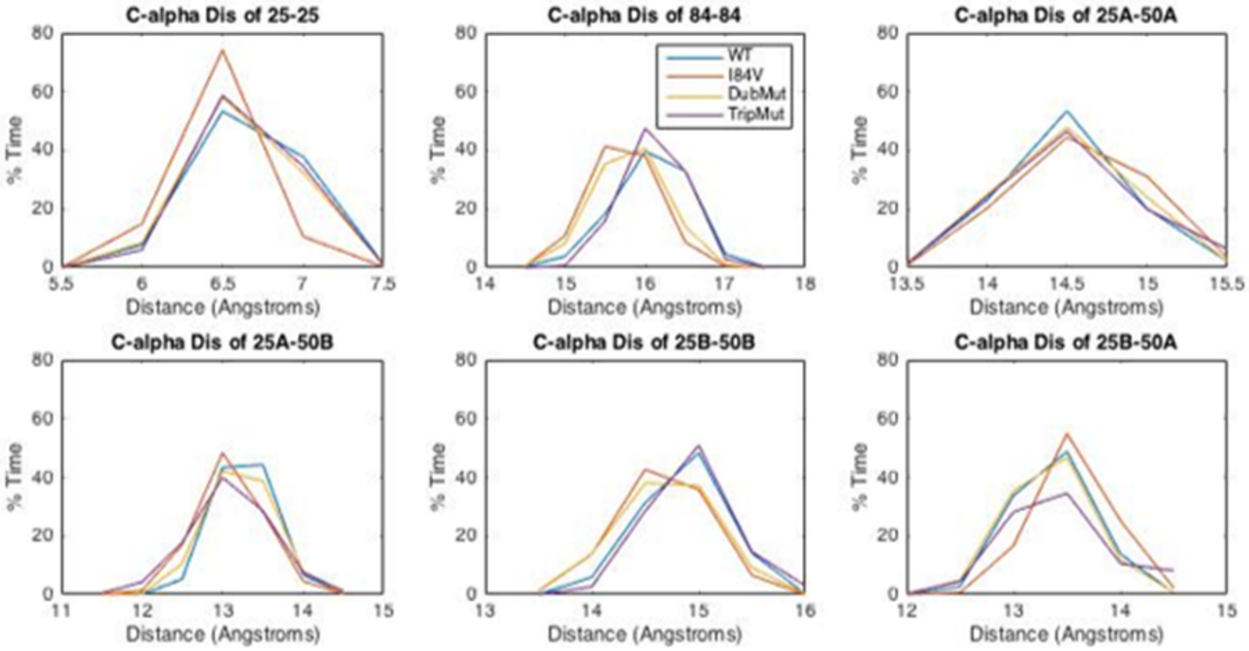


Figure 5. 18: Average Alpha Carbon Distances

The I84V variant alpha-carbon distances showed the most deviance from the wild type's distances. The distance was smaller from wild type in all residues except for 25-50, which was a similar distance, and 25'-50, which was a larger distance. V82F+I84V had similar alpha carbon differences to wild type for residues 25-25', 25-50, 25-50', and 25'-50. Smaller distances were seen between 84-84' and 25'-50'. The M46I+V82F+I84V variant has alpha carbon distances similar to wild type for residues 25-25', 84-84', 25-50, and 25'-50'. Distances between residues 25'-50 and 25-50' for the M46I+V82F+I84V variant were smaller than wild type. Overall, there were no significant differences or patterns seen between the C-alpha Distances of the mutation variants compared to the wild type variants.

Another way to visualize the alpha-carbon distances of the mutated variants were color coding the lines between the residues, of which the distances were measured. The colors corresponded to the average distance being larger, the same, or smaller than wild type. A red line represents a higher average C-alpha distance, a green line represents the same average C-alpha distance, and a blue line represents a lower average C-alpha distance to wild type. The C-alpha distances of the I84V variant is shown in the figure below (Fig. 5.19).

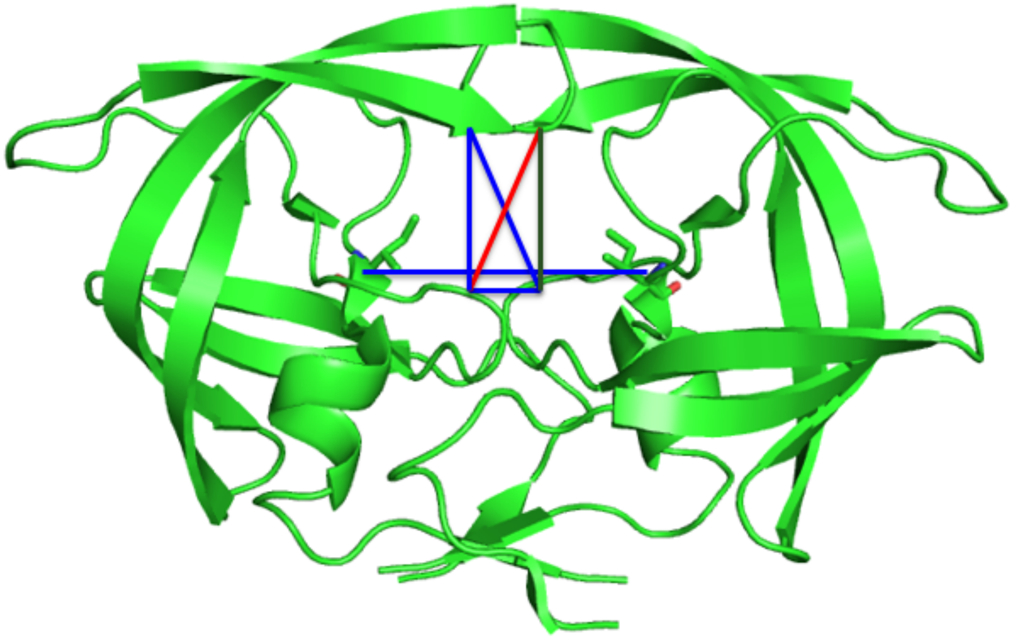


Figure 5. 19: I84V C-alpha Distances Compared to WT

The C-alpha distances between residues 25-25’, 84-84’, 25-50’, and 25’-50’ are represented by a blue line, 25-50 by a green line, and 25’-50 by a red line. The C-alpha distances of the I84V variant are mostly smaller than wild type. Only one distance is similar to wild type and another distance larger than wild type.

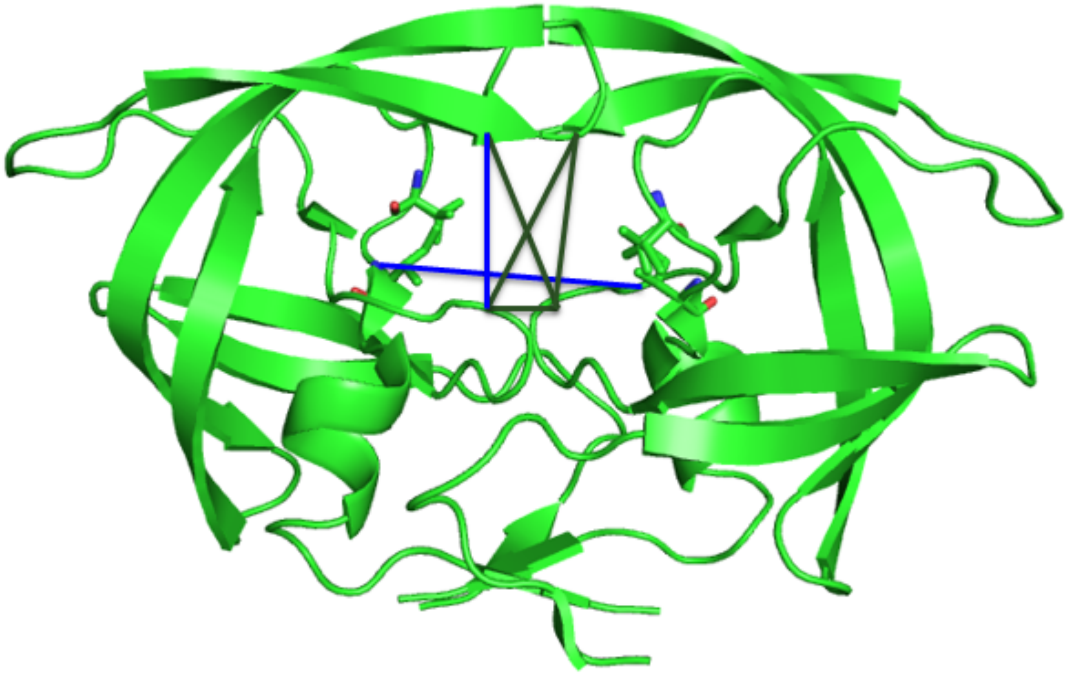


Figure 5. 20: V82F+I84V C-Alpha Distances Compared to WT

V82F+I84V C-alpha results were also compared to the wild type. The C-alpha distances between residues 84-84’ and 25’-50’ are represented by a blue line, while 25-25’, 25-50, 25-50’, and 25’-50 are represented by a green line. Generally, the average C-alpha distances of V82F+I84V variant were similar to wild type. Only two C-alpha distances had a lower average than wild type.

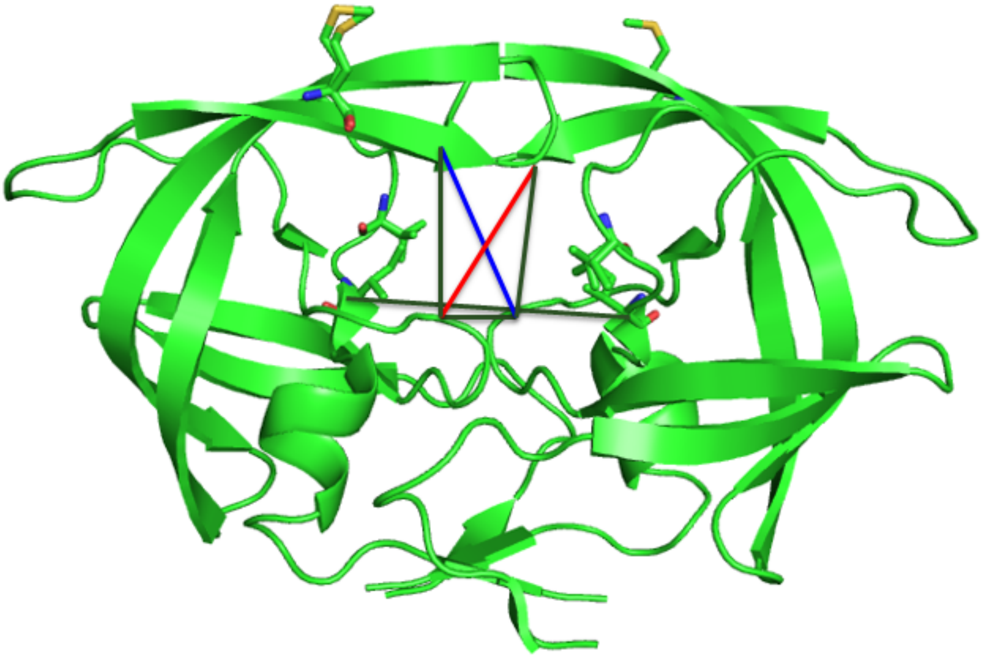


Figure 5. 21: M46I+V82F+I84V C-alpha Distances Compared to WT

M46I+V82F+I84V C-alpha distance between residues 25-50 is represented by a blue line, 25-25', 84-84', 25-50, and 25'-50'by a green line, and 25'-50 by a red line. Most of the C-alpha distances of the M46I+V82F+I84V variant were similar to wild type. There was one C-alpha distances for both a larger distance than wild type and smaller distance than wild type.

All the mutated variants experienced a smaller active site compared to Wild Type. The larger averages were seen between residues 25'-50. The C-alpha distances of the M46I+V82F+I84V variant and the V82F+I84V variant are similar to wild type, while the I84V C-alpha distances were smaller compared to wild type.

5.5 Van der Waals

The van der Waals interactions between the ligand and the protein were calculated for each variant of HIV-1 protease. The following figures show the van der Waal interactions for each replicate of each variant. The interactions are divided into chain A and chain B residues for visual purposes (Fig. 5.22 and Fig. 5.23, respectively).

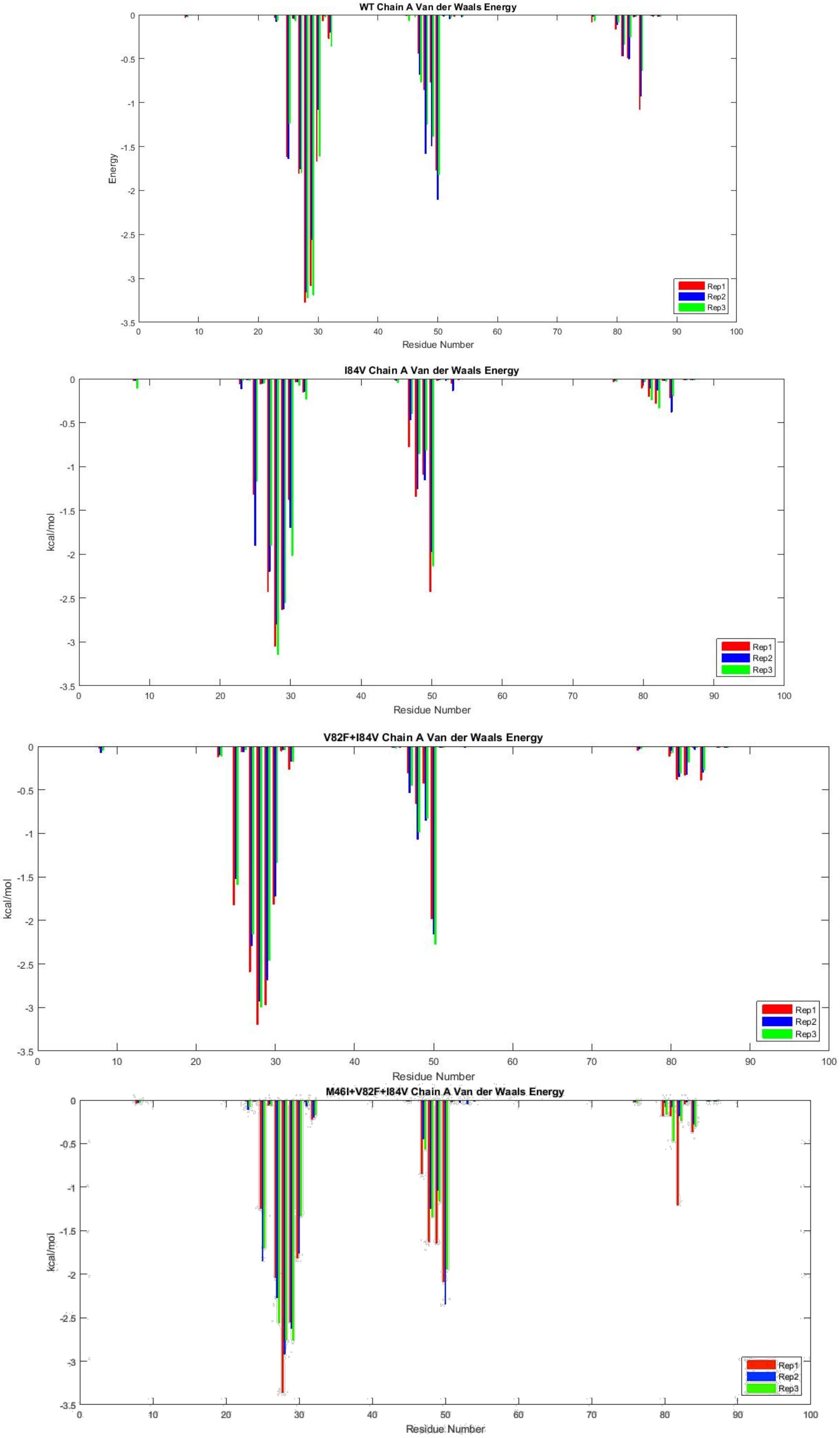


Figure 5. 22: Chain A van der Waals Energies for WT, I84V, V82F+I84V, and M46I+V82F+I84V

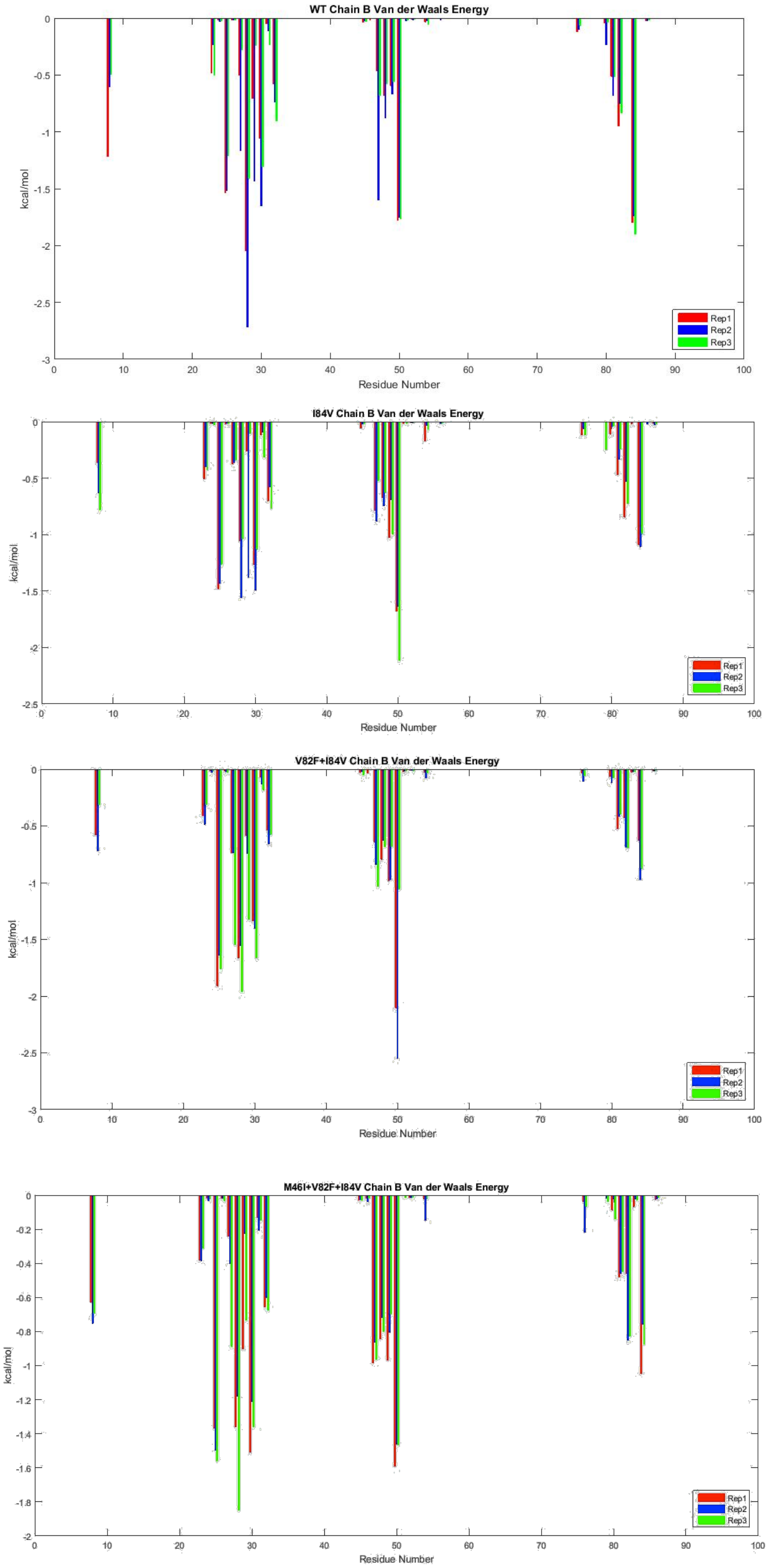


Figure 5. 23: Chain B van der Waals Energies for WT, I84V, V82F+I84V, and M46I+V82F+I84V

Each variant follows a similar trend across all replicates. Residues 25-31 generally had the greatest interactions with the inhibitor, with the exception of I84V chain B and V82F+I84V chain B. Although following a similar trend, residues 28 and 29 in chain A of wild type, I84V, V82F+I84V have energies of approximately -3.25 kcal/mol, compared to M46I+V82F+I84V with -2.75 kcal/mol. M46I+V82F+I84V chain A also had a lower energy at residue 60 with approximately -2.75 kcal/mol, compared to -2.25 to -2.5 kcal/mol seen in the other variants. With respect to chain B, wild type and V82F+I84V had the lowest energy at residue 60, approximately -1.75 and -1.8, respectively. Further, in the chain B variants there is a visible van der Waals interaction at residue 9 ranging from about -0.75 to -1.25 kcal/mol.

The average of the van der Waals energies of each variant were calculated. Similar to RMSF data, several residues had negligible Van der Waals interactions differences. The team determined through discussion with our sponsor that a difference greater than 0.02 kcal/mol was significant and is shown below for each residue. In these figures, chain A and chain B are both displayed on one graph (Fig. 5.24).

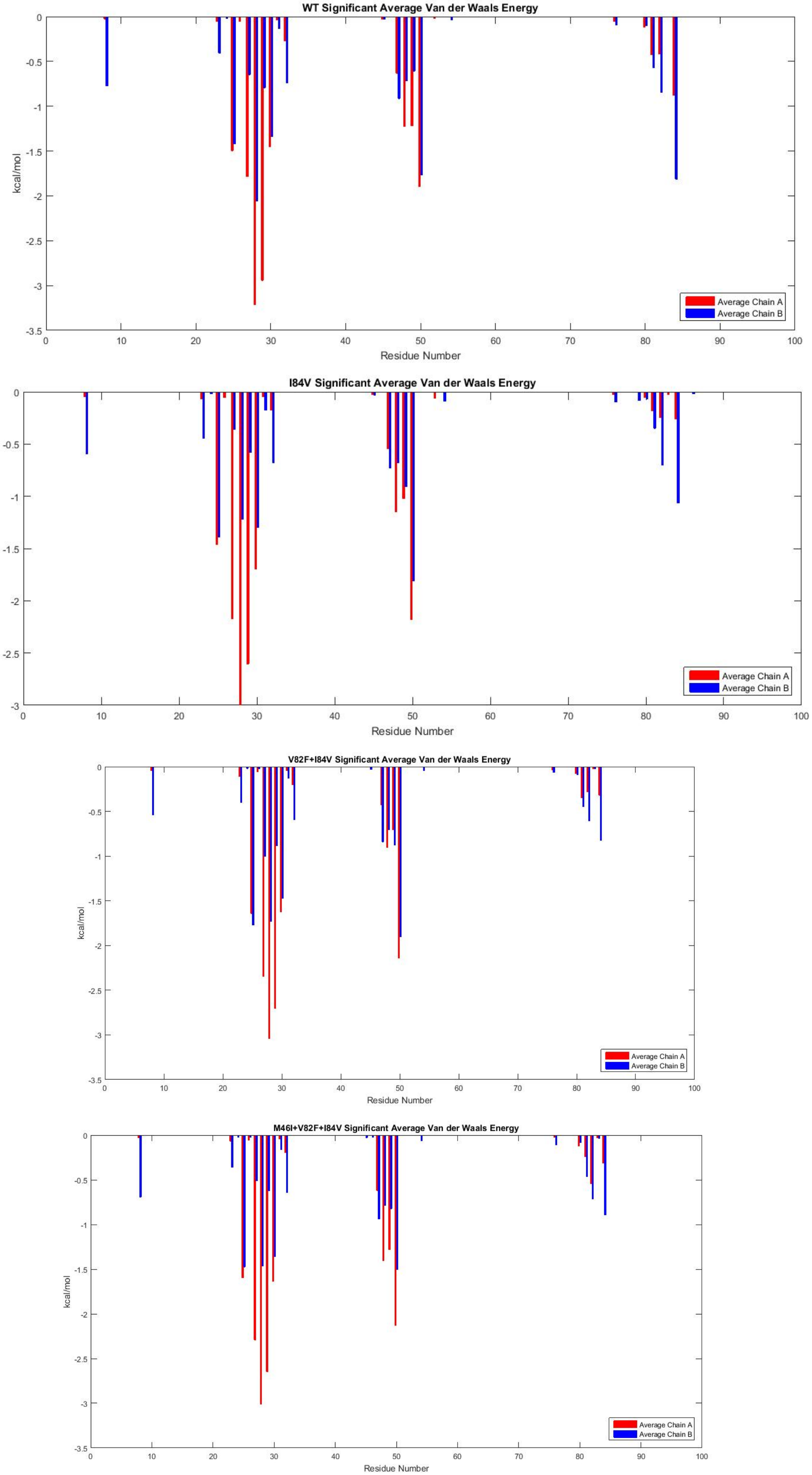


Figure 5. 24: Significant Average van der Waal Energies for WT, I84V, V82F, and M46I+V82F+I84V

Comparing the van der Waals interactions less than -0.02 kcal/mol with chain A and chain B plotted on the same graph, chain A generally had greater interactions than chain B. Especially at residues 28 and 29, the average energy is significantly greater in chain A, and in the case of I84V and M46I+V82F+I84V is more than doubled.

Next, the average of the mutated variants' van der Waals energies were compared to the Wild Type's van der Waals energies. To provide a clearer portrayal of the significant difference data, residues with a difference less than 0.02 kcal/mol were removed and all three residues were plotted in the two figures below, which one is chain A and the other is chain B (Fig 5.25).

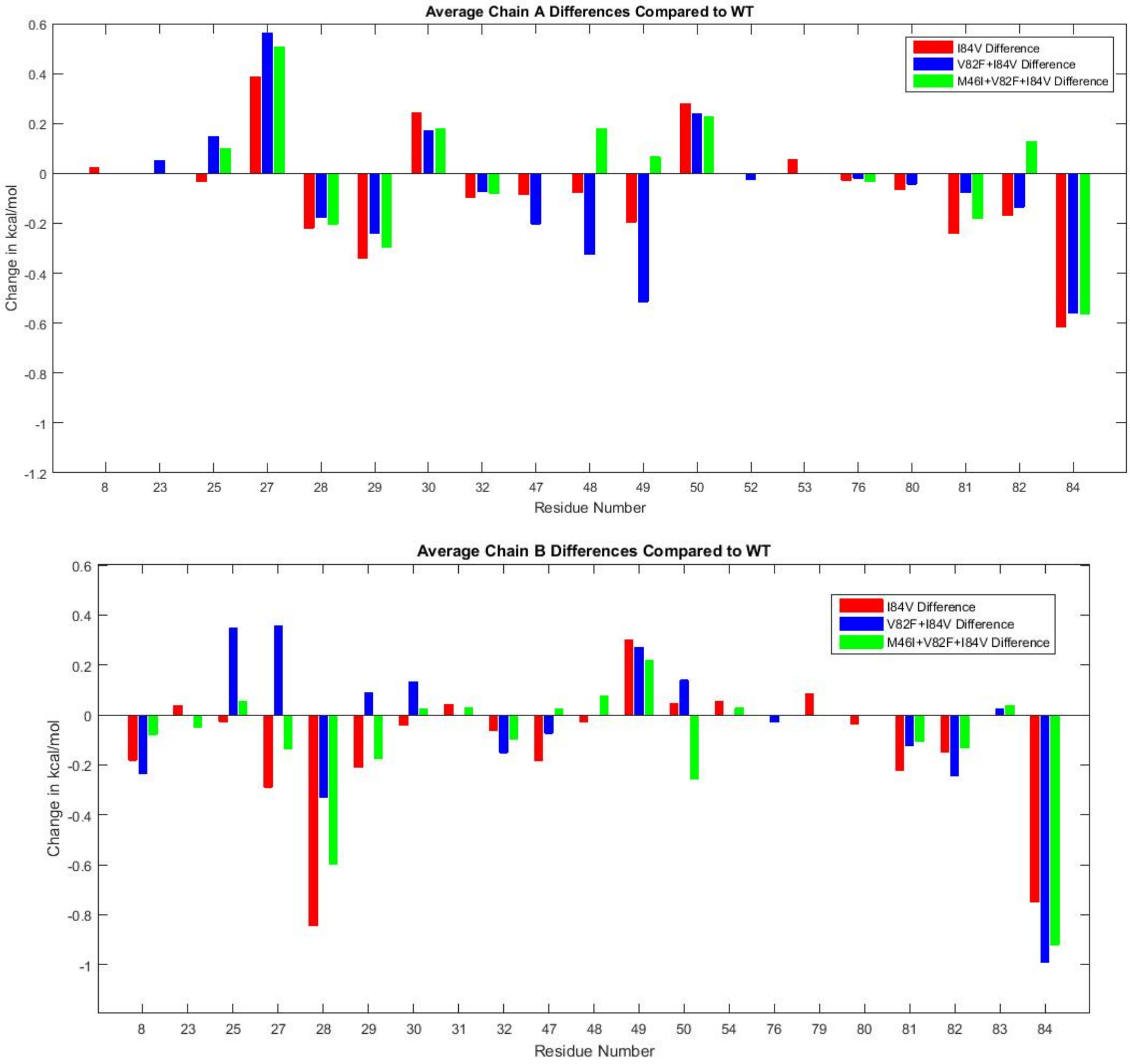


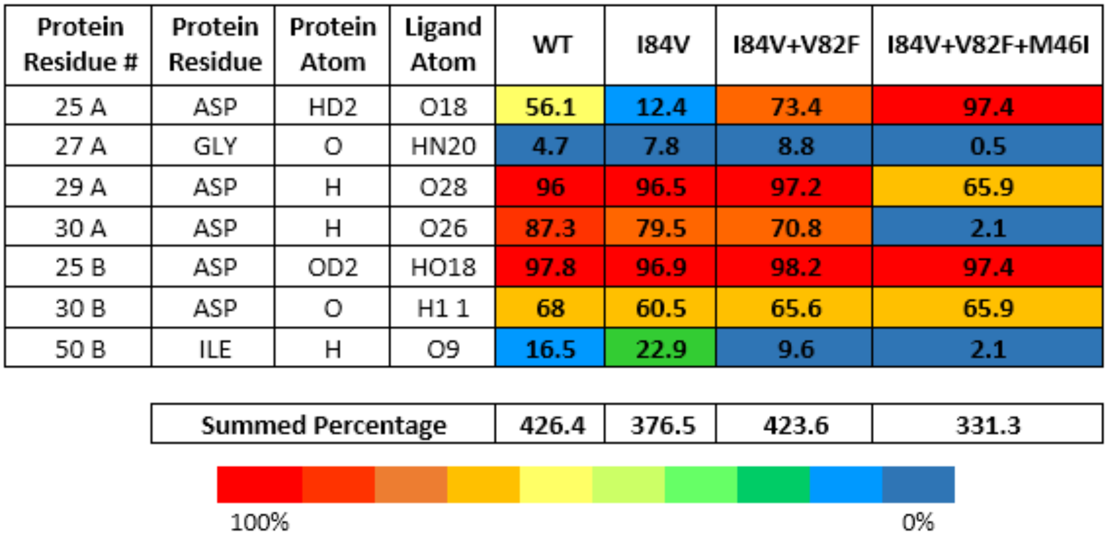
Figure 5. 25: Significant van der Waals Difference to WT

The van der Waals energies of all the residues were added together for each variant, including the mutated variants and wild type, to compare the total energy between the ligand and the protein. Wild Type had the largest van der Waals energies -34.2 +/- 2.2 kcal/mol. V82F+I84V and M46I+V82F+I84V were next with average van der Waals energies of -32.2. +/- 1.2 kcal/mol. Lastly, I84V had an average of -30.6 +/- 0.9.

5.6 Hydrogen Bonds

The hydrogen bonds between the protein and the ligand were calculated for every 10 picoseconds over the 100 nanosecond simulation. The percentage of time that each hydrogen bond was present throughout the simulation was determined. The average of the hydrogen bond percentages for each variant was compared to the wild type. These average hydrogen bond percentages are displayed in Table 5.1 and shaded according to percentage. Additionally, the hydrogen bond percentages were added together to achieve total amount of hydrogen bonds for each variant during the simulation.

Table 5. 1: Average Hydrogen Bond Percentages of WT, I84V, V82F+I84V, and M46I+V82F+I84V



Protein Residue #	Protein Residue	Protein Atom	Ligand Atom	WT	I84V	I84V+V82F	I84V+V82F+M46I
25 A	ASP	HD2	O18	56.1	12.4	73.4	97.4
27 A	GLY	O	HN20	4.7	7.8	8.8	0.5
29 A	ASP	H	O28	96	96.5	97.2	65.9
30 A	ASP	H	O26	87.3	79.5	70.8	2.1
25 B	ASP	OD2	HO18	97.8	96.9	98.2	97.4
30 B	ASP	O	H1 1	68	60.5	65.6	65.9
50 B	ILE	H	O9	16.5	22.9	9.6	2.1

Summed Percentage	426.4	376.5	423.6	331.3

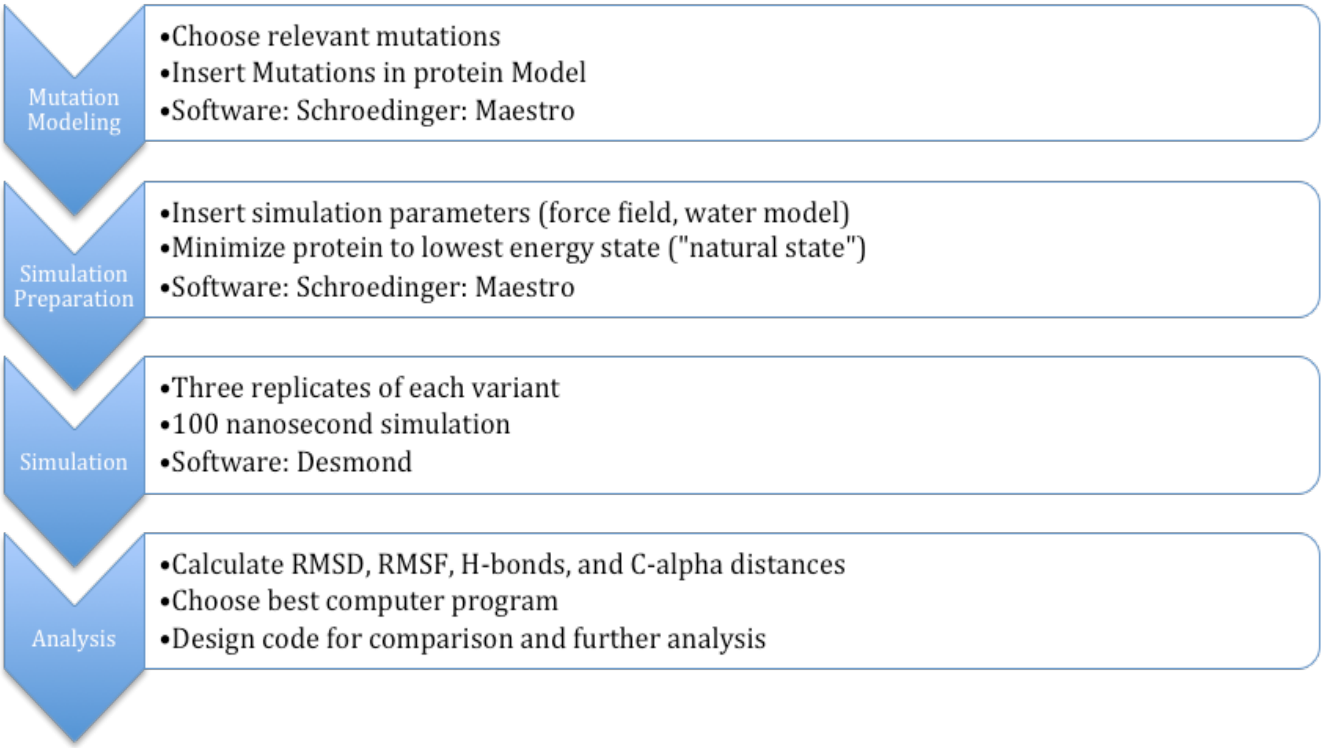
Significant differences of the hydrogen bonds seen in *Table 5.6.1* include the catalytic residue of chain A, which is residue 25 A to ligand atom number 18. I84V drastically decreases the occurrence of that hydrogen bonds while V82F+I84V and M46I+V82F+I84V drastically increase the occurrence of that hydrogen bonds. Another significant difference is the decrease seen between residue 29 in chain A and ligand number 28 of the M46I+V82F+I84V variant. The hydrogen bond occurrence of residue 30 of chain A and ligand number 26 decreases with the addition of mutations. The hydrogen bond percentage for residue 50 of chain B increases with I84V but decreases with V82F+I84V and M46I+V82F+I84V.

Lastly, the sum of the percentages show that the mutations overall decrease the number of hydrogen bonds between the ligand and the protein throughout the simulation. They also show that M46I+V82F+I84V has the least amount of hydrogen bonds with I84V having a similar decrease in hydrogen bond percentages. However, V82F+I84V has a similar hydrogen bond character as the wild type.

6.0 Final Design and Validation

6.1 Experimental Process

Molecular dynamic simulations were chosen as the experimental method to compute the structural dynamics and patterns of resistance to Darunavir (DRV) in mutant HIV-1 variants. The steps considered when running molecular dynamic (MD) simulations using Schroedinger's Maestro are outlined in Figure 6.1.



Mutation Modeling

- •Choose relevant mutations
- •Insert Mutations in protein Model
- •Software: Schroedinger: Maestro

Simulation Preparation

- •Insert simulation parameters (force field, water model)
- •Minimize protein to lowest energy state ("natural state")
- •Software: Schroedinger: Maestro

Simulation

- •Three replicates of each variant
- •100 nanosecond simulation
- •Software: Desmond

Analysis

- •Calculate RMSD, RMSF, H-bonds, and C-alpha distances
- •Choose best computer program
- •Design code for comparison and further analysis

Figure 6. 1: Steps of Molecular Dynamics Simulation

6.1.1 Preparation

Preparation involves importing the protein's information into the modeling software, modifying the model with the desired mutations, and adding virtual experimental conditions to the model. The modeling software used for the preparation step was Schrodinger's Maestro. The methodology followed during the preparation step to build the homology model is as follows:

1. Under the tasks tab, select homology model.

2. Click add files and import the genetic code sequence in the form of a PDB.
3. Click next followed by blast homology search.
4. Select and modify amino acid residues of interest in the genome sequence.
5. Slide chain B residue 1 to the right until it is properly aligned following chain A.
6. Click next.
7. Select energy-based and homo-multimer check boxes.
8. Select the inhibitor from the list, where DRV is denoted by 017.
9. Click options, and select preserve residue number.
10. Click build model.

Completion of the above steps created a homology model of each desired mutation. However, this model is missing chain B residue 1, but it will be added and bonded to the protein during minimization. A PDB file of the HIV-1 protease DRV complex was imported instead of simply typing in a fasta sequence to provide the program with atom coordinates. The atom coordinates used in the preparation step directly impact the protein conformation and interactions observed from running the simulation. The PDB used was obtained from Research Collaboratory for Structural Bioinformatics (RCSB) protein data bank, an industry standard database, which will ensure the accuracy of the protein crystal structure. A homology model was created for each of our mutations of interest: I84V, V82F+I84V, and M46I+V82F+I84. In addition to modifying the amino acid residues for each mutation, a homology model without any mutations was generated to serve as the wild type control.

6.1.2 Minimization

After generating the homology model, it is then processed using Maestro's protein preparation wizard to change certain residues into their most minimized state, or the position requiring the least energy. This process also includes adding water molecules that are present in *in vivo* scenarios. Lastly, minimization ensures there are no clashing molecules, such as an overlapping water molecules. Minimization occurs through the following steps:

1. Under the tasks tab, select protein refinement, and click minimize.
2. Change the force field to OPLS_2005.
3. Under atoms, click the plus sign and then select.
4. Under the sequence tab:
    a. Click chain A, sequence number and add
    b. Click chain B, sequence number and add
    c. On the left of the sequence tab, click backbone/side chain
    d. Select side chain and click intersect
    e. Click run.
5. Import the original PDB.
6. Delete all but chain B residue 1 by selecting the delete icon, select residue, invert, ok.
7. Click the pencil icon and draw the bond connecting chain B residue 1 to 2.
8. Import WT PDB to add crystallographic waters.
9. Delete all molecules except waters by selecting the delete icon, click molecule, molecular type, water, invert, ok.
10. Merge the water only model and the protein homology model.
11. Check inhibitor bond orders and stereochemistry.
12. Open the protein preparation wizard.
13. Under the pre-process protein tab, check off:
    a. Assign bond orders
    b. Remove original hydrogens
    c. Convert selenomethionines to methionines
    d. Create zero-order bonds to metals
    e. Fill in missing side chains using prime.
14. Under the pre-process protein tab, uncheck:
    a. Create disulfide bonds

    b. Delete waters.

15. Click pre-process.
16. Click view problems. If there are overlapping atoms, delete the interfering waters atom(s) and click update.
17. Under the review and modify tab:
    a. Click analyze workspace
    b. Click generate states and select the lowest energy state.
18. On the refine tab:
    a. Under H-Bond assignment, click sample water orientations, minimize hydrogens of altered species, click PROPKA and pH of 7.0 and click optimize.
    b. Under restrained minimization, select hydrogens only and click minimize
    c. Check problems, reports and plots
    d. Check Asp25 and note which chain is protonated.

Completion of the above steps successfully minimized the homology model created during preparation. The final step in minimization prior to simulation is creating an orthorhombic water box. This box has dimensions of 10x10x10 angstroms, contains no additional salt buffers and is neutralized. Prior to simulation, hydrogen atoms are also deleted from the model.

6.1.3 Simulate

After using Maestro's protein preparation wizard, the system is ready for the simulation process using Desmond. The system is sent to a computer with the Desmond molecular dynamics script to run the simulation process. The time of the simulation and the type of computer you run the simulation on is specified in the submission of the Desmond job. The team chose 100 nanoseconds for the simulation time and GPU computers to run the simulation on.

To submit the simulation, the team followed these instructions:

1. Open a new terminal tab and enter the UMASS cluster.
2. Go into the desired project folder and make a new folder for the specific simulation.
3. Go back to the original terminal tab and go into the folder with the Desmond_setup_out file
4. Transfer the Desmond_setup_out file into the new folder of the UMASS cluster with the command scp
5. Copy the molecular dynamics protocol into the new folder of the UMASS cluster with the command cp
6. Submit the job into the cluster with the Schrödinger submit command

After the simulation is submitted, the Desmond script "heats" the system until equilibration at 300 K. Once equilibration is reached, the system continues with the simulation. The simulation process involves applying an energy force field and environmental parameters to the system and recording the atoms' coordinates and energies. These coordinates are the movement of the protein. The energies can be analyzed further to describe how well DRV was attached in the active site.

The coordinates and energies are recorded for the 100 ns in a file. This file's information can be analyzed to determine the RMSF, RMSD, hydrogen bonds, van der Waals, alpha-carbon distances, and much more of the protein and DRV. The team had three different simulations of each mutation and the wild type variant.

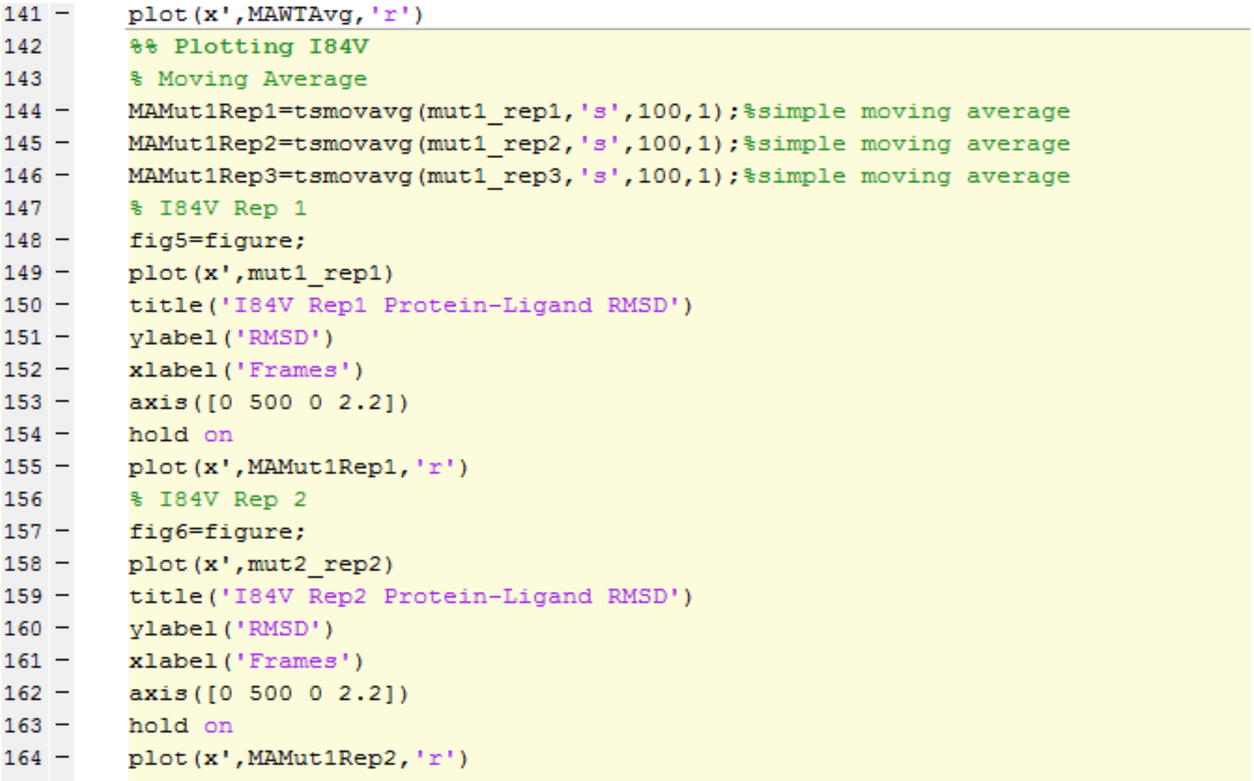
Once the simulations were finished, the information was analyzed using Schrodinger's Maestro, VMD scripts, and GFortran scripts. For Schrodinger's Maestro, the protein interactions tool was utilized. This tool gave the protein RMSF, protein RMSD, ligand RMSF, hydrogen Bonds, and alpha-carbon Distances. VMD and GFortran was used to analyze the van der Waals of the ligand to protein.

6.2 Data Analysis Process

The second component of the final project design is the analysis process and developed software. With the exception of the one python script to overwrite a PBD file, all scripting was conducted in MATLAB. This section contains highlights of the code written, and fully published code can be found in appendix B.

6.2.1 Protein-Ligand RMSD

A script for protein root-mean-squared deviation (RMSD) was written to initialize comma delimited value files. RMSD indicates the fluctuation of the alpha-carbon atoms (backbone of the protein) during the simulation. A higher RMSD indicates less protein stability. The program inputs comma delimited value files with a .csv file extension for each mutation, reads the data, and appends each RMSD value to an array. Additionally, the moving average for the entire protein RMSD was calculated using MATLAB's built-in "tsmovavg" function (Fig. 6.2 line 144-146). The major component of this script is the graphical representation of protein RMSD (line 148-164). For each mutation replicate a line graph is generated with each of the three replicate data as well as an average. The moving average was then over laid on the graph.

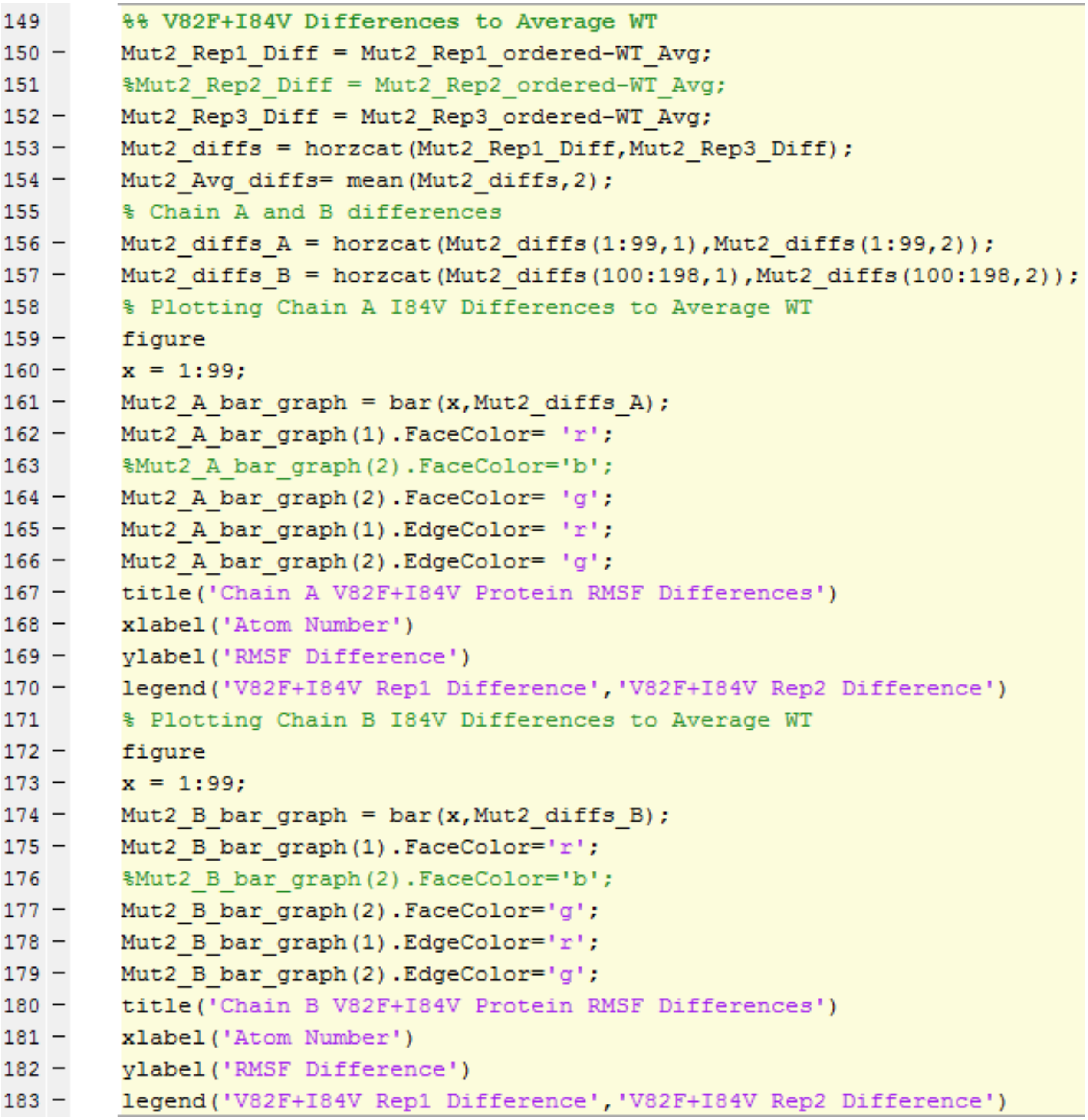


```
141 -    plot(x',MAWTAvg,'r')
142      %% Plotting I84V
143      % Moving Average
144 -    MAMut1Rep1=tsmovavg(mut1_rep1,'s',100,1);%simple moving average
145 -    MAMut1Rep2=tsmovavg(mut1_rep2,'s',100,1);%simple moving average
146 -    MAMut1Rep3=tsmovavg(mut1_rep3,'s',100,1);%simple moving average
147      % I84V Rep 1
148 -    fig5=figure;
149 -    plot(x',mut1_rep1)
150 -    title('I84V Rep1 Protein-Ligand RMSD')
151 -    ylabel('RMSD')
152 -    xlabel('Frames')
153 -    axis([0 500 0 2.2])
154 -    hold on
155 -    plot(x',MAMut1Rep1,'r')
156      % I84V Rep 2
157 -    fig6=figure;
158 -    plot(x',mut2_rep2)
159 -    title('I84V Rep2 Protein-Ligand RMSD')
160 -    ylabel('RMSD')
161 -    xlabel('Frames')
162 -    axis([0 500 0 2.2])
163 -    hold on
164 -    plot(x',MAMut1Rep2,'r')
```

Figure 6. 2: Portion of RMSD Script

6.2.2 Protein RMSF

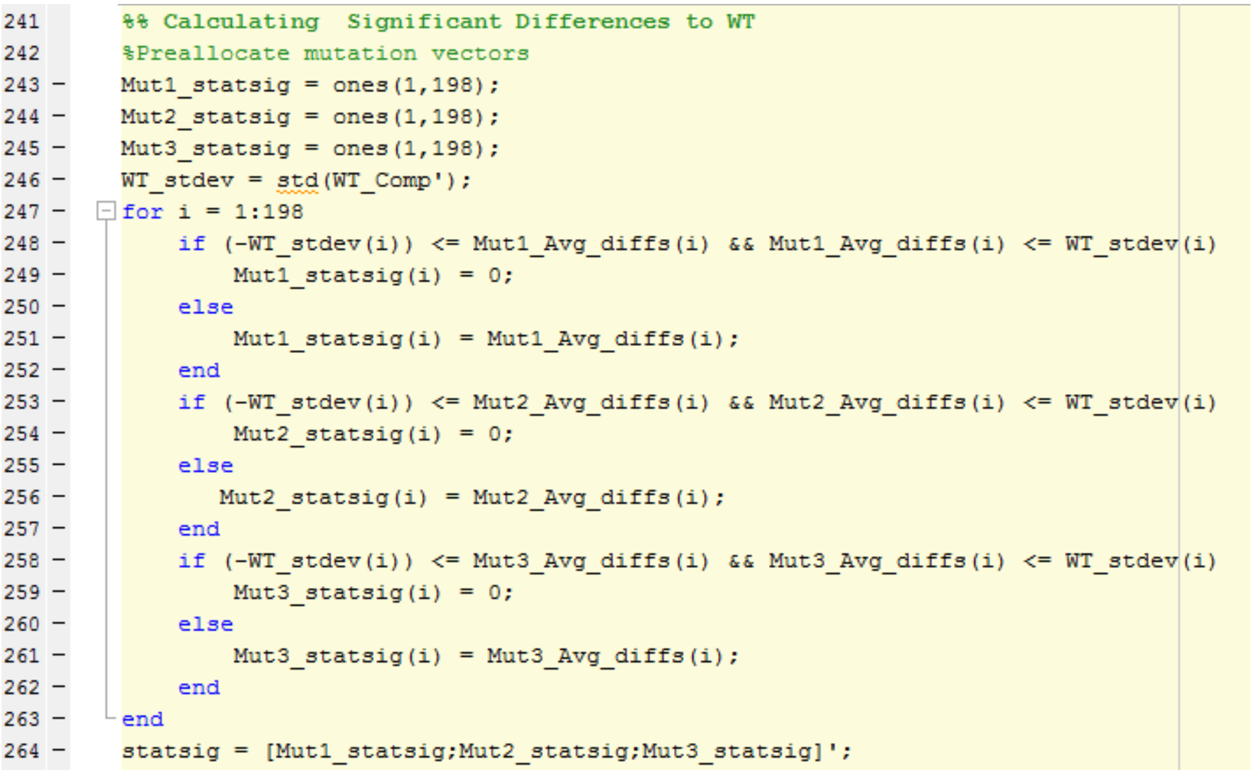
A script for protein root-mean-squared fluctuation (RMSF) was created to analyze the proteins movement during the simulation. The program utilizes text files generated from the molecular modeling software by reading the data and appending each RMSF value into an array. This script was created to help quantify the data collected during simulations into a graphical visual. For each mutation a line graph is generated with each of the three replicate data. Additionally, a line graph is generated displaying average RMSF mutation replicates against wild type RMSF replicates.



```
149     %% V82F+I84V Differences to Average WT
150 -   Mut2_Rep1_Diff = Mut2_Rep1_ordered-WT_Avg;
151     %Mut2_Rep2_Diff = Mut2_Rep2_ordered-WT_Avg;
152 -   Mut2_Rep3_Diff = Mut2_Rep3_ordered-WT_Avg;
153 -   Mut2_diffs = horzcat(Mut2_Rep1_Diff,Mut2_Rep3_Diff);
154 -   Mut2_Avg_diffs= mean(Mut2_diffs,2);
155     % Chain A and B differences
156 -   Mut2_diffs_A = horzcat(Mut2_diffs(1:99,1),Mut2_diffs(1:99,2));
157 -   Mut2_diffs_B = horzcat(Mut2_diffs(100:198,1),Mut2_diffs(100:198,2));
158     % Plotting Chain A I84V Differences to Average WT
159 -   figure
160 -   x = 1:99;
161 -   Mut2_A_bar_graph = bar(x,Mut2_diffs_A);
162 -   Mut2_A_bar_graph(1).FaceColor= 'r';
163     %Mut2_A_bar_graph(2).FaceColor='b';
164 -   Mut2_A_bar_graph(2).FaceColor= 'g';
165 -   Mut2_A_bar_graph(1).EdgeColor= 'r';
166 -   Mut2_A_bar_graph(2).EdgeColor= 'g';
167 -   title('Chain A V82F+I84V Protein RMSF Differences')
168 -   xlabel('Atom Number')
169 -   ylabel('RMSF Difference')
170 -   legend('V82F+I84V Rep1 Difference','V82F+I84V Rep2 Difference')
171     % Plotting Chain B I84V Differences to Average WT
172 -   figure
173 -   x = 1:99;
174 -   Mut2_B_bar_graph = bar(x,Mut2_diffs_B);
175 -   Mut2_B_bar_graph(1).FaceColor='r';
176     %Mut2_B_bar_graph(2).FaceColor='b';
177 -   Mut2_B_bar_graph(2).FaceColor='g';
178 -   Mut2_B_bar_graph(1).EdgeColor='r';
179 -   Mut2_B_bar_graph(2).EdgeColor='g';
180 -   title('Chain B V82F+I84V Protein RMSF Differences')
181 -   xlabel('Atom Number')
182 -   ylabel('RMSF Difference')
183 -   legend('V82F+I84V Rep1 Difference','V82F+I84V Rep2 Difference')
```

Figure 6. 3: Portion of Protein RMSF

The RMSF values of each variant were then compared to the wild type. The average data set for each mutation was subtracted from the average wild type data set and appended into a new array (Fig. 6.3 line 150-154). If the mutant had a larger fluctuation this difference would be positive and in the case there was less movement in the mutant ligand this value would be negative. This refined RMSF difference data was plotted on a bar plot. Also, bar plots with a numerical threshold were coded to display significant fluctuations as shown in Figure 6.4.



```
%% Calculating  Significant Differences to WT
%Preallocate mutation vectors
Mut1_statsig = ones(1,198);
Mut2_statsig = ones(1,198);
Mut3_statsig = ones(1,198);
WT_stdev = std(WT_Comp');
for i = 1:198
    if (-WT_stdev(i)) <= Mut1_Avg_diffs(i) && Mut1_Avg_diffs(i) <= WT_stdev(i)
        Mut1_statsig(i) = 0;
    else
        Mut1_statsig(i) = Mut1_Avg_diffs(i);
    end
    if (-WT_stdev(i)) <= Mut2_Avg_diffs(i) && Mut2_Avg_diffs(i) <= WT_stdev(i)
        Mut2_statsig(i) = 0;
    else
       Mut2_statsig(i) = Mut2_Avg_diffs(i);
    end
    if (-WT_stdev(i)) <= Mut3_Avg_diffs(i) && Mut3_Avg_diffs(i) <= WT_stdev(i)
        Mut3_statsig(i) = 0;
    else
        Mut3_statsig(i) = Mut3_Avg_diffs(i);
    end
end
statsig = [Mut1_statsig;Mut2_statsig;Mut3_statsig]';
```

Figure 6. 4: Significant Protein RMSF Differences

6.2.3 Ligand RMSF

A script for ligand root-mean-squared fluctuation (RMSF) was written that initializes the text files output from the simulation. The program inputs the text file with a .txt file extension for each mutation, reads the data, and appends each RMSF value to an array. In the cases of mutant variants, the ligand atom numbers are not in proper order (Fig. 6.5). The correct ligand order was obtained from the PDB, which states the correct atom number of each element in the inhibitor.

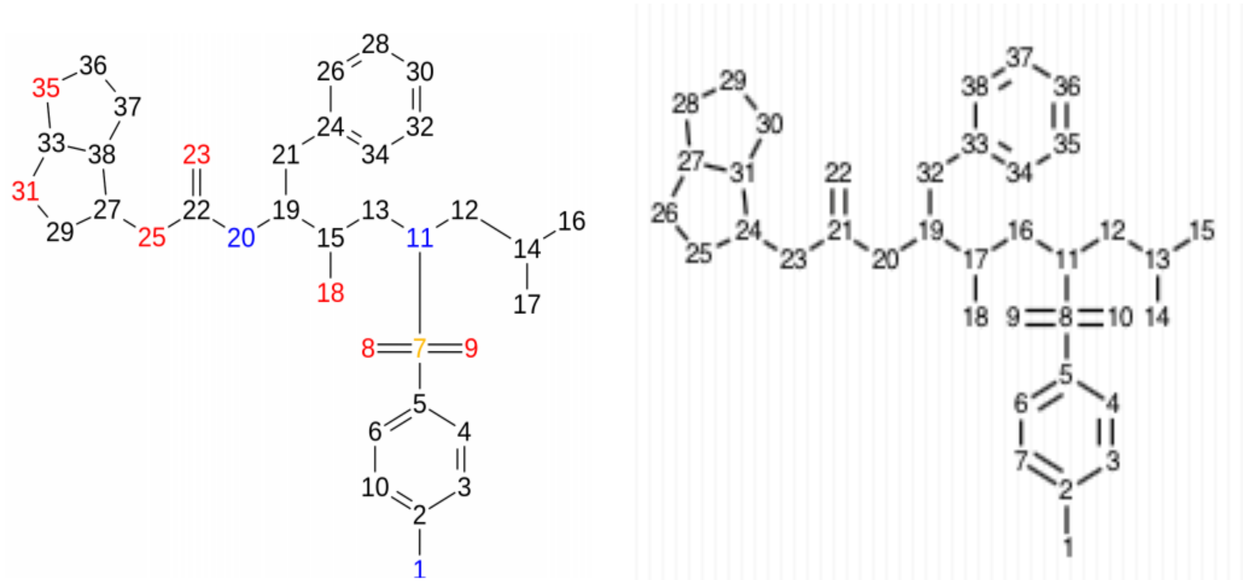
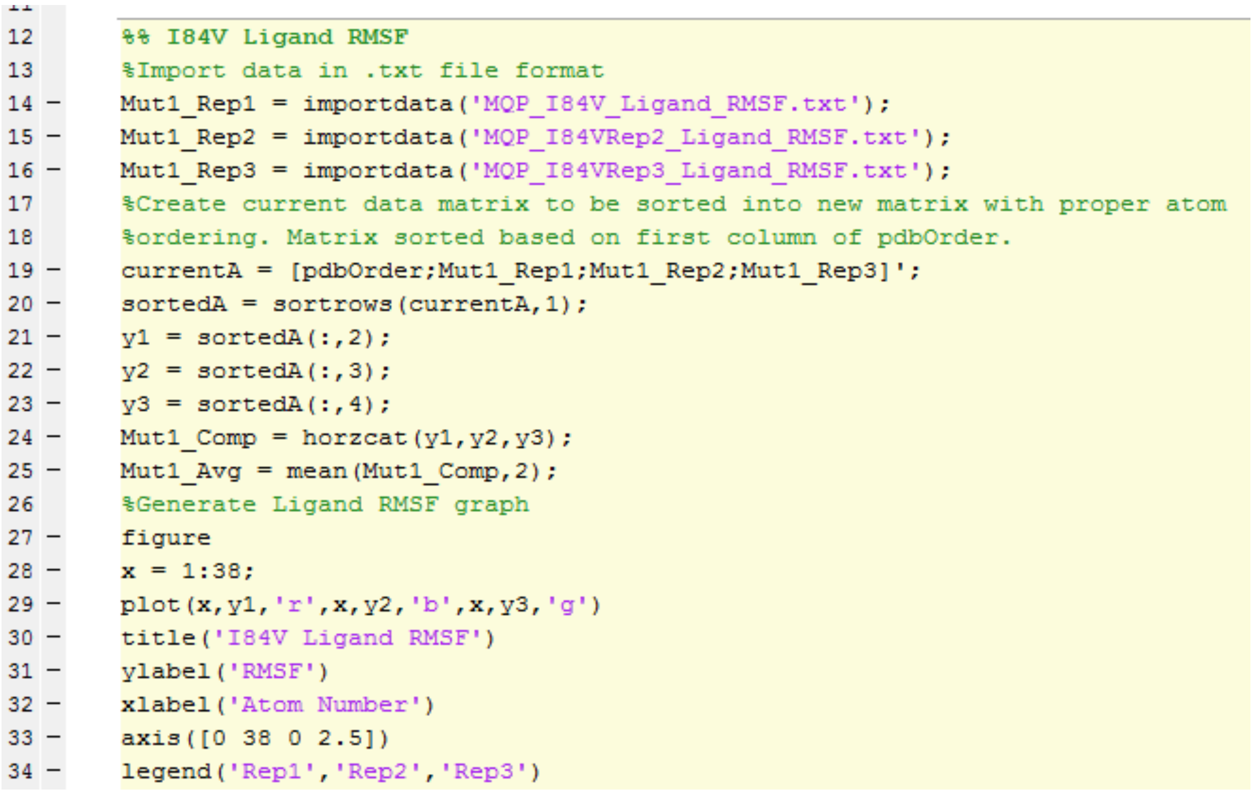


Figure 6. 5: Mis Ordered (left) and Properly Ordered (right) Ligand

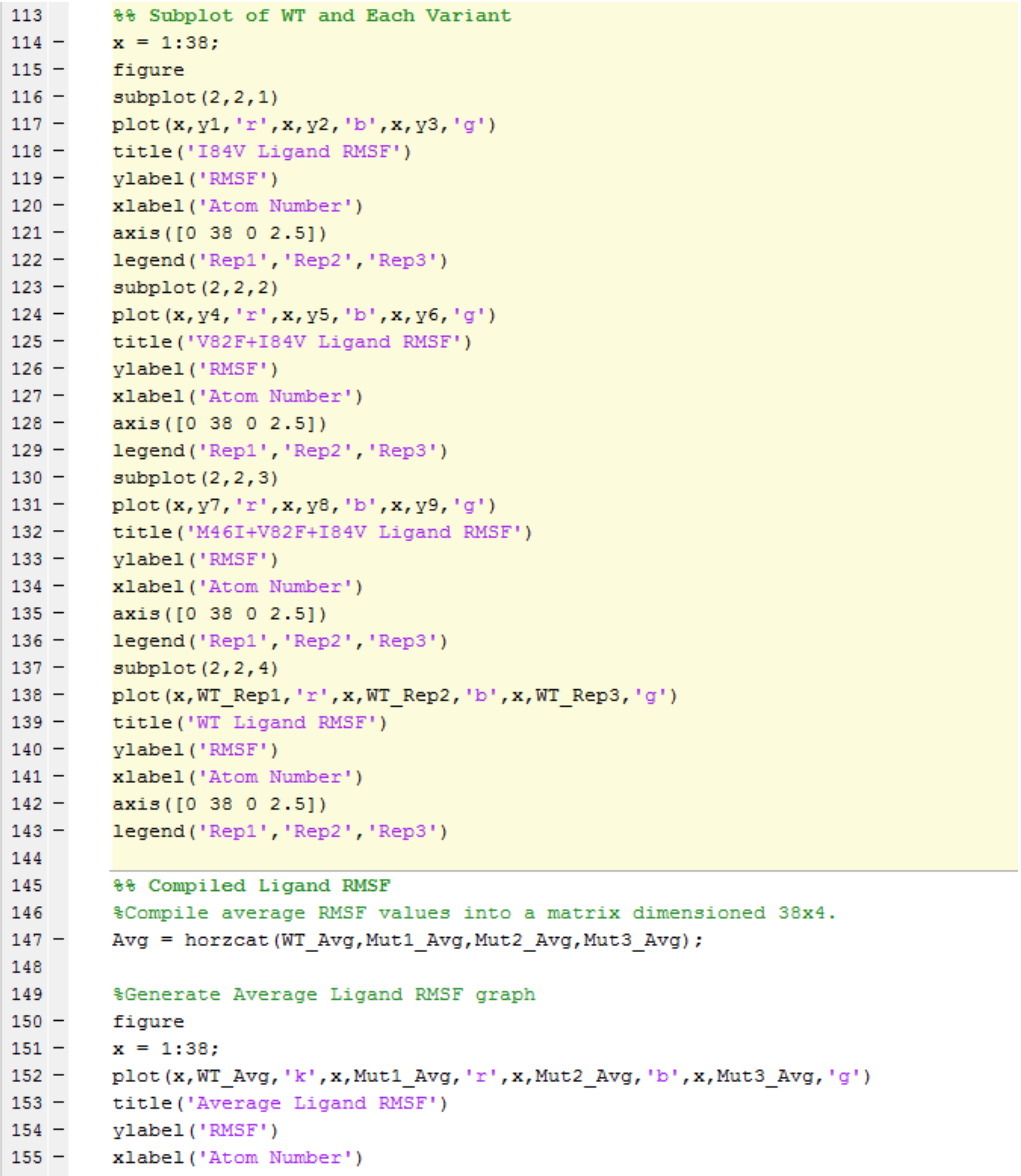
To correct for this, the script properly orders the RMSF values of the three mutant variants using the sort function and appends to a new, properly ordered 3 x 38 array with each mutation as a column (Fig. 6.6). Similar code was written for each mutant variant and wild type. However, in the case of the wild type, the ligand is properly ordered. Therefore, the wild type portion of the code is similar but omits lines 20 through 24. As seen below, variable names were kept vague, such as Mut1_Rep1 or y1, to allow the script to be used for future analysis without requiring major editing.



```
12      %% I84V Ligand RMSF
13      %Import data in .txt file format
14 —    Mut1_Rep1 = importdata('MQP_I84V_Ligand_RMSF.txt');
15 —    Mut1_Rep2 = importdata('MQP_I84VRep2_Ligand_RMSF.txt');
16 —    Mut1_Rep3 = importdata('MQP_I84VRep3_Ligand_RMSF.txt');
17      %Create current data matrix to be sorted into new matrix with proper atom
18      %ordering. Matrix sorted based on first column of pdbOrder.
19 —    currentA = [pdbOrder;Mut1_Rep1;Mut1_Rep2;Mut1_Rep3]';
20 —    sortedA = sortrows(currentA,1);
21 —    y1 = sortedA(:,2);
22 —    y2 = sortedA(:,3);
23 —    y3 = sortedA(:,4);
24 —    Mut1_Comp = horzcat(y1,y2,y3);
25 —    Mut1_Avg = mean(Mut1_Comp,2);
26      %Generate Ligand RMSF graph
27 —    figure
28 —    x = 1:38;
29 —    plot(x,y1,'r',x,y2,'b',x,y3,'g')
30 —    title('I84V Ligand RMSF')
31 —    ylabel('RMSF')
32 —    xlabel('Atom Number')
33 —    axis([0 38 0 2.5])
34 —    legend('Rep1','Rep2','Rep3')
```

Figure 6. 6: Ligand RMSF Data Import, Sort, and Plot

A major component of this script is the graphical representation of ligand RMSF. For each mutation a line graph is generated with each of the three replicate data as well as an average (Fig. 6.6, lines 27-34). To provide additional data representation, a subplot of the four line graphs was also generated. A final line graph is generated displaying average of each mutation and wild type RMSF (Fig. 6.7).

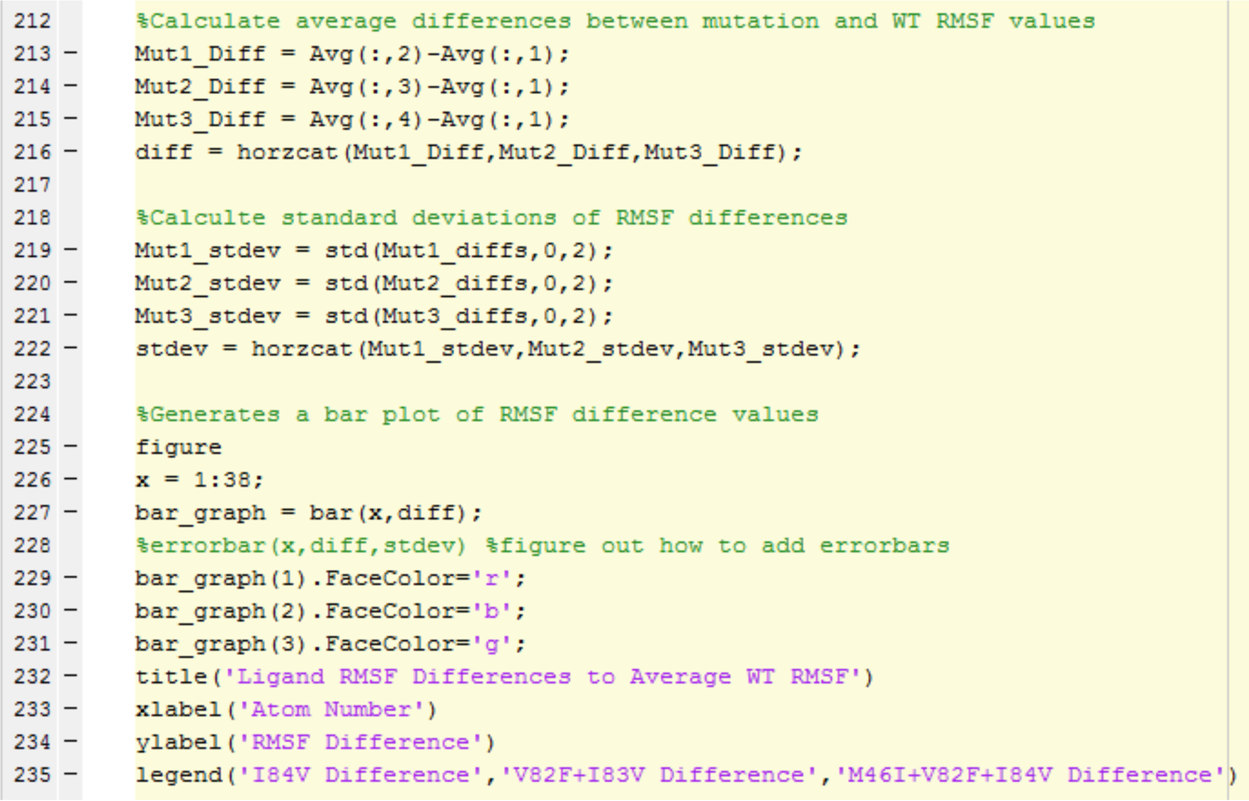


```
%% Subplot of WT and Each Variant
x = 1:38;
figure
subplot(2,2,1)
plot(x,y1,'r',x,y2,'b',x,y3,'g')
title('I84V Ligand RMSF')
ylabel('RMSF')
xlabel('Atom Number')
axis([0 38 0 2.5])
legend('Rep1','Rep2','Rep3')
subplot(2,2,2)
plot(x,y4,'r',x,y5,'b',x,y6,'g')
title('V82F+I84V Ligand RMSF')
ylabel('RMSF')
xlabel('Atom Number')
axis([0 38 0 2.5])
legend('Rep1','Rep2','Rep3')
subplot(2,2,3)
plot(x,y7,'r',x,y8,'b',x,y9,'g')
title('M46I+V82F+I84V Ligand RMSF')
ylabel('RMSF')
xlabel('Atom Number')
axis([0 38 0 2.5])
legend('Rep1','Rep2','Rep3')
subplot(2,2,4)
plot(x,WT_Rep1,'r',x,WT_Rep2,'b',x,WT_Rep3,'g')
title('WT Ligand RMSF')
ylabel('RMSF')
xlabel('Atom Number')
axis([0 38 0 2.5])
legend('Rep1','Rep2','Rep3')

%% Compiled Ligand RMSF
%Compile average RMSF values into a matrix dimensioned 38x4.
Avg = horzcat(WT_Avg,Mut1_Avg,Mut2_Avg,Mut3_Avg);

%Generate Average Ligand RMSF graph
figure
x = 1:38;
plot(x,WT_Avg,'k',x,Mut1_Avg,'r',x,Mut2_Avg,'b',x,Mut3_Avg,'g')
title('Average Ligand RMSF')
ylabel('RMSF')
xlabel('Atom Number')
```

Figure 6. 7: Ligand RMSF Replicate Subplot and Average Plot Code

The RMSF values of each variant were then compared to the wild type. The average data set for each mutation was subtracted from the average wild type data set and appended into a new array (Fig.

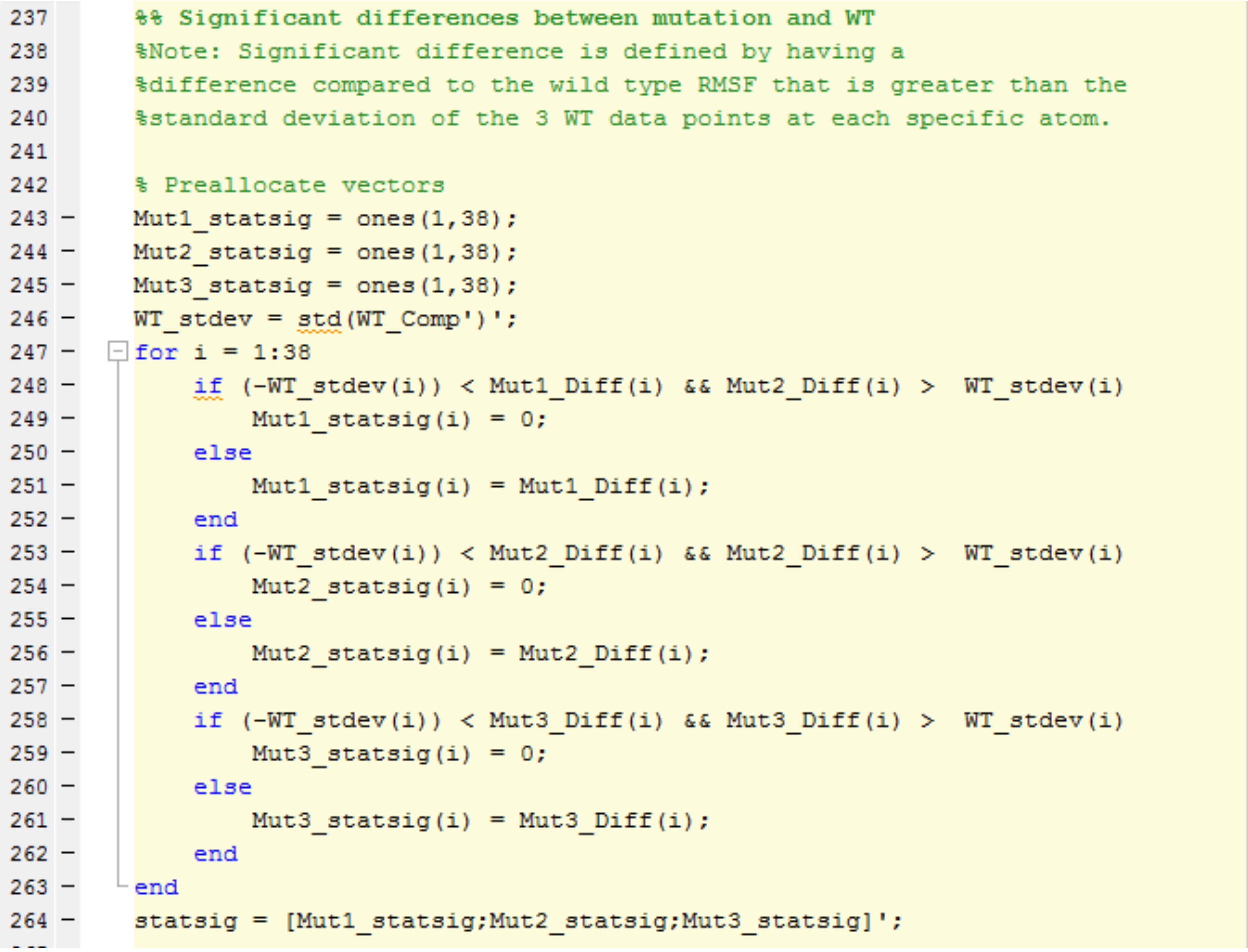
6.8). In this case, the wild type average RMSF data is in the first column of if the matrix "Avg". If mutant had a larger fluctuation this difference would be positive and in the case there was less movement in the mutant ligand this value would be negative. The standard deviations of the mutant variants were also calculated and appended into a matrix of standard deviation values. Lastly, code was written to generate a bar plot ligand RMSF differences compared to wild type.



```
%Calculate average differences between mutation and WT RMSF values
Mut1_Diff = Avg(:,2)-Avg(:,1);
Mut2_Diff = Avg(:,3)-Avg(:,1);
Mut3_Diff = Avg(:,4)-Avg(:,1);
diff = horzcat(Mut1_Diff,Mut2_Diff,Mut3_Diff);

%Calculte standard deviations of RMSF differences
Mut1_stdev = std(Mut1_diffs,0,2);
Mut2_stdev = std(Mut2_diffs,0,2);
Mut3_stdev = std(Mut3_diffs,0,2);
stdev = horzcat(Mut1_stdev,Mut2_stdev,Mut3_stdev);

%Generates a bar plot of RMSF difference values
figure
x = 1:38;
bar_graph = bar(x,diff);
%errorbar(x,diff,stdev) %figure out how to add errorbars
bar_graph(1).FaceColor='r';
bar_graph(2).FaceColor='b';
bar_graph(3).FaceColor='g';
title('Ligand RMSF Differences to Average WT RMSF')
xlabel('Atom Number')
ylabel('RMSF Difference')
legend('I84V Difference','V82F+I83V Difference','M46I+V82F+I84V Difference')
```

Figure 6. 8: Ligand RMSF Differences to WT Calculation and Bar Plot

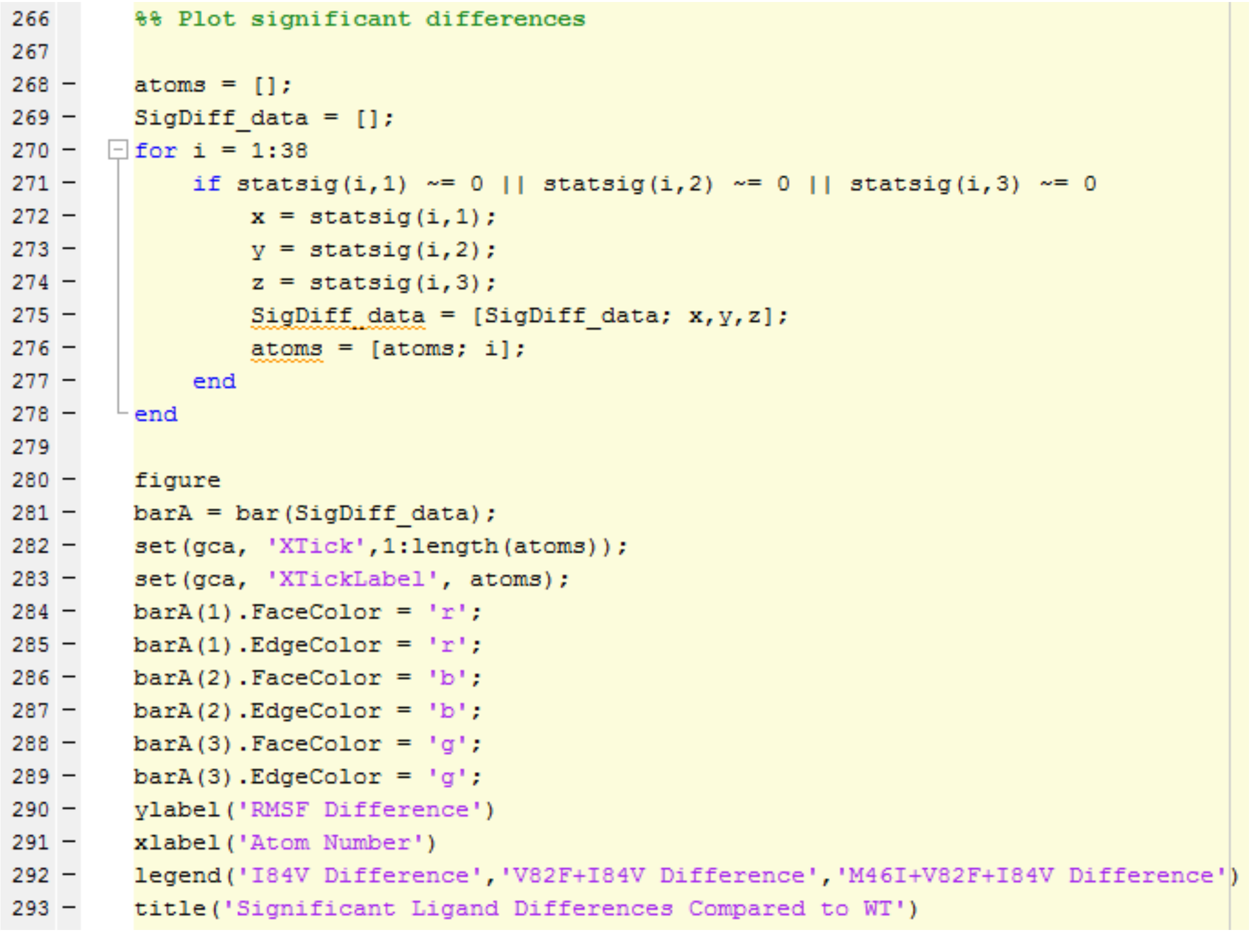
Through discussion with the sponsor, the team determined some differences in RMSF values to be insignificant. Significant differences are defined as an absolute difference value that is greater than the standard deviation of the three wild type replicate data points for each residue. This was determined through a for loop where pre-allocated vectors of zeros remained a zero value if the RMSF difference was between the negative and positive wild type standard deviation for each residue (Fig. 6.9). If the difference was outside these bounds, then the zero place holder would be changed to the RMSF difference.



```
%% Significant differences between mutation and WT
%Note: Significant difference is defined by having a
%difference compared to the wild type RMSF that is greater than the
%standard deviation of the 3 WT data points at each specific atom.

% Preallocate vectors
Mut1_statsig = ones(1,38);
Mut2_statsig = ones(1,38);
Mut3_statsig = ones(1,38);
WT_stdev = std(WT_Comp')';
for i = 1:38
    if (-WT_stdev(i)) < Mut1_Diff(i) && Mut2_Diff(i) >  WT_stdev(i)
        Mut1_statsig(i) = 0;
    else
        Mut1_statsig(i) = Mut1_Diff(i);
    end
    if (-WT_stdev(i)) < Mut2_Diff(i) && Mut2_Diff(i) >  WT_stdev(i)
        Mut2_statsig(i) = 0;
    else
        Mut2_statsig(i) = Mut2_Diff(i);
    end
    if (-WT_stdev(i)) < Mut3_Diff(i) && Mut3_Diff(i) >  WT_stdev(i)
        Mut3_statsig(i) = 0;
    else
        Mut3_statsig(i) = Mut3_Diff(i);
    end
end
statsig = [Mut1_statsig;Mut2_statsig;Mut3_statsig]';
```

Figure 6. 9: Determining Significant Ligand RMSF

Since some values were left as a zero in the “statsig” array, an additional for loop was written to remove the condition of a zero (Fig. 6.10). This for loop appends to a new array if at least one of the mutations has a difference that is not equal to zero. If all three mutation RMSF differences are 0 in the “statsig” array, this for loop moves on to the next atom until reaching 38 atoms. The new data array is then plotted as a bar graph with the atoms array as the tick label. Lastly, the ligand RMSF script prints the significant differences as a 2 column matrix with the atom number in the first and the difference value in the second column.



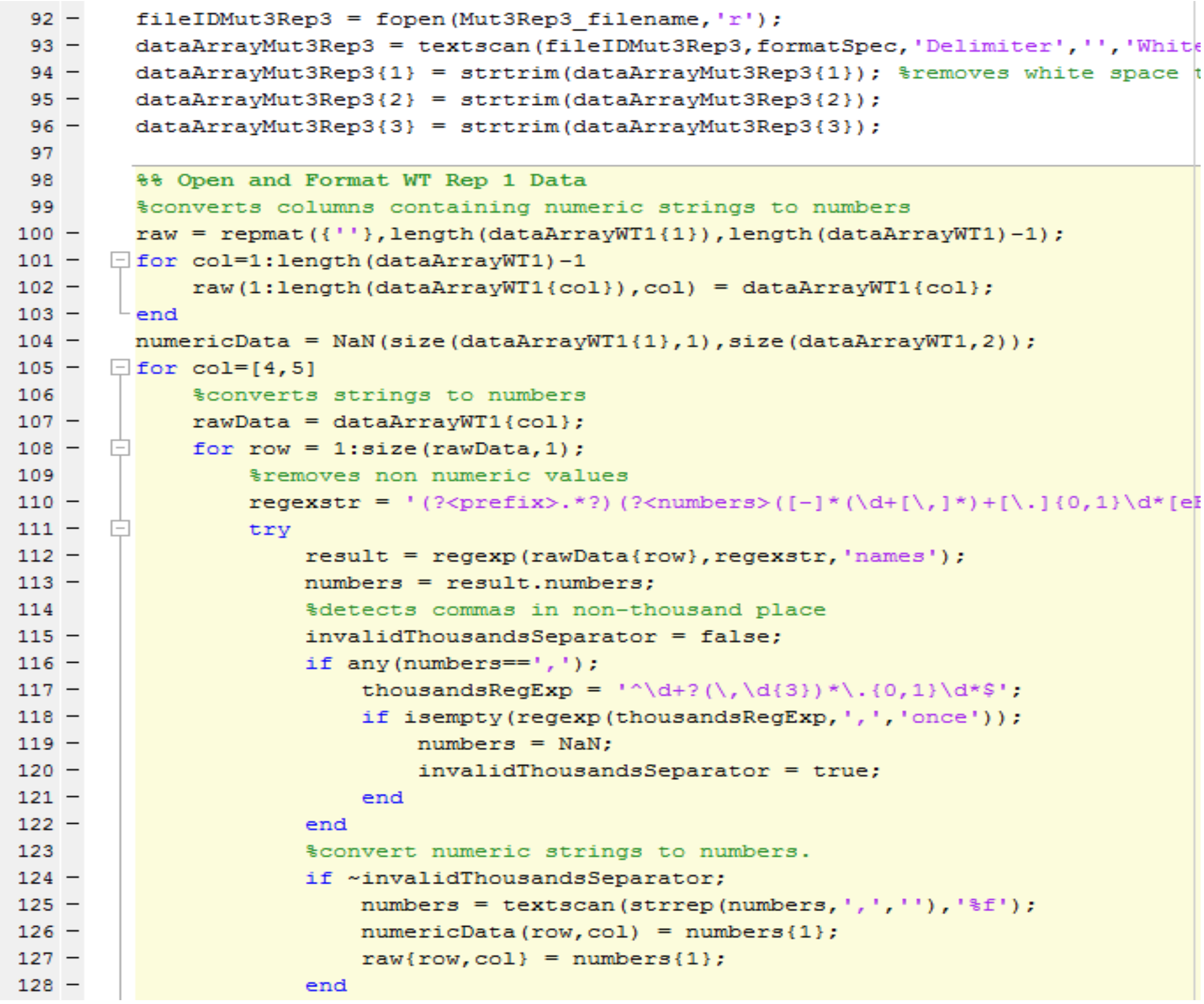
```
%% Plot significant differences

atoms = [];
SigDiff_data = [];
for i = 1:38
    if statsig(i,1) ~= 0 || statsig(i,2) ~= 0 || statsig(i,3) ~= 0
        x = statsig(i,1);
        y = statsig(i,2);
        z = statsig(i,3);
        SigDiff_data = [SigDiff_data; x,y,z];
        atoms = [atoms; i];
    end
end

figure
barA = bar(SigDiff_data);
set(gca, 'XTick',1:length(atoms));
set(gca, 'XTickLabel', atoms);
barA(1).FaceColor = 'r';
barA(1).EdgeColor = 'r';
barA(2).FaceColor = 'b';
barA(2).EdgeColor = 'b';
barA(3).FaceColor = 'g';
barA(3).EdgeColor = 'g';
ylabel('RMSF Difference')
xlabel('Atom Number')
legend('I84V Difference','V82F+I84V Difference','M46I+V82F+I84V Difference')
title('Significant Ligand Differences Compared to WT')
```

Figure 6. 10: Removing zeros from significant differences

6.2.4 Van der Waals Interactions

A script for van der Waals interactions was written in MATLAB to process interaction data from a file generated through the Schiffer Laboratory's GFortran script. The GFortran script imports the data from the simulation and outputs a file with the extension, vdwen, that has varying columns and rows depending on the frame. The primary challenge with this file type is accounting for rows that are used as sub headers for each frame and a varying amount of residues within the frame.

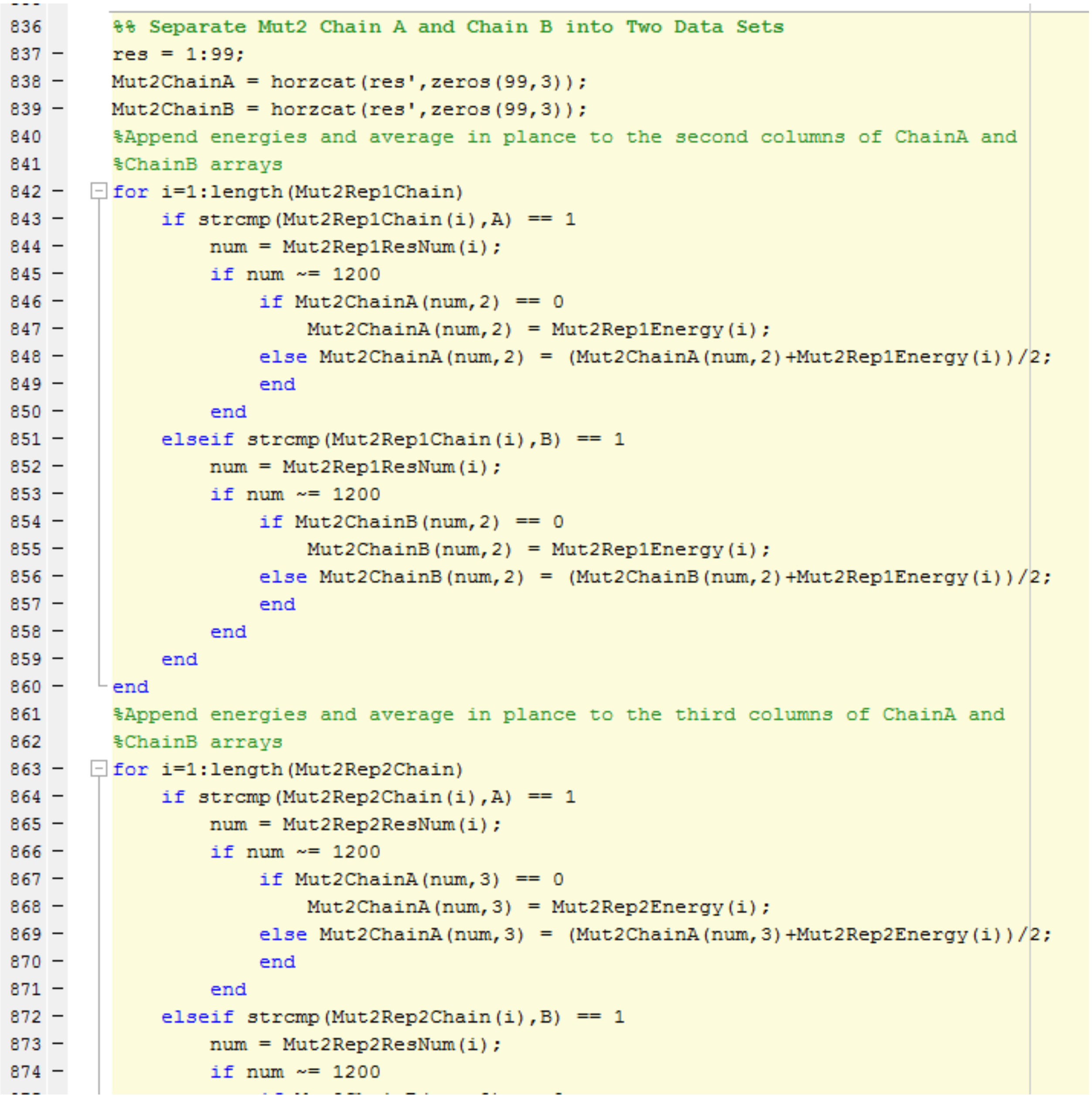
The files were first imported based on their file location and data extracted using textscan. MATLAB's string trim function, "strtrim", was used to remove the white space surrounding the numbers and characters in the file. Numeric strings were then converted into numbers and non-numeric values, such as "Frame n of x" and residue names, were removed. Data was then split into proper cell columns and non-numeric cells were overwritten with “NaN” or not a number (Fig. 6.11). Cell columns were then converted to matrices and temporary variables were cleared to preserve program memory and increase script speed. The same textscan steps and for loops were conducted to initialize all 12 data files.



```
fileIDMut3Rep3 = fopen(Mut3Rep3_filename,'r');
dataArrayMut3Rep3 = textscan(fileIDMut3Rep3,formatSpec,'Delimiter','','White
dataArrayMut3Rep3{1} = strtrim(dataArrayMut3Rep3{1}); %removes white space t
dataArrayMut3Rep3{2} = strtrim(dataArrayMut3Rep3{2});
dataArrayMut3Rep3{3} = strtrim(dataArrayMut3Rep3{3});

%% Open and Format WT Rep 1 Data
%converts columns containing numeric strings to numbers
raw = repmat({''},length(dataArrayWT1{1}),length(dataArrayWT1)-1);
for col=1:length(dataArrayWT1)-1
    raw(1:length(dataArrayWT1{col}),col) = dataArrayWT1{col};
end
numericData = NaN(size(dataArrayWT1{1},1),size(dataArrayWT1,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayWT1{col};
    for row = 1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eE
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
```

Figure 6. 11: van der Waals script loading .vdwen files and extracting data

Data for each mutation was contained in three columns of interest: Chain, ResNum or residue number and Energy. The data was divided into chain A and B for each mutation through a for loop that read each line of data for the length of the chain array (Fig. 6.12). First, the for loop determined which chain the energy value belonged to based on whether there was an “A” or “B” string as the value in the chain data array. If the number was not equal to 1200, the placeholder for the rows of frame sub headers, the for loop continued. If the mutation/wild type data matrix was empty, the energy was appended into the matrix. In the figure below, this data matrix is either Mut1ChainA or Mut1ChainB. If there was a previous energy in the matrix for that residue, the energies were averaged in place. The data matrix consisted of four columns: the residue number, averaged replicate 1 energies, averaged replicate 2 energies, and averaged replicate 3 energies. Similar for loops were conducted for each mutation and wild type.



```
%% Separate Mut2 Chain A and Chain B into Two Data Sets
res = 1:99;
Mut2ChainA = horzcat(res',zeros(99,3));
Mut2ChainB = horzcat(res',zeros(99,3));
%Append energies and average in plance to the second columns of ChainA and
%ChainB arrays
for i=1:length(Mut2Rep1Chain)
    if strcmp(Mut2Rep1Chain(i),A) == 1
        num = Mut2Rep1ResNum(i);
        if num ~= 1200
            if Mut2ChainA(num,2) == 0
                Mut2ChainA(num,2) = Mut2Rep1Energy(i);
            else Mut2ChainA(num,2) = (Mut2ChainA(num,2)+Mut2Rep1Energy(i))/2;
            end
        end
    elseif strcmp(Mut2Rep1Chain(i),B) == 1
        num = Mut2Rep1ResNum(i);
        if num ~= 1200
            if Mut2ChainB(num,2) == 0
                Mut2ChainB(num,2) = Mut2Rep1Energy(i);
            else Mut2ChainB(num,2) = (Mut2ChainB(num,2)+Mut2Rep1Energy(i))/2;
            end
        end
    end
end
%Append energies and average in plance to the third columns of ChainA and
%ChainB arrays
for i=1:length(Mut2Rep2Chain)
    if strcmp(Mut2Rep2Chain(i),A) == 1
        num = Mut2Rep2ResNum(i);
        if num ~= 1200
            if Mut2ChainA(num,3) == 0
                Mut2ChainA(num,3) = Mut2Rep2Energy(i);
            else Mut2ChainA(num,3) = (Mut2ChainA(num,3)+Mut2Rep2Energy(i))/2;
            end
        end
    elseif strcmp(Mut2Rep2Chain(i),B) == 1
        num = Mut2Rep2ResNum(i);
        if num ~= 1200
```

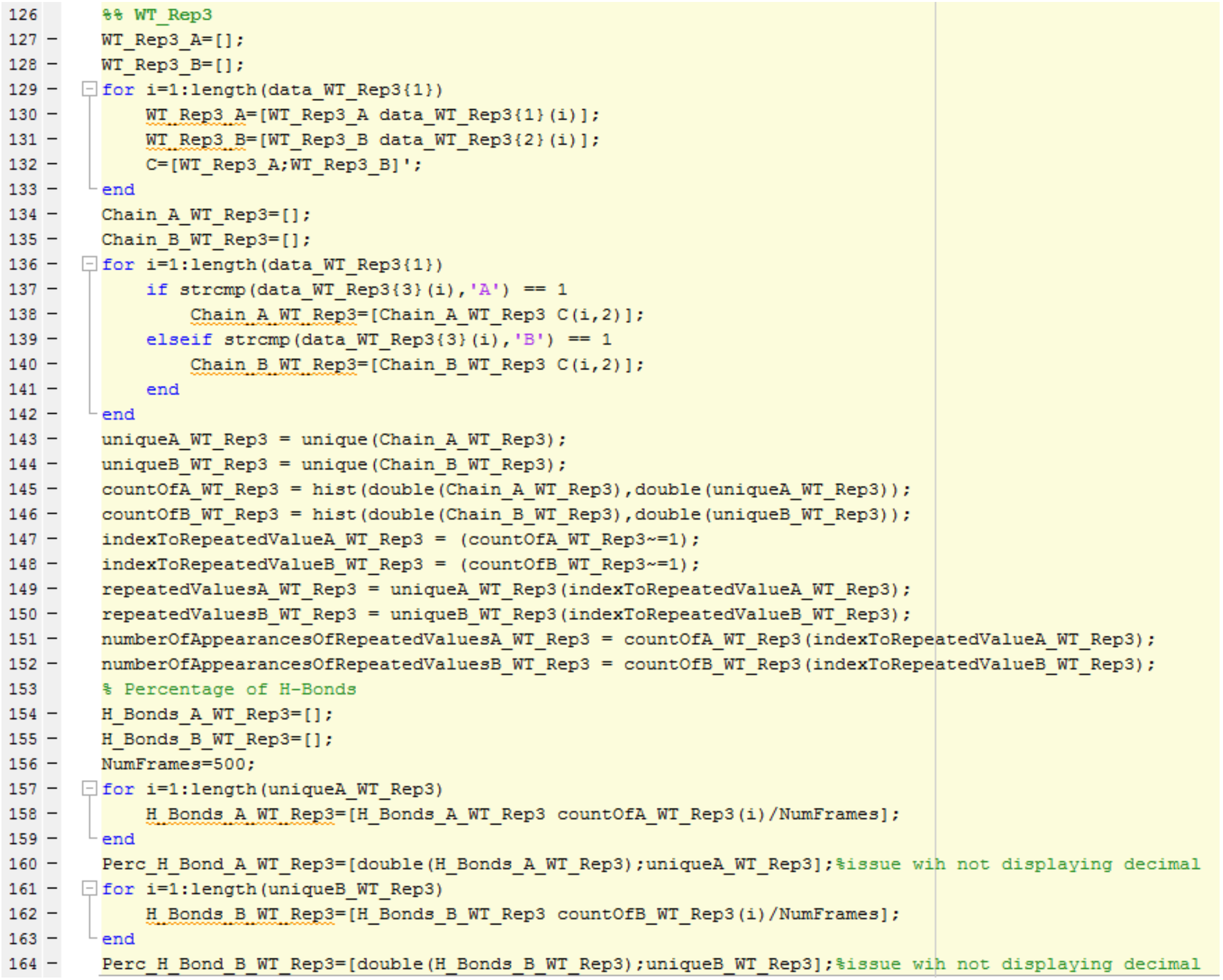
Figure 6. 12: Separating van der Waals energies into chain A and B data sets and averaging in place

Bar graphs for each mutation were generated and the average van der Waals energies of each residue across the three replicates was calculated.

6.2.5 Hydrogen Bonds

A script for hydrogen bond percentages was written to accommodate comma separated value files. The script utilizes .csv files that are directly outputed from the MD simulations in Desmond. These .csv files contains a list of HIV-1 protease residues that exhibit hydrogen bonding at each time point during the simulation. The goal of hydrogen bond analysis is to determine any changes between active site interactions when mutations are present.

In order to achieve this, each mutation replicate was imported into the program. The percentage of hydrogen bonding was determined by scanning through each file to determine how many times the similar residues appear during the 500 frames. The amount of times each specific residue appeared was stored in an array. The total appearances were then divided by the total amount of frames to determine the hydrogen bond percentage throughout the simulation.



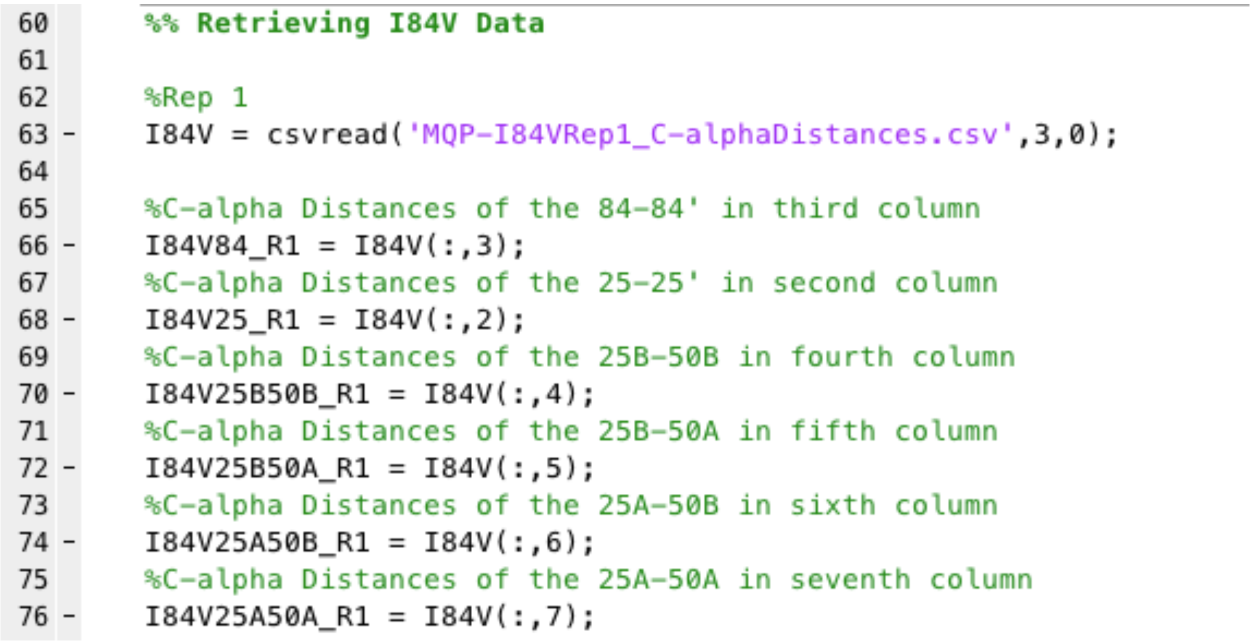
```matlab
%% WT_Rep3
WT_Rep3_A=[];
WT_Rep3_B=[];
for i=1:length(data_WT_Rep3{1})
    WT_Rep3_A=[WT_Rep3_A data_WT_Rep3{1}(i)];
    WT_Rep3_B=[WT_Rep3_B data_WT_Rep3{2}(i)];
    C=[WT_Rep3_A;WT_Rep3_B]';
end
Chain_A_WT_Rep3=[];
Chain_B_WT_Rep3=[];
for i=1:length(data_WT_Rep3{1})
    if strcmp(data_WT_Rep3{3}(i),'A') == 1
        Chain_A_WT_Rep3=[Chain_A_WT_Rep3 C(i,2)];
    elseif strcmp(data_WT_Rep3{3}(i),'B') == 1
        Chain_B_WT_Rep3=[Chain_B_WT_Rep3 C(i,2)];
    end
end
uniqueA_WT_Rep3 = unique(Chain_A_WT_Rep3);
uniqueB_WT_Rep3 = unique(Chain_B_WT_Rep3);
countOfA_WT_Rep3 = hist(double(Chain_A_WT_Rep3),double(uniqueA_WT_Rep3));
countOfB_WT_Rep3 = hist(double(Chain_B_WT_Rep3),double(uniqueB_WT_Rep3));
indexToRepeatedValueA_WT_Rep3 = (countOfA_WT_Rep3~=1);
indexToRepeatedValueB_WT_Rep3 = (countOfB_WT_Rep3~=1);
repeatedValuesA_WT_Rep3 = uniqueA_WT_Rep3(indexToRepeatedValueA_WT_Rep3);
repeatedValuesB_WT_Rep3 = uniqueB_WT_Rep3(indexToRepeatedValueB_WT_Rep3);
numberOfAppearancesOfRepeatedValuesA_WT_Rep3 = countOfA_WT_Rep3(indexToRepeatedValueA_WT_Rep3);
numberOfAppearancesOfRepeatedValuesB_WT_Rep3 = countOfB_WT_Rep3(indexToRepeatedValueB_WT_Rep3);
% Percentage of H-Bonds
H_Bonds_A_WT_Rep3=[];
H_Bonds_B_WT_Rep3=[];
NumFrames=500;
for i=1:length(uniqueA_WT_Rep3)
    H_Bonds_A_WT_Rep3=[H_Bonds_A_WT_Rep3 countOfA_WT_Rep3(i)/NumFrames];
end
Perc_H_Bond_A_WT_Rep3=[double(H_Bonds_A_WT_Rep3);uniqueA_WT_Rep3];%issue wih not displaying decimal
for i=1:length(uniqueB_WT_Rep3)
    H_Bonds_B_WT_Rep3=[H_Bonds_B_WT_Rep3 countOfB_WT_Rep3(i)/NumFrames];
end
Perc_H_Bond_B_WT_Rep3=[double(H_Bonds_B_WT_Rep3);uniqueB_WT_Rep3];%issue wih not displaying decimal
```

Figure 6. 13: Portion of hydrogen bonding percentage

6.2.6 Alpha-Carbon Distances

A script for organizing C-alpha distances was developed using MATLAB. The C-alpha distances data files were .csv files for each mutation. Each comma separated value file had the time steps, which designated each frame of the 500 frames for a 100 nanosecond simulation, as the first column. The six following columns were for each type of measurement that included the distances between residues 25 to 25’, 84 to 84’, 25 to 50, 25’ to 50’, 25’ to 50, and 25 to 50’. The residue numbers without the prime symbol are the residues of chain A and the residue numbers with the prime symbol are residues of chain B.

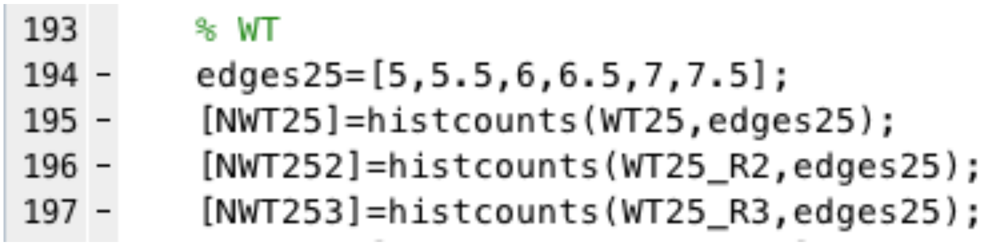
In the MATLAB script, the data from the comma separated value files for each replicate of each mutation were imported, which an example is seen in Figure 6.14.



```
%% Retrieving I84V Data

%Rep 1
I84V = csvread('MQP-I84VRep1_C-alphaDistances.csv',3,0);

%C-alpha Distances of the 84-84' in third column
I84V84_R1 = I84V(:,3);
%C-alpha Distances of the 25-25' in second column
I84V25_R1 = I84V(:,2);
%C-alpha Distances of the 25B-50B in fourth column
I84V25B50B_R1 = I84V(:,4);
%C-alpha Distances of the 25B-50A in fifth column
I84V25B50A_R1 = I84V(:,5);
%C-alpha Distances of the 25A-50B in sixth column
I84V25A50B_R1 = I84V(:,6);
%C-alpha Distances of the 25A-50A in seventh column
I84V25A50A_R1 = I84V(:,7);
```

Figure 6. 14: I84V C-alpha distances

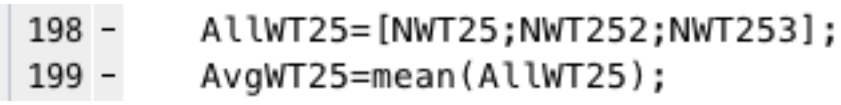
Once all the data for each replicate and mutation were imported, the histograms of each C-alpha measurement were developed. The bins for the histograms were made according to the data. For example, the C-alpha distances between the catalytics residues (25-25') fell in a range of 5 Angstroms to 7.5 Angstroms.The bins were separated between a value of 0.5 Angstroms. Next, histograms were created for each replicate of each HIV-1 protease variant using the bins specified. An example for wild type C-alpha distances between the catalytic residues are shown below.



```
% WT
edges25=[5,5.5,6,6.5,7,7.5];
[NWT25]=histcounts(WT25,edges25);
[NWT252]=histcounts(WT25_R2,edges25);
[NWT253]=histcounts(WT25_R3,edges25);
```

Figure 6. 15: C-alpha Histograms

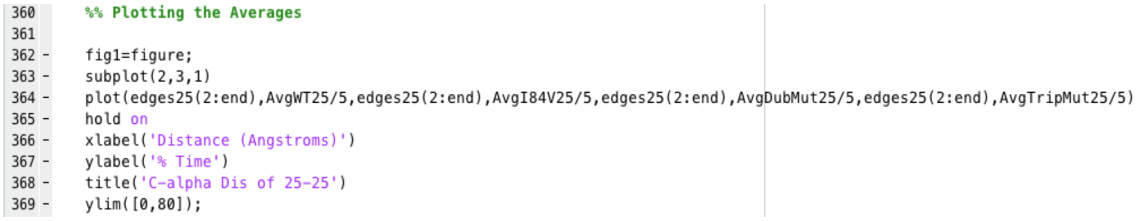
The average of each of the variant's histogram was taken, which is seen in Figure 6.16.



```
AllWT25=[NWT25;NWT252;NWT253];
AvgWT25=mean(AllWT25);
```

Figure 6. 16: Histogram averages

The averages of each variant's C-alpha distances were then compiled together for comparison between the different variants. Six graphs were created for each type of C-alpha distance. To illustrate, the script for the graph of the C-alpha distances between the catalytic residues are shown in Figure 6.17.



```
%% Plotting the Averages

fig1=figure;
subplot(2,3,1)
plot(edges25(2:end),AvgWT25/5,edges25(2:end),AvgI84V25/5,edges25(2:end),AvgDubMut25/5,edges25(2:end),AvgTripMut25/5)
hold on
xlabel('Distance (Angstroms)')
ylabel('% Time')
title('C-alpha Dis of 25-25')
ylim([0,80]);
```

Figure 6. 17: C-alpha distances plots compilation

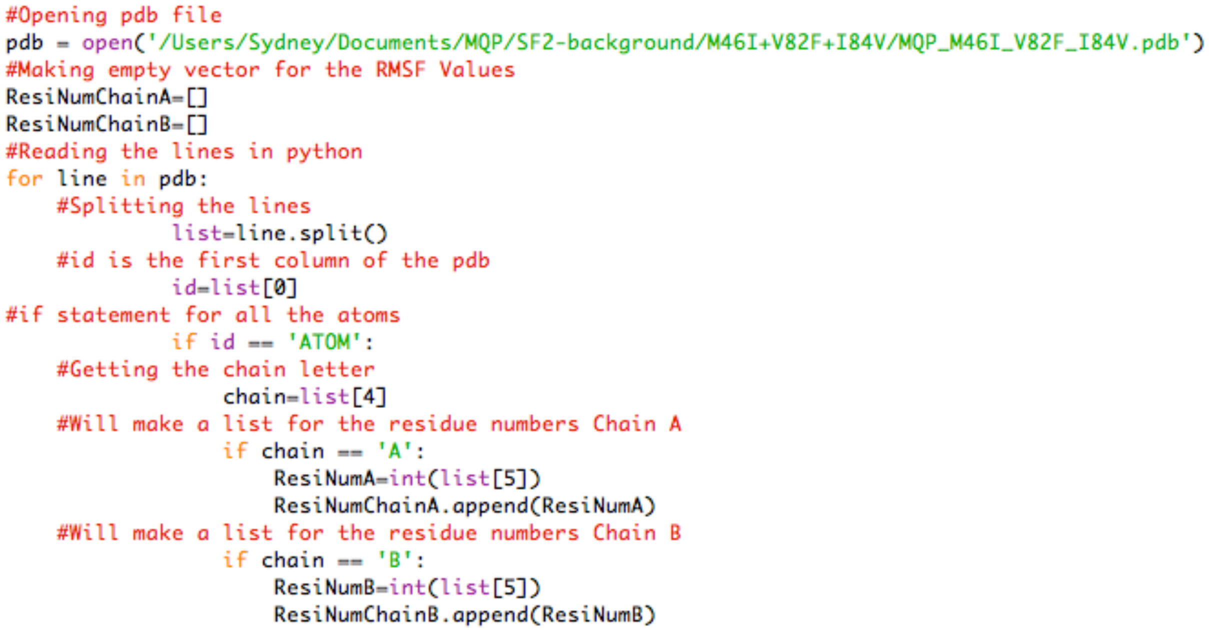
The graphs of the C-alpha distances were put into a figure of 6 different graphs, which is shown C-alpha Distances Results in Chapter 5. The script developed for the C-alpha Distances can be further modified for different mutations, number of replicates, and different input files.

6.2.7 Modification of PDB File

To display the protein RMSF values visually on the cartoon display of the protein, the PDB files needed to be modified. The beta-factors of the PDB file can be displayed on a color spectrum in the computer program PyMol, which is a three-dimensional molecular program. The beta-factors can be modified to any values desired. However, the PDB file's structure is unique and therefore cannot be easily imported and modified using MATLAB. The goal of the Python script was used to read the PDB file and create an output file of the protein RMSF values that correspond the protein RMSF value for each atom.

Protein RMSF values are designated to each residue, but the PDB file has a beta-factor for each atom. Therefore, the protein RMSF value needs to be repeated for each atom of the residue, which each type of residue has a different number of atoms in it. To obtain the number of atoms for each residue, the

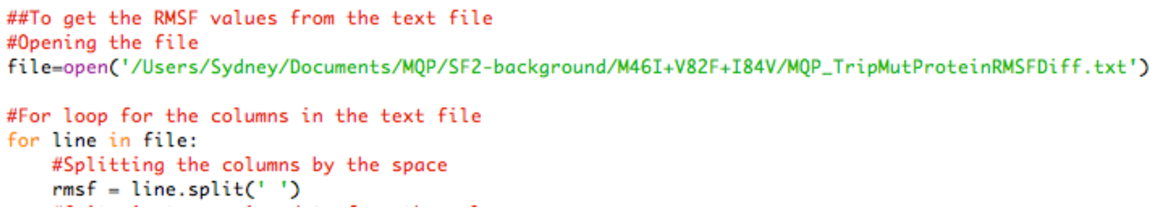
Python script counts how many atoms are in each residue. The reading of the PDB file and obtaining the number of atoms for each residue is shown in Figure 6.18.



```
#Opening pdb file
pdb = open('/Users/Sydney/Documents/MQP/SF2-background/M46I+V82F+I84V/MQP_M46I_V82F_I84V.pdb')
#Making empty vector for the RMSF Values
ResiNumChainA=[]
ResiNumChainB=[]
#Reading the lines in python
for line in pdb:
    #Splitting the lines
             list=line.split()
    #id is the first column of the pdb
             id=list[0]
#if statement for all the atoms
             if id == 'ATOM':
    #Getting the chain letter
                 chain=list[4]
    #Will make a list for the residue numbers Chain A
                 if chain == 'A':
                     ResiNumA=int(list[5])
                     ResiNumChainA.append(ResiNumA)
    #Will make a list for the residue numbers Chain B
                 if chain == 'B':
                     ResiNumB=int(list[5])
                     ResiNumChainB.append(ResiNumB)
```

Figure 6. 18: Beta Factor Modification

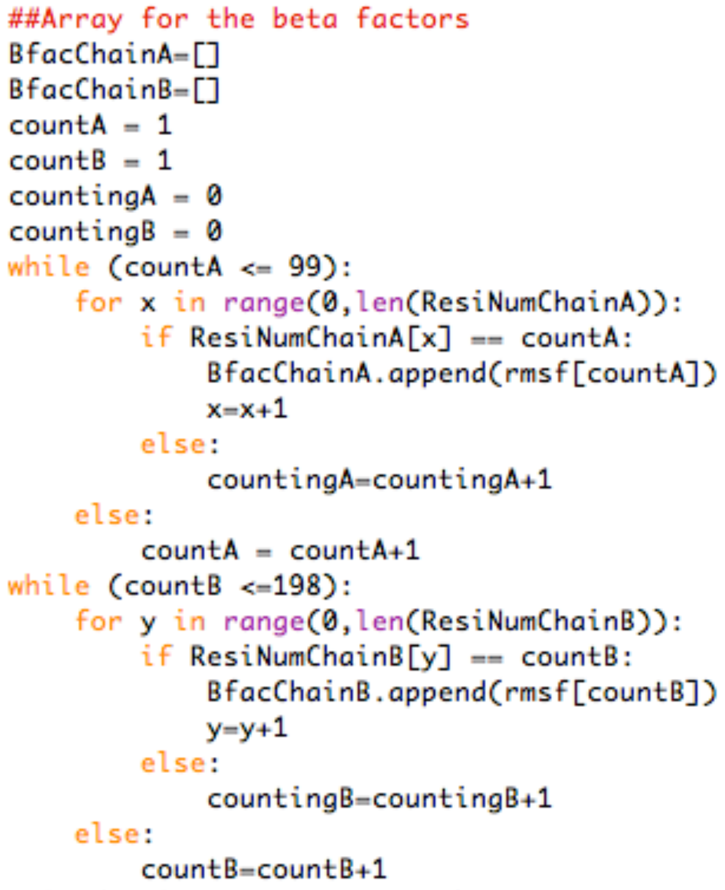
This section of the Python script reads the pdb file using the function "open." Empty arrays for chain A and chain B are defined because they will create a list of the residue numbers of each atom. For example, if residue 12 has eight atoms, then the number 12 will appear eight times in the array. Next, each line is read in the pdb. There are several column in the PDB file, so the "line.split()" function splits the line into columns. The script then looks at the first column with the function "list[0]" and checks if it says 'ATOM' with if statement "if id == 'ATOM.' There are several other atoms in the PDB file, such as water molecules, but atoms of the protein are labeled 'ATOM'. Then the chain letter in column 4 is read and if it is an 'A', then the residue number in column 5 is copied into the chain A array, and respectively if it is a 'B.' Along with having the PDB file read and the arrays being filled, the RMSF values need to be imported into the function also. This section of the script is shown in Figure 6.19.



```
##To get the RMSF values from the text file
#Opening the file
file=open('/Users/Sydney/Documents/MQP/SF2-background/M46I+V82F+I84V/MQP_TripMutProteinRMSFDiff.txt')

#For loop for the columns in the text file
for line in file:
    #Splitting the columns by the space
    rmsf = line.split(' ')
```

Figure 6. 19: Protein RMSF Text File Import

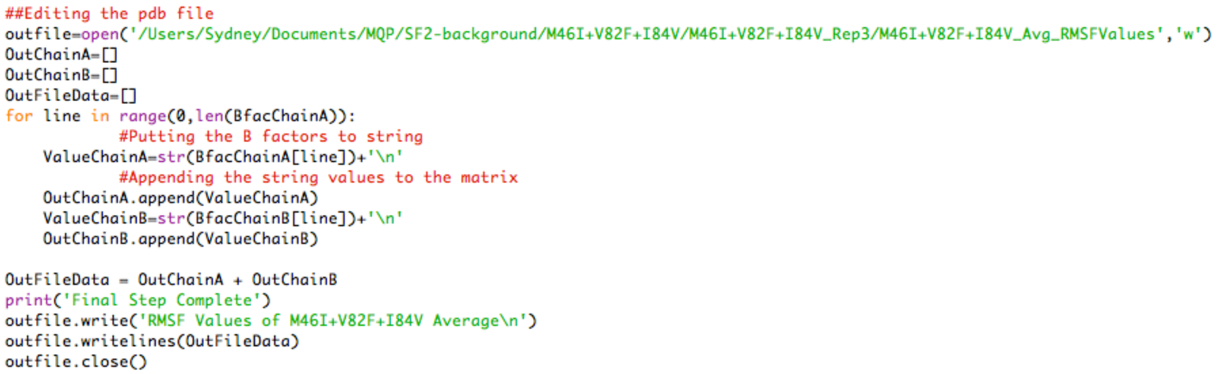
The data of the file were imported with the function “open” and each line of the file, which was the protein RMSF values, were imported into an array named “rmsf”. The next step of the script was to create a new array of numbers that put the protein RMSF value with its corresponding atom, which is shown in Figure 6.20.



```
##Array for the beta factors
BfacChainA=[]
BfacChainB=[]
countA = 1
countB = 1
countingA = 0
countingB = 0
while (countA <= 99):
    for x in range(0,len(ResiNumChainA)):
        if ResiNumChainA[x] == countA:
            BfacChainA.append(rmsf[countA])
            x=x+1
        else:
            countingA=countingA+1
    else:
        countA = countA+1
while (countB <=198):
    for y in range(0,len(ResiNumChainB)):
        if ResiNumChainB[y] == countB:
            BfacChainB.append(rmsf[countB])
            y=y+1
        else:
            countingB=countingB+1
    else:
        countB=countB+1
```

Figure 6. 20: Data array corresponding to PDB file

Two new arrays for the protein RMSF values that correspond to the order and number of atoms in the PDB file are labeled as “BfacChainA” for chain A and “BfacChainB” for chain B. The while loop goes through the list of residue numbers created from pdb file, which repeats the same residue number for the number of atoms in the residue, and matches the correct RMSF value for the residue. The resulting lists, for each chain, have the respective RMSF value for each atom. The last section of the script outputs these lists into a text file, which is shown in Figure 6.21.



```
##Editing the pdb file
outfile=open('/Users/Sydney/Documents/MQP/SF2-background/M46I+V82F+I84V/M46I+V82F+I84V_Rep3/M46I+V82F+I84V_Avg_RMSFValues','w')
OutChainA=[]
OutChainB=[]
OutFileData=[]
for line in range(0,len(BfacChainA)):
            #Putting the B factors to string
    ValueChainA=str(BfacChainA[line])+'\n'
            #Appending the string values to the matrix
    OutChainA.append(ValueChainA)
    ValueChainB=str(BfacChainB[line])+'\n'
    OutChainB.append(ValueChainB)

OutFileData = OutChainA + OutChainB
print('Final Step Complete')
outfile.write('RMSF Values of M46I+V82F+I84V Average\n')
outfile.writelines(OutFileData)
outfile.close()
```

Figure 6. 21: Outputs Protein RMSF for each atom of PDB

An outfile is created with the function write, which is signified by “w.” The protein RMSF values that will be exported are converted into a column of strings with the for loop and inserted in the new arrays labeled as “OutChainA” for chain A and “OutChainB” for chain B. Next, the new arrays are put together in another array labeled as “OutFileData.” The combined array is exported into the outfile with the command “writelines” and closed to successfully create the text file.

The values in the outfile file are copied into the PDB file in the column of the beta-factors. The newly modified PDB file can then be uploaded in PyMol and colored according to the protein RMSF values, which is referenced as the beta-factors in the PyMol software.

6.3 Relevant Industry Standards Met

Our project was research based within a graduate school laboratory. The molecular dynamic approach to computationally evaluating protein dynamics is utilized in many protein based research labs. However, there is minimal regulation between different laboratories due to differences in molecular dynamic software and protocols. Currently, there are no overarching regulations for educational based research using molecular dynamic simulations. Although, commercial biologics companies are held to an ISO standard. Industry is limited to regulation from the biomimetics ISO standard 18458:2015 [58]. This

]

ISO standard outlines common programming languages, a terminology framework, and provides a proper definition of a biomimetics systems. Standards for this type of computational analysis are limited due to the recent advances in this technology.

6.4 Design Considerations

6.4.1 Economics

Our project is based on the computational analysis of drug resistance patterns within viral HIV-1 protease. The project's goal was to identify any common patterns that resulted in proper protein function in the presence of protease inhibitors. The project was strictly theoretical research which could be applied to protease inhibitor design. However, if protease inhibitors were to be designed based off this research the drug would be substantially more expensive than the average HIV-1 treatment. HIV-1 treatment for affected patients can cost upwards of $23,000 annually and can be higher depending on using name brands [9]. Currently, the most potent protease inhibitor commercially available is Darunavir which typically costs $1,500 per month [30]. Protease inhibitors based off this research would exhibit potency greater than Darunavir, which would make this treatment more expensive.

6.4.2 Environmental Impact

Our project is completely computationally based only utilizing computers for analysis. Although convenient for the data analysis, computers contribute to a greater environmental impact than expected. Firstly, the manufacturing process consumes high amounts of electricity and consists of processing heavy metals and toxic chemicals. Secondly, once obtained and in the lab, the computers used to run simulations use high amounts of electricity and are inefficient due to their age. Lastly, if the computers used to complete this project are upgraded in the future, there is significant environmental impact involved in their disposable.

6.4.3 Societal Influence

Promising drug resistance related research, including this project, can provide hope within society as a whole. For many generations of human existence the contraction of a disease, such as HIV, inevitably resulted in death. Current problems facing modern drug use for disease treatment is resistance, which can provoke fear across a society. Being able to report findings that show drug resistance can be determined and accounted for in drug production is very hopeful. This type of research will allow ordinary people with peace of mind if they contract any type of viral disease.

6.4.4 Political ramifications

Some political ramifications that might concern this project is the funding and which types of HIV we examine drug resistance for. The lab's funding might have an obligation to research certain types of HIV. In our case, it's HIV-1 that affects the majority of the HIV population within the United States and western world.

6.4.5 Ethical concern

This project only focuses on one serotype of HIV and is neglecting other forms that are prevalent throughout third world countries. Although we are focusing on the majority of the population with HIV (HIV-1), we do not consider the others.

6.4.6 Health and safety

Specifically this project does not pose any health or safety issues. The research and analysis are all executed on computer programs.

However, there are several health and safety issues with researching drug resistance. Our project requires a crystal structure of the protein, which is the PDB for the molecular software. This PDB gives different energy parameters and atom coordinates that dictate the protein's behavior in the simulations. The making of the crystal structure poses some health and safety risks from obtaining the protein and manipulating the protein into growing a crystal, as the operator is in contact with the virus. This crystal is then examined by Nuclear Magnetic Resonance imaging (NMR) and solved, with molecular software, to determine the atomic structure and energies of the protein. Although there is associated risks with developing the crystal structure, overall, computational approaches pose significantly reduce health related risks.

6.4.7 Manufacturability

This project is not manufacturable, rather is intended to be used as a research resource. The program written in MATLAB with a supplemental python script that could be reproduced and altered towards specific research aims. Manufacturing could be considered if this research was to be conducted with collaborative labs to provide consistent experimentation.

6.4.8 Sustainability

The analysis program produced through this project is sustainable, as it can be easily adapted towards specific needs. Code variables were kept vague intentionally so it can be a sustainable analysis solution used for a variety of mutations and tests. Another factor that plays a role in sustainability is software updates, both in terms of molecular modeling programs and scripting languages.

7.0 Discussion

In an effort to elucidate potential resistance patterns that specific mutations to HIV-1 protease create, the team performed molecular dynamic studies. Further analysis was conducted using information obtained from molecular dynamic simulations separated into two categories: inhibitor movement analysis and inhibitor interactions analysis. Inhibitor movement analysis quantifies the effectiveness of protein and ligand binding through molecular movement. Protein-ligand RMSD, protein RMSF, ligand RMSF, and alpha carbon distances were the dynamic factors investigated that attribute to drug resistance. However, inhibitor interactions analysis quantifies the effectiveness of protein and ligand binding through atomic interactions. van der Waals and hydrogen bond percentages were the interactive factors investigated that attribute to drug resistance.

7.1 Inhibitor Movement Analysis

7.1.1 Protein-Ligand RMSD

Protein-Ligand RMSD was the first analysis that needed to be conducted before progressing to further stages of analysis. This dynamic attribute between the protein and ligand directly correlates to protein stability, thus feasibility *in vivo.* If the RMSD values are either too high or low, information gathered from the simulation is irrelevant due to the lack of equilibrium. The Schiffer Lab has determined that relevant information can be obtained from HIV-1 protease RMSD values that equilibrate between roughly 1-1.5 angstroms over the course of 100 nanosecond simulations. The average protein-ligand RMSD plots for wild type, I84V, V82F+I84V, and M46I+V82F+I84V variants all met this requirement. All further analysis was determined to be valid.

7.1.2 Protein RMSF

The protein RMSF was analyzed to determine the average protein mobility across the simulation. This takes the dynamic nature of the protein into account, allowing residues with the most movement and fluctuation from their starting position to be easily identified. With the exception of V82F+I84V chain B, residues 1 and 2, RMSF values of each replicate were consistent, varying slightly due to natural *in vivo* movement. This supports our original choice of computational analysis and molecular dynamics, as the differences depicted in each RMSF replicate demonstrate the dynamic process is being accurately simulated.

When comparing average RMSF values, each mutant variant followed a similar trend, with some difference observed at residues 1-5, 15-18, 50-60, 70-75, 99-104 (chain B, residues 1-5), and 150-160. The differences seen in residues 1-5 in both chains is to be expected, as that is the dimerization region, which is highly motile. The fluctuation observed in the 10's loop, residues 16-18, was present in all three mutations. I84V and V82F+I84V had more fluctuation through the 10s loop compared to the wild type. Conversely, M46I+V82F+I84V exhibited a more rigid 10's loop with a negative change in angstroms compared to wild type. Fluctuation in residues in the flap regions, residues 50-60 and 150-160, were observed in all three variants. M46I+V82F+I84 had the greatest change in fluctuation compared to wild type, followed by I84V then V82F+I84V. Interestingly, the change in RMSF in V82F+I84V compared to wild type was not significant for these residues, although is visible when comparing the heatmaps. This is most likely caused by the variation among the wild type values for those residues, as significance was determined through wild type standard deviation. The flap region is expected to be a very dynamic region of the protein and this may be a source of possible error, as the change in RMSF for V82F+I84V may be greater than shown depending on the variation in wild type replicates. The final residues where a significant difference in RMSF was observed was in residues 70-75, the back of the beta sheet in chain A. Although difficult to observe on the heat maps, there is fluctuation through this area, especially in V82F+I84V. In V82F+I84V, at residue 70, the values are significantly negative, and change to significantly positive the next residue. This varies from what was observed in I84V and M46I+V82F+I84V, which had a negative RMSF difference compared to wild type.

Lastly, RMSF values of each variant and wild type were summed. V82F+I84V had the lowest total RMSF, however, when comparing to the protein RMSF significant differences bar graph, the majority of the RMSF values were greater than the wild type RMSF, giving a negative RMSF difference.

Although in some instances M46I+V82F+I84V had the greatest difference from wild type, when compared to the summed RMSF values, it most closely matched wild type. This suggests rigidity may play a role in conferring drug resistance, as V82F+I84V was the most rigid structure over the 500ns but had the greatest total difference to wild type.

7.1.3 Ligand RMSF

Ligand RMSF was analyzed to determine the effects of the mutant variants on inhibitor mobility. Average ligand RMSF follows a general trend with the wild type having the lowest fluctuation, followed by I84V, then V82F+I84V, and M46I+V82F+I84V with the greatest fluctuation. The most significant differences from wild type occur at atom numbers 9-19, 25 and 26, and 32-38. Atoms 9-19 include two double bonded oxygen atoms at numbers 9 and 10 to a sulfur atom and an oxygen bound to a carbon atom at 18. The fluctuation in these atoms was primarily observed in M46I+V82+I84V. V82F+I84V had one significant difference compared to wild type in this atom range at atom 15, the carbon oxygen is bound to. However, at atoms 25 and 26, V82F+I84V is the only variant with a significant RMSF difference. The most fluctuation was observed in atoms 32-38, with M46I+V82F+I84V having the greatest fluctuation, more than double that of I84V and V82F+I84V at atoms 34, 35, 36, and 37.

The RMSF values of each variant were also summed and followed an increasing trend with wild type having the lowest total RMSF and M46I+V82F+I84V the greatest. When considering the difference to wild type bar graph, the same pattern is followed with very few instances of a negative ligand RMSF difference.

7.1.4 Alpha Carbon Distances

The carbon-alpha distances between several residues across the active site of all variants of HIV-1 protease were measured and analyzed. These distances were between residues 25-25’, 84-84’, 25-50, 25-50’, 25’-50, and 25’-50’. The average distances between the residues were calculated for each variant. The average distances of the mutated variants were then compared to the wild type distances. The significant findings of this comparison is that the active site decreased in size with the mutated variants.

However, C-alpha distances of the M46I+V82F+I84V variant and the V82F+I84V variant were closer to the wild type, while the I84V C-alpha distances were smaller compared to wild type. This suggests that the active site size of the M46I+V82F+I84V variant and the V82F+I84V variant are closer to wild type, while the I84V active site size was smaller compared to wild type.

7.2 Inhibitor Interactions Analysis

7.2.1 Van der Waals

Van der Waals interactions are one of the intermolecular forces that keep the ligand attached to the protein. Thus, the changes in van der Waals energies due to mutations directly impact the interactions between the ligand and the protein. These energies were calculated between the ligand atoms and the protein residues for each HIV-1 protease variant. Between the different replicates of each variant of the protein, the difference in van der Waals energies were small and thus showing a consistent behavior between replicates.

To compare the mutated variants' van der Waals energies to the wild type, the average mutated variant van der Waals were subtracted from the average wild type. Several differences were seen between the mutated variants' van der Waals energies. However, many of these differences were insignificant, which we decided was less than a difference of 0.02 kcal/mol.

The significant differences between the mutated variants and the wild type for chain A were seen in residues 27-29,47-50, 81, and 84. Residues 27 to 29 are located at the bottom of the active site. The positive differences in residue 27 for all mutated variants suggest a stronger interaction between the ligand and the residue. V82F+I84V had the largest increase, with M46I+V82F+I84V following and I84V with the least increase. However, the negative differences in residue 28 and 29 for all mutated variants suggest a stronger interaction between the ligand and the residue.

Residues 47 to 50 are located at the top of the active site. For residue 47 and 48, the only variant that had a significant difference was V82F+I84V. V82F+I84V mutation caused a decrease in van der Waals energies between the ligand and residues 47 and 48. Both V82F+I84V and I84V mutations caused a decrease in van der Waals energies between the ligand and residue 49, but V82F+I84V had a much larger decrease. However, all mutated variants had a stronger interaction between the ligand and residue 50, which is the residue at the tip of the flap.

For residue 81, only the I84V mutation showed a significant difference of the van der Waals energies. The difference was negative and thus suggesting a decreased interaction between the ligand and the residue. Residue 84 was changed for each mutated variant in the team's project and each mutated variant showed a large decrease in van der Waals energies. This suggests that a mutation in residue 84 has a significant impact on the interaction between the ligand and the protein.

Van der Waals energy changes were also seen in Chain B. Significant differences were seen in residues 8, 25-28, 49, 50, 81, 82, and 84. Residue 8 is located below the active site, but the isoleucine amino acid reaches into the active site. Only mutated variant V82F+I84V had a significant difference in van der Waals energies between the ligand and the protein. The difference is negative, which suggests a stronger interaction between the ligand and the protein. Residues 25 to 28 of chain B also experienced significant difference in van der Waals energies compared to wild type. Residue 25 is the catalytic residue of the protein and only mutated variant V82F+I84V had a significant difference in van der Waals energies for that residue. V82F+I84V had a large increase in van der Waals energies, suggesting that the interaction between the ligand and that residue increased. Van der Waals energies of residue 26 compared to wild type experienced a large increase for V82F+I84V variant and a significant decrease for the I84V variant. Residue 28 had a significant decrease for all mutated variants, with I84V having the largest decrease, followed by M46I+V82F+I84V and lastly V82F+I84V.

Residues 49 and 50 also show significant differences in the van der Waals energies of chain B. All mutated variants experience an increased significant difference in van der Waals energies of residue 49 compared to wild type. This suggests an decrease in the strength of the interaction between the ligand and residue 49. Yet, only M46I+V82F+I84V mutated variant had a significant difference in van der Waals energies compared to wild type for residue 50. The difference was negative, suggesting the interaction strength between the tip of the flap (residue 50) and the ligand increased.

Residues 81, 82, and 84 experienced significant differences in van der Waals energies between the mutated variants and the wild type. They are all located in the active site. Residue 81 had a significant difference with only the I84V mutated variant. The difference was negative, which suggests a increased interaction between the ligand and residue 81. Residue 82 had a significant difference in only V82F+I84V mutated variant. The difference was also negative, which suggests an increased interaction between the ligand and residue 82. Residue 84 had large significant differences between all mutated variants. V82F+I84V had the largest negative difference with M46I+V82F+I84V closely following. I84V had the least negative difference.

Overall, there were no significant patterns between the different variants of the protein. The only distinction seen was that the mutation of residue 84 had a large impact on the van der Waals energies.

7.2.2 Hydrogen Bonds

Hydrogen bond percentages were analyzed since they play a major role in inhibitor binding. Darunavir, the most potent protease inhibitor commercially available, was chosen as the modeling ligand since it's mechanism of binding relies heavily on hydrogen bonding. Molecular dynamic analysis provided the team with 7 active site residues (4 chain A and 3 chain B) that exhibit hydrogen bondage to the ligand. The residues that displayed the most significant changes across variants compared to wild type were chain A residues 25, 29, and 30 as well as chain B residue 50. Chain A residue 25 , aspartic acid, is the catalytic component that facilitates peptide bond cleavage.

The I84V variant displayed notable interaction related drug resistance with a 43.7% decrease in hydrogen bonds compared to wild type. Although, both I84V+V82F and M46I+V82F+I84V variants increase inhibitor binding efficacy with 17.3% and 41.3% bondage increase. The I84V single point mutation switching isoleucine with valine provided the change necessary to alter the binding domain such that the aspartic acid residue had a difficult time creating hydrogen bonds with the oxygen atom (18) on the ligand. I84V+V82F and M46I+V82F+I84V variants were capable of changing the binding pocket conformation such that the aspartic acid residue was in a closer proximity to the ligands oxygen atom (18) resulting in a high degree of stability.

Another noteworthy hydrogen bond based interaction was the negative effect of M46I+V82F+I84V variant on chain A residue 29 bondage. Wild type, I84V, and I84V+V82F variants all displayed a high degree of hydrogen bonding with 96%, 96.5%, and 97.2%, respectively. Yet, the M46I+V82F+I84V variant decreased residue 29's hydrogen binding affinity to the oxygen atom (28) of the ligand by 30.1%. Residue 29 on chain B is an aspartic acid that is a catalytic component. The M46I mutation caused a binding pocket change great enough to increase the distance between chain A residue 29 and the oxygen atom (28) of the ligand.

Additionally, the hydrogen bonding affinities for chain B residue 50 to the oxygen atom (9) of the ligand varied between variants. Reside 50 is isoleucine that plays a minimal role in hydrogen bondage displaying 16.5% bondage in wild type. The single I84V point mutation increased hydrogen bondage by 6.4% while I84V+V82F and M46I+V82F+I84V variants decreased bondage by 6.9% and 14.4% respectively.

Finally, all of the residues contributing hydrogen bonding in each variant were summed then analyzed for overall trends. As expected, the wild type had the highest summed percentage of bonding totaling 426.4%. Similarly, the I84V+V82F variant retained a high degree of hydrogen bondage with a summed percentage of 423.6%. The I84V variant displayed a significant drop of 47.1% in summed hydrogen bond affinity compared I84V+V82F. The I84V variant had an overall summed hydrogen bond percentage of 376.5%. The M46I+V82F+I84V variant showed the least total amount of hydrogen bondage with 331.3%, a 95.1% decrease compared to wild type.

Based on the total hydrogen bonding percentages, the M46I+V82F+I84V variant displayed the greatest drug resistance.

8.0 Conclusions and Recommendations

The team successfully simulated three mutated variants of HIV-1 protease and developed programs to efficiently interpret the inhibitor interaction data and the protein dynamic behavior. These programs were used to compare and present the data in a manner where conclusions could be drawn. The significant findings of this analysis and recommendations for future work are discussed in this chapter.

8.1 Significant Findings

The team hypothesized there would be a difference in inhibitor interactions and protein dynamic behavior in mutant variants compared to wild type. Specifically, as the number of mutations increased, it was thought there would be increased drug resistance. Ligand RMSF data may support this; however, further analysis displayed inconsistent trends with respect to drug resistance. V82F+I84V showed the most inhibitor fluctuation compared to wild type, followed by M46I+V82F+I84V, and I84V. However, in the case of protein RMSF, van der Waals interactions and hydrogen bond percentages, increasing resistance is not supported. Van der Waals interactions provided inconclusive data, with all variants having a similar amount of energies. With respect to protein RMSF, V82F+I84V had the greatest fluctuation compared to wild type, followed by M46I+V82F+I84V, then I84V. Similar to protein RMSF, V82F+I84V demonstrated the highest percentage of hydrogen bonds of the mutant variants, followed by M46I+V82F+I84V, then I84V. This however, partially supports the hypothesis, as bond percentages decreased from V82F+I84V to M46I+V82F+I84V.

8.2 Future Direction

Although, this project provided insightful results, there is additional research that can be conducted to strengthen our conclusions. Due to time constraints and feasibility, the team was limited to analysis of only three mutated variants. We suggest further MD analysis of different single point mutations within the active site to investigate whether increased mutations increase drug resistance.

Specifically, V82F and M46I single point mutations should be analyzed to determine their roles in compounded mutation resistance. Additionally, non-active site mutations should be examined to determine their effect on protein-ligand affinity. Evaluating different simulation parameters can strengthen future behavior patterns. Finally, the data generated from this project can be used in the development of protease inhibitors that are designed to retain potency across compounded mutations that may confer increased resistance.

References

[1] J. L. Clever, M. Daniel, Jr., and T. G. Parslow, "RNA Structure and Packaging Signals in the 5′ Leader Region of the Human Immunodeficiency Virus Type 1 Genome," *Journal of Virology,* vol. 76, pp. 12381-12387, 2002.

[2] N. I. o. Health. Compound Summary for CID 213039 [Online]. Available: https://pubchem.ncbi.nlm.nih.gov/compound/Darunavir

[3] S. G. Deeks, B. Autran, B. Berkhout, M. Benkirane, S. Cairns, N. Chomont*, et al.*, "Towards an HIV cure: A global scientific strategy," *Nature Reviews Immunology,* vol. 12, pp. 607-614, 2012.

[4] a. T. F. f. A. Research. (2015). *Statistics: Worldwide*. Available: http://www.amfar.org/worldwide-aids-stats/

[5] C. f. D. C. a. Prevention. (2015). *HIV in the United States: At A Glance*. Available: http://www.cdc.gov/hiv/statistics/overview/ataglance.html

[6] R. P. Walensky, J. D. Auerbach, A. R. A. C. Off, A. R. A. C. H. I. V. A. R. P. R. W. G. Office of, and A. R. A. C. Office of, "Focusing National Institutes of Health HIV/AIDS Research for Maximum Population Impact," *CLINICAL INFECTIOUS DISEASES,* vol. 60, pp. 937-940, 2015.

[7] I. T. Weber and J. Agniswamy, "HIV-1 Protease: Structural Perspectives on Drug Resistance," *VIRUSES-BASEL,* vol. 1, pp. 1110-1136, 2009.

[8] A. Brik and C.-H. Wong, "HIV-1 protease: Mechanism and drug discovery," *Organic and Biomolecular Chemistry,* vol. 1, pp. 5-14, 2003.

[9] C. f. D. C. a. Prevention. (2015). *HIV Cost-effectiveness*. Available: http://www.cdc.gov/hiv/programresources/guidance/costeffectiveness/index.html

[10] M. O. Johnson, E. Charlebois, S. F. Morin, S. L. Catz, R. B. Goldstein, R. H. Remien*, et al.*, "Perceived Adverse Effects of Antiretroviral Therapy," *Journal of Pain and Symptom Management,* vol. 29, pp. 193-205, 2005.

[11] J.-M. Molina, J. Andrade-Villanueva, J. Echevarria, P. Chetchotisakd, J. Corral, N. David*, et al.*, "Once-daily atazanavir/ritonavir versus twice-daily lopinavir/ritonavir, each in combination with tenofovir and emtricitabine, for management of antiretroviral-naive HIV-1-infected patients: 48 week efficacy and safety results of the CASTLE study," *The Lancet,* vol. 372, pp. 646-655, 2008.

[12] E. De Clercq, "Non-nucleoside reverse transcriptase inhibitors (NNRTIs): Past, present, and future," *Chemistry and Biodiversity,* vol. 1, pp. 44-64, 2004.

[13] B. Turanli-Yildiz, C. Alkim, and Z. P. Cakar, *Protein Engineering Methods and Applications, Protein Engineering*, 2012.

[14] D. C. Chan and P. S. Kim, "HIV Entry and Its Inhibition," vol. 93, ed. CAMBRIDGE: Elsevier Inc, 1998, pp. 681-684.

[15] M. I. Qadir and S. A. Malik, "HIV fusion inhibitors," *Reviews in Medical Virology,* vol. 20, pp. 23-33, 2010.

[16] E. A. Abbondanzieri, X. Zhuang, J. W. Rausch, J. X. Zhang, G. Bokinsky, and S. F. J. Le Grice, "Dynamic binding orientations direct activity of HIV reverse transcriptase," *Nature,* vol. 453, pp. 184-189, 2008.

[17] R. Di Santo, "Inhibiting the HIV Integration Process: Past, Present, and the Future," *JOURNAL OF MEDICINAL CHEMISTRY,* vol. 57, pp. 539-566, 2014.

[18] R. Craigie and F. D. Bushman, "HIV DNA Integration," *Cold Spring Harbor Perspectives in Medicine,* vol. 2, p. a006890, 2012.

[19] J. Alcami, T. Lain de Lera, L. Folgueira, M. A. Pedraza, J. M. Jacque, F. Bachelerie*, et al.*, "Absolute dependence on κB responsive elements for initiation and tat-mediated amplification of HIV transcription in blood CD4 T lymphocytes," *EMBO Journal,* vol. 14, pp. 1552-1560, 1995.

[20] L. Houzet, J. C. Paillart, F. Smagulova, S. Maurel, Z. Morichaud, R. Marquet*, et al.*, "HIV controls the selective packaging of genomic, spliced viral and cellular RNAs into virions through different mechanisms," *Nucleic Acids Research,* vol. 35, pp. 2695-2704, 2007.

[21] S. Pagans, A. Pedal, B. J. North, K. Kaehlcke, B. L. Marshall, A. Dorr*, et al.*, "SIRT1 regulates HIV transcription via Tat deacetylation," *PLoS Biology,* vol. 3, pp. 0210-0220, 2005.

[22] B. Meng and A. M. L. Lever, "Wrapping up the bad news - HIV assembly and release," *RETROVIROLOGY,* vol. 10, pp. 5-5, 2013.

[23] U. Schubert, D. E. Ott, E. N. Chertova, R. Welker, U. Tessmer, M. F. Princiotta*, et al.*, "Proteasome Inhibition Interferes with Gag Polyprotein Processing, Release, and Maturation of HIV-1 and HIV-2," *Proceedings of the National Academy of Sciences of the United States of America,* vol. 97, pp. 13057-13062, 2000.

[24] B. K. Ganser-Pornillos, M. Yeager, and W. I. Sundquist, "The structural biology of HIV assembly," *Current Opinion in Structural Biology,* vol. 18, pp. 203-217, 2008.

[25] J. A. Esté and A. Telenti, "HIV entry inhibitors," *The Lancet,* vol. 370, pp. 81-88.

[26] A. A. Johnson, C. Marchand, and Y. Pommier, "Integrase inhibitors to treat HIV/Aids," *Nature Reviews Drug Discovery,* vol. 4, pp. 236-248, 2005.

[27] V. Summa, A. Petrocchi, F. Bonelli, B. Crescenzi, M. Donghi, M. Ferrara*, et al.*, "Discovery of raltegravir, a potent, selective orally bioavailable HIV-integrase inhibitor for the treatment of HIV-AIDS infection," *Journal of Medicinal Chemistry,* vol. 51, pp. 5843-5855, 2008.

[28] N. Sluis-Cremer, D. Arion, and M. A. Parniak, "Molecular mechanisms of HIV-1 resistance to nucleoside reverse transcriptase inhibitors (NRTIs)," *Cellular and Molecular Life Sciences,* vol. 57, pp. 1408-1422, 2000.

[29] E. M. Connor, R. S. Sperling, R. Gelber, P. Kiselev, G. Scott, M. J. O'Sullivan*, et al.*, "Reduction of Maternal-Infant Transmission of Human Immunodeficiency Virus Type 1 with Zidovudine Treatment," *The New England Journal of Medicine,* vol. 331, pp. 1173-1180, 1994.

[30] N. I. o. Health. *Guidelines for the Use of Antiretroviral Agents in HIV-1-Infected Adults and Adolescents*. Available: https://aidsinfo.nih.gov/guidelines/html/1/adult-and-adolescent-arv-guidelines/459/cost-considerations-and-antiretroviral-therapy

[31] K. McKeage, C. M. Perry, and S. J. Keam, "Darunavir: A Review of its Use in the Management of HIV Infection in Adults," *Drugs,* vol. 69, pp. 477-503, 2009.

[32] M. L. Vazquez, R. A. Mueller, J. J. Talley, D. P. Getman, G. A. DeCrescenzo, J. N. Freskos*, et al.*, "α- and β-amino acid hydroxyethylamino sulfonamides useful as retroviral protease inhibitors," ed: Google Patents, 2001.

[33] I. Dierynck, M. D. Wit, E. Gustin, I. Keuleers, J. Vandersmissen, S. Hallenberger*, et al.*, "Binding Kinetics of Darunavir to Human Immunodeficiency Virus Type 1 Protease Explain the Potent Antiviral Activity and High Genetic Barrier," *Journal of Virology,* vol. 81, pp. 13845-13851, 2007.

[34] D. Finzi, M. Hermankova, T. Pierson, L. M. Carruth, C. Buck, R. E. Chaisson*, et al.*, "Identification of a Reservoir for HIV-1 in Patients on Highly Active Antiretroviral Therapy," *Science,* vol. 278, pp. 1295-1300, 1997.

[35] Dharmananda. COUNTERACTING
SIDE EFFECTS OF THE HIV-INHIBITING DRUG COCKTAIL [Online].

[36] J. Aguirre, "Cost Of Treatment Still A Challenge For HIV Patients In U.S.," in *All Things Considered*, J. Aguirre, Ed., ed, 2012.

[37] A. T. i. L. I. C. S. Group, "Cohort profile: Antiretroviral Therapy in Lower Income Countries (ART-LINC): international collaboration of treatment cohorts," *International Journal of Epidemiology,* vol. 34, pp. 979-986, 2005.

[38] L. K. Nicholson, T. Yamazaki, D. A. Torchia, S. Grzesiek, A. Bax, S. J. Stahl*, et al.*, "Flexibility and function in HIV-1 protease," *Nature structural biology,* vol. 2, pp. 274-280, 1995.

[39] W. R. P. Scott and C. A. Schiffer, "Curling of Flap Tips in HIV-1 Protease as a Mechanism for Substrate Entry and Tolerance of Drug Resistance," *Structure,* vol. 8, pp. 1259-1265, 2000.

[40] J. C. Clemente, R. E. Moose, R. Hemrajani, L. R. S. Whitford, L. Govindasamy, R. Reutzel*, et al.*, "Comparing the accumulation of active- and nonactive-site mutations in the HIV-1 protease," *Biochemistry,* vol. 43, pp. 12141-12151, 2004.

[41] J. E. Foulkes-Murzycki, W. R. P. Scott, and C. A. Schiffer, "Hydrophobic sliding: a possible mechanism for drug resistance in human immunodeficiency virus type 1 protease," *Structure,* vol. 15, pp. 225-233, 2007.

[42] D. N. Levy, G. M. Aldrovandi, O. Kutsch, and G. M. Shaw, "Dynamics of HIV-1 recombination in its natural target cells," *Proceedings of the National Academy of Sciences,* vol. 101, pp. 4204-4209, 2004.

[43] S. Piana, P. Carloni, and U. Rothlisberger, "Drug resistance in HIV‐1 protease: Flexibility‐assisted mechanism of compensatory mutations," *Protein Science,* vol. 11, pp. 2393-2402, 2002.

[44] W. I. Sundquist and H.-G. Kräusslich, "HIV-1 assembly, budding, and maturation," *Cold Spring Harbor Perspectives in Medicine,* vol. 2, p. a006924, 2012.

[45] A. D. Frankel and J. A. Young, "HIV-1: fifteen proteins and an RNA," *Annual review of biochemistry,* vol. 67, pp. 1-25, 1998.

[46] M. Prabu-Jeyabalan, E. Nalivaika, and C. A. Schiffer, "Substrate shape determines specificity of recognition for HIV-1 protease: Analysis of crystal structures of six substrate complexes," *Structure,* vol. 10, pp. 369-381, 2002.

[47] A. M. Silva, R. E. Cachau, H. L. Sham, and J. W. Erickson, "Inhibition and catalytic mechanism of HIV-1 aspartic protease," *Journal of Molecular Biology,* vol. 255, pp. 321-340, // 1996.

[48] T. Hansson, C. Oostenbrink, and W. van Gunsteren, "Molecular dynamics simulations," *Current opinion in structural biology,* vol. 12, pp. 190-196, 2002.

[49] S. Press, "Protein Preparation Guide," ed. New York, NY: Schrödinger LLC, 2009.

[50] S. Press, "Prime 2.1: User Manual," ed. New York, NY: Schrödinger LLC, 2013.

[51] K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen*, et al.*, "Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters," pp. 43-43.

[52] Julich, "Analysis of MD Simulations," ed: Forschungszentrum.

[53] Schrodinger, "Maestro 9.0," ed: Schrodinger, 2009.

[54] W. Humphrey, A. Dalke, and K. Schulten, "VMD: Visual molecular dynamics," *JOURNAL OF MOLECULAR GRAPHICS & MODELLING,* vol. 14, pp. 33-38, 1996.

[55] (2016). *Schiffer Laboratory*. Available: http://www.umassmed.edu/schifferlab/

[56] A. Y. Kuksin, I. V. Morozov, G. E. Norman, V. V. Stegailov, and I. A. Valuev, "Standards for molecular dynamics modelling and simulation of relaxation," *Molecular Simulation,* vol. 31, pp. 1005-1017, 2005.

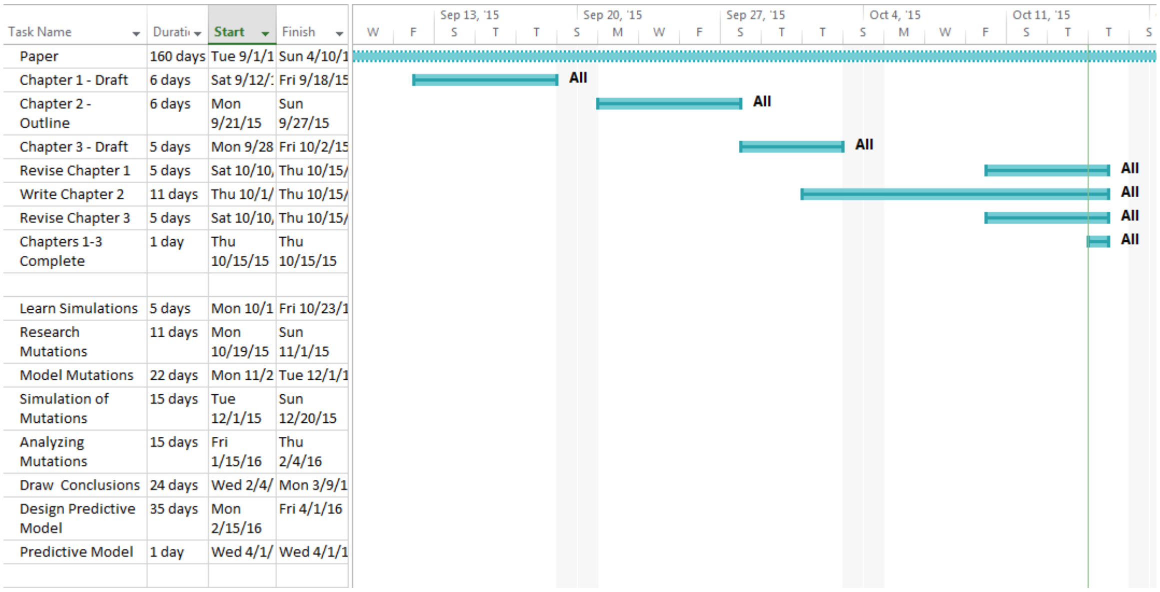
[57] R. Johnson, "MATLAB Programming Style Guidelines," ed: MathWorks, 2002.

[58] I. O. f. Standardization, "ISO 18458: Biomimetics — Terminology, concepts and methodology," ed: International Organizational for Standardization, 2015.

Appendix A

The team created a Gantt chart to make sure the project is completed within time constraints (Figures A.1 through A.4).

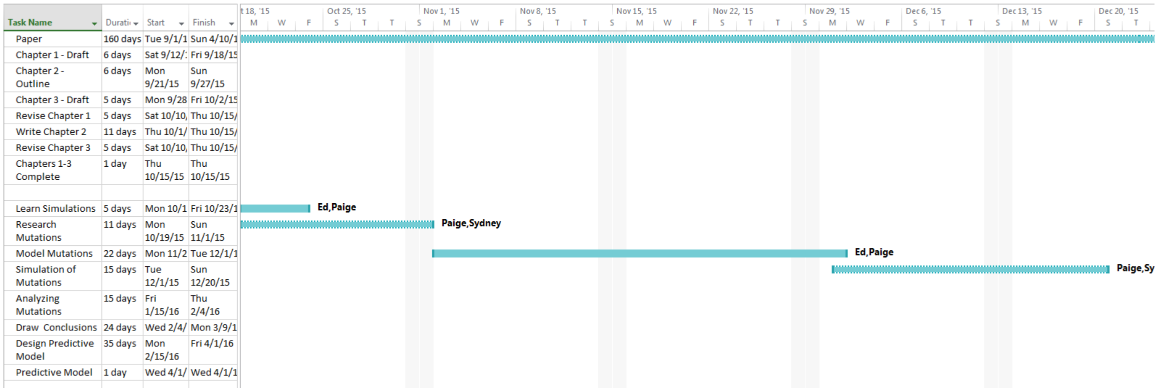
A.1 A Term Gantt Chart



Task Name	Durati	Start	Finish
Paper	160 days	Tue 9/1/1	Sun 4/10/1
Chapter 1 - Draft	6 days	Sat 9/12/:	Fri 9/18/15
Chapter 2 - Outline	6 days	Mon 9/21/15	Sun 9/27/15
Chapter 3 - Draft	5 days	Mon 9/28	Fri 10/2/15
Revise Chapter 1	5 days	Sat 10/10,	Thu 10/15/
Write Chapter 2	11 days	Thu 10/1/	Thu 10/15/
Revise Chapter 3	5 days	Sat 10/10,	Thu 10/15/
Chapters 1-3 Complete	1 day	Thu 10/15/15	Thu 10/15/15
Learn Simulations	5 days	Mon 10/1	Fri 10/23/1
Research Mutations	11 days	Mon 10/19/15	Sun 11/1/15
Model Mutations	22 days	Mon 11/2	Tue 12/1/1
Simulation of Mutations	15 days	Tue 12/1/15	Sun 12/20/15
Analyzing Mutations	15 days	Fri 1/15/16	Thu 2/4/16
Draw Conclusions	24 days	Wed 2/4/	Mon 3/9/1
Design Predictive Model	35 days	Mon 2/15/16	Fri 4/1/16
Predictive Model	1 day	Wed 4/1/	Wed 4/1/1

Figure A.1: A Term Gantt Chart

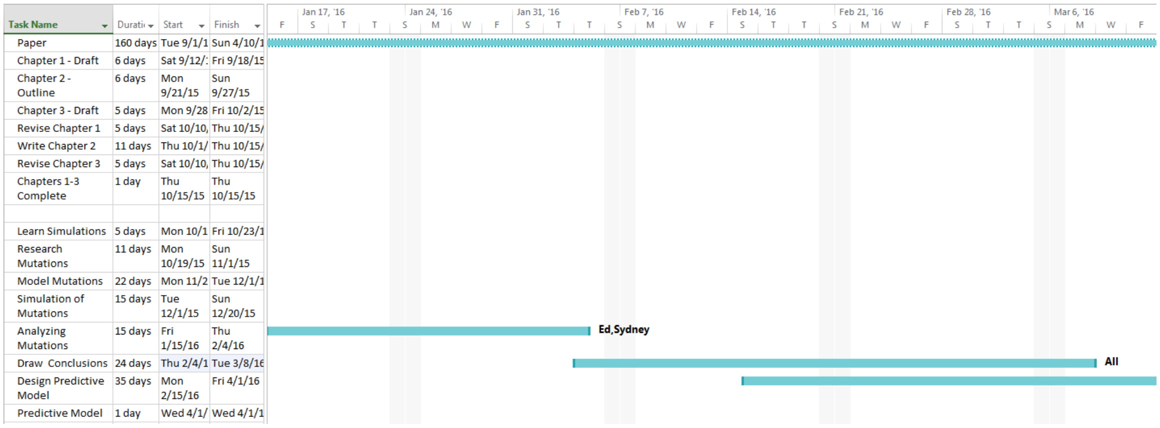
A.2 B Term Gantt Chart



Task Name	Durati	Start	Finish
Paper	160 days	Tue 9/1/1	Sun 4/10/1
Chapter 1 - Draft	6 days	Sat 9/12/:	Fri 9/18/15
Chapter 2 - Outline	6 days	Mon 9/21/15	Sun 9/27/15
Chapter 3 - Draft	5 days	Mon 9/28	Fri 10/2/15
Revise Chapter 1	5 days	Sat 10/10,	Thu 10/15/
Write Chapter 2	11 days	Thu 10/1/	Thu 10/15/
Revise Chapter 3	5 days	Sat 10/10,	Thu 10/15/
Chapters 1-3 Complete	1 day	Thu 10/15/15	Thu 10/15/15
Learn Simulations	5 days	Mon 10/1	Fri 10/23/1
Research Mutations	11 days	Mon 10/19/15	Sun 11/1/15
Model Mutations	22 days	Mon 11/2	Tue 12/1/1
Simulation of Mutations	15 days	Tue 12/1/15	Sun 12/20/15
Analyzing Mutations	15 days	Fri 1/15/16	Thu 2/4/16
Draw Conclusions	24 days	Wed 2/4/	Mon 3/9/1
Design Predictive Model	35 days	Mon 2/15/16	Fri 4/1/16
Predictive Model	1 day	Wed 4/1/	Wed 4/1/1

Figure A.2: B Term Gantt Chart

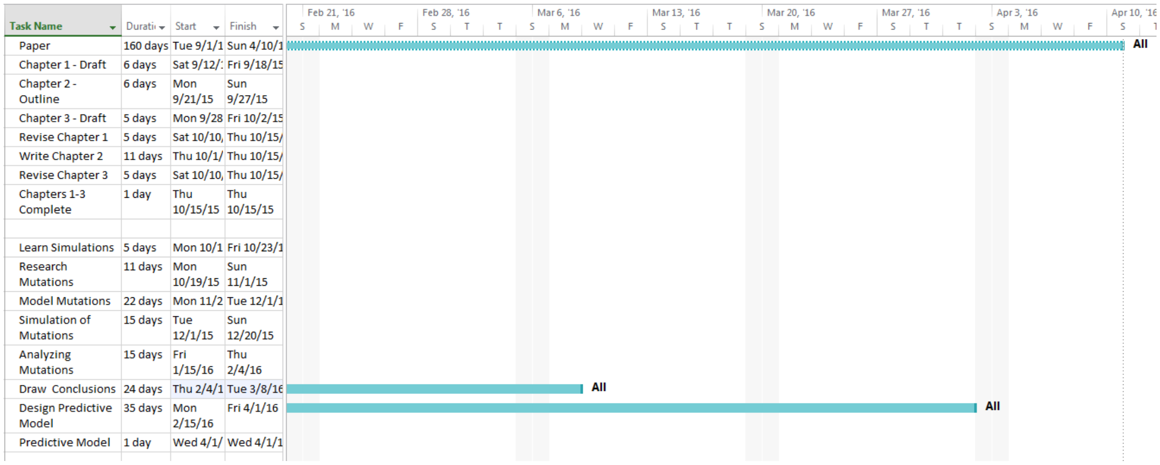
A.3 C Term Gantt Chart



Task Name	Durati	Start	Finish
Paper	160 days	Tue 9/1/1	Sun 4/10/1
Chapter 1 - Draft	6 days	Sat 9/12/1	Fri 9/18/15
Chapter 2 - Outline	6 days	Mon 9/21/15	Sun 9/27/15
Chapter 3 - Draft	5 days	Mon 9/28	Fri 10/2/15
Revise Chapter 1	5 days	Sat 10/10,	Thu 10/15/
Write Chapter 2	11 days	Thu 10/1/	Thu 10/15/
Revise Chapter 3	5 days	Sat 10/10,	Thu 10/15/
Chapters 1-3 Complete	1 day	Thu 10/15/15	Thu 10/15/15
Learn Simulations	5 days	Mon 10/1	Fri 10/23/1
Research Mutations	11 days	Mon 10/19/15	Sun 11/1/15
Model Mutations	22 days	Mon 11/2	Tue 12/1/1
Simulation of Mutations	15 days	Tue 12/1/15	Sun 12/20/15
Analyzing Mutations	15 days	Fri 1/15/16	Thu 2/4/16
Draw Conclusions	24 days	Thu 2/4/1	Tue 3/8/16
Design Predictive Model	35 days	Mon 2/15/16	Fri 4/1/16
Predictive Model	1 day	Wed 4/1/	Wed 4/1/1

Figure A.3: C Term Gantt Chart

A.4 D Term Gantt Chart

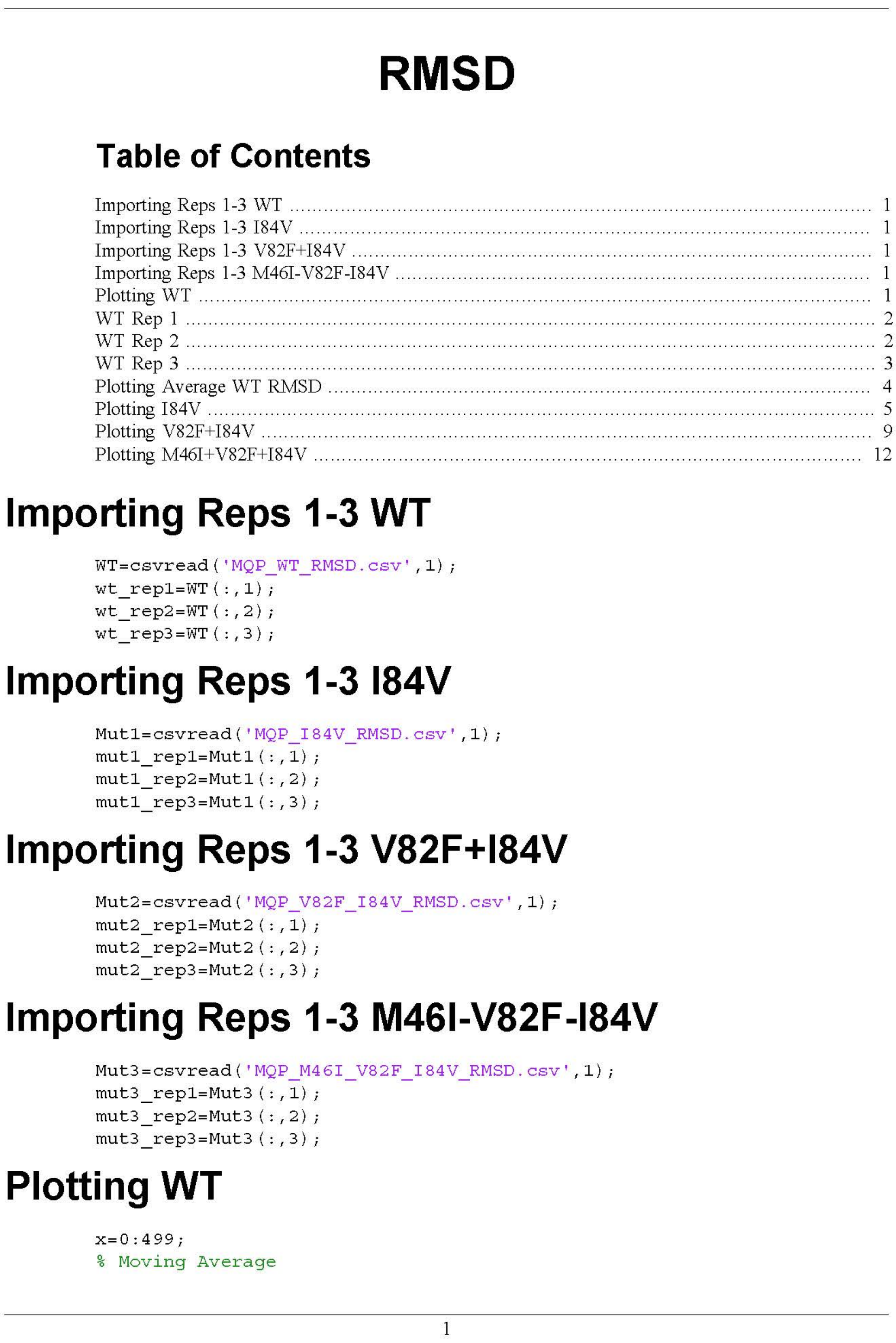


Task Name	Durati	Start	Finish
Paper	160 days	Tue 9/1/1	Sun 4/10/1
Chapter 1 - Draft	6 days	Sat 9/12/1	Fri 9/18/15
Chapter 2 - Outline	6 days	Mon 9/21/15	Sun 9/27/15
Chapter 3 - Draft	5 days	Mon 9/28	Fri 10/2/15
Revise Chapter 1	5 days	Sat 10/10,	Thu 10/15/
Write Chapter 2	11 days	Thu 10/1/	Thu 10/15/
Revise Chapter 3	5 days	Sat 10/10,	Thu 10/15/
Chapters 1-3 Complete	1 day	Thu 10/15/15	Thu 10/15/15
Learn Simulations	5 days	Mon 10/1	Fri 10/23/1
Research Mutations	11 days	Mon 10/19/15	Sun 11/1/15
Model Mutations	22 days	Mon 11/2	Tue 12/1/1
Simulation of Mutations	15 days	Tue 12/1/15	Sun 12/20/15
Analyzing Mutations	15 days	Fri 1/15/16	Thu 2/4/16
Draw Conclusions	24 days	Thu 2/4/1	Tue 3/8/16
Design Predictive Model	35 days	Mon 2/15/16	Fri 4/1/16
Predictive Model	1 day	Wed 4/1/	Wed 4/1/1

Figure A.4: D Term Gantt Chart

Appendix B

B.1 Protein-Ligand RMSD Script



RMSD

Table of Contents

Importing Reps 1-3 WT

```
WT=csvread('MQP_WT_RMSD.csv',1);
wt_rep1=WT(:,1);
wt_rep2=WT(:,2);
wt_rep3=WT(:,3);
```

Importing Reps 1-3 I84V

```
Mut1=csvread('MQP_I84V_RMSD.csv',1);
mut1_rep1=Mut1(:,1);
mut1_rep2=Mut1(:,2);
mut1_rep3=Mut1(:,3);
```

Importing Reps 1-3 V82F+I84V

```
Mut2=csvread('MQP_V82F_I84V_RMSD.csv',1);
mut2_rep1=Mut2(:,1);
mut2_rep2=Mut2(:,2);
mut2_rep3=Mut2(:,3);
```

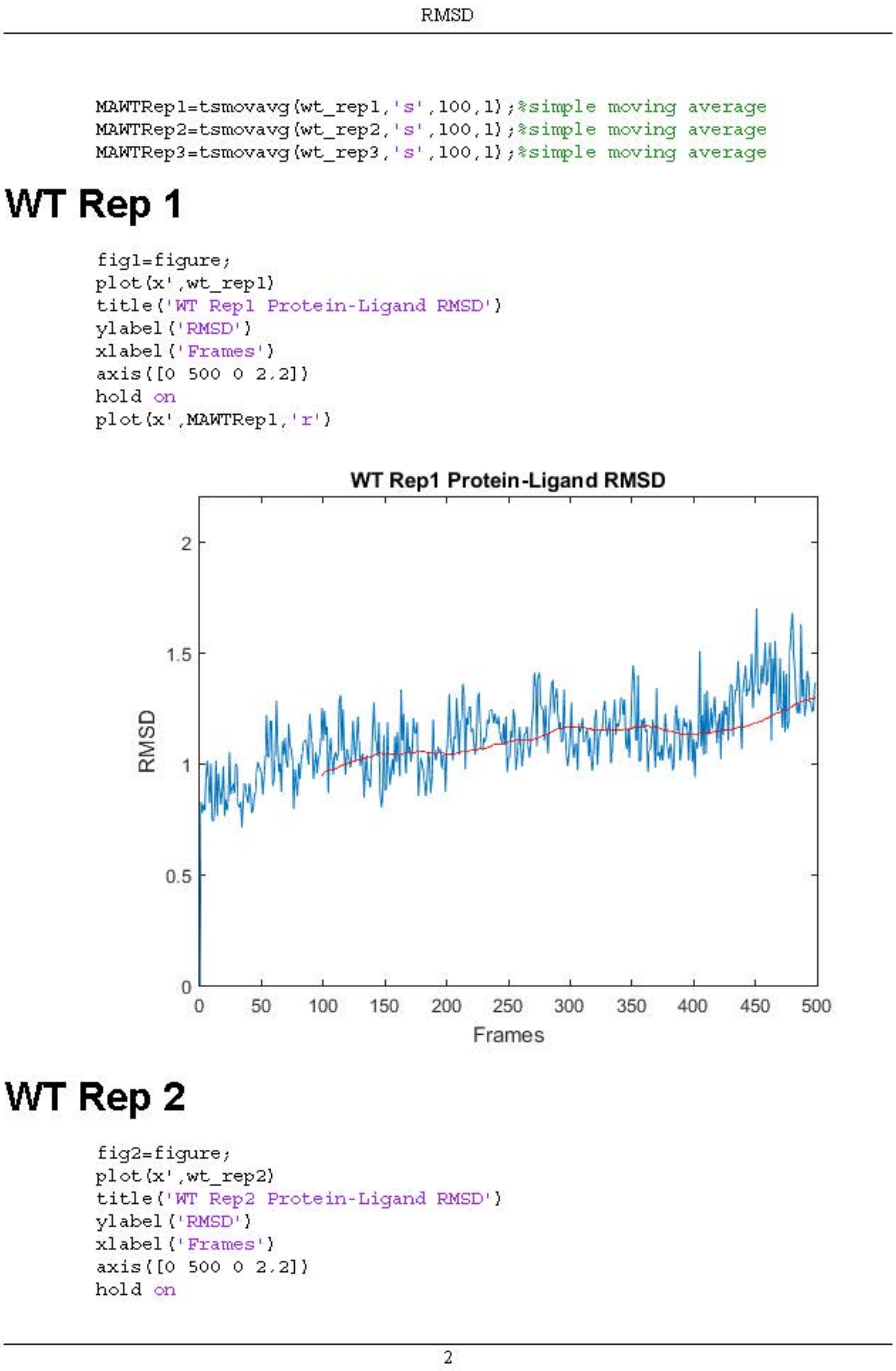
Importing Reps 1-3 M46I-V82F-I84V

```
Mut3=csvread('MQP_M46I_V82F_I84V_RMSD.csv',1);
mut3_rep1=Mut3(:,1);
mut3_rep2=Mut3(:,2);
mut3_rep3=Mut3(:,3);
```

Plotting WT

```
x=0:499;
% Moving Average
```

1



```
MAWTRep1=tsmovavg(wt_rep1,'s',100,1);%simple moving average
MAWTRep2=tsmovavg(wt_rep2,'s',100,1);%simple moving average
MAWTRep3=tsmovavg(wt_rep3,'s',100,1);%simple moving average
```

WT Rep 1

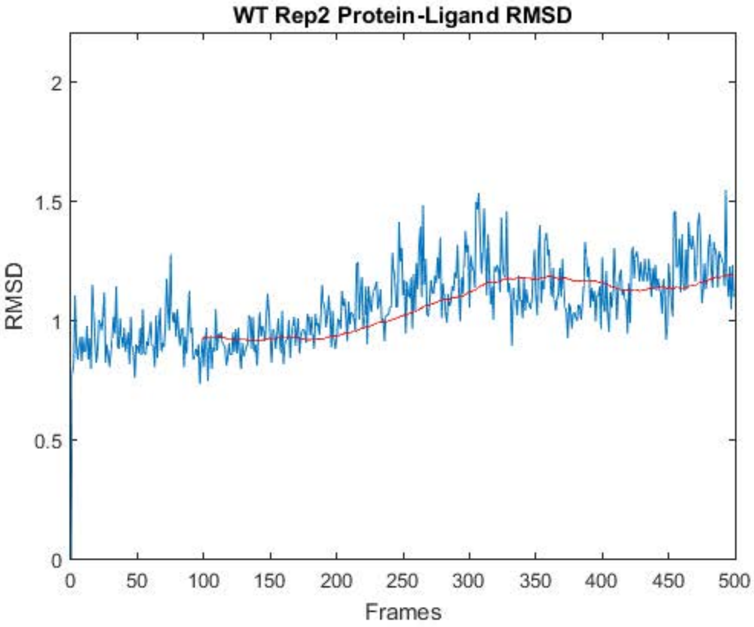
```
fig1=figure;
plot(x',wt_rep1)
title('WT Rep1 Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAWTRep1,'r')
```

WT Rep1 Protein-Ligand RMSD
RMSD
0
0.5
1
1.5
2
0
50
100
150
200
250
300
350
400
450
500
Frames

WT Rep 2

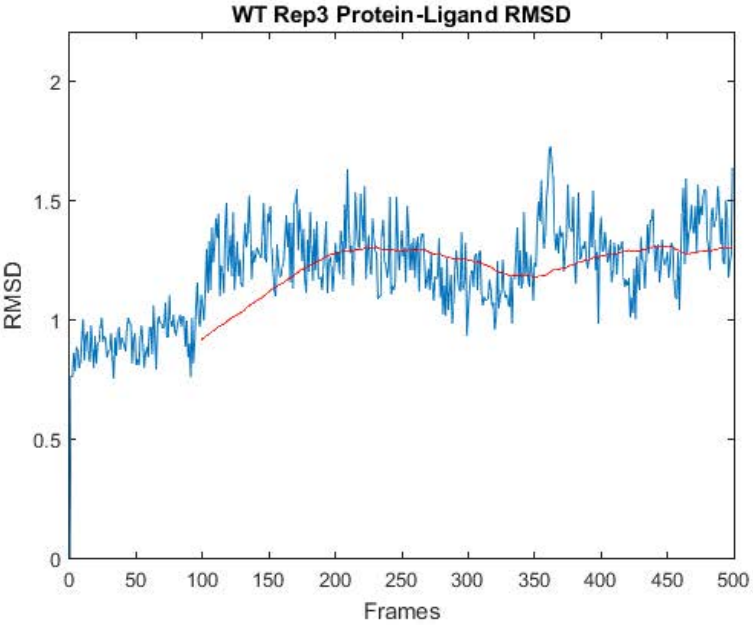
```
fig2=figure;
plot(x',wt_rep2)
title('WT Rep2 Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
```

```
plot(x',MAWTRep2,'r')
```



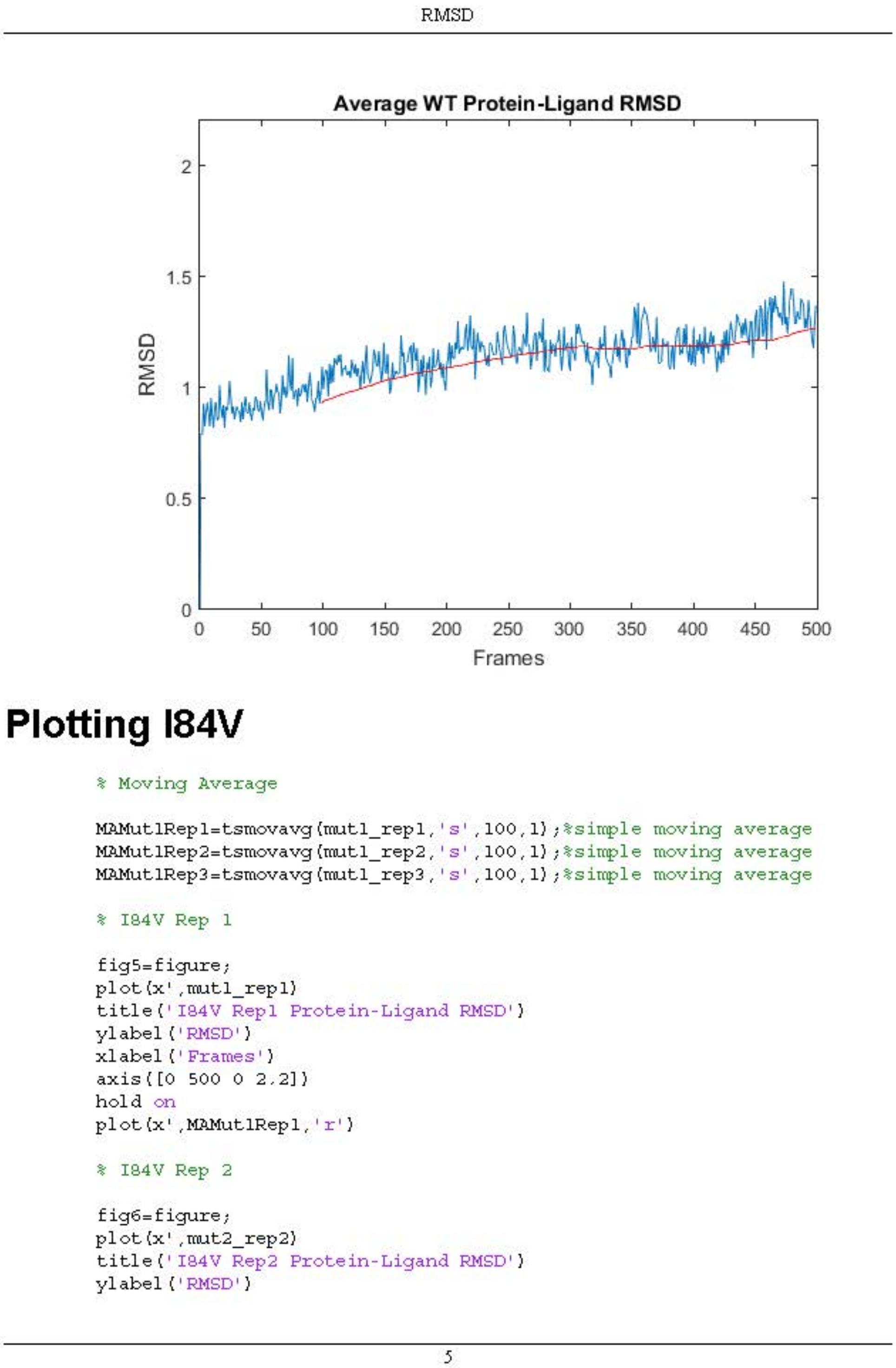
WT Rep 3

```
fig3=figure;
plot(x',wt_rep3)
title('WT Rep3 Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAWTRep3,'r')
```



Plotting Average WT RMSD

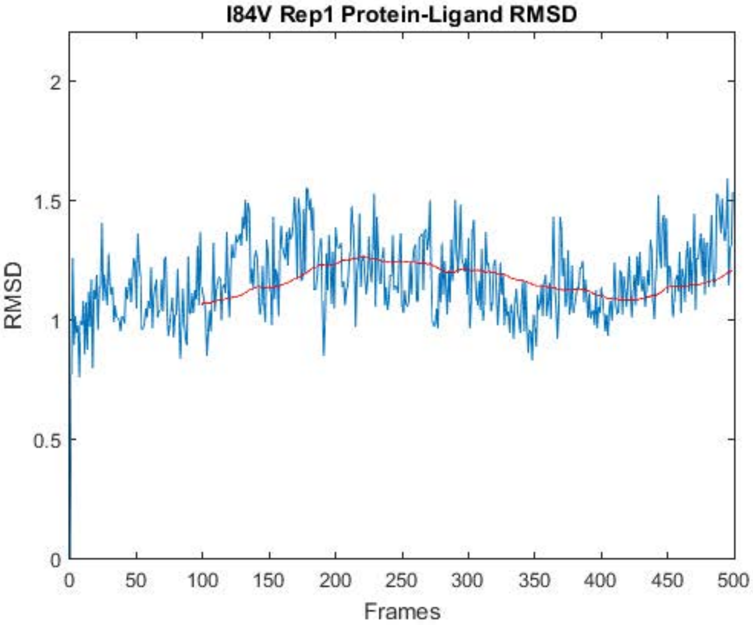
```
WT_Average=(wt_rep1+wt_rep2+wt_rep3)/3;
MAWTAvg=tsmovavg(WT_Average,'s',100,1);%simple moving average

fig4=figure;
plot(x',WT_Average)
title('Average WT Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAWTAvg,'r')
```

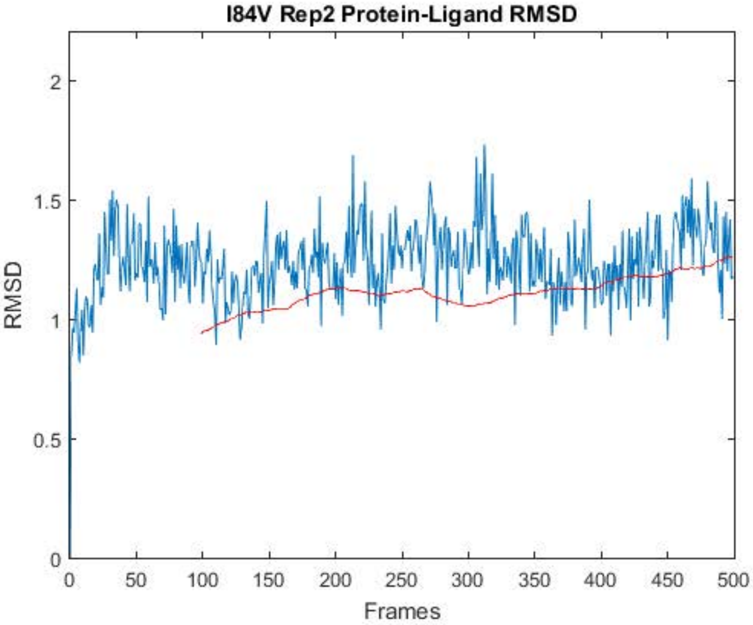


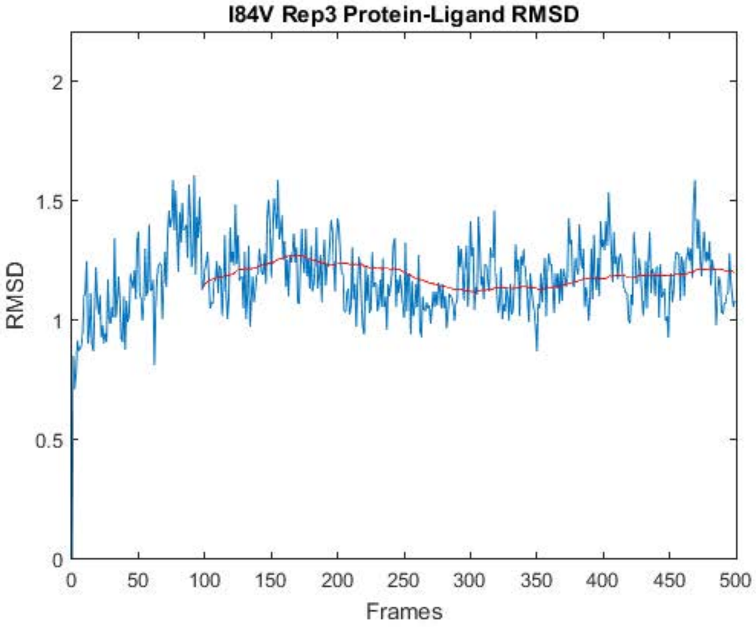
Plotting I84V

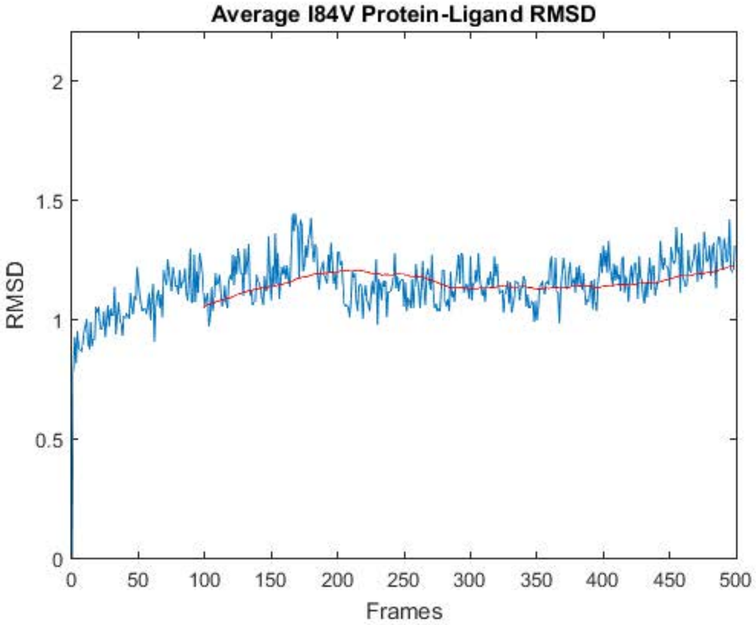
```
% Moving Average

MAMut1Rep1=tsmovavg(mut1_rep1,'s',100,1);%simple moving average
MAMut1Rep2=tsmovavg(mut1_rep2,'s',100,1);%simple moving average
MAMut1Rep3=tsmovavg(mut1_rep3,'s',100,1);%simple moving average

% I84V Rep 1

fig5=figure;
plot(x',mut1_rep1)
title('I84V Rep1 Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAMut1Rep1,'r')

% I84V Rep 2

fig6=figure;
plot(x',mut2_rep2)
title('I84V Rep2 Protein-Ligand RMSD')
ylabel('RMSD')
```

```
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAMut1Rep2,'r')

% I84V Rep 3

fig7=figure;
plot(x',mut1_rep3)
title('I84V Rep3 Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAMut1Rep3,'r')

%Plotting Average I84V RMSD

Mut1_Average=(mut1_rep1+mut1_rep2+mut1_rep3)/3;
MAMut1Avg=tsmovavg(Mut1_Average,'s',100,1);%simple moving average

fig8=figure;
plot(x',Mut1_Average)
title('Average I84V Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAMut1Avg,'r')
```


I84V Rep1 Protein-Ligand RMSD


I84V Rep2 Protein-Ligand RMSD

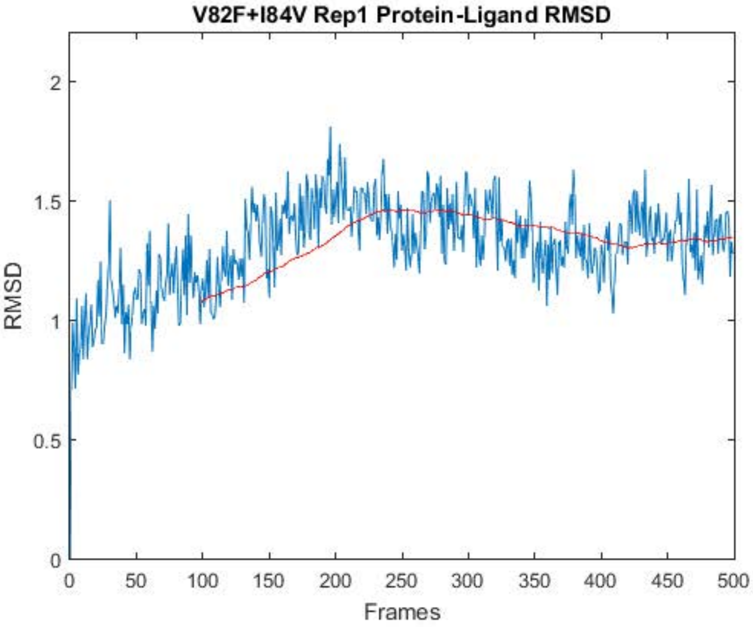

I84V Rep3 Protein-Ligand RMSD
RMSD
Frames
0
0.5
1
1.5
2
0
50
100
150
200
250
300
350
400
450
500

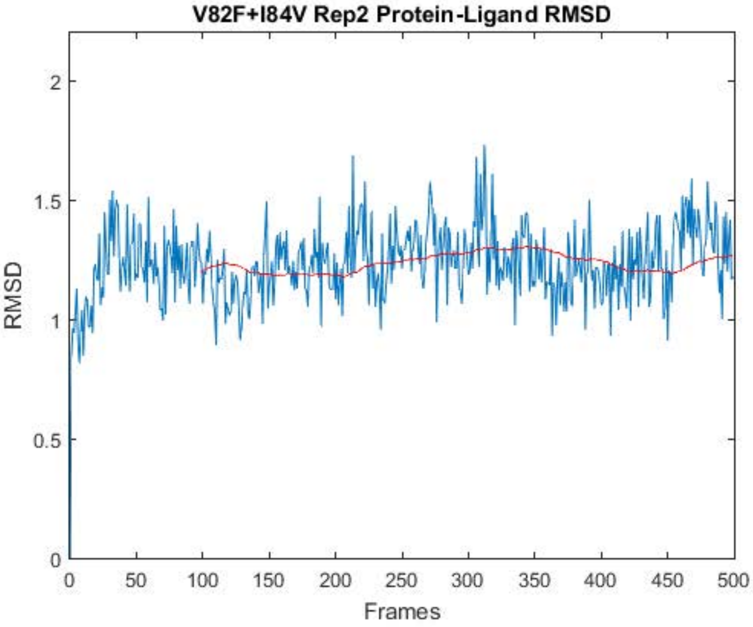

Average I84V Protein-Ligand RMSD
RMSD
Frames
0
0.5
1
1.5
2
0
50
100
150
200
250
300
350
400
450
500

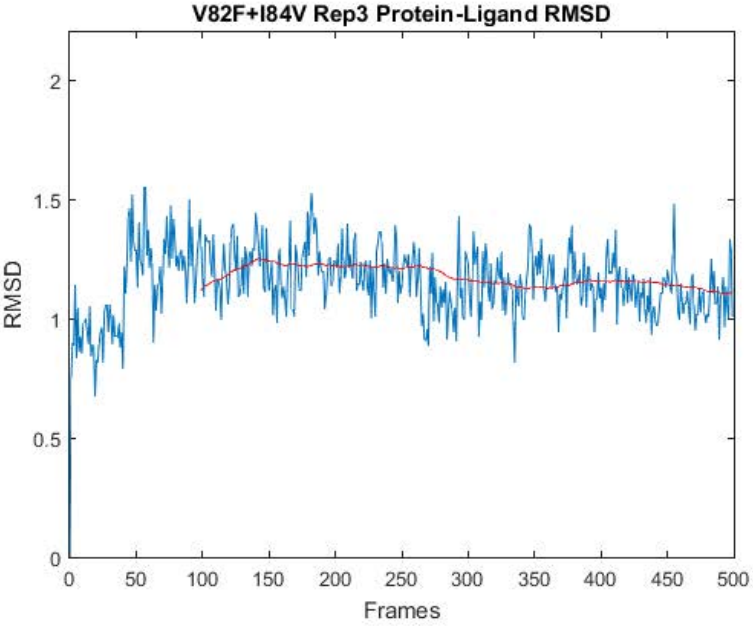
Plotting V82F+I84V

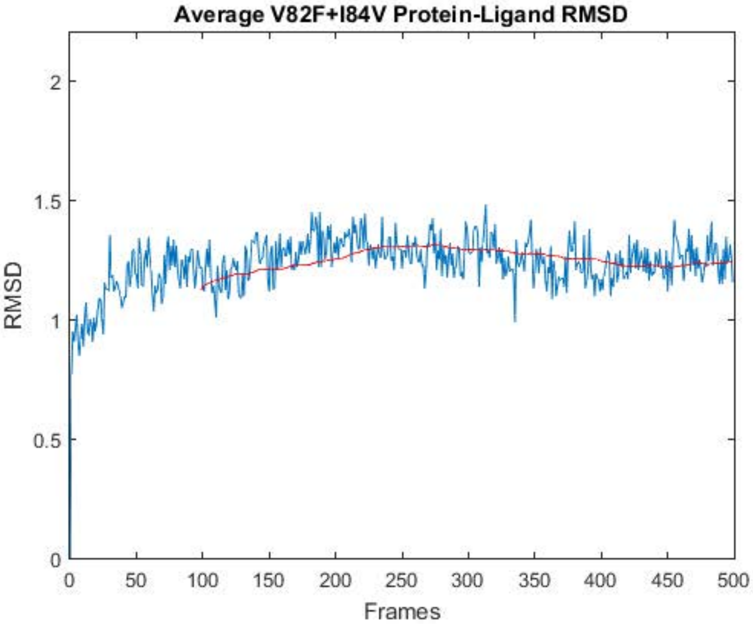
```
% Moving Average

MAMut2Rep1=tsmovavg(mut2_rep1,'s',100,1);%simple moving average
MAMut2Rep2=tsmovavg(mut2_rep2,'s',100,1);%simple moving average
MAMut2Rep3=tsmovavg(mut2_rep3,'s',100,1);%simple moving average

% V82F+I84V Rep 1

fig9=figure;
plot(x',mut2_rep1)
title('V82F+I84V Rep1 Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAMut2Rep1,'r')

% V82F+I84V Rep 2

fig10=figure;
plot(x',mut2_rep2)
title('V82F+I84V Rep2 Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAMut2Rep2,'r')

% V82F+I84V Rep 3

fig11=figure;
plot(x',mut2_rep3)
title('V82F+I84V Rep3 Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAMut2Rep3,'r')

%Plotting Average V82F+I84V RMSD

Mut2_Average=(mut2_rep1+mut2_rep2+mut2_rep3)/3;
MAMut2Avg=tsmovavg(Mut2_Average,'s',100,1);%simple moving average

fig12=figure;
plot(x',Mut2_Average)
title('Average V82F+I84V Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
```

```
plot(x',MAMut2Avg,'r')
```




V82F+I84V Rep2 Protein-Ligand RMSD
RMSD
0
0.5
1
1.5
2
0
50
100
150
200
250
300
350
400
450
500
Frames

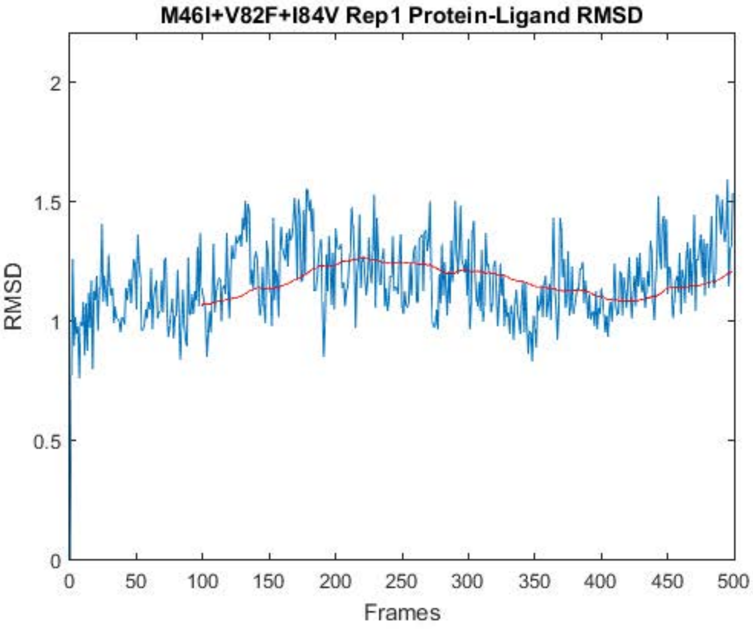

V82F+I84V Rep3 Protein-Ligand RMSD
RMSD
0
0.5
1
1.5
2
0
50
100
150
200
250
300
350
400
450
500
Frames

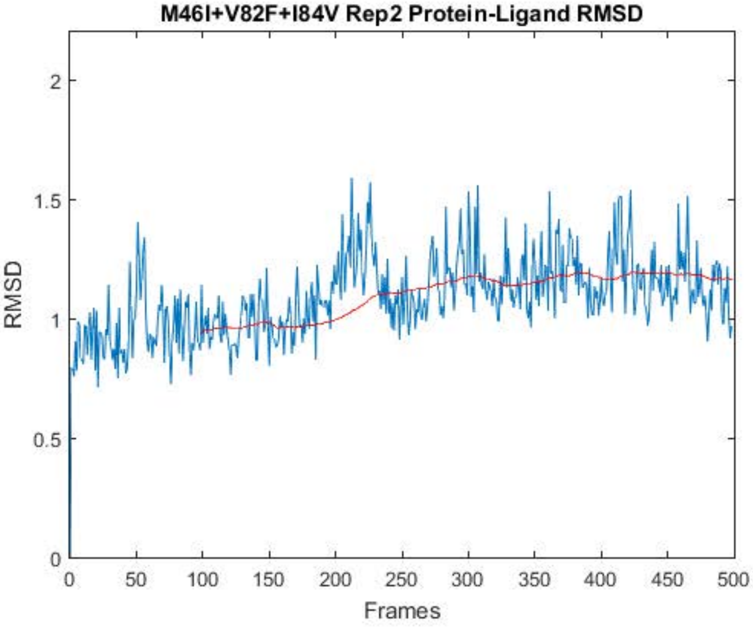


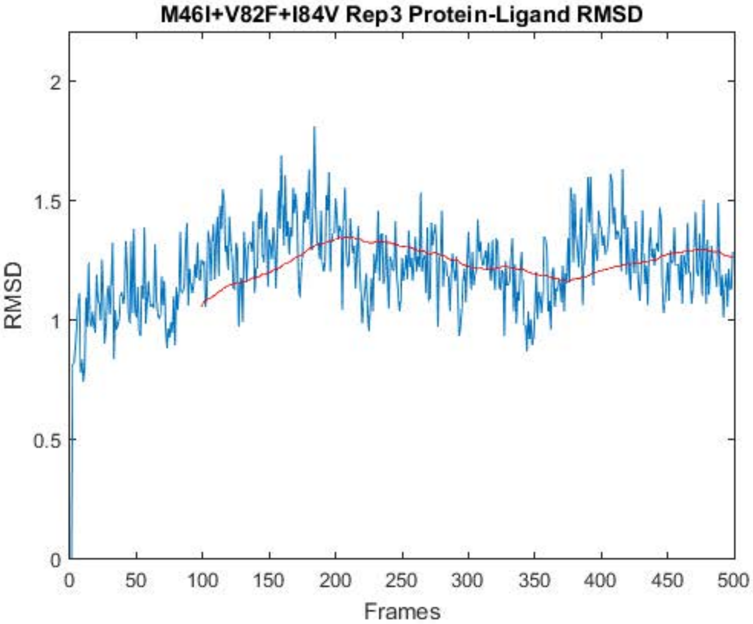
Plotting M46I+V82F+I84V

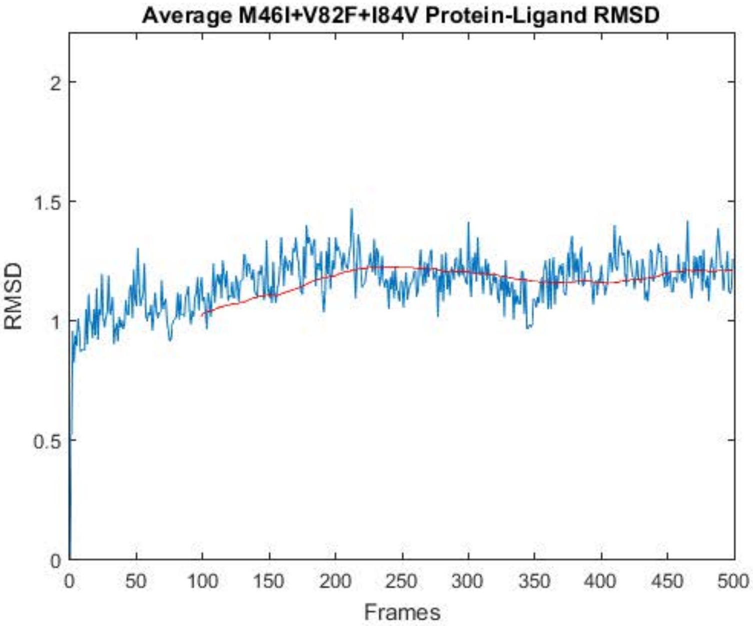
```
% Moving Average

MAMut3Rep1=tsmovavg(mut3_rep1,'s',100,1);%simple moving average
MAMut3Rep2=tsmovavg(mut3_rep2,'s',100,1);%simple moving average
MAMut3Rep3=tsmovavg(mut3_rep3,'s',100,1);%simple moving average

% M46I+V82F+I84V Rep 1

fig13=figure;
plot(x',mut3_rep1)
title('M46I+V82F+I84V Rep1 Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAMut3Rep1,'r')

% M46I+V82F+I84V Rep 2

fig14=figure;
plot(x',mut3_rep2)
title('M46I+V82F+I84V Rep2 Protein-Ligand RMSD')
ylabel('RMSD')
```

```
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAMut3Rep2,'r')

% M46I+V82F+I84V Rep 3

fig15=figure;
plot(x',mut3_rep3)
title('M46I+V82F+I84V Rep3 Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAMut3Rep3,'r')

%Plotting Average M46I+V82F+I84V RMSD

Mut3_Average=(mut3_rep1+mut3_rep2+mut3_rep3)/3;
MAMut3Avg=tsmovavg(Mut3_Average,'s',100,1);%simple moving average

fig16=figure;
plot(x',Mut3_Average)
title('Average M46I+V82F+I84V Protein-Ligand RMSD')
ylabel('RMSD')
xlabel('Frames')
axis([0 500 0 2.2])
hold on
plot(x',MAMut3Avg,'r')
```


M46I+V82F+I84V Rep1 Protein-Ligand RMSD
RMSD
0
0.5
1
1.5
2
0
50
100
150
200
250
300
350
400
450
500
Frames


M46I+V82F+I84V Rep2 Protein-Ligand RMSD
RMSD
0
0.5
1
1.5
2
0
50
100
150
200
250
300
350
400
450
500
Frames


M46I+V82F+I84V Rep3 Protein-Ligand RMSD
RMSD
Frames
0
0.5
1
1.5
2
0
50
100
150
200
250
300
350
400
450
500


Average M46I+V82F+I84V Protein-Ligand RMSD
RMSD
Frames
0
0.5
1
1.5
2
0
50
100
150
200
250
300
350
400
450
500

B.2 Protein RMSF Script

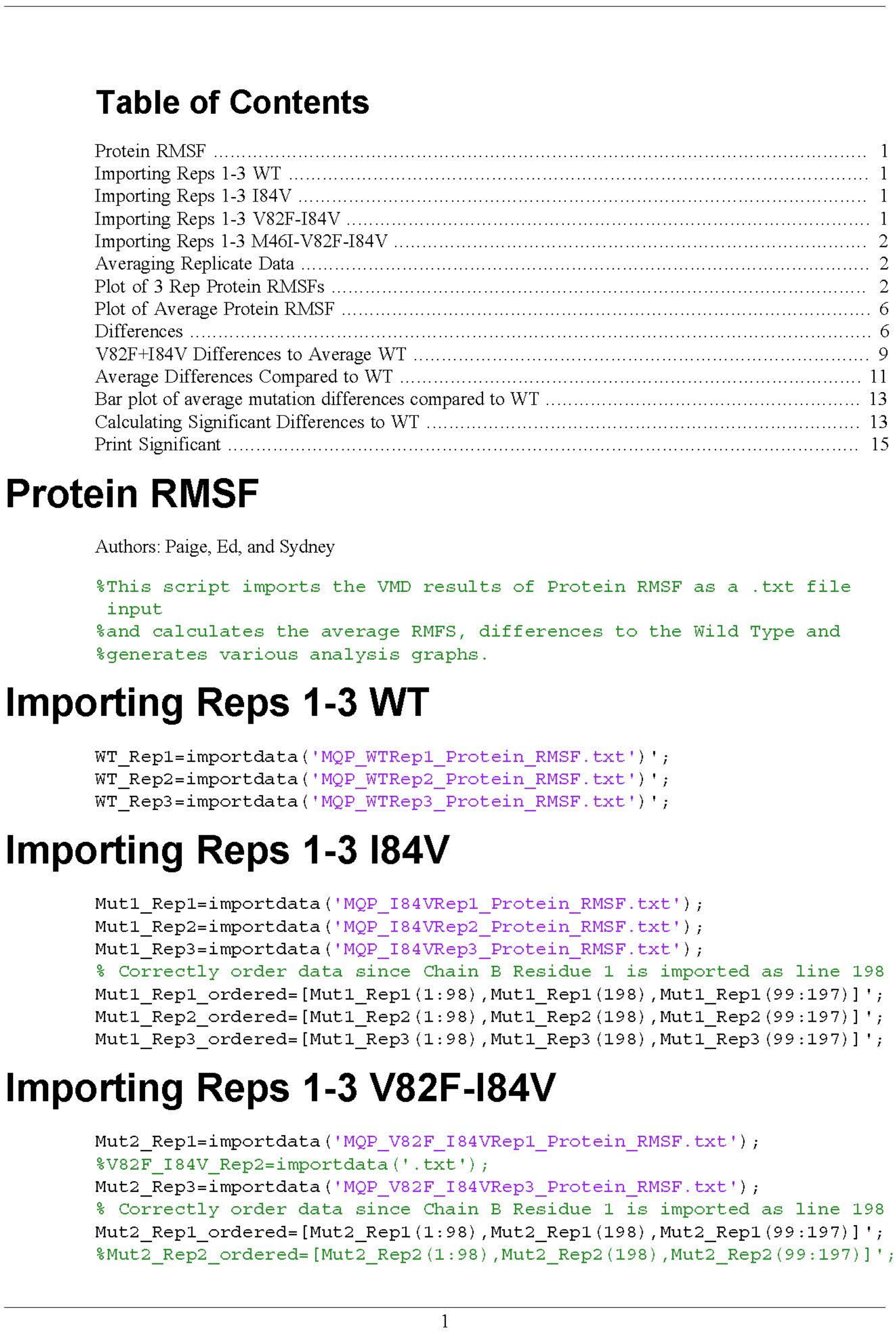


Table of Contents

Protein RMSF

Authors: Paige, Ed, and Sydney

```
%This script imports the VMD results of Protein RMSF as a .txt file
 input
%and calculates the average RMFS, differences to the Wild Type and
%generates various analysis graphs.
```

Importing Reps 1-3 WT

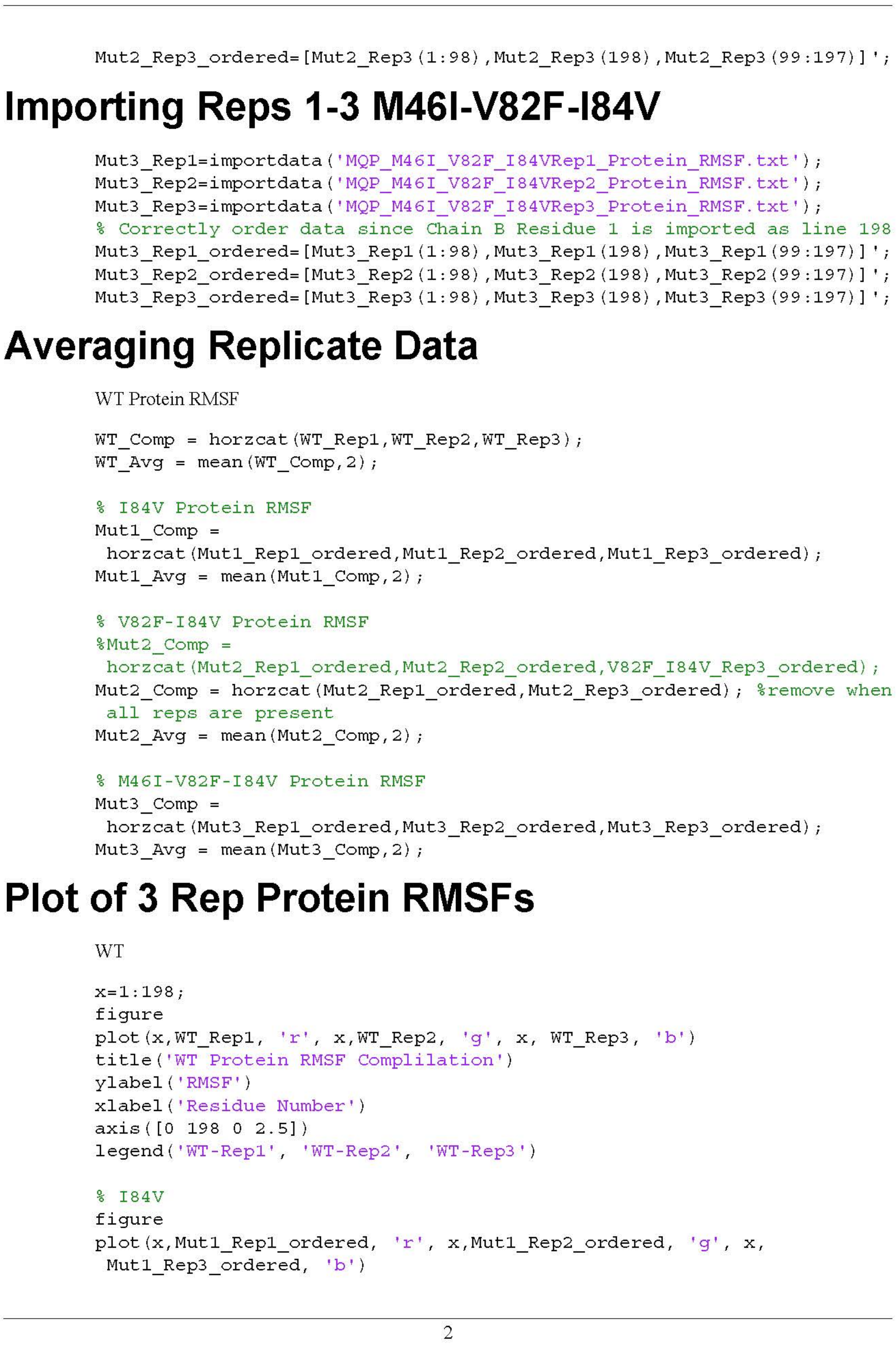
```
WT_Rep1=importdata('MQP_WTRep1_Protein_RMSF.txt')';
WT_Rep2=importdata('MQP_WTRep2_Protein_RMSF.txt')';
WT_Rep3=importdata('MQP_WTRep3_Protein_RMSF.txt')';
```

Importing Reps 1-3 I84V

```
Mut1_Rep1=importdata('MQP_I84VRep1_Protein_RMSF.txt');
Mut1_Rep2=importdata('MQP_I84VRep2_Protein_RMSF.txt');
Mut1_Rep3=importdata('MQP_I84VRep3_Protein_RMSF.txt');
% Correctly order data since Chain B Residue 1 is imported as line 198
Mut1_Rep1_ordered=[Mut1_Rep1(1:98),Mut1_Rep1(198),Mut1_Rep1(99:197)]';
Mut1_Rep2_ordered=[Mut1_Rep2(1:98),Mut1_Rep2(198),Mut1_Rep2(99:197)]';
Mut1_Rep3_ordered=[Mut1_Rep3(1:98),Mut1_Rep3(198),Mut1_Rep3(99:197)]';
```

Importing Reps 1-3 V82F-I84V

```
Mut2_Rep1=importdata('MQP_V82F_I84VRep1_Protein_RMSF.txt');
%V82F_I84V_Rep2=importdata('.txt');
Mut2_Rep3=importdata('MQP_V82F_I84VRep3_Protein_RMSF.txt');
% Correctly order data since Chain B Residue 1 is imported as line 198
Mut2_Rep1_ordered=[Mut2_Rep1(1:98),Mut2_Rep1(198),Mut2_Rep1(99:197)]';
%Mut2_Rep2_ordered=[Mut2_Rep2(1:98),Mut2_Rep2(198),Mut2_Rep2(99:197)]';
```



```
Mut2_Rep3_ordered=[Mut2_Rep3(1:98),Mut2_Rep3(198),Mut2_Rep3(99:197)]';
```

Importing Reps 1-3 M46I-V82F-I84V

```
Mut3_Rep1=importdata('MQP_M46I_V82F_I84VRep1_Protein_RMSF.txt');
Mut3_Rep2=importdata('MQP_M46I_V82F_I84VRep2_Protein_RMSF.txt');
Mut3_Rep3=importdata('MQP_M46I_V82F_I84VRep3_Protein_RMSF.txt');
% Correctly order data since Chain B Residue 1 is imported as line 198
Mut3_Rep1_ordered=[Mut3_Rep1(1:98),Mut3_Rep1(198),Mut3_Rep1(99:197)]';
Mut3_Rep2_ordered=[Mut3_Rep2(1:98),Mut3_Rep2(198),Mut3_Rep2(99:197)]';
Mut3_Rep3_ordered=[Mut3_Rep3(1:98),Mut3_Rep3(198),Mut3_Rep3(99:197)]';
```

Averaging Replicate Data

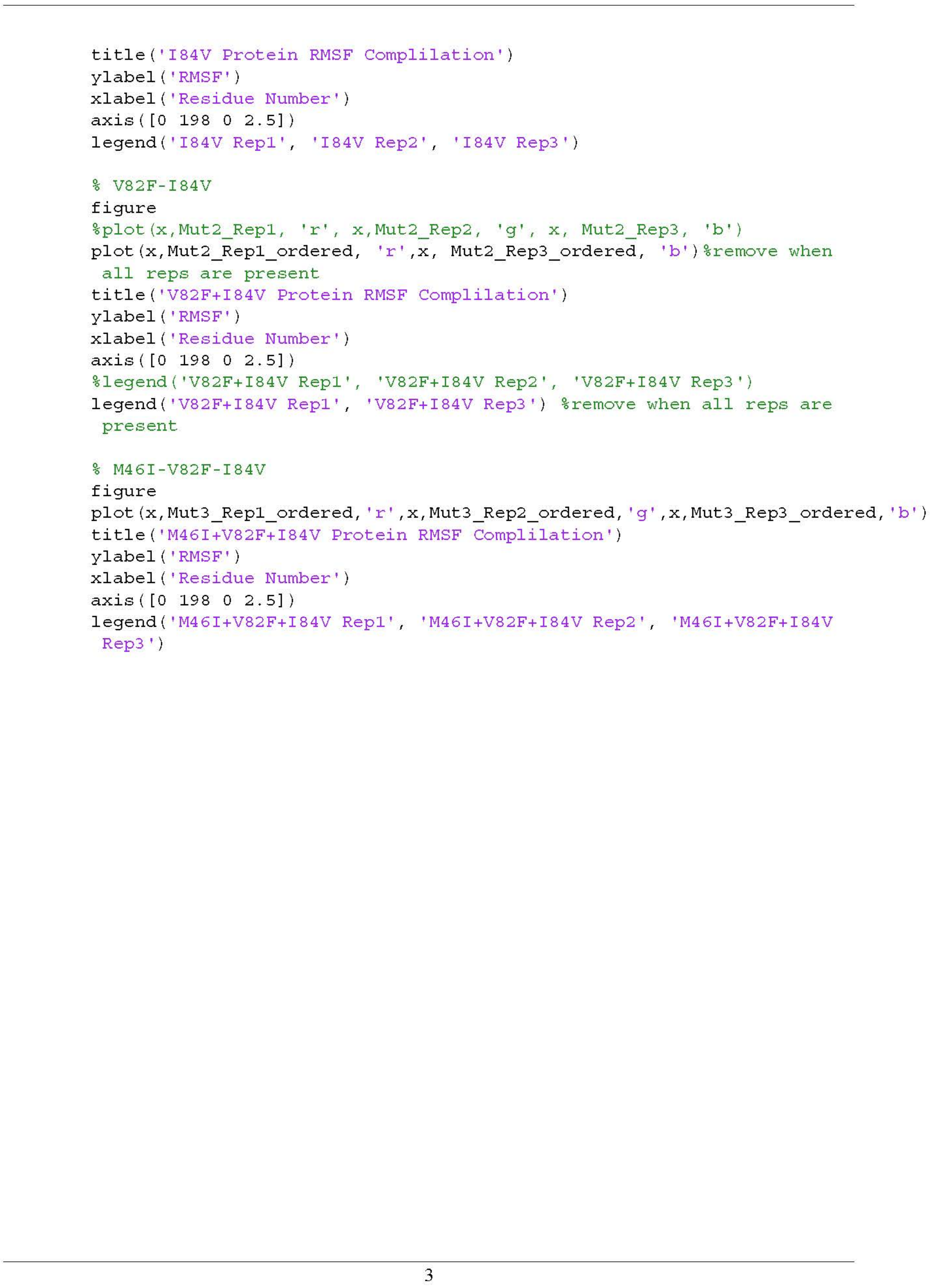
WT Protein RMSF

```
WT_Comp = horzcat(WT_Rep1,WT_Rep2,WT_Rep3);
WT_Avg = mean(WT_Comp,2);

% I84V Protein RMSF
Mut1_Comp =
 horzcat(Mut1_Rep1_ordered,Mut1_Rep2_ordered,Mut1_Rep3_ordered);
Mut1_Avg = mean(Mut1_Comp,2);

% V82F-I84V Protein RMSF
%Mut2_Comp =
 horzcat(Mut2_Rep1_ordered,Mut2_Rep2_ordered,V82F_I84V_Rep3_ordered);
Mut2_Comp = horzcat(Mut2_Rep1_ordered,Mut2_Rep3_ordered); %remove when
 all reps are present
Mut2_Avg = mean(Mut2_Comp,2);

% M46I-V82F-I84V Protein RMSF
Mut3_Comp =
 horzcat(Mut3_Rep1_ordered,Mut3_Rep2_ordered,Mut3_Rep3_ordered);
Mut3_Avg = mean(Mut3_Comp,2);
```

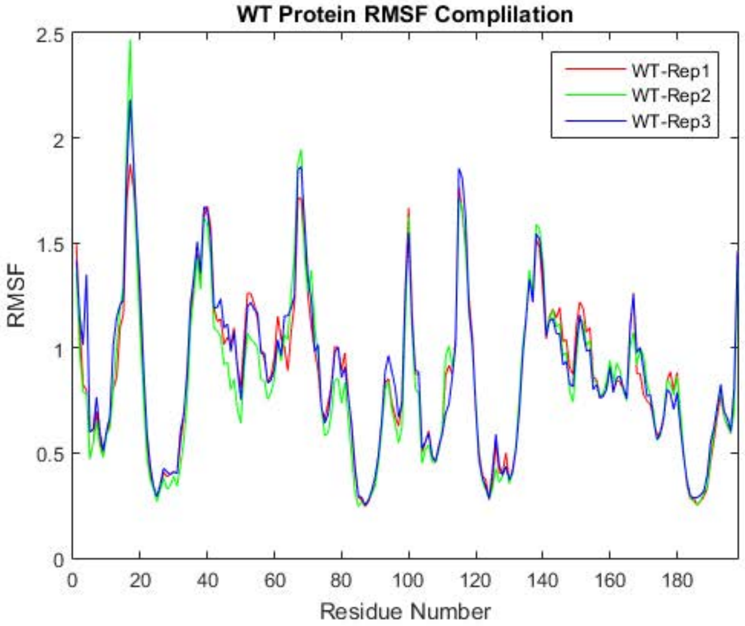
Plot of 3 Rep Protein RMSFs

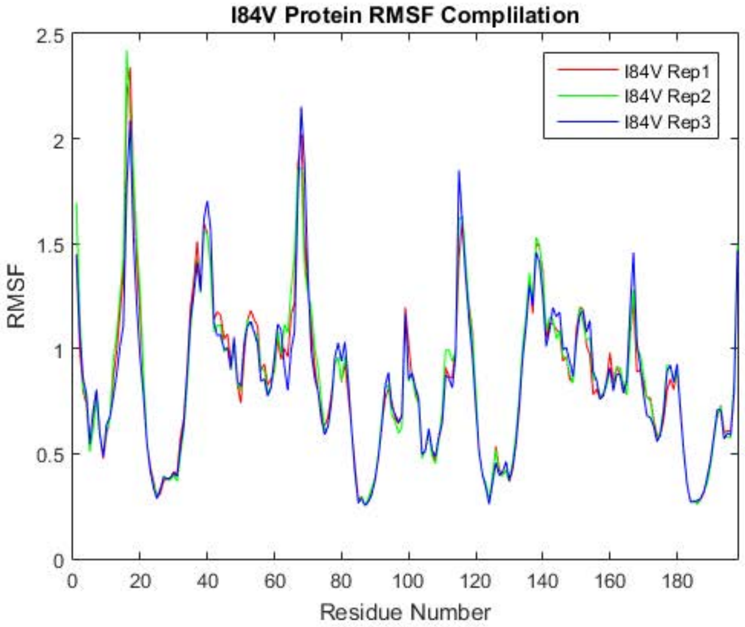
WT

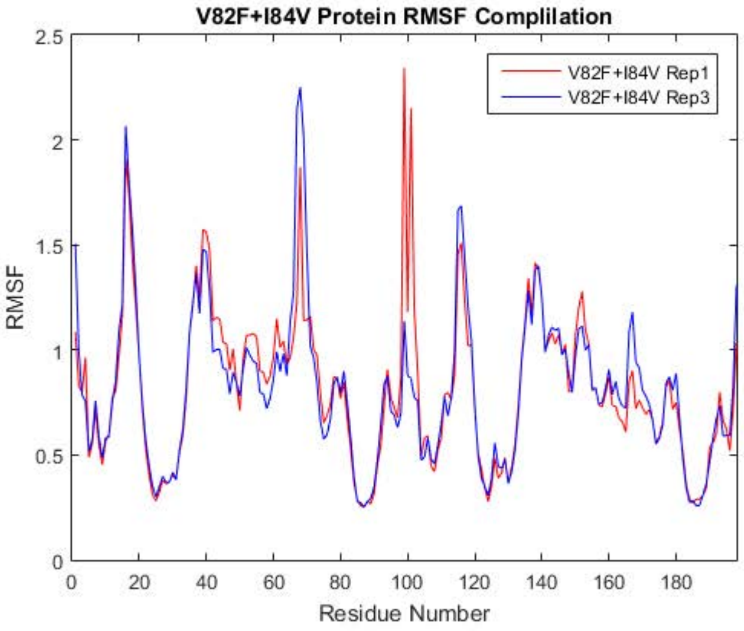
```
x=1:198;
figure
plot(x,WT_Rep1, 'r', x,WT_Rep2, 'g', x, WT_Rep3, 'b')
title('WT Protein RMSF Complilation')
ylabel('RMSF')
xlabel('Residue Number')
axis([0 198 0 2.5])
legend('WT-Rep1', 'WT-Rep2', 'WT-Rep3')

% I84V
figure
plot(x,Mut1_Rep1_ordered, 'r', x,Mut1_Rep2_ordered, 'g', x,
 Mut1_Rep3_ordered, 'b')
```

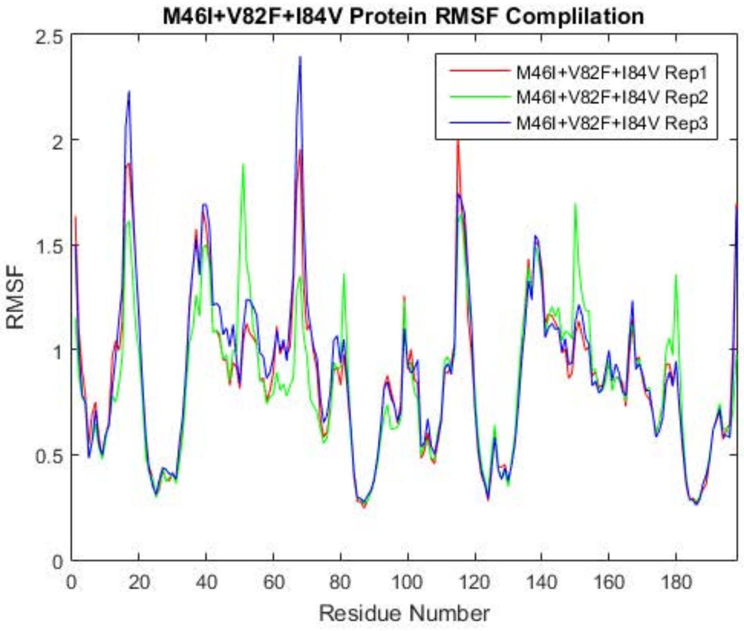


```
title('I84V Protein RMSF Complilation')
ylabel('RMSF')
xlabel('Residue Number')
axis([0 198 0 2.5])
legend('I84V Rep1', 'I84V Rep2', 'I84V Rep3')

% V82F-I84V
figure
%plot(x,Mut2_Rep1, 'r', x,Mut2_Rep2, 'g', x, Mut2_Rep3, 'b')
plot(x,Mut2_Rep1_ordered, 'r',x, Mut2_Rep3_ordered, 'b')%remove when
 all reps are present
title('V82F+I84V Protein RMSF Complilation')
ylabel('RMSF')
xlabel('Residue Number')
axis([0 198 0 2.5])
%legend('V82F+I84V Rep1', 'V82F+I84V Rep2', 'V82F+I84V Rep3')
legend('V82F+I84V Rep1', 'V82F+I84V Rep3') %remove when all reps are
 present

% M46I-V82F-I84V
figure
plot(x,Mut3_Rep1_ordered,'r',x,Mut3_Rep2_ordered,'g',x,Mut3_Rep3_ordered,'b')
title('M46I+V82F+I84V Protein RMSF Complilation')
ylabel('RMSF')
xlabel('Residue Number')
axis([0 198 0 2.5])
legend('M46I+V82F+I84V Rep1', 'M46I+V82F+I84V Rep2', 'M46I+V82F+I84V
 Rep3')
```


WT Protein RMSF Complilation
WT-Rep1
WT-Rep2
WT-Rep3
RMSF
0
0.5
1
1.5
2
2.5
Residue Number
0
20
40
60
80
100
120
140
160
180

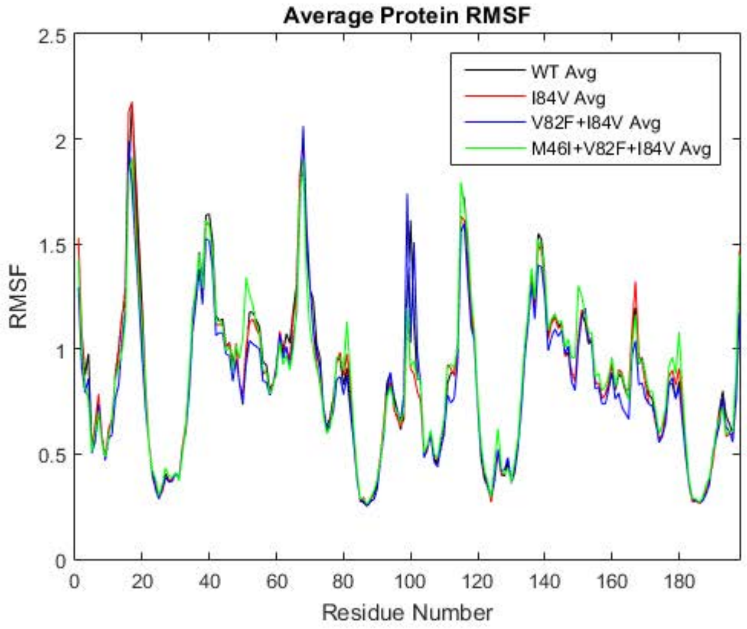

I84V Protein RMSF Complilation
I84V Rep1
I84V Rep2
I84V Rep3
RMSF
0
0.5
1
1.5
2
2.5
Residue Number
0
20
40
60
80
100
120
140
160
180


V82F+I84V Protein RMSF Complilation
V82F+I84V Rep1
V82F+I84V Rep3
RMSF
0
0.5
1
1.5
2
2.5
0
20
40
60
80
100
120
140
160
180
Residue Number


M46I+V82F+I84V Protein RMSF Complilation
M46I+V82F+I84V Rep1
M46I+V82F+I84V Rep2
M46I+V82F+I84V Rep3
RMSF
0
0.5
1
1.5
2
2.5
0
20
40
60
80
100
120
140
160
180
Residue Number

Plot of Average Protein RMSF

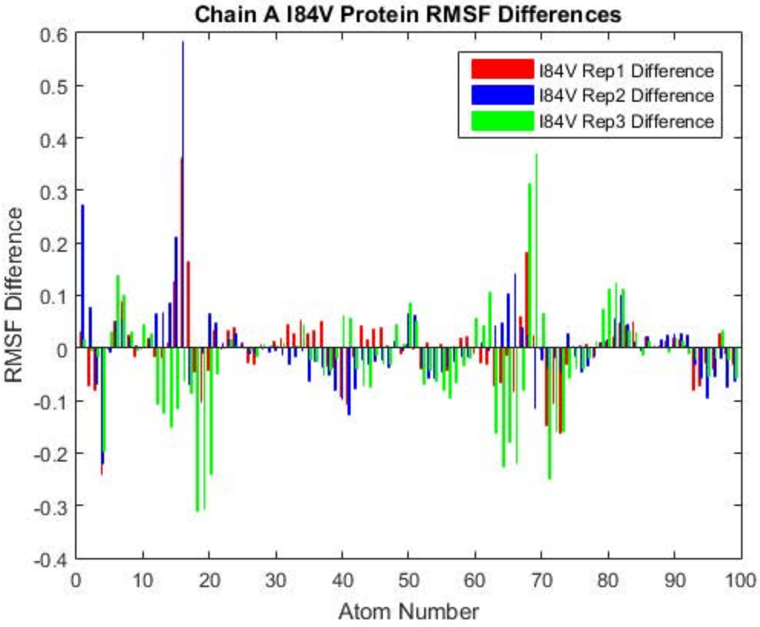
```
figure
x = 1:198;
plot(x,WT_Avg','k',x,Mut1_Avg','r',x,Mut2_Avg','b',x,Mut3_Avg','g')
title('Average Protein RMSF')
ylabel('RMSF')
xlabel('Residue Number')
axis([0 198 0 2.5])
legend('WT Avg','I84V Avg','V82F+I84V Avg','M46I+V82F+I84V Avg')
```

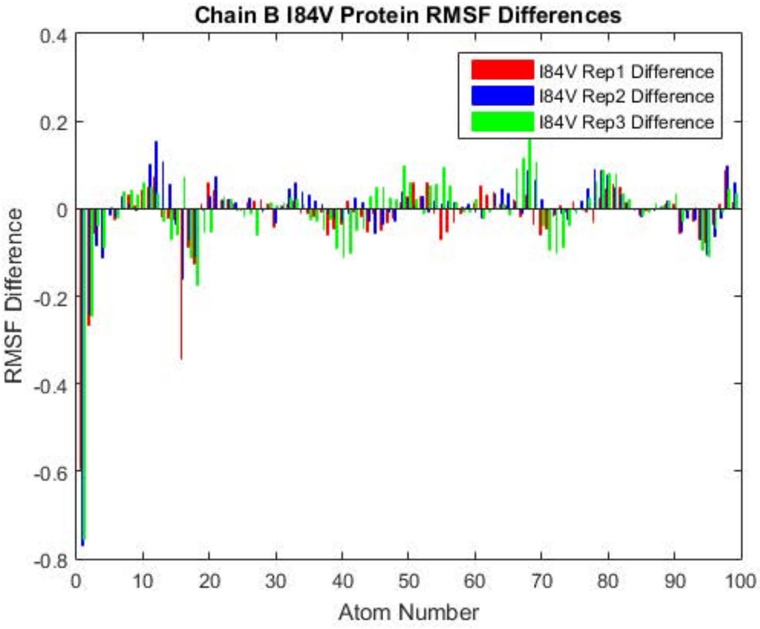


Differences

```
% I84V Differences to Average WT
Mut1_Rep1_Diff = Mut1_Rep1_ordered-WT_Avg;
Mut1_Rep2_Diff = Mut1_Rep2_ordered-WT_Avg;
Mut1_Rep3_Diff = Mut1_Rep3_ordered-WT_Avg;
Mut1_diffs = horzcat(Mut1_Rep1_Diff,Mut1_Rep2_Diff,Mut1_Rep3_Diff);
Mut1_Avg_diffs = mean(Mut1_diffs,2);
% Chain A and B Differences
Mut1_diffs_A =
 horzcat(Mut1_diffs(1:99,1),Mut1_diffs(1:99,2),Mut1_diffs(1:99,3));
Mut1_diffs_B =
 horzcat(Mut1_diffs(100:198,1),Mut1_diffs(100:198,2),Mut1_diffs(100:198,3));
```

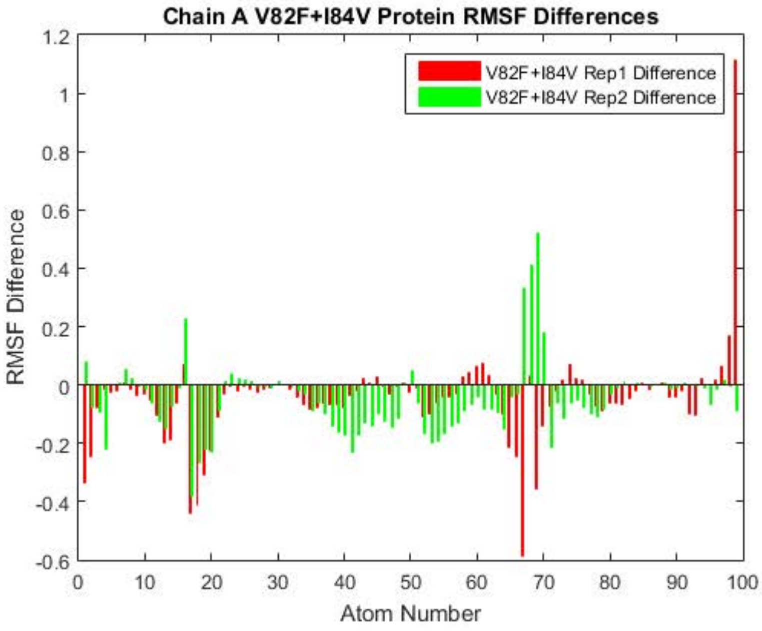
```
% Plotting Chain A I84V Differences to Average WT
figure
x = 1:99;
Mut1_A_bar_graph = bar(x,Mut1_diffs_A);
Mut1_A_bar_graph(1).FaceColor='r';
Mut1_A_bar_graph(2).FaceColor='b';
Mut1_A_bar_graph(3).FaceColor='g';
Mut1_A_bar_graph(1).EdgeColor='r';
Mut1_A_bar_graph(2).EdgeColor='b';
Mut1_A_bar_graph(3).EdgeColor='g';
title('Chain A I84V Protein RMSF Differences')
xlabel('Atom Number')
ylabel('RMSF Difference')
legend('I84V Rep1 Difference','I84V Rep2 Difference','I84V Rep3
 Difference')

% Plotting Chain B I84V Differences to Average WT
figure
x = 1:99;
Mut1_B_bar_graph = bar(x,Mut1_diffs_B);
Mut1_B_bar_graph(1).FaceColor='r';
Mut1_B_bar_graph(2).FaceColor='b';
Mut1_B_bar_graph(3).FaceColor='g';
Mut1_B_bar_graph(1).EdgeColor='r';
Mut1_B_bar_graph(2).EdgeColor='b';
Mut1_B_bar_graph(3).EdgeColor='g';
title('Chain B I84V Protein RMSF Differences')
xlabel('Atom Number')
ylabel('RMSF Difference')
legend('I84V Rep1 Difference','I84V Rep2 Difference','I84V Rep3
 Difference')
```


Chain A I84V Protein RMSF Differences
I84V Rep1 Difference
I84V Rep2 Difference
I84V Rep3 Difference
RMSF Difference
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0
10
20
30
40
50
60
70
80
90
100
Atom Number

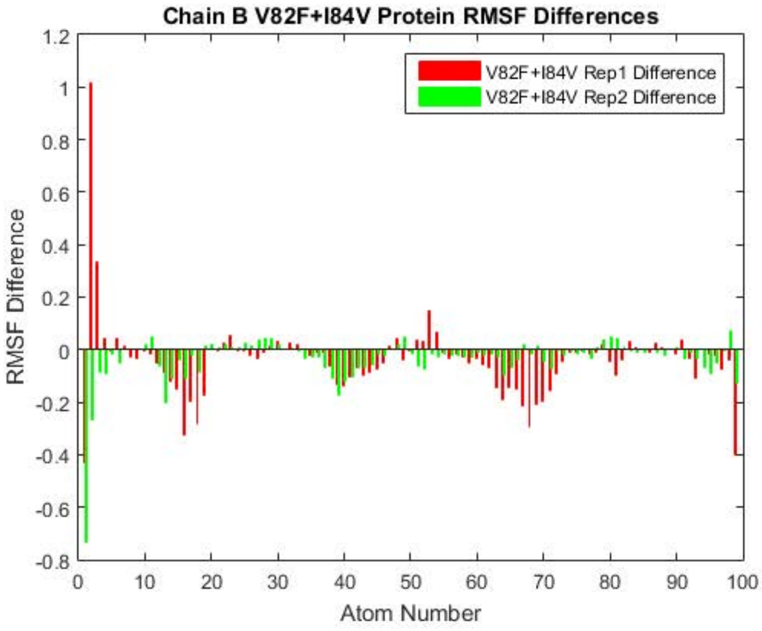

Chain B I84V Protein RMSF Differences
I84V Rep1 Difference
I84V Rep2 Difference
I84V Rep3 Difference
RMSF Difference
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
10
20
30
40
50
60
70
80
90
100
Atom Number

V82F+I84V Differences to Average WT

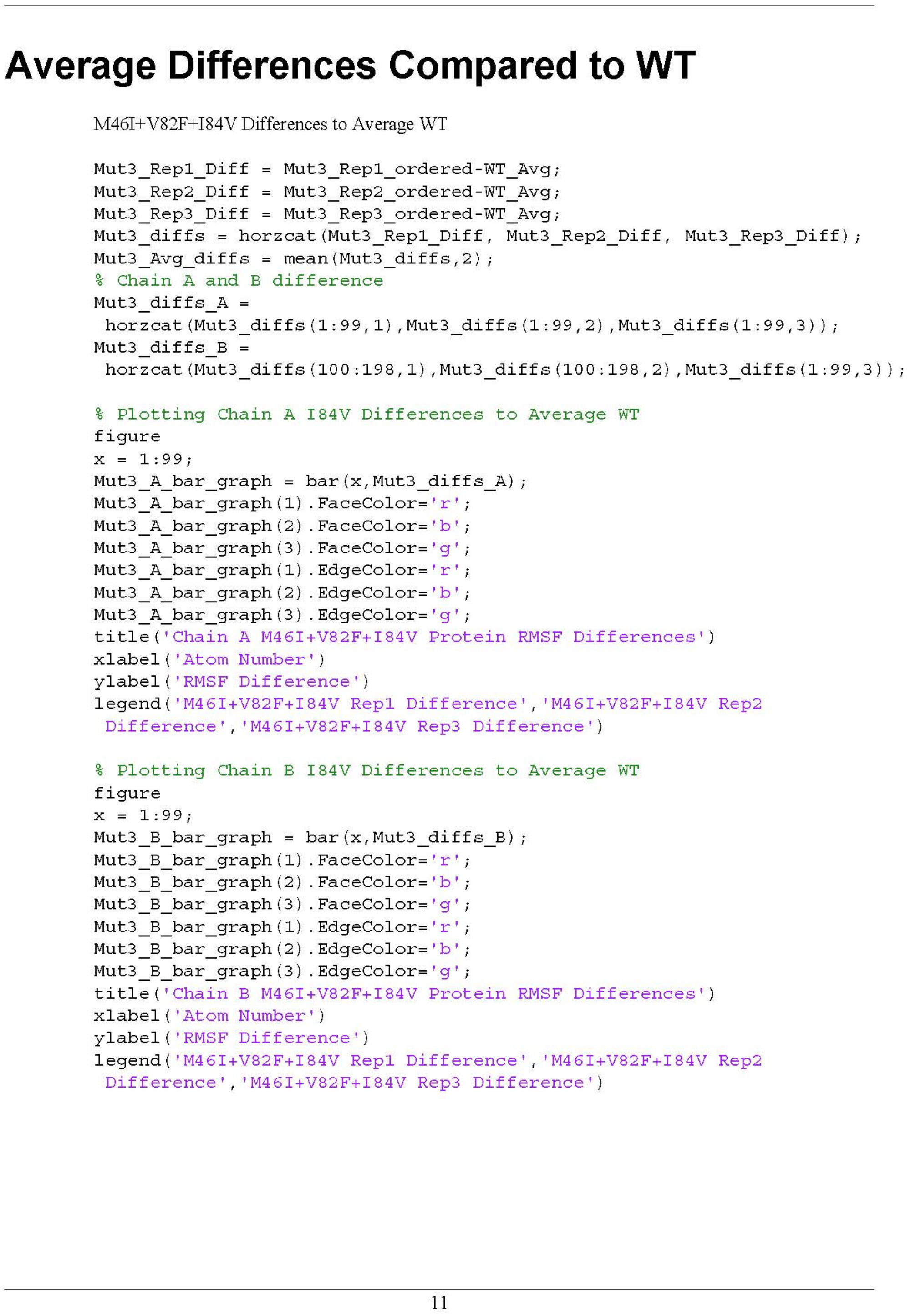
```
Mut2_Rep1_Diff = Mut2_Rep1_ordered-WT_Avg;
%Mut2_Rep2_Diff = Mut2_Rep2_ordered-WT_Avg;
Mut2_Rep3_Diff = Mut2_Rep3_ordered-WT_Avg;
Mut2_diffs = horzcat(Mut2_Rep1_Diff,Mut2_Rep3_Diff);
Mut2_Avg_diffs= mean(Mut2_diffs,2);
% Chain A and B differences
Mut2_diffs_A = horzcat(Mut2_diffs(1:99,1),Mut2_diffs(1:99,2));
Mut2_diffs_B = horzcat(Mut2_diffs(100:198,1),Mut2_diffs(100:198,2));

% Plotting Chain A I84V Differences to Average WT
figure
x = 1:99;
Mut2_A_bar_graph = bar(x,Mut2_diffs_A);
Mut2_A_bar_graph(1).FaceColor= 'r';
%Mut2_A_bar_graph(2).FaceColor='b';
Mut2_A_bar_graph(2).FaceColor= 'g';
Mut2_A_bar_graph(1).EdgeColor= 'r';
Mut2_A_bar_graph(2).EdgeColor= 'g';
title('Chain A V82F+I84V Protein RMSF Differences')
xlabel('Atom Number')
ylabel('RMSF Difference')
legend('V82F+I84V Rep1 Difference','V82F+I84V Rep2 Difference')

% Plotting Chain B I84V Differences to Average WT
figure
x = 1:99;
Mut2_B_bar_graph = bar(x,Mut2_diffs_B);
Mut2_B_bar_graph(1).FaceColor='r';
%Mut2_B_bar_graph(2).FaceColor='b';
Mut2_B_bar_graph(2).FaceColor='g';
Mut2_B_bar_graph(1).EdgeColor='r';
Mut2_B_bar_graph(2).EdgeColor='g';
title('Chain B V82F+I84V Protein RMSF Differences')
xlabel('Atom Number')
ylabel('RMSF Difference')
legend('V82F+I84V Rep1 Difference','V82F+I84V Rep2 Difference')
```



Chain A V82F+I84V Protein RMSF Differences



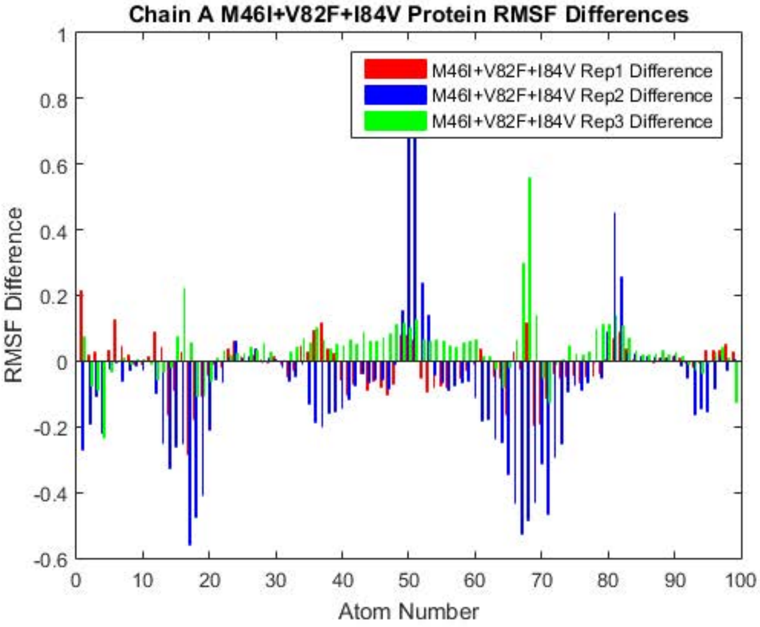
Chain B V82F+I84V Protein RMSF Differences

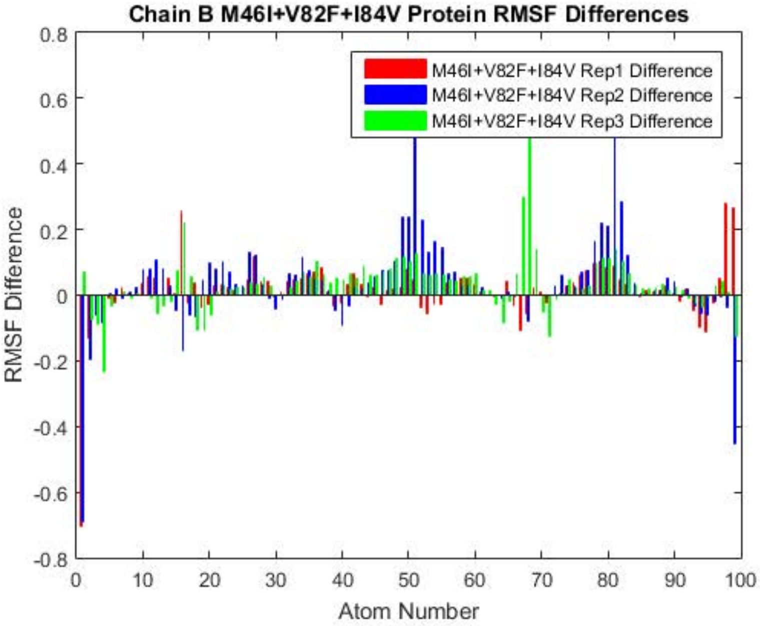


Average Differences Compared to WT

M46I+V82F+I84V Differences to Average WT

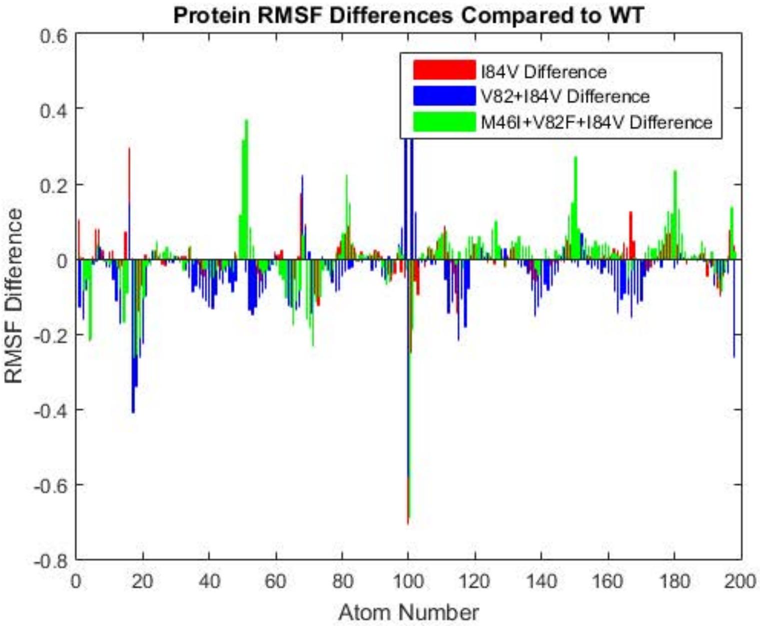
```
Mut3_Rep1_Diff = Mut3_Rep1_ordered-WT_Avg;
Mut3_Rep2_Diff = Mut3_Rep2_ordered-WT_Avg;
Mut3_Rep3_Diff = Mut3_Rep3_ordered-WT_Avg;
Mut3_diffs = horzcat(Mut3_Rep1_Diff, Mut3_Rep2_Diff, Mut3_Rep3_Diff);
Mut3_Avg_diffs = mean(Mut3_diffs,2);
% Chain A and B difference
Mut3_diffs_A =
 horzcat(Mut3_diffs(1:99,1),Mut3_diffs(1:99,2),Mut3_diffs(1:99,3));
Mut3_diffs_B =
 horzcat(Mut3_diffs(100:198,1),Mut3_diffs(100:198,2),Mut3_diffs(1:99,3));

% Plotting Chain A I84V Differences to Average WT
figure
x = 1:99;
Mut3_A_bar_graph = bar(x,Mut3_diffs_A);
Mut3_A_bar_graph(1).FaceColor='r';
Mut3_A_bar_graph(2).FaceColor='b';
Mut3_A_bar_graph(3).FaceColor='g';
Mut3_A_bar_graph(1).EdgeColor='r';
Mut3_A_bar_graph(2).EdgeColor='b';
Mut3_A_bar_graph(3).EdgeColor='g';
title('Chain A M46I+V82F+I84V Protein RMSF Differences')
xlabel('Atom Number')
ylabel('RMSF Difference')
legend('M46I+V82F+I84V Rep1 Difference','M46I+V82F+I84V Rep2
 Difference','M46I+V82F+I84V Rep3 Difference')

% Plotting Chain B I84V Differences to Average WT
figure
x = 1:99;
Mut3_B_bar_graph = bar(x,Mut3_diffs_B);
Mut3_B_bar_graph(1).FaceColor='r';
Mut3_B_bar_graph(2).FaceColor='b';
Mut3_B_bar_graph(3).FaceColor='g';
Mut3_B_bar_graph(1).EdgeColor='r';
Mut3_B_bar_graph(2).EdgeColor='b';
Mut3_B_bar_graph(3).EdgeColor='g';
title('Chain B M46I+V82F+I84V Protein RMSF Differences')
xlabel('Atom Number')
ylabel('RMSF Difference')
legend('M46I+V82F+I84V Rep1 Difference','M46I+V82F+I84V Rep2
 Difference','M46I+V82F+I84V Rep3 Difference')
```


Chain A M46I+V82F+I84V Protein RMSF Differences
M46I+V82F+I84V Rep1 Difference
M46I+V82F+I84V Rep2 Difference
M46I+V82F+I84V Rep3 Difference
RMSF Difference
Atom Number
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
10
20
30
40
50
60
70
80
90
100


Chain B M46I+V82F+I84V Protein RMSF Differences
M46I+V82F+I84V Rep1 Difference
M46I+V82F+I84V Rep2 Difference
M46I+V82F+I84V Rep3 Difference
RMSF Difference
Atom Number
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
10
20
30
40
50
60
70
80
90
100

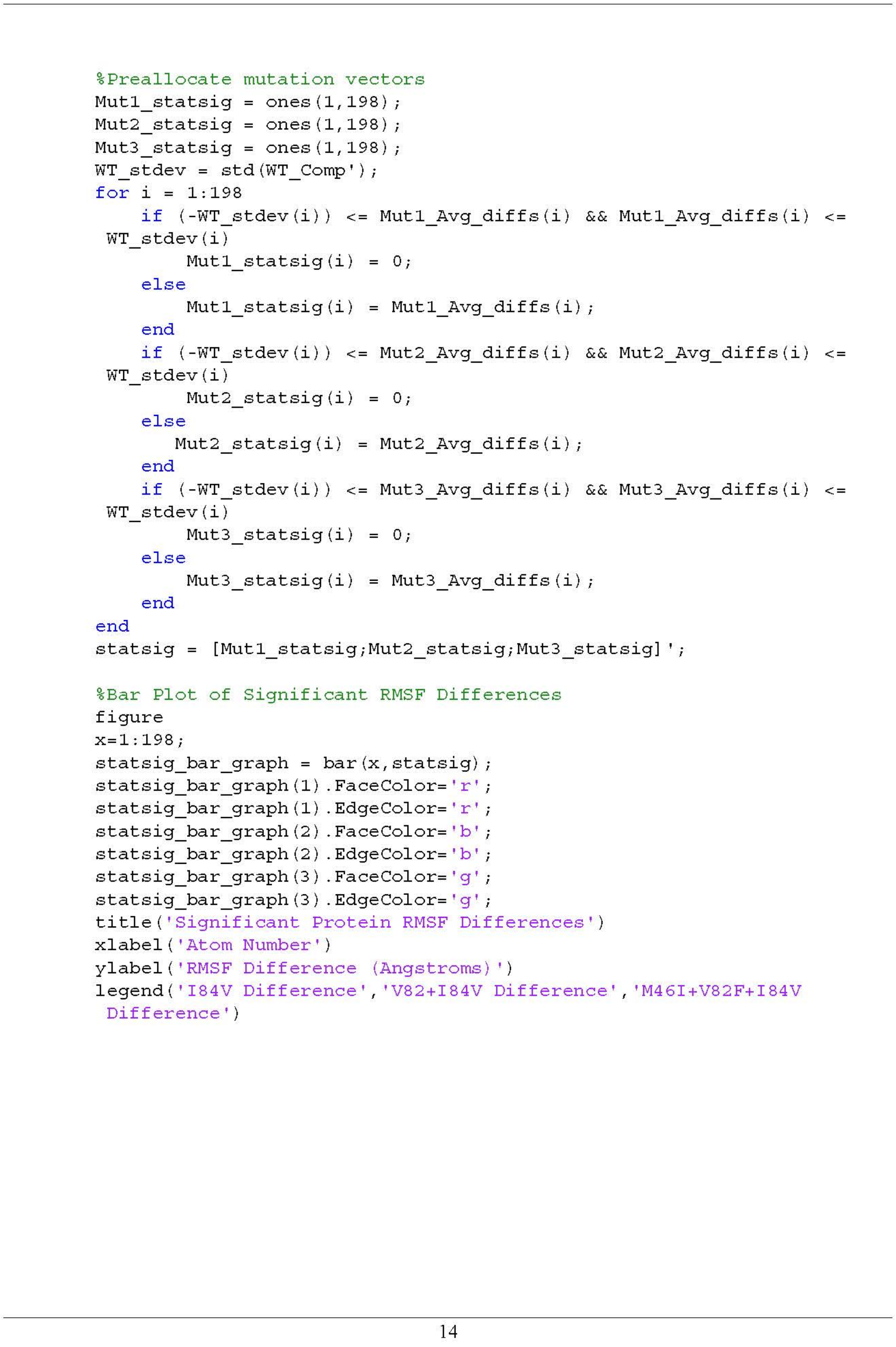
Bar plot of average mutation differences compared to WT

```
Avg_diffs = horzcat(Mut1_Avg_diffs,Mut2_Avg_diffs,Mut3_Avg_diffs);
figure
x=1:198;
bar_graph = bar(x,Avg_diffs);
bar_graph(1).FaceColor='r';
bar_graph(1).EdgeColor='r';
bar_graph(2).FaceColor='b';
bar_graph(2).EdgeColor='b';
bar_graph(3).FaceColor='g';
bar_graph(3).EdgeColor='g';
title('Protein RMSF Differences Compared to WT')
xlabel('Atom Number')
ylabel('RMSF Difference')
legend('I84V Difference','V82+I84V Difference','M46I+V82F+I84V
 Difference')
```

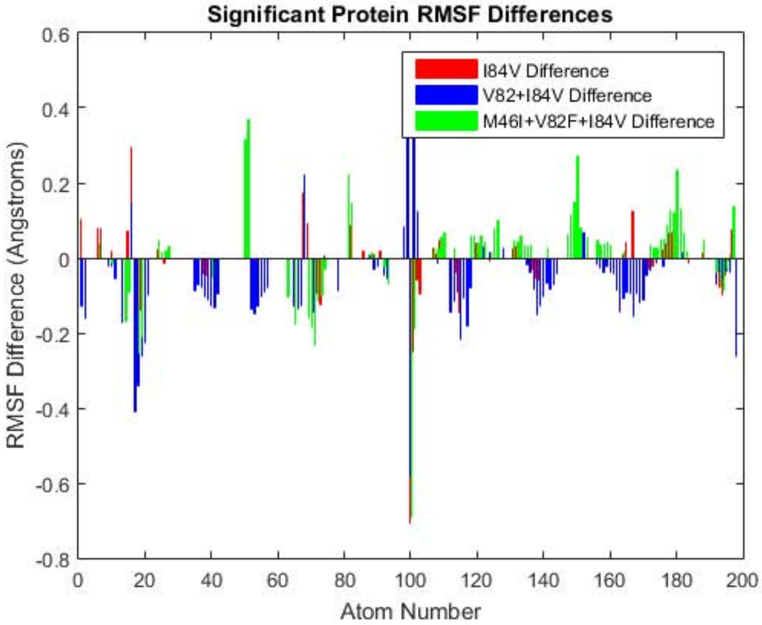


Calculating Significant Differences to WT

```
%Note: Significant difference is defined by having a
%difference compared to the wild type RMSF that is greater than the
%standard deviation of the 3 WT data points for each residue.
```



```
%Preallocate mutation vectors
Mut1_statsig = ones(1,198);
Mut2_statsig = ones(1,198);
Mut3_statsig = ones(1,198);
WT_stdev = std(WT_Comp');
for i = 1:198
    if (-WT_stdev(i)) <= Mut1_Avg_diffs(i) && Mut1_Avg_diffs(i) <=
 WT_stdev(i)
        Mut1_statsig(i) = 0;
    else
        Mut1_statsig(i) = Mut1_Avg_diffs(i);
    end
    if (-WT_stdev(i)) <= Mut2_Avg_diffs(i) && Mut2_Avg_diffs(i) <=
 WT_stdev(i)
        Mut2_statsig(i) = 0;
    else
       Mut2_statsig(i) = Mut2_Avg_diffs(i);
    end
    if (-WT_stdev(i)) <= Mut3_Avg_diffs(i) && Mut3_Avg_diffs(i) <=
 WT_stdev(i)
        Mut3_statsig(i) = 0;
    else
        Mut3_statsig(i) = Mut3_Avg_diffs(i);
    end
end
statsig = [Mut1_statsig;Mut2_statsig;Mut3_statsig]';

%Bar Plot of Significant RMSF Differences
figure
x=1:198;
statsig_bar_graph = bar(x,statsig);
statsig_bar_graph(1).FaceColor='r';
statsig_bar_graph(1).EdgeColor='r';
statsig_bar_graph(2).FaceColor='b';
statsig_bar_graph(2).EdgeColor='b';
statsig_bar_graph(3).FaceColor='g';
statsig_bar_graph(3).EdgeColor='g';
title('Significant Protein RMSF Differences')
xlabel('Atom Number')
ylabel('RMSF Difference (Angstroms)')
legend('I84V Difference','V82+I84V Difference','M46I+V82F+I84V
 Difference')
```



Print Significant

```
Mut1_sigdiff = [];
Mut2_sigdiff = [];
Mut3_sigdiff = [];
for i = 1:38;
    if Mut1_statsig(i) ~= 0
        Mut1_sigdiff = [Mut1_sigdiff; i,Mut1_statsig(i)];
    if Mut2_statsig(i) ~= 0
        Mut2_sigdiff = [Mut2_sigdiff; i,Mut2_statsig(i)];
    if Mut3_statsig(i) ~= 0
        Mut3_sigdiff = [Mut3_sigdiff; i,Mut3_statsig(i)];
    end
    end
    end
end
```

Published with MATLAB® R2015a

B.3 Ligand RMSF Script

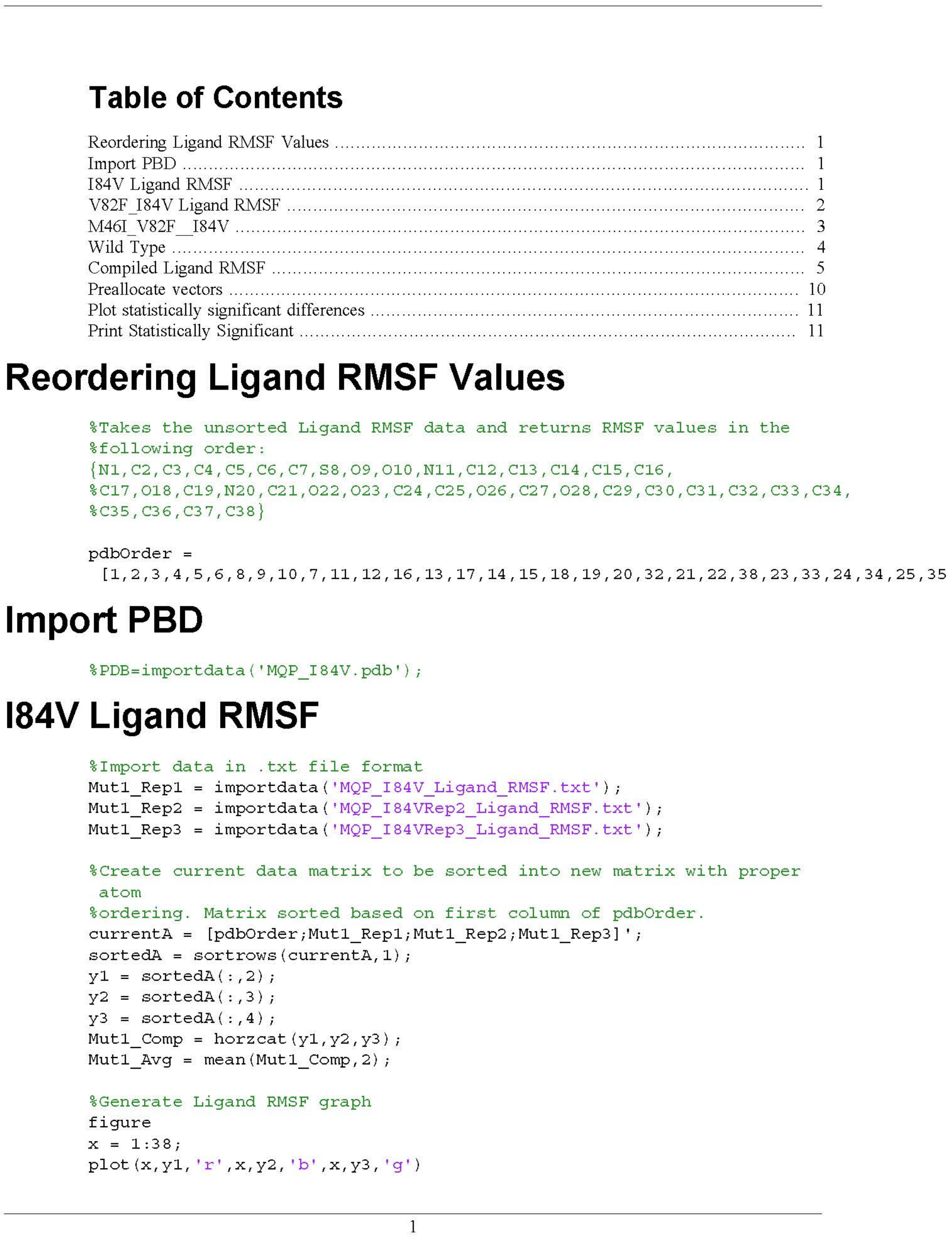


Table of Contents

Reordering Ligand RMSF Values

```
%Takes the unsorted Ligand RMSF data and returns RMSF values in the
%following order:
{N1,C2,C3,C4,C5,C6,C7,S8,O9,O10,N11,C12,C13,C14,C15,C16,
%C17,O18,C19,N20,C21,O22,O23,C24,C25,O26,C27,O28,C29,C30,C31,C32,C33,C34,
%C35,C36,C37,C38}

pdbOrder =
 [1,2,3,4,5,6,8,9,10,7,11,12,16,13,17,14,15,18,19,20,32,21,22,38,23,33,24,34,25,35
```

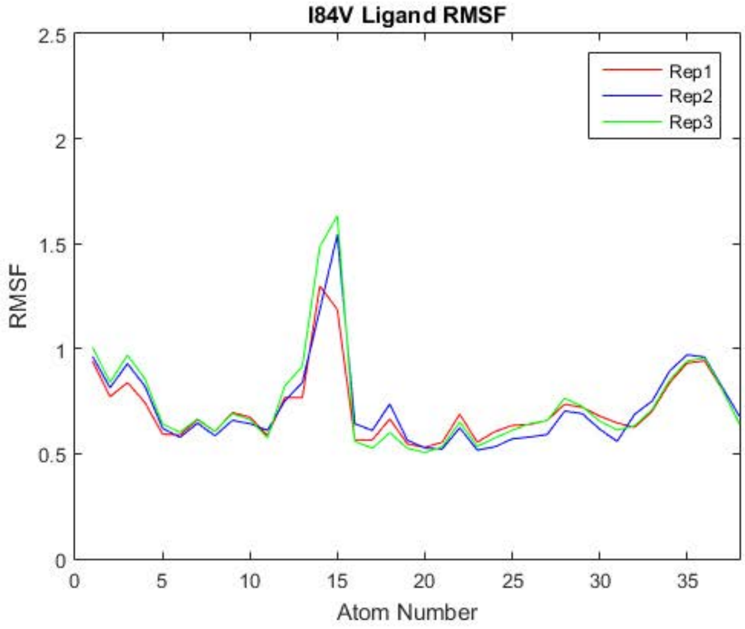
Import PBD

```
%PDB=importdata('MQP_I84V.pdb');
```

I84V Ligand RMSF

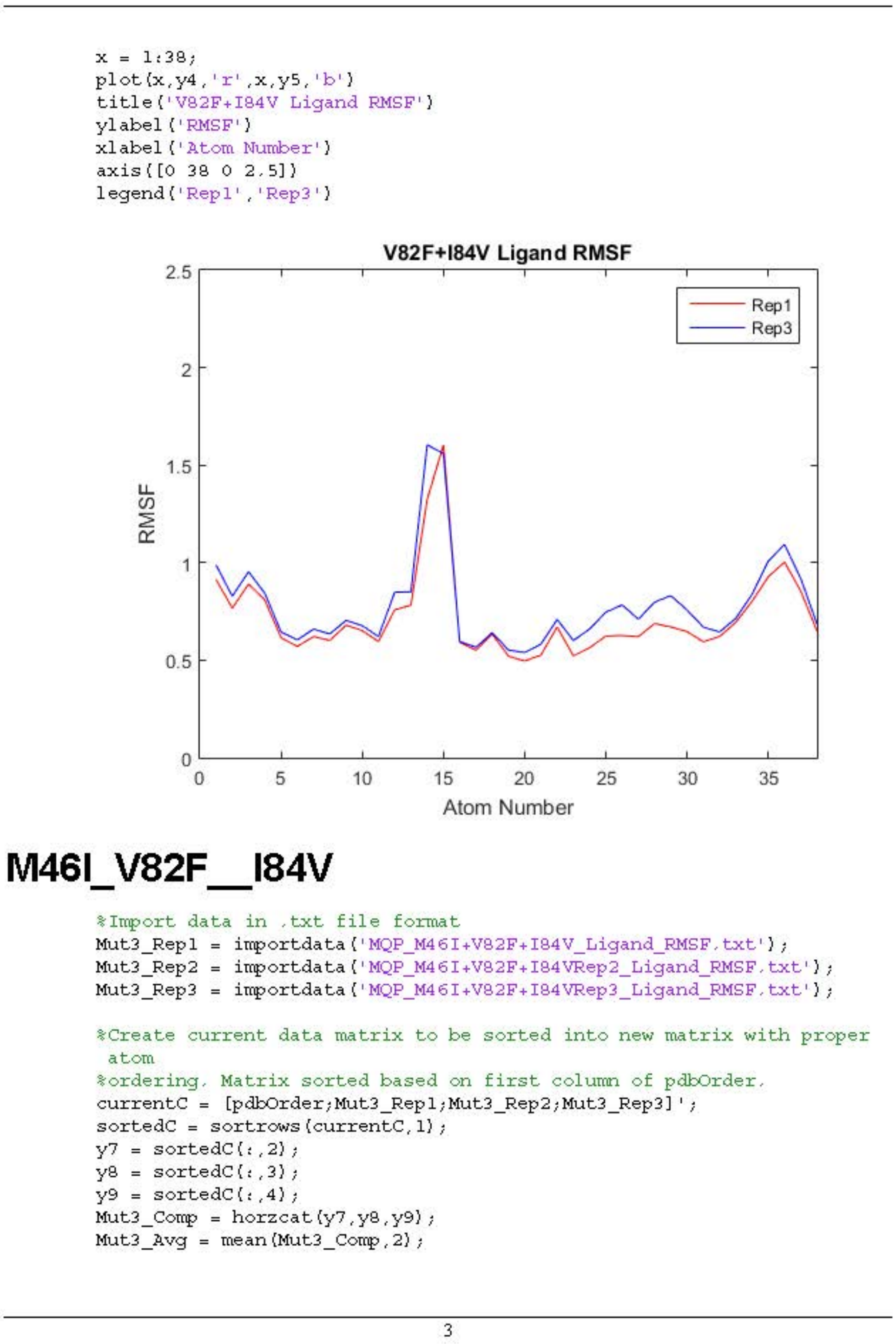
```
%Import data in .txt file format
Mut1_Rep1 = importdata('MQP_I84V_Ligand_RMSF.txt');
Mut1_Rep2 = importdata('MQP_I84VRep2_Ligand_RMSF.txt');
Mut1_Rep3 = importdata('MQP_I84VRep3_Ligand_RMSF.txt');

%Create current data matrix to be sorted into new matrix with proper
 atom
%ordering. Matrix sorted based on first column of pdbOrder.
currentA = [pdbOrder;Mut1_Rep1;Mut1_Rep2;Mut1_Rep3]';
sortedA = sortrows(currentA,1);
y1 = sortedA(:,2);
y2 = sortedA(:,3);
y3 = sortedA(:,4);
Mut1_Comp = horzcat(y1,y2,y3);
Mut1_Avg = mean(Mut1_Comp,2);

%Generate Ligand RMSF graph
figure
x = 1:38;
plot(x,y1,'r',x,y2,'b',x,y3,'g')
```

```
title('I84V Ligand RMSF')
ylabel('RMSF')
xlabel('Atom Number')
axis([0 38 0 2.5])
legend('Rep1','Rep2','Rep3')
```



V82F_I84V Ligand RMSF

```
%Import data in .txt file format
Mut2_Rep1 = importdata('MQP_V82F+I84V_Ligand_RMSF.txt');
%Mut2_Rep2 = importdata('');
Mut2_Rep3 = importdata('MQP_V82F+I84VRep3_Ligand_RMSF.txt');

%Create current data matrix to be sorted into new matrix with proper
 atom
%ordering. Matrix sorted based on first column of pdbOrder.
currentB = [pdbOrder;Mut2_Rep1;Mut2_Rep3]';
sortedB = sortrows(currentB,1);
y4 = sortedB(:,2);
y5 = sortedB(:,3);
%y6 = sortedB(:,4);
Mut2_Comp = horzcat(y4,y5);
Mut2_Avg = mean(Mut2_Comp,2);

%Generate Ligand RMSF graph
figure
```

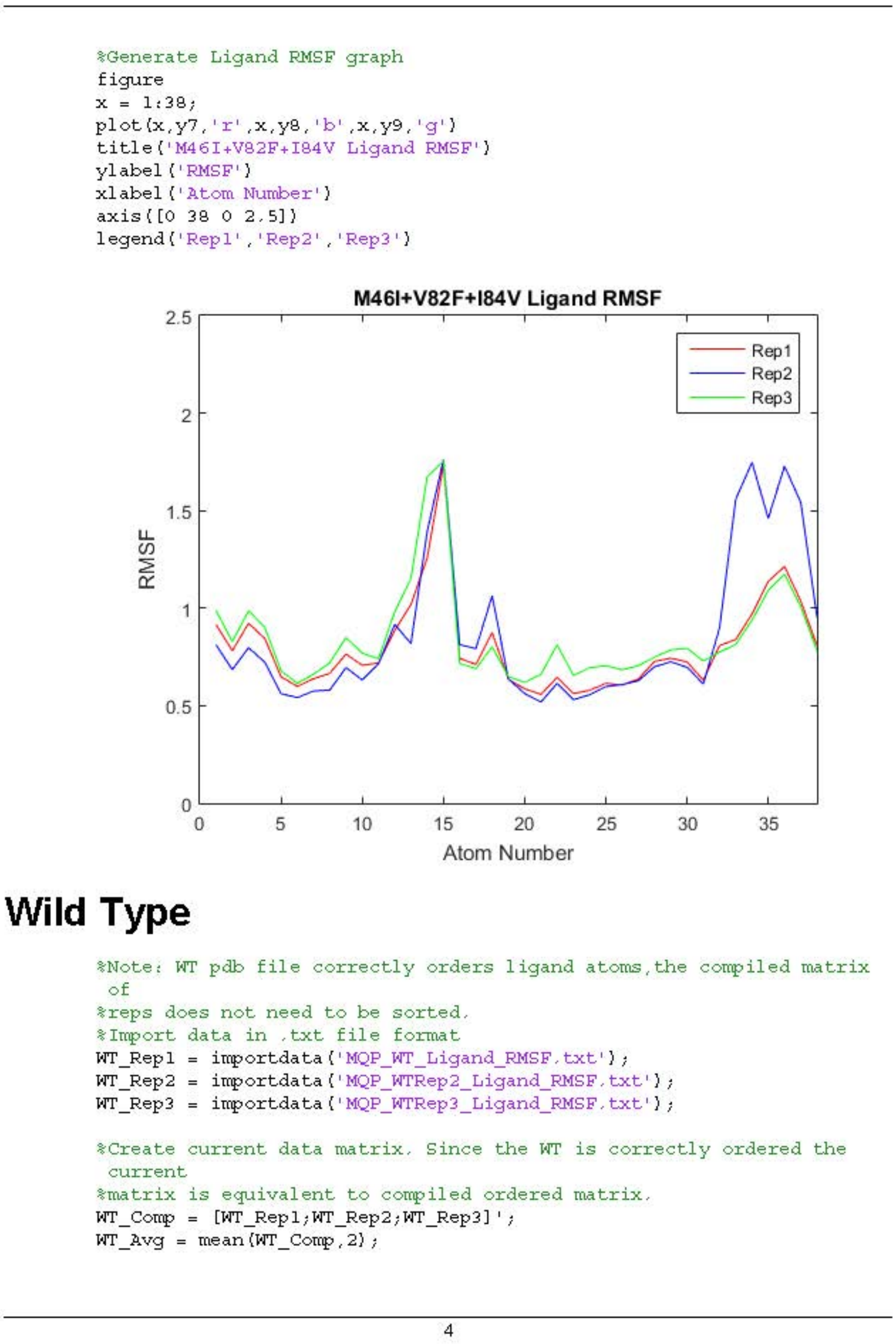


```
x = 1:38;
plot(x,y4,'r',x,y5,'b')
title('V82F+I84V Ligand RMSF')
ylabel('RMSF')
xlabel('Atom Number')
axis([0 38 0 2.5])
legend('Rep1','Rep3')
```

V82F+I84V Ligand RMSF
RMSF
Atom Number
0
0.5
1
1.5
2
2.5
0
5
10
15
20
25
30
35
Rep1
Rep3

M46I_V82F__I84V

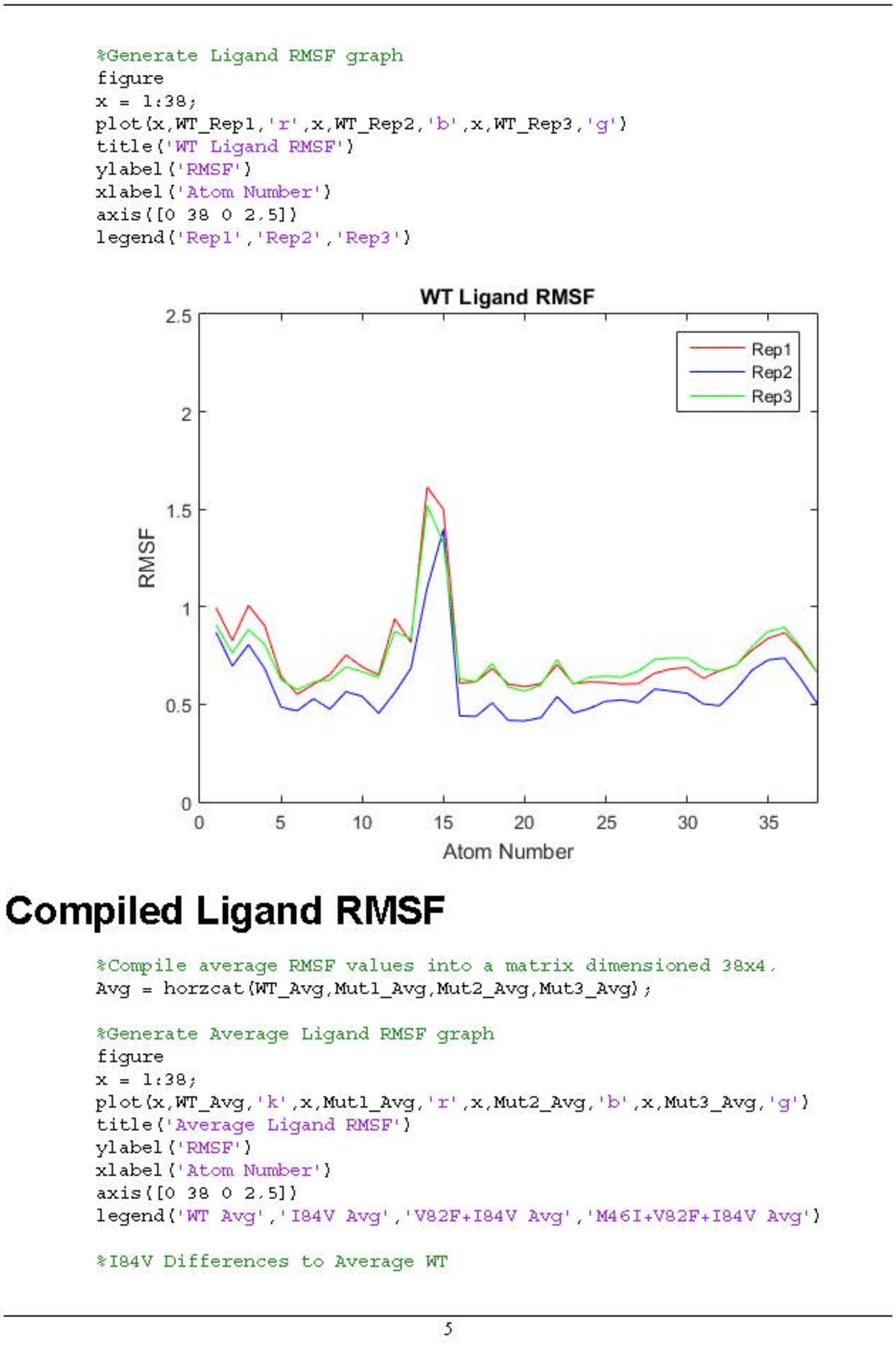
```
%Import data in .txt file format
Mut3_Rep1 = importdata('MQP_M46I+V82F+I84V_Ligand_RMSF.txt');
Mut3_Rep2 = importdata('MQP_M46I+V82F+I84VRep2_Ligand_RMSF.txt');
Mut3_Rep3 = importdata('MQP_M46I+V82F+I84VRep3_Ligand_RMSF.txt');

%Create current data matrix to be sorted into new matrix with proper
 atom
%ordering. Matrix sorted based on first column of pdbOrder.
currentC = [pdbOrder;Mut3_Rep1;Mut3_Rep2;Mut3_Rep3]';
sortedC = sortrows(currentC,1);
y7 = sortedC(:,2);
y8 = sortedC(:,3);
y9 = sortedC(:,4);
Mut3_Comp = horzcat(y7,y8,y9);
Mut3_Avg = mean(Mut3_Comp,2);
```



```
%Generate Ligand RMSF graph
figure
x = 1:38;
plot(x,y7,'r',x,y8,'b',x,y9,'g')
title('M46I+V82F+I84V Ligand RMSF')
ylabel('RMSF')
xlabel('Atom Number')
axis([0 38 0 2.5])
legend('Rep1','Rep2','Rep3')
```

Wild Type

```
%Note: WT pdb file correctly orders ligand atoms,the compiled matrix
 of
%reps does not need to be sorted.
%Import data in .txt file format
WT_Rep1 = importdata('MQP_WT_Ligand_RMSF.txt');
WT_Rep2 = importdata('MQP_WTRep2_Ligand_RMSF.txt');
WT_Rep3 = importdata('MQP_WTRep3_Ligand_RMSF.txt');

%Create current data matrix. Since the WT is correctly ordered the
 current
%matrix is equivalent to compiled ordered matrix.
WT_Comp = [WT_Rep1;WT_Rep2;WT_Rep3]';
WT_Avg = mean(WT_Comp,2);
```

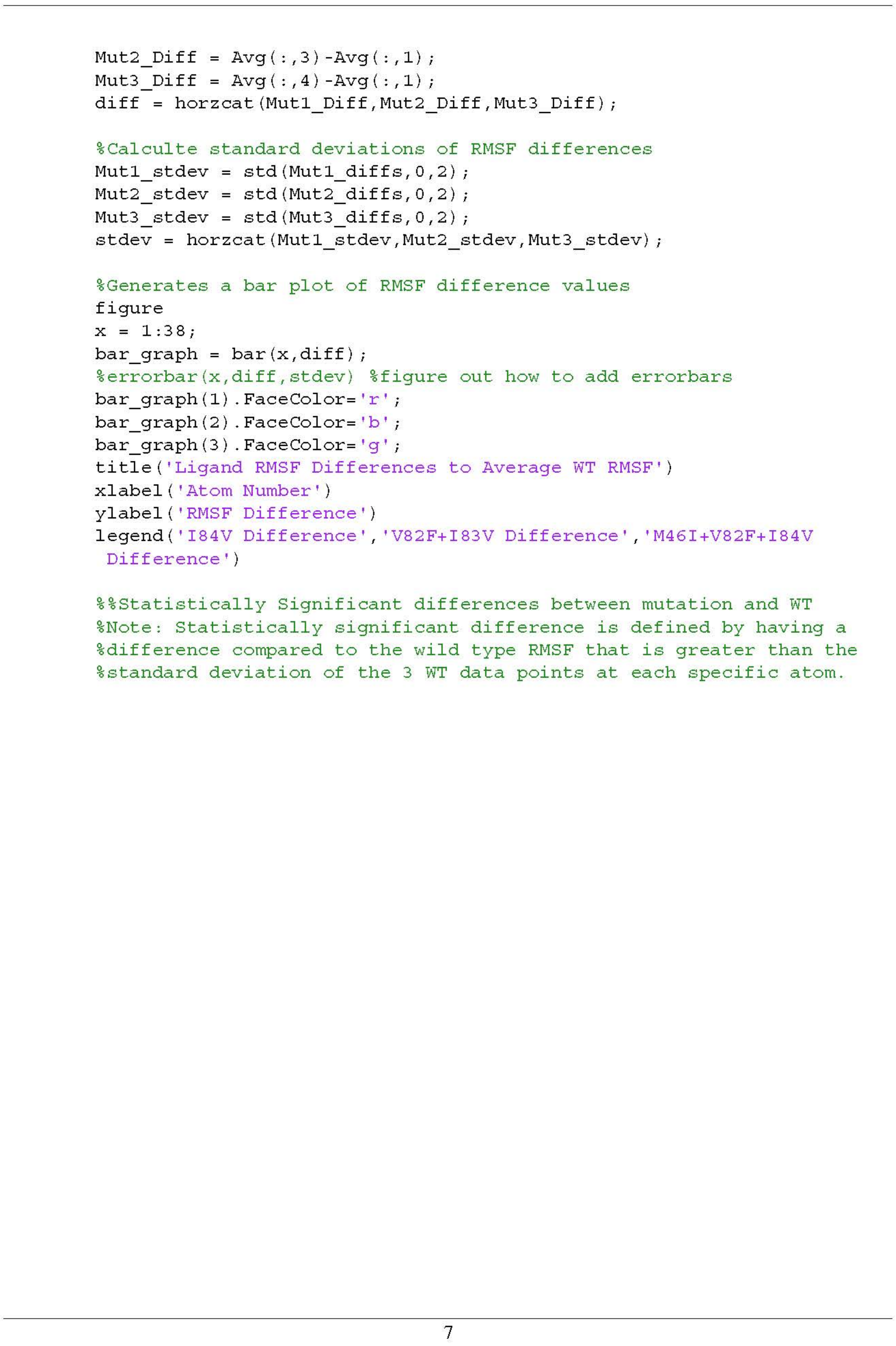


```
%Generate Ligand RMSF graph
figure
x = 1:38;
plot(x,WT_Rep1,'r',x,WT_Rep2,'b',x,WT_Rep3,'g')
title('WT Ligand RMSF')
ylabel('RMSF')
xlabel('Atom Number')
axis([0 38 0 2.5])
legend('Rep1','Rep2','Rep3')
```

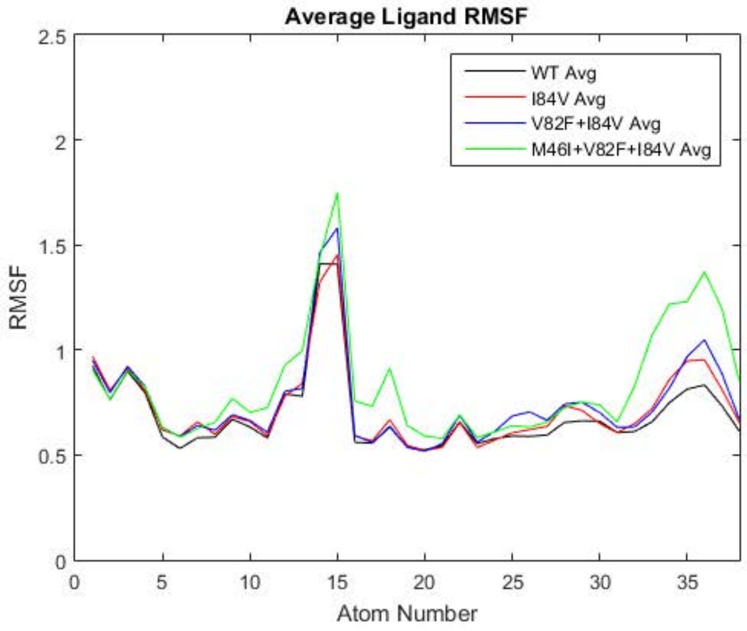
Compiled Ligand RMSF

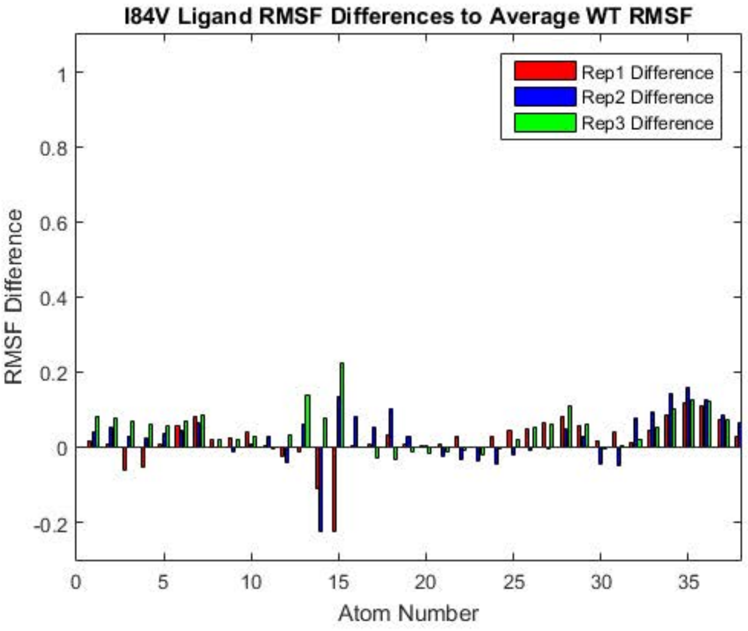
```
%Compile average RMSF values into a matrix dimensioned 38x4.
Avg = horzcat(WT_Avg,Mut1_Avg,Mut2_Avg,Mut3_Avg);

%Generate Average Ligand RMSF graph
figure
x = 1:38;
plot(x,WT_Avg,'k',x,Mut1_Avg,'r',x,Mut2_Avg,'b',x,Mut3_Avg,'g')
title('Average Ligand RMSF')
ylabel('RMSF')
xlabel('Atom Number')
axis([0 38 0 2.5])
legend('WT Avg','I84V Avg','V82F+I84V Avg','M46I+V82F+I84V Avg')

%I84V Differences to Average WT
```

```
Mut1_Rep1_Diff = y1-WT_Avg;
Mut1_Rep2_Diff = y2-WT_Avg;
Mut1_Rep3_Diff = y3-WT_Avg;
Mut1_diffs = horzcat(Mut1_Rep1_Diff, Mut1_Rep2_Diff, Mut1_Rep3_Diff);

figure
x = 1:38;
Mut1_bar_graph = bar(x,Mut1_diffs);
Mut1_bar_graph(1).FaceColor='r';
Mut1_bar_graph(2).FaceColor='b';
Mut1_bar_graph(3).FaceColor='g';
title('I84V Ligand RMSF Differences to Average WT RMSF')
xlabel('Atom Number')
ylabel('RMSF Difference')
axis([0 38 -0.3 1.1])
legend('Rep1 Difference','Rep2 Difference','Rep3 Difference')

%V82F+I84V Differences to Average WT
Mut2_Rep1_Diff = y4-WT_Avg;
Mut2_Rep2_Diff = y3-WT_Avg;
Mut2_Rep3_Diff = y5-WT_Avg;
Mut2_diffs = horzcat(Mut2_Rep1_Diff,Mut2_Rep3_Diff);

figure
x = 1:38;
Mut2_bar_graph = bar(x,Mut2_diffs);
Mut2_bar_graph(1).FaceColor='r';
Mut2_bar_graph(2).FaceColor='b';
title('V82F+I84V Ligand RMSF Differences to Average WT RMSF')
xlabel('Atom Number')
ylabel('RMSF Difference')
axis([0 38 -0.3 1.1])
legend('Rep1 Difference','Rep3 Difference')

%M46I+V82F+I84V Differences to Average WT
Mut3_Rep1_Diff = y7-WT_Avg;
Mut3_Rep2_Diff = y8-WT_Avg;
Mut3_Rep3_Diff = y9-WT_Avg;
Mut3_diffs = horzcat(Mut3_Rep1_Diff,Mut3_Rep2_Diff,Mut3_Rep3_Diff);

figure
x = 1:38;
Mut3_bar_graph = bar(x,Mut3_diffs);
Mut3_bar_graph(1).FaceColor='r';
Mut3_bar_graph(2).FaceColor='b';
Mut3_bar_graph(3).FaceColor='g';
title('M46I+V82F+I84V Ligand RMSF Differences to Average WT RMSF')
xlabel('Atom Number')
ylabel('RMSF Difference')
axis([0 38 -0.3 1.1])
legend('Rep1 Difference','Rep2 Difference','Rep3 Difference')

%Calculate average differences between mutation and WT RMSF values
Mut1_Diff = Avg(:,2)-Avg(:,1);
```

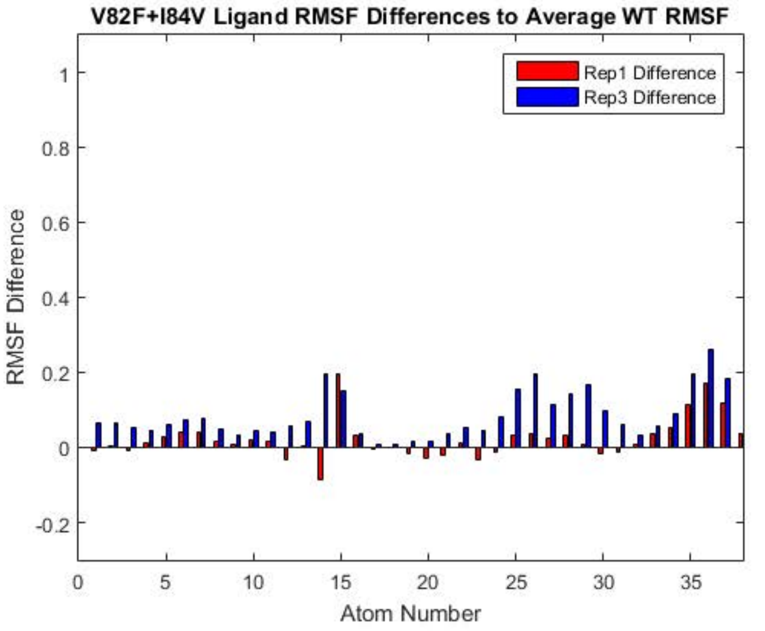


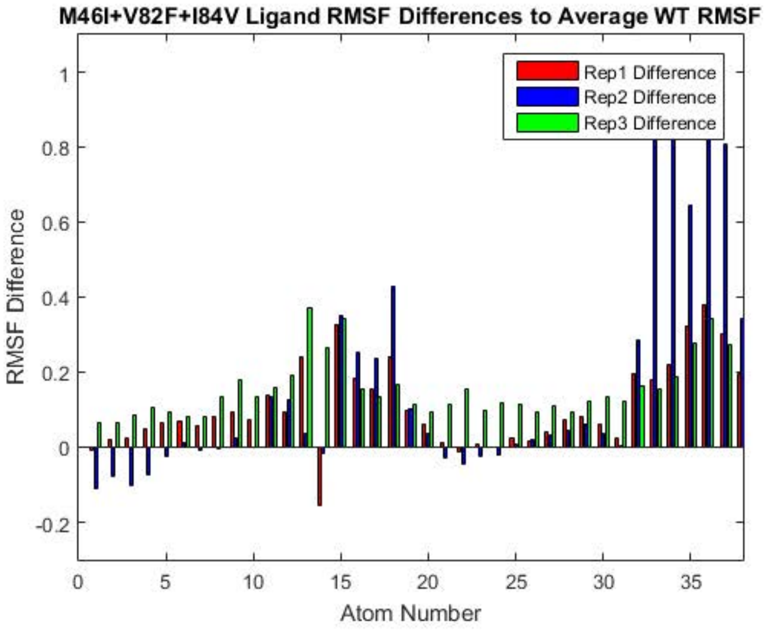
```
Mut2_Diff = Avg(:,3)-Avg(:,1);
Mut3_Diff = Avg(:,4)-Avg(:,1);
diff = horzcat(Mut1_Diff,Mut2_Diff,Mut3_Diff);

%Calculte standard deviations of RMSF differences
Mut1_stdev = std(Mut1_diffs,0,2);
Mut2_stdev = std(Mut2_diffs,0,2);
Mut3_stdev = std(Mut3_diffs,0,2);
stdev = horzcat(Mut1_stdev,Mut2_stdev,Mut3_stdev);

%Generates a bar plot of RMSF difference values
figure
x = 1:38;
bar_graph = bar(x,diff);
%errorbar(x,diff,stdev) %figure out how to add errorbars
bar_graph(1).FaceColor='r';
bar_graph(2).FaceColor='b';
bar_graph(3).FaceColor='g';
title('Ligand RMSF Differences to Average WT RMSF')
xlabel('Atom Number')
ylabel('RMSF Difference')
legend('I84V Difference','V82F+I83V Difference','M46I+V82F+I84V
 Difference')

%%Statistically Significant differences between mutation and WT
%Note: Statistically significant difference is defined by having a
%difference compared to the wild type RMSF that is greater than the
%standard deviation of the 3 WT data points at each specific atom.
```

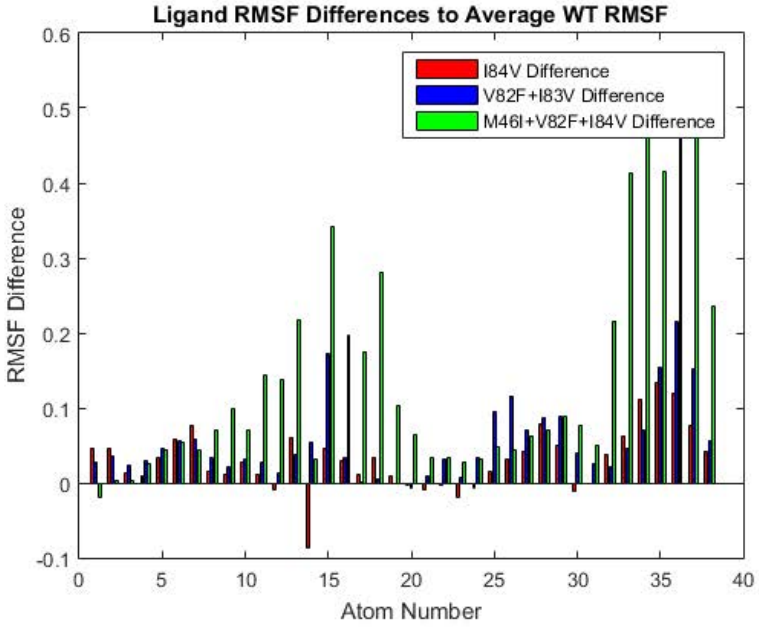

Average Ligand RMSF
RMSF
Atom Number
0
0.5
1
1.5
2
2.5
0
5
10
15
20
25
30
35
WT Avg
I84V Avg
V82F+I84V Avg
M46I+V82F+I84V Avg


I84V Ligand RMSF Differences to Average WT RMSF
RMSF Difference
Atom Number
-0.2
0
0.2
0.4
0.6
0.8
1
0
5
10
15
20
25
30
35
Rep1 Difference
Rep2 Difference
Rep3 Difference

8


V82F+I84V Ligand RMSF Differences to Average WT RMSF
🟥 Rep1 Difference
🟦 Rep3 Difference
RMSF Difference
1
0.8
0.6
0.4
0.2
0
-0.2
0
5
10
15
20
25
30
35
Atom Number


M46I+V82F+I84V Ligand RMSF Differences to Average WT RMSF
🟥 Rep1 Difference
🟦 Rep2 Difference
🟩 Rep3 Difference
RMSF Difference
1
0.8
0.6
0.4
0.2
0
-0.2
0
5
10
15
20
25
30
35
Atom Number

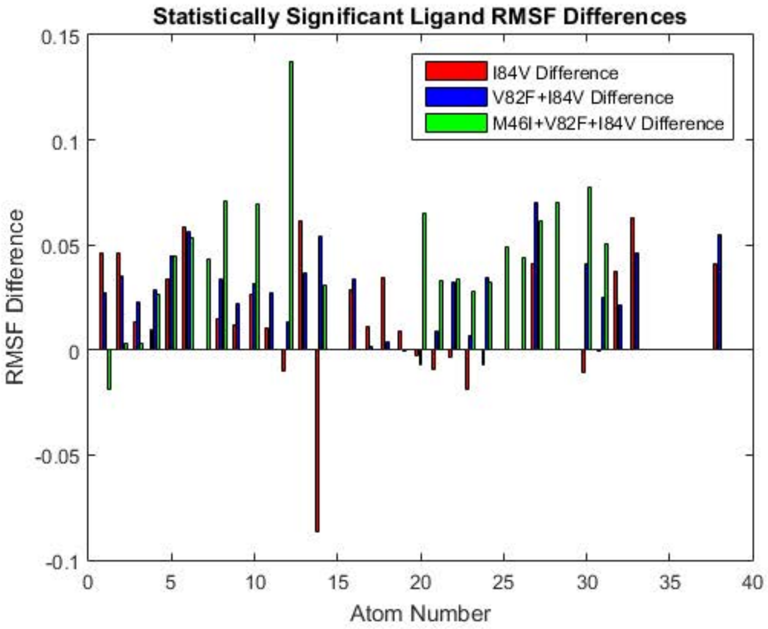


Preallocate vectors

```
Mut1_statsig = ones(1,38);
Mut2_statsig = ones(1,38);
Mut3_statsig = ones(1,38);
WT_stdev = std(WT_Comp')';
for i = 1:38
 if (-WT_stdev(i)) < Mut1_Diff(i) && Mut2_Diff(i) >  WT_stdev(i)
  Mut1_statsig(i) = 0;
 else
  Mut1_statsig(i) = Mut1_Diff(i);
    end
    if (-WT_stdev(i)) < Mut2_Diff(i) && Mut2_Diff(i) >  WT_stdev(i)
  Mut2_statsig(i) = 0;
 else
  Mut2_statsig(i) = Mut2_Diff(i);
    end
    if (-WT_stdev(i)) < Mut3_Diff(i) && Mut3_Diff(i) >  WT_stdev(i)
  Mut3_statsig(i) = 0;
 else
  Mut3_statsig(i) = Mut3_Diff(i);
    end
end
statsig = [Mut1_statsig;Mut2_statsig;Mut3_statsig]';
```

Plot statistically significant differences

```
figure
x = 1:38;
statsig_bar_graph = bar(x,statsig);
statsig_bar_graph(1).FaceColor='r';
statsig_bar_graph(2).FaceColor='b';
statsig_bar_graph(3).FaceColor='g';
title('Statistically Significant Ligand RMSF Differences')
xlabel('Atom Number')
ylabel('RMSF Difference')
legend('I84V Difference','V82F+I84V Difference','M46I+V82F+I84V
 Difference')
```



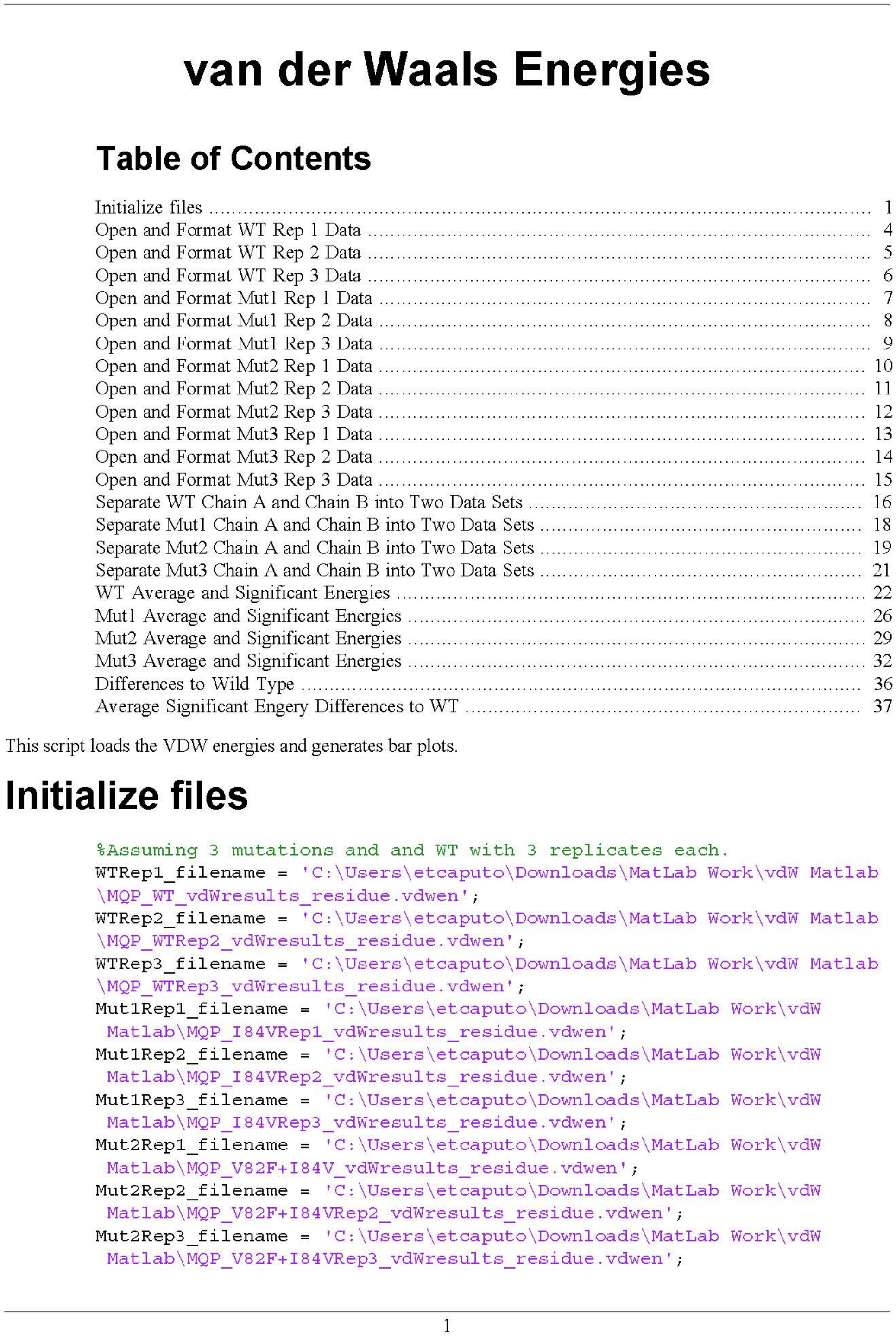
Print Statistically Significant

```
Mut1_sigdiff = [];
Mut2_sigdiff = [];
Mut3_sigdiff = [];
for i = 1:38;
    if Mut1_statsig(i) ~= 0
        Mut1_sigdiff = [Mut1_sigdiff; i,Mut1_statsig(i)];
    if Mut2_statsig(i) ~= 0
        Mut2_sigdiff = [Mut2_sigdiff; i,Mut2_statsig(i)];
    if Mut3_statsig(i) ~= 0
```

```
            Mut3_sigdiff = [Mut3_sigdiff; i,Mut3_statsig(i)];
        end
        end
        end
    end
```

Published with MATLAB® R2015a

B.4 Van der Waals Interactions Script



van der Waals Energies

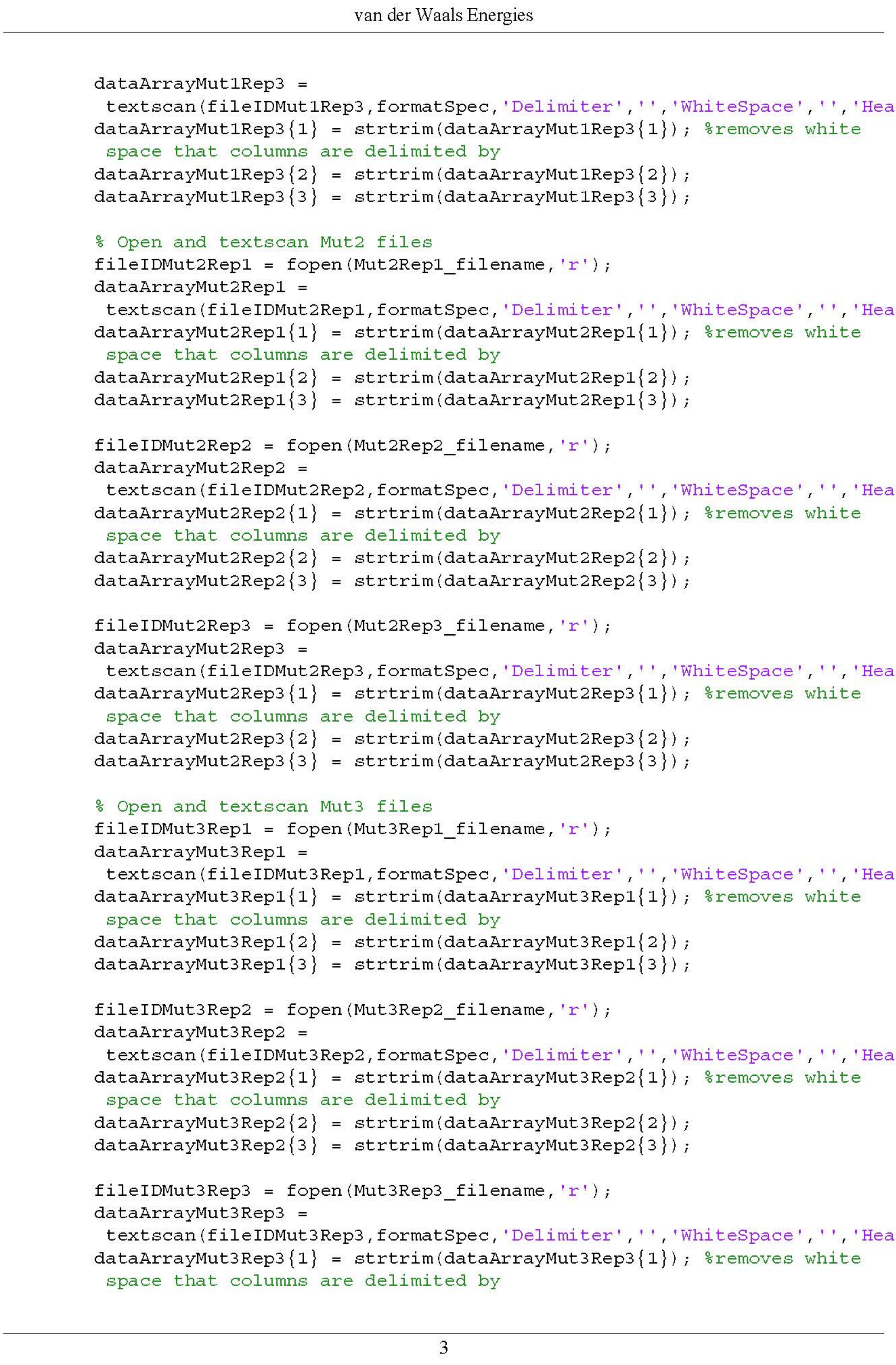
Table of Contents

This script loads the VDW energies and generates bar plots.

Initialize files

```
%Assuming 3 mutations and and WT with 3 replicates each.
WTRep1_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW Matlab
\MQP_WT_vdWresults_residue.vdwen';
WTRep2_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW Matlab
\MQP_WTRep2_vdWresults_residue.vdwen';
WTRep3_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW Matlab
\MQP_WTRep3_vdWresults_residue.vdwen';
Mut1Rep1_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW
 Matlab\MQP_I84VRep1_vdWresults_residue.vdwen';
Mut1Rep2_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW
 Matlab\MQP_I84VRep2_vdWresults_residue.vdwen';
Mut1Rep3_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW
 Matlab\MQP_I84VRep3_vdWresults_residue.vdwen';
Mut2Rep1_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW
 Matlab\MQP_V82F+I84V_vdWresults_residue.vdwen';
Mut2Rep2_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW
 Matlab\MQP_V82F+I84VRep2_vdWresults_residue.vdwen';
Mut2Rep3_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW
 Matlab\MQP_V82F+I84VRep3_vdWresults_residue.vdwen';
```

```
Mut3Rep1_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW
 Matlab\MQP_M46I+V82F+I84V_vdWresults_residue.vdwen';
Mut3Rep2_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW
 Matlab\MQP_TripleMutRep2_vdWresults_residue.vdwen';
Mut3Rep3_filename = 'C:\Users\etcaputo\Downloads\MatLab Work\vdW
 Matlab\M46I+V82F+I84VRep3_vdWresults_residue.vdwen';
startRow = 3;
formatSpec = '%3s%3s%13s%4s%s%[^\n\r]';

% Open and textscan WT files
fileIDWT1 = fopen(WTRep1_filename,'r');
dataArrayWT1 =
 textscan(fileIDWT1,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',startR
dataArrayWT1{1} = strtrim(dataArrayWT1{1}); %removes white space that
 columns are delimited by
dataArrayWT1{2} = strtrim(dataArrayWT1{2});
dataArrayWT1{3} = strtrim(dataArrayWT1{3});
fclose(fileIDWT1);

fileIDWT2 = fopen(WTRep2_filename,'r');
dataArrayWT2 =
 textscan(fileIDWT2,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',startR
dataArrayWT2{1} = strtrim(dataArrayWT2{1}); %removes white space that
 columns are delimited by
dataArrayWT2{2} = strtrim(dataArrayWT2{2});
dataArrayWT2{3} = strtrim(dataArrayWT2{3});

fileIDWT3 = fopen(WTRep3_filename,'r');
dataArrayWT3 =
 textscan(fileIDWT3,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',startR
dataArrayWT3{1} = strtrim(dataArrayWT3{1}); %removes white space that
 columns are delimited by
dataArrayWT3{2} = strtrim(dataArrayWT3{2});
dataArrayWT3{3} = strtrim(dataArrayWT3{3});

% Open and textscan Mut1 files
fileIDMut1Rep1 = fopen(Mut1Rep1_filename,'r');
dataArrayMut1Rep1 =
 textscan(fileIDMut1Rep1,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',s
dataArrayMut1Rep1{1} = strtrim(dataArrayMut1Rep1{1}); %removes white
 space that columns are delimited by
dataArrayMut1Rep1{2} = strtrim(dataArrayMut1Rep1{2});
dataArrayMut1Rep1{3} = strtrim(dataArrayMut1Rep1{3});

fileIDMut1Rep2 = fopen(Mut1Rep2_filename,'r');
dataArrayMut1Rep2 =
 textscan(fileIDMut1Rep2,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',s
dataArrayMut1Rep2{1} = strtrim(dataArrayMut1Rep2{1}); %removes white
 space that columns are delimited by
dataArrayMut1Rep2{2} = strtrim(dataArrayMut1Rep2{2});
dataArrayMut1Rep2{3} = strtrim(dataArrayMut1Rep2{3});

fileIDMut1Rep3 = fopen(Mut1Rep3_filename,'r');
```

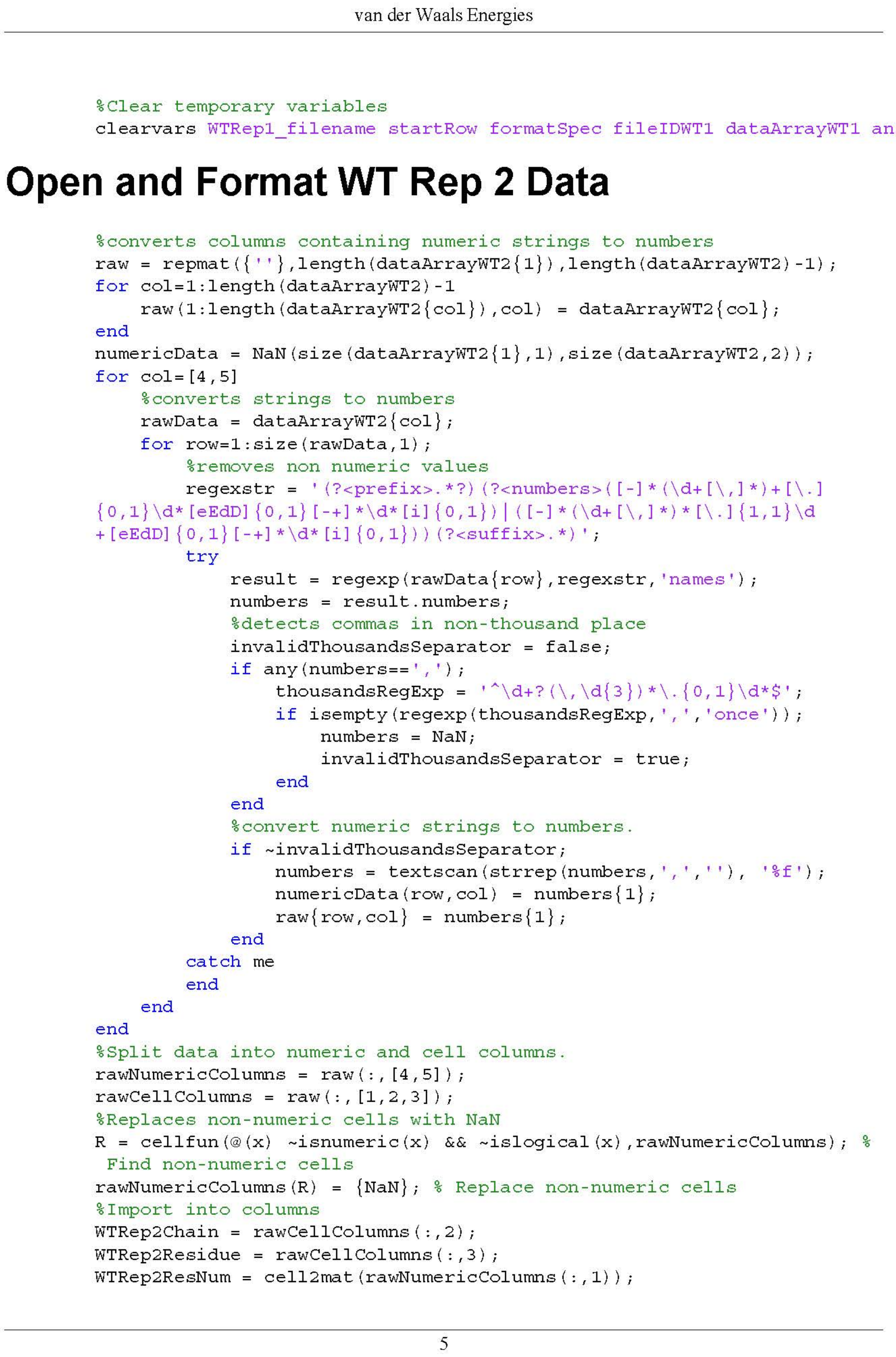


```
dataArrayMut1Rep3 =
 textscan(fileIDMut1Rep3,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',s
dataArrayMut1Rep3{1} = strtrim(dataArrayMut1Rep3{1}); %removes white
 space that columns are delimited by
dataArrayMut1Rep3{2} = strtrim(dataArrayMut1Rep3{2});
dataArrayMut1Rep3{3} = strtrim(dataArrayMut1Rep3{3});

% Open and textscan Mut2 files
fileIDMut2Rep1 = fopen(Mut2Rep1_filename,'r');
dataArrayMut2Rep1 =
 textscan(fileIDMut2Rep1,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',s
dataArrayMut2Rep1{1} = strtrim(dataArrayMut2Rep1{1}); %removes white
 space that columns are delimited by
dataArrayMut2Rep1{2} = strtrim(dataArrayMut2Rep1{2});
dataArrayMut2Rep1{3} = strtrim(dataArrayMut2Rep1{3});

fileIDMut2Rep2 = fopen(Mut2Rep2_filename,'r');
dataArrayMut2Rep2 =
 textscan(fileIDMut2Rep2,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',s
dataArrayMut2Rep2{1} = strtrim(dataArrayMut2Rep2{1}); %removes white
 space that columns are delimited by
dataArrayMut2Rep2{2} = strtrim(dataArrayMut2Rep2{2});
dataArrayMut2Rep2{3} = strtrim(dataArrayMut2Rep2{3});

fileIDMut2Rep3 = fopen(Mut2Rep3_filename,'r');
dataArrayMut2Rep3 =
 textscan(fileIDMut2Rep3,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',s
dataArrayMut2Rep3{1} = strtrim(dataArrayMut2Rep3{1}); %removes white
 space that columns are delimited by
dataArrayMut2Rep3{2} = strtrim(dataArrayMut2Rep3{2});
dataArrayMut2Rep3{3} = strtrim(dataArrayMut2Rep3{3});

% Open and textscan Mut3 files
fileIDMut3Rep1 = fopen(Mut3Rep1_filename,'r');
dataArrayMut3Rep1 =
 textscan(fileIDMut3Rep1,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',s
dataArrayMut3Rep1{1} = strtrim(dataArrayMut3Rep1{1}); %removes white
 space that columns are delimited by
dataArrayMut3Rep1{2} = strtrim(dataArrayMut3Rep1{2});
dataArrayMut3Rep1{3} = strtrim(dataArrayMut3Rep1{3});

fileIDMut3Rep2 = fopen(Mut3Rep2_filename,'r');
dataArrayMut3Rep2 =
 textscan(fileIDMut3Rep2,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',s
dataArrayMut3Rep2{1} = strtrim(dataArrayMut3Rep2{1}); %removes white
 space that columns are delimited by
dataArrayMut3Rep2{2} = strtrim(dataArrayMut3Rep2{2});
dataArrayMut3Rep2{3} = strtrim(dataArrayMut3Rep2{3});

fileIDMut3Rep3 = fopen(Mut3Rep3_filename,'r');
dataArrayMut3Rep3 =
 textscan(fileIDMut3Rep3,formatSpec,'Delimiter','','WhiteSpace','','HeaderLines',s
dataArrayMut3Rep3{1} = strtrim(dataArrayMut3Rep3{1}); %removes white
 space that columns are delimited by
```

```
dataArrayMut3Rep3{2} = strtrim(dataArrayMut3Rep3{2});
dataArrayMut3Rep3{3} = strtrim(dataArrayMut3Rep3{3});
```

Open and Format WT Rep 1 Data

```
%converts columns containing numeric strings to numbers
raw = repmat({''},length(dataArrayWT1{1}),length(dataArrayWT1)-1);
for col=1:length(dataArrayWT1)-1
    raw(1:length(dataArrayWT1{col}),col) = dataArrayWT1{col};
end
numericData = NaN(size(dataArrayWT1{1},1),size(dataArrayWT1,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayWT1{col};
    for row = 1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
WTRep1Chain = rawCellColumns(:,2);
WTRep1Residue = rawCellColumns(:,3);
WTRep1ResNum = cell2mat(rawNumericColumns(:,1));
WTRep1Energy = cell2mat(rawNumericColumns(:,2));
```



```
%Clear temporary variables
clearvars WTRep1_filename startRow formatSpec fileIDWT1 dataArrayWT1 ans raw col n
```

Open and Format WT Rep 2 Data

```
%converts columns containing numeric strings to numbers
raw = repmat({''},length(dataArrayWT2{1}),length(dataArrayWT2)-1);
for col=1:length(dataArrayWT2)-1
    raw(1:length(dataArrayWT2{col}),col) = dataArrayWT2{col};
end
numericData = NaN(size(dataArrayWT2{1},1),size(dataArrayWT2,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayWT2{col};
    for row=1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''), '%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
WTRep2Chain = rawCellColumns(:,2);
WTRep2Residue = rawCellColumns(:,3);
WTRep2ResNum = cell2mat(rawNumericColumns(:,1));
```

```
WTRep2Energy = cell2mat(rawNumericColumns(:,2));

%Clear temporary variables
clearvars WTRep2_filename startRow formatSpec fileIDWT2 dataArrayWT2 ans raw col n
```

Open and Format WT Rep 3 Data

```
%converts columns containing numeric strings to numbers
raw = repmat({''},length(dataArrayWT3{1}),length(dataArrayWT3)-1);
for col=1:length(dataArrayWT3)-1
    raw(1:length(dataArrayWT3{col}),col) = dataArrayWT3{col};
end
numericData = NaN(size(dataArrayWT3{1},1),size(dataArrayWT3,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayWT3{col};
    for row=1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
WTRep3Chain = rawCellColumns(:,2);
WTRep3Residue = rawCellColumns(:,3);
```

```
WTRep3ResNum = cell2mat(rawNumericColumns(:,1));
WTRep3Energy = cell2mat(rawNumericColumns(:,2));

%Clear temporary variables
clearvars WTRep3_filename startRow formatSpec fileIDWT3 dataArrayWT3 ans raw col n
```

Open and Format Mut1 Rep 1 Data

```
%converts columns containing numeric strings to numbers
raw =
 repmat({''},length(dataArrayMut1Rep1{1}),length(dataArrayMut1Rep1)-1);
for col=1:length(dataArrayMut1Rep1)-1
    raw(1:length(dataArrayMut1Rep1{col}),col) =
 dataArrayMut1Rep1{col};
end
numericData =
 NaN(size(dataArrayMut1Rep1{1},1),size(dataArrayMut1Rep1,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayMut1Rep1{col};
    for row=1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
```

```
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
Mut1Rep1Chain = rawCellColumns(:,2);
Mut1Rep1Residue = rawCellColumns(:,3);
Mut1Rep1ResNum = cell2mat(rawNumericColumns(:,1));
Mut1Rep1Energy = cell2mat(rawNumericColumns(:,2));

%Clear temporary variables
clearvars Mut1Rep1_filename startRow formatSpec fileIDMut1Rep1 dataArrayMut1Rep1 a
```

Open and Format Mut1 Rep 2 Data

```
%converts columns containing numeric strings to numbers
raw =
 repmat({''},length(dataArrayMut1Rep2{1}),length(dataArrayMut1Rep2)-1);
for col=1:length(dataArrayMut1Rep2)-1
    raw(1:length(dataArrayMut1Rep2{col}),col) =
 dataArrayMut1Rep2{col};
end
numericData =
 NaN(size(dataArrayMut1Rep2{1},1),size(dataArrayMut1Rep2,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayMut1Rep2{col};
    for row=1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
```

```
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
Mut1Rep2Chain = rawCellColumns(:,2);
Mut1Rep2Residue = rawCellColumns(:,3);
Mut1Rep2ResNum = cell2mat(rawNumericColumns(:,1));
Mut1Rep2Energy = cell2mat(rawNumericColumns(:,2));

%Clear temporary variables
clearvars Mut1Rep2_filename startRow formatSpec fileIDMut1Rep2 dataArrayMut1Rep2 a
```

Open and Format Mut1 Rep 3 Data

```
%converts columns containing numeric strings to numbers
raw =
 repmat({''},length(dataArrayMut1Rep3{1}),length(dataArrayMut1Rep3)-1);
for col=1:length(dataArrayMut1Rep3)-1
    raw(1:length(dataArrayMut1Rep3{col}),col) =
 dataArrayMut1Rep3{col};
end
numericData =
 NaN(size(dataArrayMut1Rep3{1},1),size(dataArrayMut1Rep3,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayMut1Rep3{col};
    for row=1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
Mut1Rep3Chain = rawCellColumns(:,2);
Mut1Rep3Residue = rawCellColumns(:,3);
Mut1Rep3ResNum = cell2mat(rawNumericColumns(:,1));
Mut1Rep3Energy = cell2mat(rawNumericColumns(:,2));

%Clear temporary variables
clearvars Mut1Rep3_filename startRow formatSpec fileIDMut1Rep3 dataArrayMut1Rep3 a
```

Open and Format Mut2 Rep 1 Data

```
%converts columns containing numeric strings to numbers
raw =
 repmat({''},length(dataArrayMut2Rep1{1}),length(dataArrayMut2Rep1)-1);
for col=1:length(dataArrayMut2Rep1)-1
    raw(1:length(dataArrayMut2Rep1{col}),col) =
 dataArrayMut2Rep1{col};
end
numericData =
 NaN(size(dataArrayMut2Rep1{1},1),size(dataArrayMut2Rep1,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayMut2Rep1{col};
    for row=1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
Mut2Rep1Chain = rawCellColumns(:,2);
Mut2Rep1Residue = rawCellColumns(:,3);
Mut2Rep1ResNum = cell2mat(rawNumericColumns(:,1));
Mut2Rep1Energy = cell2mat(rawNumericColumns(:,2));

%Clear temporary variables
clearvars Mut2Rep1_filename startRow formatSpec fileIDMut2Rep1 dataArrayMut2Rep1 a
```

Open and Format Mut2 Rep 2 Data

```
%converts columns containing numeric strings to numbers
raw =
 repmat({''},length(dataArrayMut2Rep2{1}),length(dataArrayMut2Rep2)-1);
for col=1:length(dataArrayMut2Rep2)-1
    raw(1:length(dataArrayMut2Rep2{col}),col) =
 dataArrayMut2Rep2{col};
end
numericData =
 NaN(size(dataArrayMut2Rep2{1},1),size(dataArrayMut2Rep2,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayMut2Rep2{col};
    for row=1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
Mut2Rep2Chain = rawCellColumns(:,2);
Mut2Rep2Residue = rawCellColumns(:,3);
Mut2Rep2ResNum = cell2mat(rawNumericColumns(:,1));
Mut2Rep2Energy = cell2mat(rawNumericColumns(:,2));

%Clear temporary variables
clearvars Mut2Rep2_filename startRow formatSpec fileIDMut2Rep2 dataArrayMut2Rep2 a
```

Open and Format Mut2 Rep 3 Data

```
%converts columns containing numeric strings to numbers
raw =
 repmat({''},length(dataArrayMut2Rep3{1}),length(dataArrayMut2Rep3)-1);
for col=1:length(dataArrayMut2Rep3)-1
    raw(1:length(dataArrayMut2Rep3{col}),col) =
 dataArrayMut2Rep3{col};
end
numericData =
 NaN(size(dataArrayMut2Rep3{1},1),size(dataArrayMut2Rep3,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayMut2Rep3{col};
    for row=1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
```

```
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
Mut2Rep3Chain = rawCellColumns(:,2);
Mut2Rep3Residue = rawCellColumns(:,3);
Mut2Rep3ResNum = cell2mat(rawNumericColumns(:,1));
Mut2Rep3Energy = cell2mat(rawNumericColumns(:,2));

%Clear temporary variables
clearvars Mut2Rep3_filename startRow formatSpec fileIDMut2Rep3 dataArrayMut2Rep3 a
```

Open and Format Mut3 Rep 1 Data

```
%converts columns containing numeric strings to numbers
raw =
 repmat({''},length(dataArrayMut3Rep1{1}),length(dataArrayMut3Rep1)-1);
for col=1:length(dataArrayMut3Rep1)-1
    raw(1:length(dataArrayMut3Rep1{col}),col) =
 dataArrayMut3Rep1{col};
end
numericData =
 NaN(size(dataArrayMut3Rep1{1},1),size(dataArrayMut3Rep1,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayMut3Rep1{col};
    for row=1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
Mut3Rep1Chain = rawCellColumns(:,2);
Mut3Rep1Residue = rawCellColumns(:,3);
Mut3Rep1ResNum = cell2mat(rawNumericColumns(:,1));
Mut3Rep1Energy = cell2mat(rawNumericColumns(:,2));

%Clear temporary variables
clearvars Mut3Rep1_filename startRow formatSpec fileIDMut3Rep1 dataArrayMut3Rep1 a
```

Open and Format Mut3 Rep 2 Data

```
%converts columns containing numeric strings to numbers
raw =
 repmat({''},length(dataArrayMut3Rep2{1}),length(dataArrayMut3Rep2)-1);
for col=1:length(dataArrayMut3Rep2)-1
    raw(1:length(dataArrayMut3Rep2{col}),col) =
 dataArrayMut3Rep2{col};
end
numericData =
 NaN(size(dataArrayMut3Rep2{1},1),size(dataArrayMut3Rep2,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayMut3Rep2{col};
    for row=1:size(rawData,1);
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
```

```
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
Mut3Rep2Chain = rawCellColumns(:,2);
Mut3Rep2Residue = rawCellColumns(:,3);
Mut3Rep2ResNum = cell2mat(rawNumericColumns(:,1));
Mut3Rep2Energy = cell2mat(rawNumericColumns(:,2));

%Clear temporary variables
clearvars Mut3Rep2_filename startRow formatSpec fileIDMut3Rep2 dataArrayMut3Rep2 a
```

Open and Format Mut3 Rep 3 Data

```
%converts columns containing numeric strings to numbers
raw =
 repmat({''},length(dataArrayMut3Rep3{1}),length(dataArrayMut3Rep3)-1);
for col=1:length(dataArrayMut3Rep3)-1
    raw(1:length(dataArrayMut3Rep3{col}),col) =
 dataArrayMut3Rep3{col};
end
numericData =
 NaN(size(dataArrayMut3Rep3{1},1),size(dataArrayMut3Rep3,2));
for col=[4,5]
    %converts strings to numbers
    rawData = dataArrayMut3Rep3{col};
    for row=1:size(rawData,1);
```

```
        %removes non numeric values
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row},regexstr,'names');
            numbers = result.numbers;
            %detects commas in non-thousand place
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp,',','once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            %convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers,',',''),'%f');
                numericData(row,col) = numbers{1};
                raw{row,col} = numbers{1};
            end
        catch me
        end
    end
end
%Split data into numeric and cell columns.
rawNumericColumns = raw(:,[4,5]);
rawCellColumns = raw(:,[1,2,3]);
%Replaces non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); %
 Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells
%Import into columns
Mut3Rep3Chain = rawCellColumns(:,2);
Mut3Rep3Residue = rawCellColumns(:,3);
Mut3Rep3ResNum = cell2mat(rawNumericColumns(:,1));
Mut3Rep3Energy = cell2mat(rawNumericColumns(:,2));

%Clear temporary variables
clearvars Mut3Rep3_filename startRow formatSpec fileIDMut3Rep3 dataArrayMut3Rep3 a
```

Separate WT Chain A and Chain B into Two Data Sets

```
res = 1:99;
WTChainA = horzcat(res',zeros(99,3));
WTChainB = horzcat(res',zeros(99,3));
A = 'A';
B = 'B';
%Append energies and average in plance to the second columns of ChainA
 and
```

```
%ChainB arrays
for i=1:length(WTRep1Chain)
    if strcmp(WTRep1Chain(i),A) == 1
        num = WTRep1ResNum(i);
        if num ~= 1200
            if WTChainA(num,2) == 0
                WTChainA(num,2) = WTRep1Energy(i);
            else WTChainA(num,2) =
 (WTChainA(num,2)+WTRep1Energy(i))/2;
            end
        end
    elseif strcmp(WTRep1Chain(i),B) == 1
        num = WTRep1ResNum(i);
        if num ~= 1200
            if WTChainB(num,2) == 0
                WTChainB(num,2) = WTRep1Energy(i);
            else WTChainB(num,2) =
 (WTChainB(num,2)+WTRep1Energy(i))/2;
            end
        end
    end
end
%Append energies and average in plance to the third columns of ChainA
 and
%ChainB arrays
for i=1:length(WTRep2Chain)
    if strcmp(WTRep2Chain(i),A) == 1
        num = WTRep2ResNum(i);
        if num ~= 1200
            if WTChainA(num,3) == 0
                WTChainA(num,3) = WTRep2Energy(i);
            else WTChainA(num,3) =
 (WTChainA(num,3)+WTRep2Energy(i))/2;
            end
        end
    elseif strcmp(WTRep2Chain(i),B) == 1
        num = WTRep2ResNum(i);
        if num ~= 1200
            if WTChainB(num,3) == 0
                WTChainB(num,3) = WTRep2Energy(i);
            else WTChainB(num,3) =
 (WTChainB(num,3)+WTRep2Energy(i))/2;
            end
        end
    end
end
%Append energies and average in plance to the fourth columns of ChainA
 and
%ChainB arrays
for i=1:length(WTRep3Chain)
    if strcmp(WTRep3Chain(i),A) == 1
        num = WTRep3ResNum(i);
        if num ~= 1200
            if WTChainA(num,4) == 0
```

```
                WTChainA(num,4) = WTRep3Energy(i);
            else WTChainA(num,4) =
 (WTChainA(num,4)+WTRep3Energy(i))/2;
            end
        end
    elseif strcmp(WTRep3Chain(i),B) == 1
        num = WTRep3ResNum(i);
        if num ~= 1200
            if WTChainB(num,4) == 0
                WTChainB(num,4) = WTRep3Energy(i);
            else WTChainB(num,4) =
 (WTChainB(num,4)+WTRep3Energy(i))/2;
            end
        end
    end
end
```

Separate Mut1 Chain A and Chain B into Two Data Sets

```
res = 1:99;
Mut1ChainA = horzcat(res',zeros(99,3));
Mut1ChainB = horzcat(res',zeros(99,3));
%Append energies and average in plance to the second columns of ChainA
 and
%ChainB arrays
for i=1:length(Mut1Rep1Chain)
    if strcmp(Mut1Rep1Chain(i),A) == 1
        num = Mut1Rep1ResNum(i);
        if num ~= 1200
            if Mut1ChainA(num,2) == 0
                Mut1ChainA(num,2) = Mut1Rep1Energy(i);
            else Mut1ChainA(num,2) =
 (Mut1ChainA(num,2)+Mut1Rep1Energy(i))/2;
            end
        end
    elseif strcmp(Mut1Rep1Chain(i),B) == 1
        num = Mut1Rep1ResNum(i);
        if num ~= 1200
            if Mut1ChainB(num,2) == 0
                Mut1ChainB(num,2) = Mut1Rep1Energy(i);
            else Mut1ChainB(num,2) =
 (Mut1ChainB(num,2)+Mut1Rep1Energy(i))/2;
            end
        end
    end
end
%Append energies and average in plance to the third columns of ChainA
 and
%ChainB arrays
for i=1:length(Mut1Rep2Chain)
    if strcmp(Mut1Rep2Chain(i),A) == 1
```

```
        num = Mut1Rep2ResNum(i);
        if num ~= 1200
            if Mut1ChainA(num,3) == 0
                Mut1ChainA(num,3) = Mut1Rep2Energy(i);
            else Mut1ChainA(num,3) =
 (Mut1ChainA(num,3)+Mut1Rep2Energy(i))/2;
            end
        end
    elseif strcmp(Mut1Rep2Chain(i),B) == 1
        num = Mut1Rep2ResNum(i);
        if num ~= 1200
            if Mut1ChainB(num,3) == 0
                Mut1ChainB(num,3) = Mut1Rep2Energy(i);
            else Mut1ChainB(num,3) =
 (Mut1ChainB(num,3)+Mut1Rep2Energy(i))/2;
            end
        end
    end
end
%Append energies and average in plance to the fourth columns of ChainA
 and
%ChainB arrays
for i=1:length(Mut1Rep3Chain)
    if strcmp(Mut1Rep3Chain(i),A) == 1
        num = Mut1Rep3ResNum(i);
        if num ~= 1200
            if Mut1ChainA(num,4) == 0
                Mut1ChainA(num,4) = Mut1Rep3Energy(i);
            else Mut1ChainA(num,4) =
 (Mut1ChainA(num,4)+Mut1Rep3Energy(i))/2;
            end
        end
    elseif strcmp(Mut1Rep3Chain(i),B) == 1
        num = Mut1Rep3ResNum(i);
        if num ~= 1200
            if Mut1ChainB(num,4) == 0
                Mut1ChainB(num,4) = Mut1Rep3Energy(i);
            else Mut1ChainB(num,4) =
 (Mut1ChainB(num,4)+Mut1Rep3Energy(i))/2;
            end
        end
    end
end
```

Separate Mut2 Chain A and Chain B into Two Data Sets

```
res = 1:99;
Mut2ChainA = horzcat(res',zeros(99,3));
Mut2ChainB = horzcat(res',zeros(99,3));
%Append energies and average in plance to the second columns of ChainA
 and
```

```
%ChainB arrays
for i=1:length(Mut2Rep1Chain)
    if strcmp(Mut2Rep1Chain(i),A) == 1
        num = Mut2Rep1ResNum(i);
        if num ~= 1200
            if Mut2ChainA(num,2) == 0
                Mut2ChainA(num,2) = Mut2Rep1Energy(i);
            else Mut2ChainA(num,2) =
 (Mut2ChainA(num,2)+Mut2Rep1Energy(i))/2;
            end
        end
    elseif strcmp(Mut2Rep1Chain(i),B) == 1
        num = Mut2Rep1ResNum(i);
        if num ~= 1200
            if Mut2ChainB(num,2) == 0
                Mut2ChainB(num,2) = Mut2Rep1Energy(i);
            else Mut2ChainB(num,2) =
 (Mut2ChainB(num,2)+Mut2Rep1Energy(i))/2;
            end
        end
    end
end
%Append energies and average in plance to the third columns of ChainA
 and
%ChainB arrays
for i=1:length(Mut2Rep2Chain)
    if strcmp(Mut2Rep2Chain(i),A) == 1
        num = Mut2Rep2ResNum(i);
        if num ~= 1200
            if Mut2ChainA(num,3) == 0
                Mut2ChainA(num,3) = Mut2Rep2Energy(i);
            else Mut2ChainA(num,3) =
 (Mut2ChainA(num,3)+Mut2Rep2Energy(i))/2;
            end
        end
    elseif strcmp(Mut2Rep2Chain(i),B) == 1
        num = Mut2Rep2ResNum(i);
        if num ~= 1200
            if Mut2ChainB(num,3) == 0
                Mut2ChainB(num,3) = Mut2Rep2Energy(i);
            else Mut2ChainB(num,3) =
 (Mut2ChainB(num,3)+Mut2Rep2Energy(i))/2;
            end
        end
    end
end
%Append energies and average in plance to the fourth columns of ChainA
 and
%ChainB arrays
for i=1:length(Mut2Rep3Chain)
    if strcmp(Mut2Rep3Chain(i),A) == 1
        num = Mut2Rep3ResNum(i);
        if num ~= 1200
            if Mut2ChainA(num,4) == 0
```

```
                Mut2ChainA(num,4) = Mut2Rep3Energy(i);
            else Mut2ChainA(num,4) =
 (Mut2ChainA(num,4)+Mut2Rep3Energy(i))/2;
            end
        end
    elseif strcmp(Mut2Rep3Chain(i),B) == 1
        num = Mut2Rep3ResNum(i);
        if num ~= 1200
            if Mut2ChainB(num,4) == 0
                Mut2ChainB(num,4) = Mut2Rep3Energy(i);
            else Mut2ChainB(num,4) =
 (Mut2ChainB(num,4)+Mut2Rep3Energy(i))/2;
            end
        end
    end
end
```

Separate Mut3 Chain A and Chain B into Two Data Sets

```
res = 1:99;
Mut3ChainA = horzcat(res',zeros(99,3));
Mut3ChainB = horzcat(res',zeros(99,3));
%Append energies and average in plance to the second columns of ChainA
 and
%ChainB arrays
for i=1:length(Mut3Rep1Chain)
    if strcmp(Mut3Rep1Chain(i),A) == 1
        num = Mut3Rep1ResNum(i);
        if num ~= 1200
            if Mut3ChainA(num,2) == 0
                Mut3ChainA(num,2) = Mut3Rep1Energy(i);
            else Mut3ChainA(num,2) =
 (Mut3ChainA(num,2)+Mut3Rep1Energy(i))/2;
            end
        end
    elseif strcmp(Mut3Rep1Chain(i),B) == 1
        num = Mut3Rep1ResNum(i);
        if num ~= 1200
            if Mut3ChainB(num,2) == 0
                Mut3ChainB(num,2) = Mut3Rep1Energy(i);
            else Mut3ChainB(num,2) =
 (Mut3ChainB(num,2)+Mut3Rep1Energy(i))/2;
            end
        end
    end
end
%Append energies and average in plance to the third columns of ChainA
 and
%ChainB arrays
for i=1:length(Mut3Rep2Chain)
    if strcmp(Mut3Rep2Chain(i),A) == 1
```

```
        num = Mut3Rep2ResNum(i);
        if num ~= 1200
            if Mut3ChainA(num,3) == 0
                Mut3ChainA(num,3) = Mut3Rep2Energy(i);
            else Mut3ChainA(num,3) =
 (Mut3ChainA(num,3)+Mut3Rep2Energy(i))/2;
            end
        end
    elseif strcmp(Mut3Rep2Chain(i),B) == 1
        num = Mut3Rep2ResNum(i);
        if num ~= 1200
            if Mut3ChainB(num,3) == 0
                Mut3ChainB(num,3) = Mut3Rep2Energy(i);
            else Mut3ChainB(num,3) =
 (Mut3ChainB(num,3)+Mut3Rep2Energy(i))/2;
            end
        end
    end
end
%Append energies and average in plance to the fourth columns of ChainA
 and
%ChainB arrays
for i=1:length(Mut3Rep3Chain)
    if strcmp(Mut3Rep3Chain(i),A) == 1
        num = Mut3Rep3ResNum(i);
        if num ~= 1200
            if Mut3ChainA(num,4) == 0
                Mut3ChainA(num,4) = Mut3Rep3Energy(i);
            else Mut3ChainA(num,4) =
 (Mut3ChainA(num,4)+Mut3Rep3Energy(i))/2;
            end
        end
    elseif strcmp(Mut3Rep3Chain(i),B) == 1
        num = Mut3Rep3ResNum(i);
        if num ~= 1200
            if Mut3ChainB(num,4) == 0
                Mut3ChainB(num,4) = Mut3Rep3Energy(i);
            else Mut3ChainB(num,4) =
 (Mut3ChainB(num,4)+Mut3Rep3Energy(i))/2;
            end
        end
    end
end
```

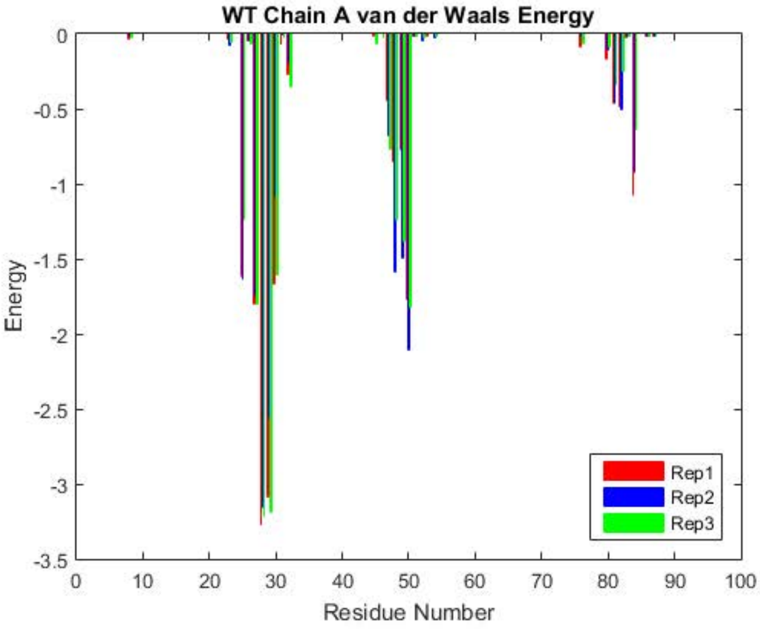
WT Average and Significant Energies

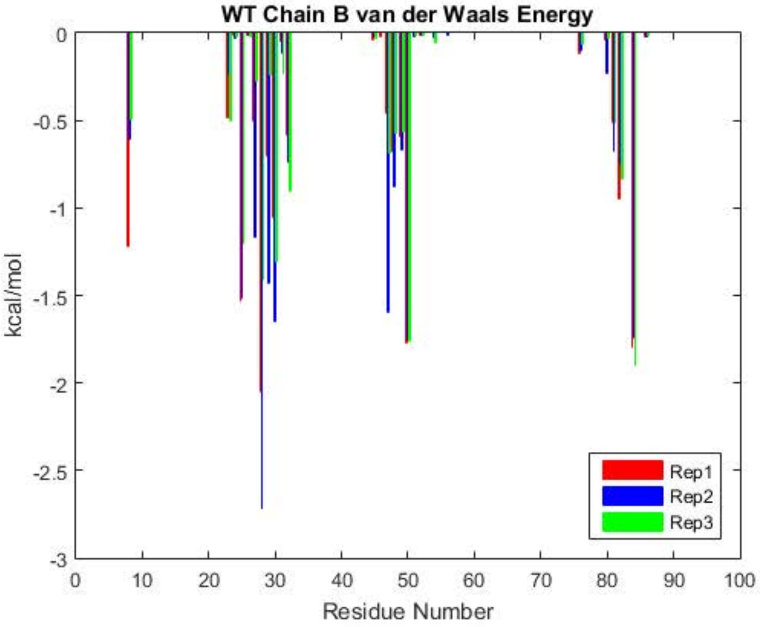
Note: significant is defined as an energy less than -0.02.

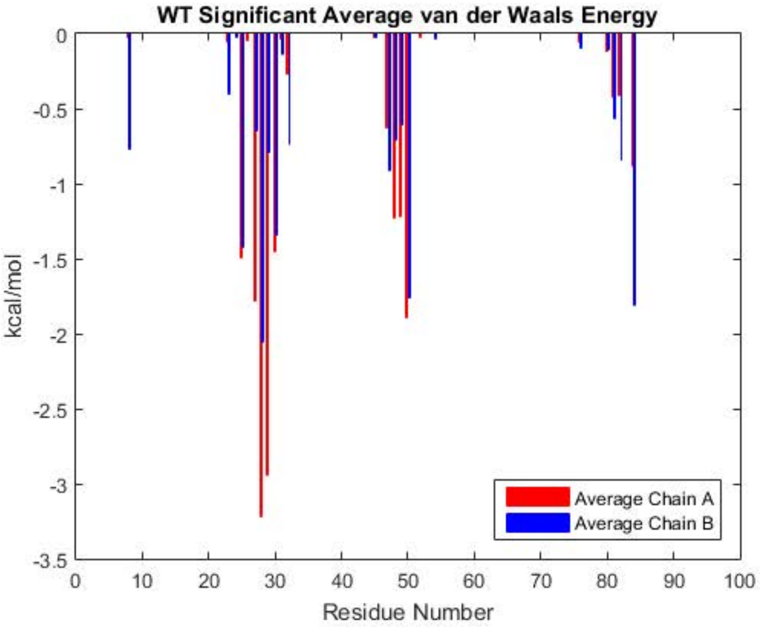
```
%Remove residue numbers
WTChainA(:,1)=[];
WTChainB(:,1)=[];

%ChainA and ChainB plots
```

```
figure
x = 1:99;
chainA_bar = bar(x,WTChainA);
chainA_bar(1).FaceColor = 'r';
chainA_bar(1).EdgeColor = 'r';
chainA_bar(2).FaceColor = 'b';
chainA_bar(2).EdgeColor = 'b';
chainA_bar(3).FaceColor = 'g';
chainA_bar(3).EdgeColor = 'g';
title('WT Chain A van der Waals Energy')
ylabel('Energy')
xlabel('Residue Number')
legend('Rep1','Rep2','Rep3','Location', 'southeast')

figure
chainB_bar = bar(x,WTChainB);
chainB_bar(1).FaceColor = 'r';
chainB_bar(1).EdgeColor = 'r';
chainB_bar(2).FaceColor = 'b';
chainB_bar(2).EdgeColor = 'b';
chainB_bar(3).FaceColor = 'g';
chainB_bar(3).EdgeColor = 'g';
title('WT Chain B van der Waals Energy')
ylabel('kcal/mol')
xlabel('Residue Number')
legend('Rep1','Rep2','Rep3','Location', 'southeast')

%calculate average and remove terms that are greater than -0.02
WTAvg_ChainA = mean(WTChainA,2);
WTAvg_ChainB = mean(WTChainB,2);

WTsig_ChainA = zeros(99,1);
WTsig_ChainB = zeros(99,1);
for i = 1:99
    if WTAvg_ChainA(i) <= -0.02
        WTsig_ChainA(i) = WTAvg_ChainA(i);
    else WTsig_ChainA(i) = 0;
    end
    if WTAvg_ChainB(i) <= -0.02
        WTsig_ChainB(i) = WTAvg_ChainB(i);
    else WTsig_ChainB(i) = 0;
    end
end

averages = horzcat(WTsig_ChainA,WTsig_ChainB);
% Average Significant Energy
figure
avg_bar = bar(x,averages);
avg_bar(1).FaceColor = 'r';
avg_bar(1).EdgeColor = 'r';
avg_bar(2).FaceColor = 'b';
avg_bar(2).EdgeColor = 'b';
title('WT Significant Average van der Waals Energy')
ylabel('kcal/mol')
```

```
xlabel('Residue Number')
legend('Average Chain A','Average Chain B','Location','Southeast')
```




WT Chain B van der Waals Energy
kcal/mol
0
-0.5
-1
-1.5
-2
-2.5
-3
0
10
20
30
40
50
60
70
80
90
100
Residue Number
Rep1
Rep2
Rep3

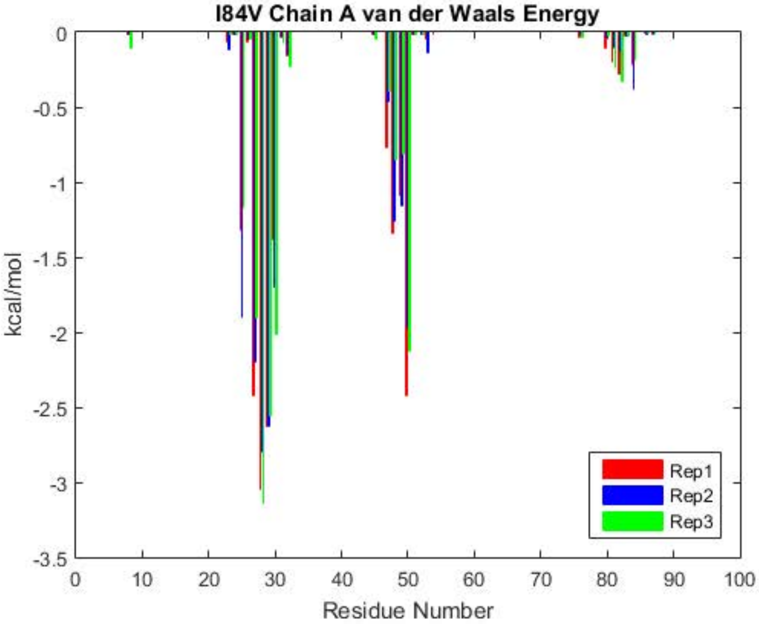

WT Significant Average van der Waals Energy
kcal/mol
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
0
10
20
30
40
50
60
70
80
90
100
Residue Number
Average Chain A
Average Chain B

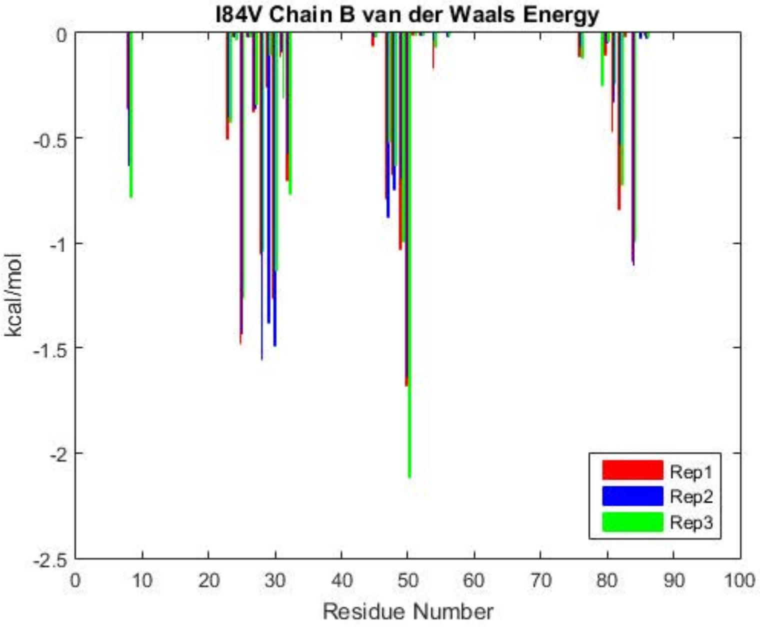
Mut1 Average and Significant Energies

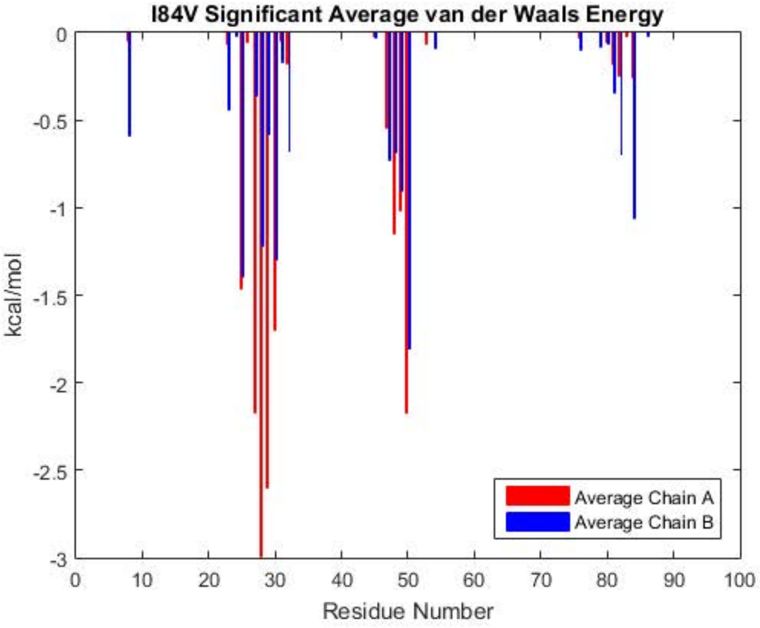
Note: significant is defined as an energy less than -0.02.

```
%remove residue numbers
Mut1ChainA(:,1)=[];
Mut1ChainB(:,1)=[];

%ChainA and ChainB plots
figure
x = 1:99;
chainA_bar = bar(x,Mut1ChainA);
chainA_bar(1).FaceColor = 'r';
chainA_bar(1).EdgeColor = 'r';
chainA_bar(2).FaceColor = 'b';
chainA_bar(2).EdgeColor = 'b';
chainA_bar(3).FaceColor = 'g';
chainA_bar(3).EdgeColor = 'g';
title('I84V Chain A van der Waals Energy')
ylabel('kcal/mol')
xlabel('Residue Number')
legend('Rep1','Rep2','Rep3','Location', 'Southeast')

figure
chainB_bar = bar(x,Mut1ChainB);
chainB_bar(1).FaceColor = 'r';
chainB_bar(1).EdgeColor = 'r';
chainB_bar(2).FaceColor = 'b';
chainB_bar(2).EdgeColor = 'b';
chainB_bar(3).FaceColor = 'g';
chainB_bar(3).EdgeColor = 'g';
title('I84V Chain B van der Waals Energy')
ylabel('kcal/mol')
xlabel('Residue Number')
legend('Rep1','Rep2','Rep3','Location','Southeast')

%calculate average and remove terms that are greater than -0.02
Mut1Avg_ChainA = mean(Mut1ChainA,2);
Mut1Avg_ChainB = mean(Mut1ChainB,2);

Mut1sig_ChainA = zeros(99,1);
Mut1sig_ChainB = zeros(99,1);
for i = 1:99
    if Mut1Avg_ChainA(i) <= -0.02
        Mut1sig_ChainA(i) = Mut1Avg_ChainA(i);
    else Mut1sig_ChainA(i) = 0;
    end
    if Mut1Avg_ChainB(i) <= -0.02
        Mut1sig_ChainB(i) = Mut1Avg_ChainB(i);
    else Mut1sig_ChainB(i) = 0;
    end
end
```

```
Mut1averages = horzcat(Mut1sig_ChainA,Mut1sig_ChainB);
% Average Significant Energy
figure
avg_bar = bar(x,Mut1averages);
avg_bar(1).FaceColor = 'r';
avg_bar(1).EdgeColor = 'r';
avg_bar(2).FaceColor = 'b';
avg_bar(2).EdgeColor = 'b';
title('I84V Significant Average van der Waals Energy')
ylabel('kcal/mol')
xlabel('Residue Number')
legend('Average Chain A','Average Chain B','Location','Southeast')
```




I84V Chain B van der Waals Energy
kcal/mol
0
-0.5
-1
-1.5
-2
-2.5
0
10
20
30
40
50
60
70
80
90
100
Residue Number
Rep1
Rep2
Rep3

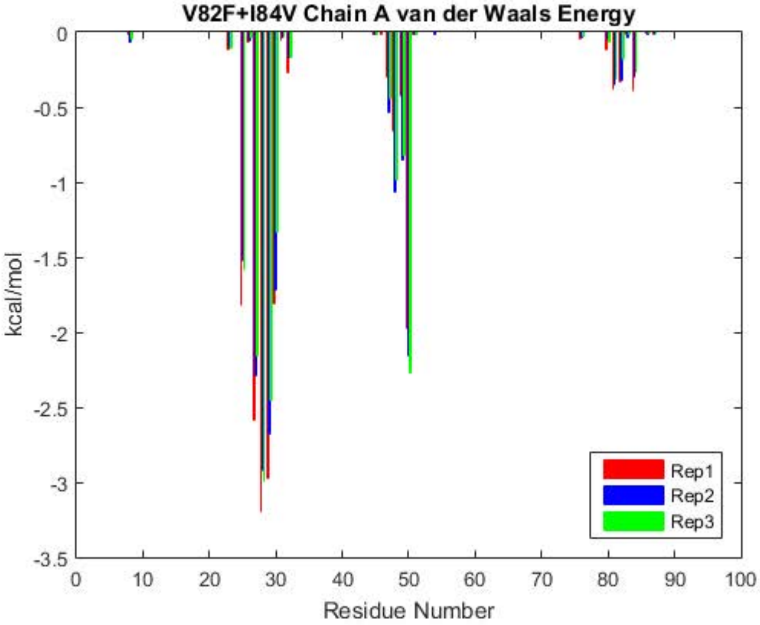

I84V Significant Average van der Waals Energy
kcal/mol
0
-0.5
-1
-1.5
-2
-2.5
-3
0
10
20
30
40
50
60
70
80
90
100
Residue Number
Average Chain A
Average Chain B

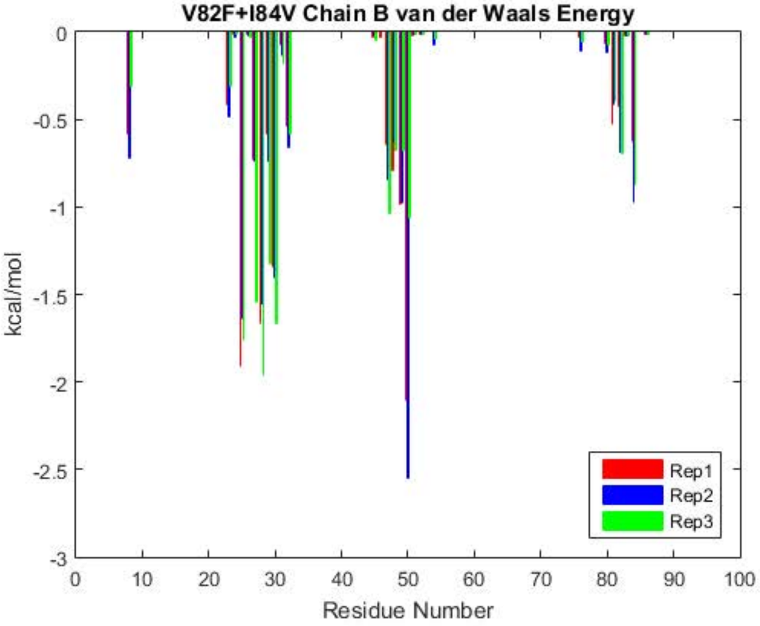
Mut2 Average and Significant Energies

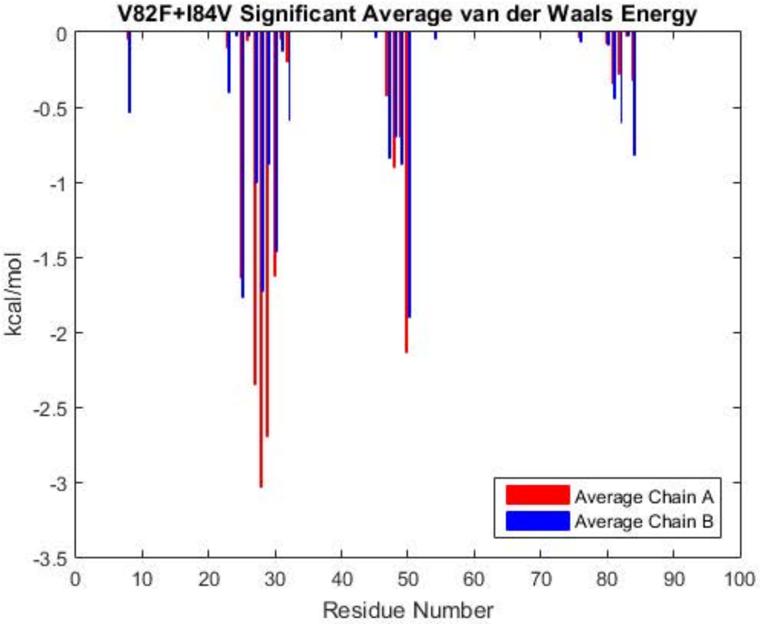
Note: significant is defined as an energy less than -0.02.

```
%remove residue numbers
Mut2ChainA(:,1)=[];
Mut2ChainB(:,1)=[];

%ChainA and ChainB plots
figure
x = 1:99;
chainA_bar = bar(x,Mut2ChainA);
chainA_bar(1).FaceColor = 'r';
chainA_bar(1).EdgeColor = 'r';
chainA_bar(2).FaceColor = 'b';
chainA_bar(2).EdgeColor = 'b';
chainA_bar(3).FaceColor = 'g';
chainA_bar(3).EdgeColor = 'g';
title('V82F+I84V Chain A van der Waals Energy')
ylabel('kcal/mol')
xlabel('Residue Number')
legend('Rep1','Rep2','Rep3','Location','Southeast')

figure
chainB_bar = bar(x,Mut2ChainB);
chainB_bar(1).FaceColor = 'r';
chainB_bar(1).EdgeColor = 'r';
chainB_bar(2).FaceColor = 'b';
chainB_bar(2).EdgeColor = 'b';
chainB_bar(3).FaceColor = 'g';
chainB_bar(3).EdgeColor = 'g';
title('V82F+I84V Chain B van der Waals Energy')
ylabel('kcal/mol')
xlabel('Residue Number')
legend('Rep1','Rep2','Rep3','Location','Southeast')

%calculate average and remove terms that are greater than -0.02
Mut2Avg_ChainA = mean(Mut2ChainA,2);
Mut2Avg_ChainB = mean(Mut2ChainB,2);

Mut2sig_ChainA = zeros(99,1);
Mut2sig_ChainB = zeros(99,1);
for i = 1:99
    if Mut2Avg_ChainA(i) <= -0.02
        Mut2sig_ChainA(i) = Mut2Avg_ChainA(i);
    else Mut2sig_ChainA(i) = 0;
    end
    if Mut2Avg_ChainB(i) <= -0.02
        Mut2sig_ChainB(i) = Mut2Avg_ChainB(i);
    else Mut2sig_ChainB(i) = 0;
    end
end
```

```
Mut2averages = horzcat(Mut2sig_ChainA,Mut2sig_ChainB);
% Average Significant Energy
figure
avg_bar = bar(x,Mut2averages);
avg_bar(1).FaceColor = 'r';
avg_bar(1).EdgeColor = 'r';
avg_bar(2).FaceColor = 'b';
avg_bar(2).EdgeColor = 'b';
title('V82F+I84V Significant Average van der Waals Energy')
ylabel('kcal/mol')
xlabel('Residue Number')
legend('Average Chain A','Average Chain B','Location','Southeast')
```




V82F+I84V Chain B van der Waals Energy
kcal/mol
Residue Number
0
-0.5
-1
-1.5
-2
-2.5
-3
0
10
20
30
40
50
60
70
80
90
100
Rep1
Rep2
Rep3


V82F+I84V Significant Average van der Waals Energy
kcal/mol
Residue Number
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
0
10
20
30
40
50
60
70
80
90
100
Average Chain A
Average Chain B

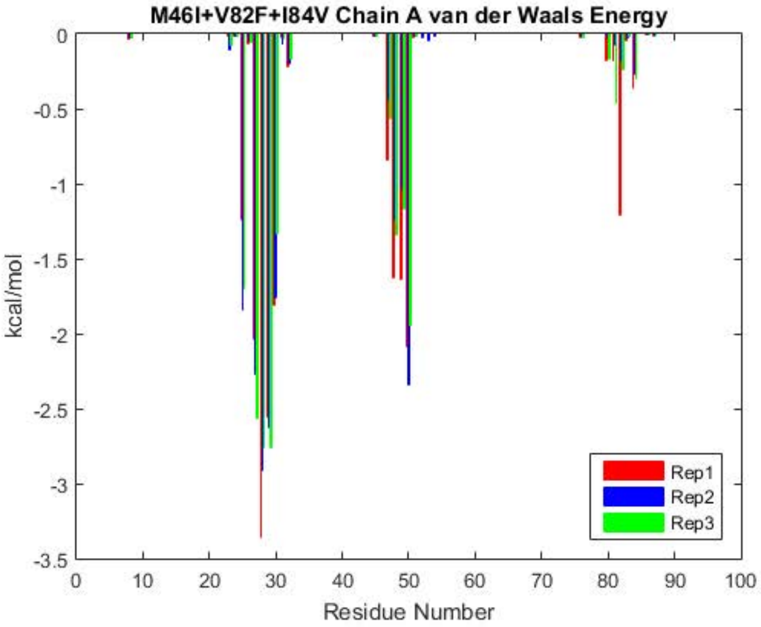
Mut3 Average and Significant Energies

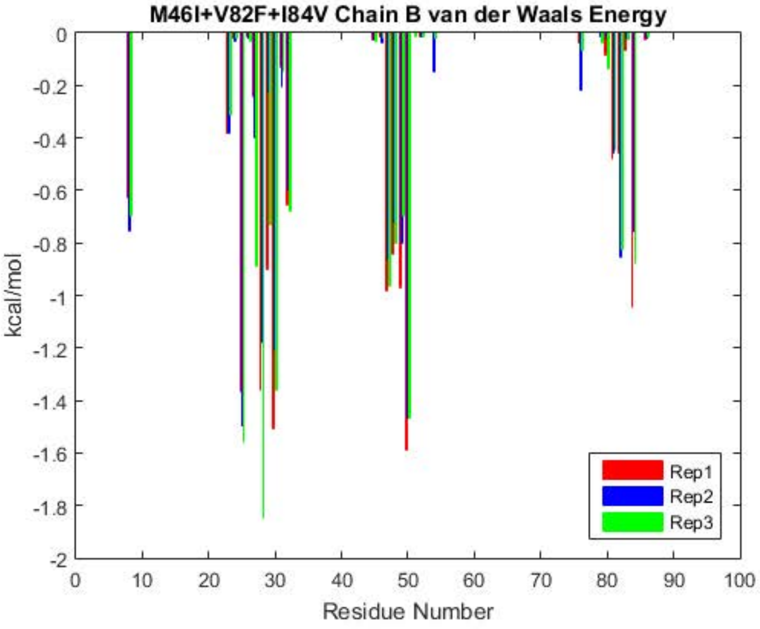
Note: significant is defined as an energy less than -0.02.

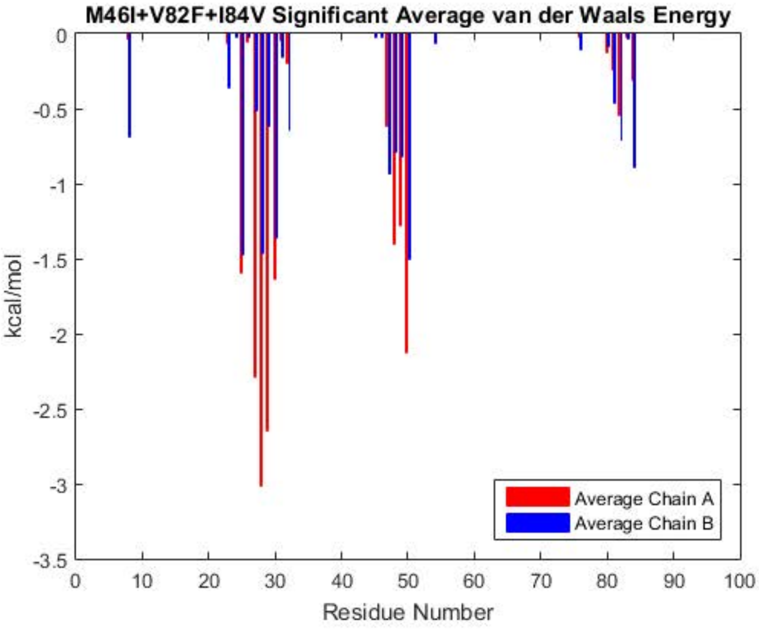
```
%remove residue numbers
Mut3ChainA(:,1)=[];
Mut3ChainB(:,1)=[];

%ChainA and ChainB plots
figure
x = 1:99
chainA_bar = bar(x,Mut3ChainA);
chainA_bar(1).FaceColor = 'r';
chainA_bar(1).EdgeColor = 'r';
chainA_bar(2).FaceColor = 'b';
chainA_bar(2).EdgeColor = 'b';
chainA_bar(3).FaceColor = 'g';
chainA_bar(3).EdgeColor = 'g';
title('M46I+V82F+I84V Chain A van der Waals Energy')
ylabel('kcal/mol')
xlabel('Residue Number')
legend('Rep1','Rep2','Rep3','Location','Southeast')

figure
chainB_bar = bar(x,Mut3ChainB);
chainB_bar(1).FaceColor = 'r';
chainB_bar(1).EdgeColor = 'r';
chainB_bar(2).FaceColor = 'b';
chainB_bar(2).EdgeColor = 'b';
chainB_bar(3).FaceColor = 'g';
chainB_bar(3).EdgeColor = 'g';
title('M46I+V82F+I84V Chain B van der Waals Energy')
ylabel('kcal/mol')
xlabel('Residue Number')
legend('Rep1','Rep2','Rep3','Location','Southeast')

%calculate average and remove terms that are greater than -0.02
Mut3Avg_ChainA = mean(Mut3ChainA,2);
Mut3Avg_ChainB = mean(Mut3ChainB,2);

Mut3sig_ChainA = zeros(99,1);
Mut3sig_ChainB = zeros(99,1);
for i = 1:99
    if Mut3Avg_ChainA(i) <= -0.02
        Mut3sig_ChainA(i) = Mut3Avg_ChainA(i);
    else Mut3sig_ChainA(i) = 0;
    end
    if Mut3Avg_ChainB(i) <= -0.02
        Mut3sig_ChainB(i) = Mut3Avg_ChainB(i);
    else Mut3sig_ChainB(i) = 0;
    end
end
```

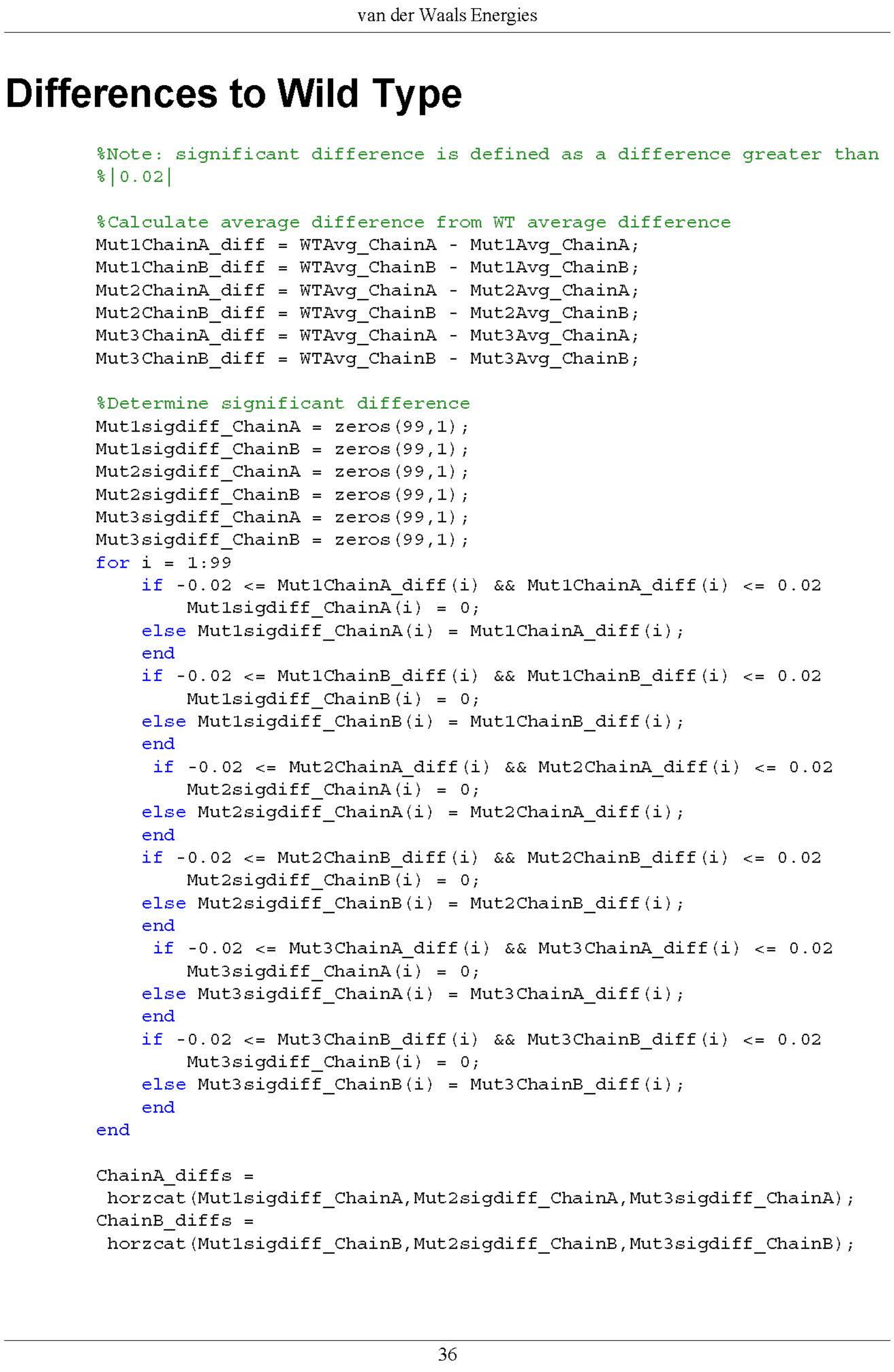
```
Mut3averages = horzcat(Mut3sig_ChainA,Mut3sig_ChainB);
% Average Significant Energy
figure
avg_bar = bar(x,Mut3averages);
avg_bar(1).FaceColor = 'r';
avg_bar(1).EdgeColor = 'r';
avg_bar(2).FaceColor = 'b';
avg_bar(2).EdgeColor = 'b';
title('M46I+V82F+I84V Significant Average van der Waals Energy')
ylabel('kcal/mol')
xlabel('Residue Number')
legend('Average Chain A','Average Chain B','Location','Southeast')
```

```
x =

  Columns 1 through 13

     1     2     3     4     5     6     7     8     9    10    11
 12    13

  Columns 14 through 26

    14    15    16    17    18    19    20    21    22    23    24
 25    26

  Columns 27 through 39

    27    28    29    30    31    32    33    34    35    36    37
 38    39

  Columns 40 through 52

    40    41    42    43    44    45    46    47    48    49    50
 51    52

  Columns 53 through 65

    53    54    55    56    57    58    59    60    61    62    63
 64    65

  Columns 66 through 78

    66    67    68    69    70    71    72    73    74    75    76
 77    78

  Columns 79 through 91

    79    80    81    82    83    84    85    86    87    88    89
 90    91

  Columns 92 through 99

    92    93    94    95    96    97    98    99
```


M46I+V82F+I84V Chain A van der Waals Energy
kcal/mol
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
0
10
20
30
40
50
60
70
80
90
100
Residue Number
Rep1
Rep2
Rep3


M46I+V82F+I84V Chain B van der Waals Energy
kcal/mol
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-2
0
10
20
30
40
50
60
70
80
90
100
Residue Number
Rep1
Rep2
Rep3


M46I+V82F+I84V Significant Average van der Waals Energy
kcal/mol
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
0
10
20
30
40
50
60
70
80
90
100
Residue Number
Average Chain A
Average Chain B

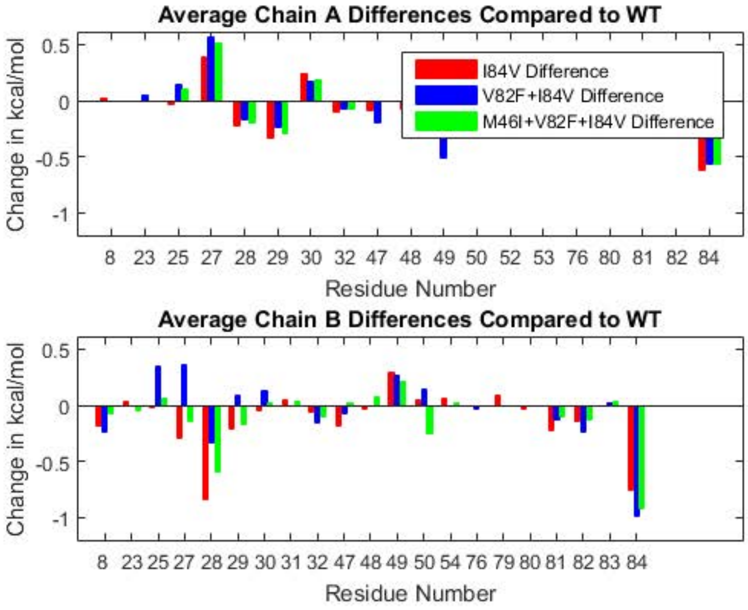


Differences to Wild Type

```
%Note: significant difference is defined as a difference greater than
%|0.02|

%Calculate average difference from WT average difference
Mut1ChainA_diff = WTAvg_ChainA - Mut1Avg_ChainA;
Mut1ChainB_diff = WTAvg_ChainB - Mut1Avg_ChainB;
Mut2ChainA_diff = WTAvg_ChainA - Mut2Avg_ChainA;
Mut2ChainB_diff = WTAvg_ChainB - Mut2Avg_ChainB;
Mut3ChainA_diff = WTAvg_ChainA - Mut3Avg_ChainA;
Mut3ChainB_diff = WTAvg_ChainB - Mut3Avg_ChainB;

%Determine significant difference
Mut1sigdiff_ChainA = zeros(99,1);
Mut1sigdiff_ChainB = zeros(99,1);
Mut2sigdiff_ChainA = zeros(99,1);
Mut2sigdiff_ChainB = zeros(99,1);
Mut3sigdiff_ChainA = zeros(99,1);
Mut3sigdiff_ChainB = zeros(99,1);
for i = 1:99
    if -0.02 <= Mut1ChainA_diff(i) && Mut1ChainA_diff(i) <= 0.02
        Mut1sigdiff_ChainA(i) = 0;
    else Mut1sigdiff_ChainA(i) = Mut1ChainA_diff(i);
    end
    if -0.02 <= Mut1ChainB_diff(i) && Mut1ChainB_diff(i) <= 0.02
        Mut1sigdiff_ChainB(i) = 0;
    else Mut1sigdiff_ChainB(i) = Mut1ChainB_diff(i);
    end
     if -0.02 <= Mut2ChainA_diff(i) && Mut2ChainA_diff(i) <= 0.02
        Mut2sigdiff_ChainA(i) = 0;
    else Mut2sigdiff_ChainA(i) = Mut2ChainA_diff(i);
    end
    if -0.02 <= Mut2ChainB_diff(i) && Mut2ChainB_diff(i) <= 0.02
        Mut2sigdiff_ChainB(i) = 0;
    else Mut2sigdiff_ChainB(i) = Mut2ChainB_diff(i);
    end
     if -0.02 <= Mut3ChainA_diff(i) && Mut3ChainA_diff(i) <= 0.02
        Mut3sigdiff_ChainA(i) = 0;
    else Mut3sigdiff_ChainA(i) = Mut3ChainA_diff(i);
    end
    if -0.02 <= Mut3ChainB_diff(i) && Mut3ChainB_diff(i) <= 0.02
        Mut3sigdiff_ChainB(i) = 0;
    else Mut3sigdiff_ChainB(i) = Mut3ChainB_diff(i);
    end
end

ChainA_diffs =
 horzcat(Mut1sigdiff_ChainA,Mut2sigdiff_ChainA,Mut3sigdiff_ChainA);
ChainB_diffs =
 horzcat(Mut1sigdiff_ChainB,Mut2sigdiff_ChainB,Mut3sigdiff_ChainB);
```

Average Significant Engery Differences to WT

```
residuesA = [];
ChainA_data = [];
for i = 1:99
    if ChainA_diffs(i,1) ~= 0 | ChainA_diffs(i,2) ~= 0 |
 ChainA_diffs(i,3) ~= 0
        x = ChainA_diffs(i,1);
        y = ChainA_diffs(i,2);
        z = ChainA_diffs(i,3);
        ChainA_data = [ChainA_data; x,y,z];
        residuesA = [residuesA; i];
    end
end
residuesB = [];
ChainB_data = [];
for i = 1:99
    if ChainB_diffs(i,1) ~= 0 | ChainB_diffs(i,2) ~= 0 |
 ChainB_diffs(i,3) ~= 0
        x = ChainB_diffs(i,1);
        y = ChainB_diffs(i,2);
        z = ChainB_diffs(i,3);
        ChainB_data = [ChainB_data; x,y,z];
        residuesB = [residuesB; i];
    end
end

fig1=figure;
subplot(2,1,1)
barA = bar(ChainA_data);
hold on
set(gca, 'XTick',1:length(residuesA));
set(gca, 'XTickLabel', residuesA);
barA(1).FaceColor = 'r';
barA(1).EdgeColor = 'r';
barA(2).FaceColor = 'b';
barA(2).EdgeColor = 'b';
barA(3).FaceColor = 'g';
barA(3).EdgeColor = 'g';
ylabel('Change in kcal/mol')
xlabel('Residue Number')
legend('I84V Difference','V82F+I84V Difference','M46I+V82F+I84V
 Difference')
title('Average Chain A Differences Compared to WT')
ylim([-1.2,0.6])

subplot(2,1,2)
barB = bar(ChainB_data);
set(gca, 'XTick', 1:length(residuesB));
set(gca, 'XTickLabel', residuesB);
barB(1).FaceColor = 'r';
barB(1).EdgeColor = 'r';
barB(2).FaceColor = 'b';
barB(2).EdgeColor = 'b';
barB(3).FaceColor = 'g';
barB(3).EdgeColor = 'g';
ylabel('Change in kcal/mol')
xlabel('Residue Number')
title('Average Chain B Differences Compared to WT')
ylim([-1.2,0.6])
```



Published with MATLAB® R2015a

B.5 Hydrogen Bonds Script

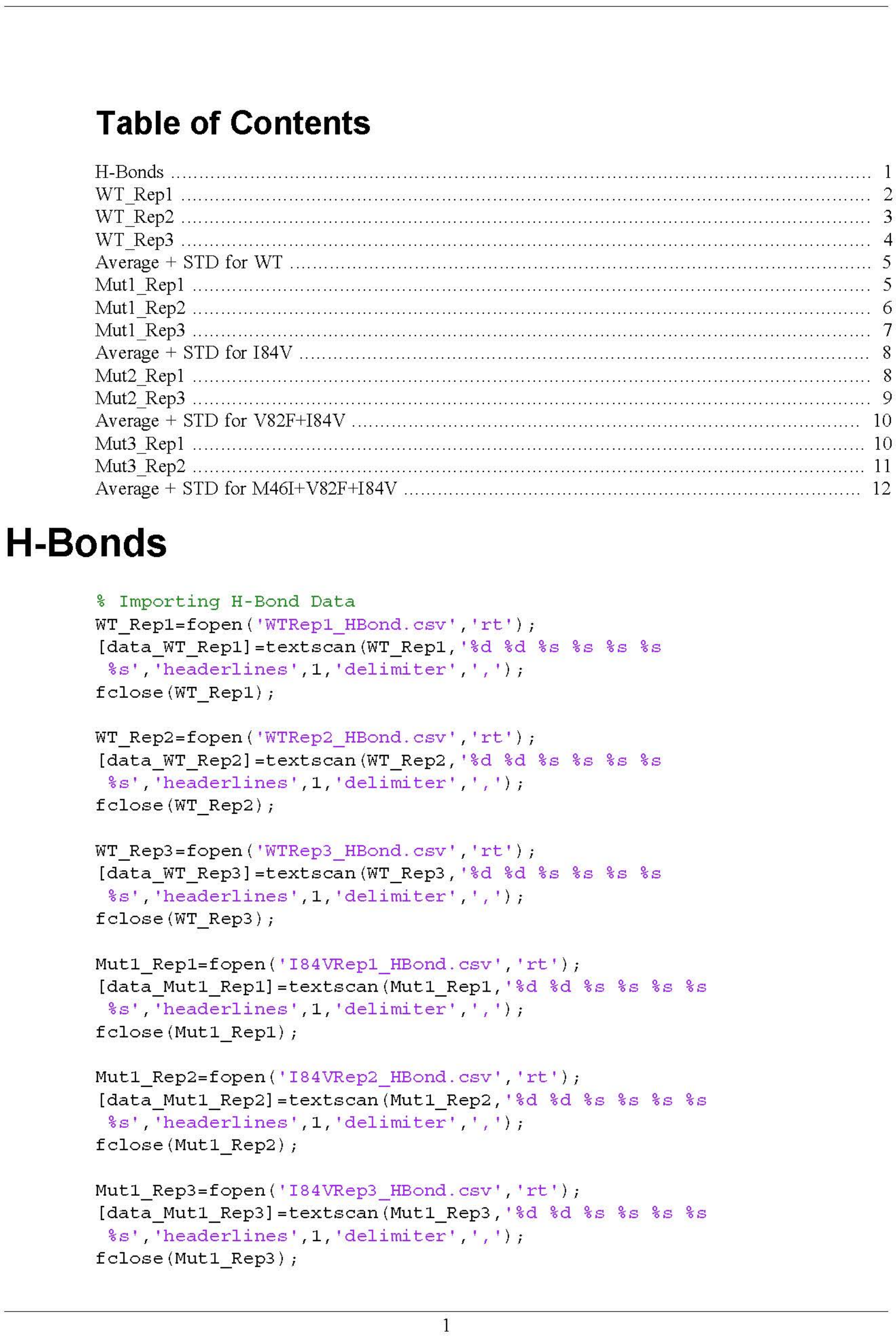
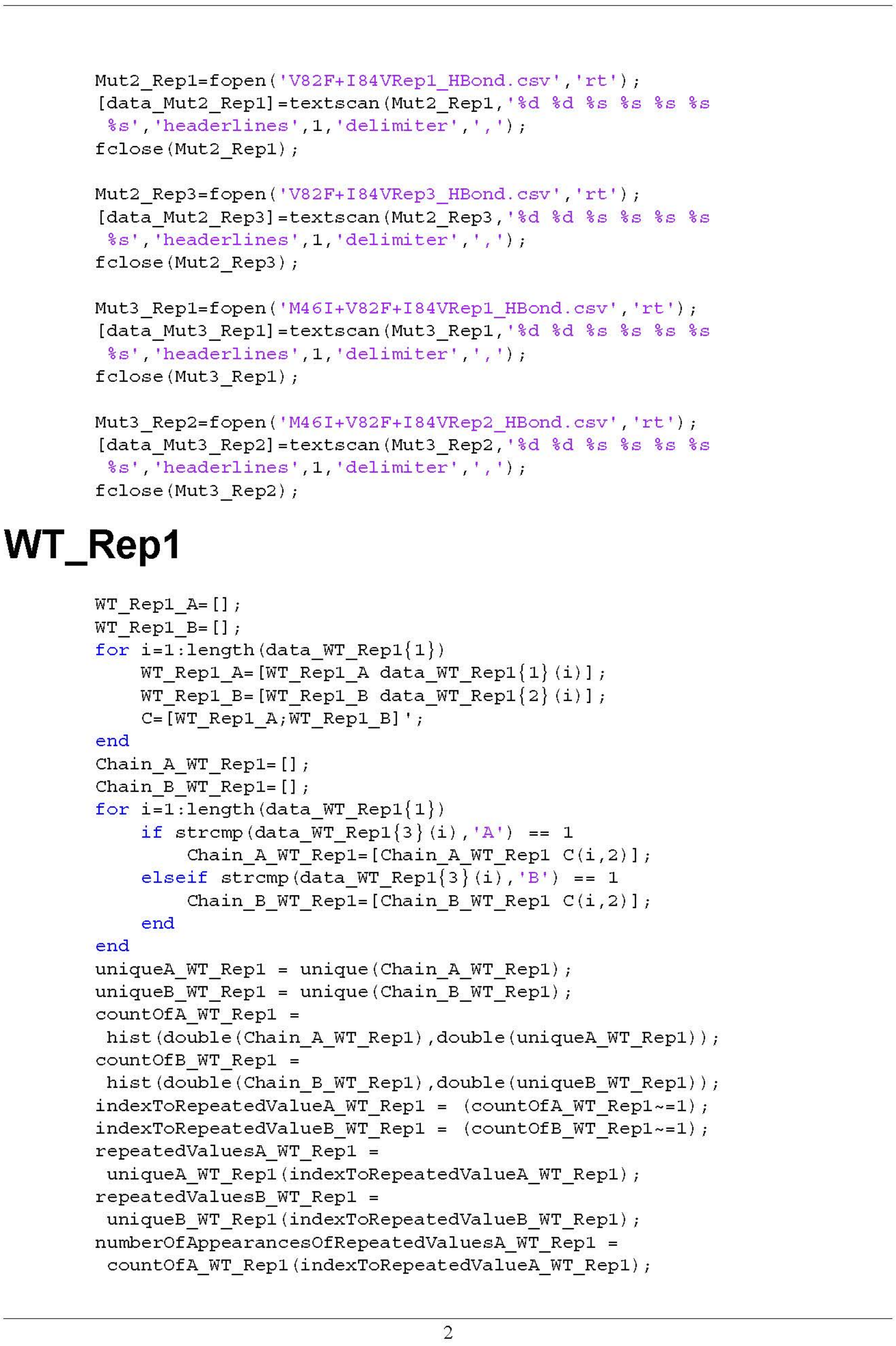


Table of Contents

H-Bonds

```
% Importing H-Bond Data
WT_Rep1=fopen('WTRep1_HBond.csv','rt');
[data_WT_Rep1]=textscan(WT_Rep1,'%d %d %s %s %s %s
 %s','headerlines',1,'delimiter',',');
fclose(WT_Rep1);

WT_Rep2=fopen('WTRep2_HBond.csv','rt');
[data_WT_Rep2]=textscan(WT_Rep2,'%d %d %s %s %s %s
 %s','headerlines',1,'delimiter',',');
fclose(WT_Rep2);

WT_Rep3=fopen('WTRep3_HBond.csv','rt');
[data_WT_Rep3]=textscan(WT_Rep3,'%d %d %s %s %s %s
 %s','headerlines',1,'delimiter',',');
fclose(WT_Rep3);

Mut1_Rep1=fopen('I84VRep1_HBond.csv','rt');
[data_Mut1_Rep1]=textscan(Mut1_Rep1,'%d %d %s %s %s %s
 %s','headerlines',1,'delimiter',',');
fclose(Mut1_Rep1);

Mut1_Rep2=fopen('I84VRep2_HBond.csv','rt');
[data_Mut1_Rep2]=textscan(Mut1_Rep2,'%d %d %s %s %s %s
 %s','headerlines',1,'delimiter',',');
fclose(Mut1_Rep2);

Mut1_Rep3=fopen('I84VRep3_HBond.csv','rt');
[data_Mut1_Rep3]=textscan(Mut1_Rep3,'%d %d %s %s %s %s
 %s','headerlines',1,'delimiter',',');
fclose(Mut1_Rep3);
```



```
Mut2_Rep1=fopen('V82F+I84VRep1_HBond.csv','rt');
[data_Mut2_Rep1]=textscan(Mut2_Rep1,'%d %d %s %s %s %s
 %s','headerlines',1,'delimiter',',');
fclose(Mut2_Rep1);

Mut2_Rep3=fopen('V82F+I84VRep3_HBond.csv','rt');
[data_Mut2_Rep3]=textscan(Mut2_Rep3,'%d %d %s %s %s %s
 %s','headerlines',1,'delimiter',',');
fclose(Mut2_Rep3);

Mut3_Rep1=fopen('M46I+V82F+I84VRep1_HBond.csv','rt');
[data_Mut3_Rep1]=textscan(Mut3_Rep1,'%d %d %s %s %s %s
 %s','headerlines',1,'delimiter',',');
fclose(Mut3_Rep1);

Mut3_Rep2=fopen('M46I+V82F+I84VRep2_HBond.csv','rt');
[data_Mut3_Rep2]=textscan(Mut3_Rep2,'%d %d %s %s %s %s
 %s','headerlines',1,'delimiter',',');
fclose(Mut3_Rep2);
```

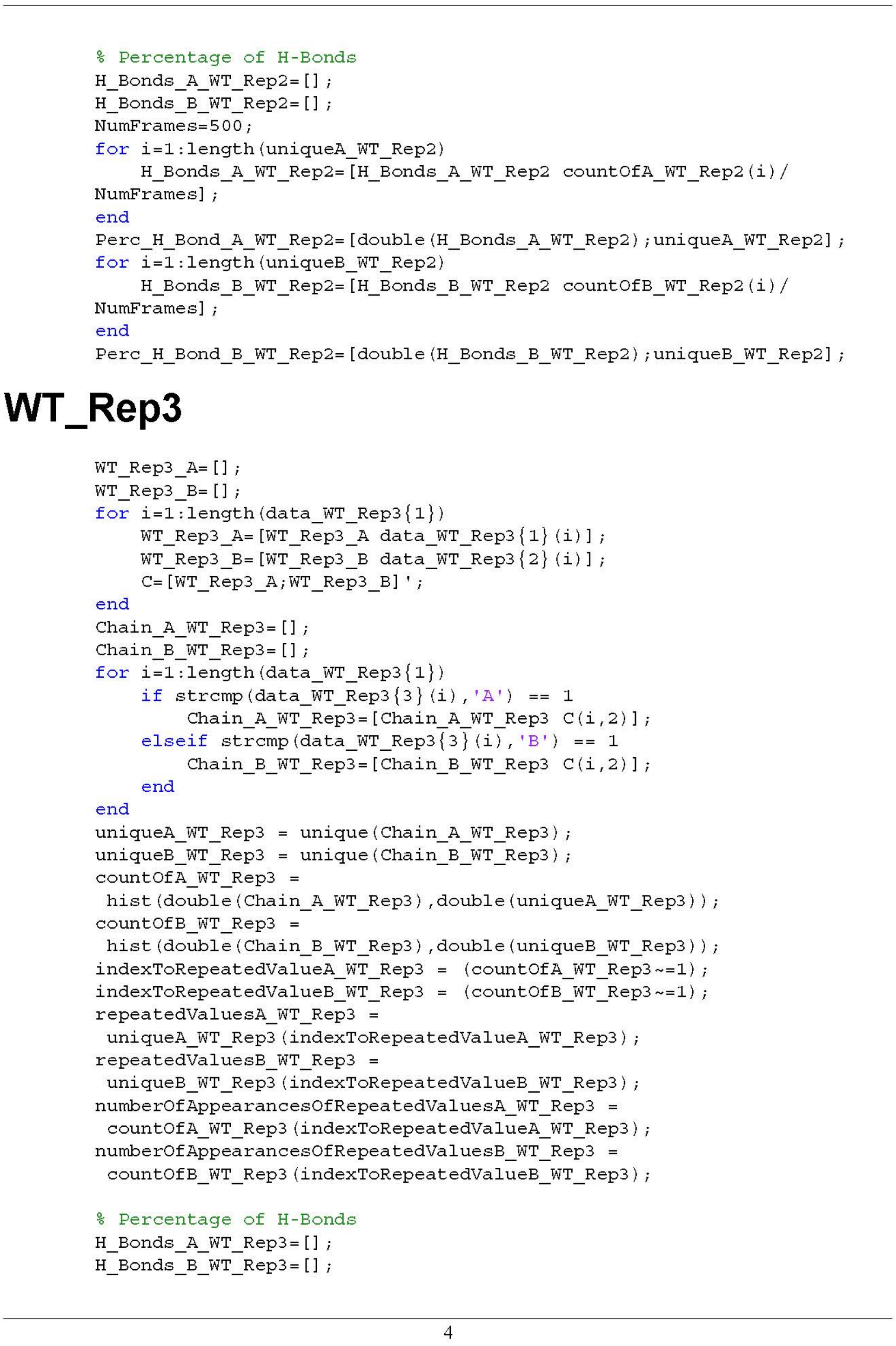
WT_Rep1

```
WT_Rep1_A=[];
WT_Rep1_B=[];
for i=1:length(data_WT_Rep1{1})
    WT_Rep1_A=[WT_Rep1_A data_WT_Rep1{1}(i)];
    WT_Rep1_B=[WT_Rep1_B data_WT_Rep1{2}(i)];
    C=[WT_Rep1_A;WT_Rep1_B]';
end
Chain_A_WT_Rep1=[];
Chain_B_WT_Rep1=[];
for i=1:length(data_WT_Rep1{1})
    if strcmp(data_WT_Rep1{3}(i),'A') == 1
        Chain_A_WT_Rep1=[Chain_A_WT_Rep1 C(i,2)];
    elseif strcmp(data_WT_Rep1{3}(i),'B') == 1
        Chain_B_WT_Rep1=[Chain_B_WT_Rep1 C(i,2)];
    end
end
uniqueA_WT_Rep1 = unique(Chain_A_WT_Rep1);
uniqueB_WT_Rep1 = unique(Chain_B_WT_Rep1);
countOfA_WT_Rep1 =
 hist(double(Chain_A_WT_Rep1),double(uniqueA_WT_Rep1));
countOfB_WT_Rep1 =
 hist(double(Chain_B_WT_Rep1),double(uniqueB_WT_Rep1));
indexToRepeatedValueA_WT_Rep1 = (countOfA_WT_Rep1~=1);
indexToRepeatedValueB_WT_Rep1 = (countOfB_WT_Rep1~=1);
repeatedValuesA_WT_Rep1 =
 uniqueA_WT_Rep1(indexToRepeatedValueA_WT_Rep1);
repeatedValuesB_WT_Rep1 =
 uniqueB_WT_Rep1(indexToRepeatedValueB_WT_Rep1);
numberOfAppearancesOfRepeatedValuesA_WT_Rep1 =
 countOfA_WT_Rep1(indexToRepeatedValueA_WT_Rep1);
```

```
numberOfAppearancesOfRepeatedValuesB_WT_Rep1 =
 countOfB_WT_Rep1(indexToRepeatedValueB_WT_Rep1);

% Percentage of H-Bonds
H_Bonds_A_WT_Rep1=[];
H_Bonds_B_WT_Rep1=[];
NumFrames=500;
for i=1:length(uniqueA_WT_Rep1)
    H_Bonds_A_WT_Rep1=[H_Bonds_A_WT_Rep1 countOfA_WT_Rep1(i)/
NumFrames];
end
Perc_H_Bond_A_WT_Rep1=[double(H_Bonds_A_WT_Rep1);uniqueA_WT_Rep1];
for i=1:length(uniqueB_WT_Rep1)
    H_Bonds_B_WT_Rep1=[H_Bonds_B_WT_Rep1 countOfB_WT_Rep1(i)/
NumFrames];
end
Perc_H_Bond_B_WT_Rep1=[double(H_Bonds_B_WT_Rep1);uniqueB_WT_Rep1];
```

WT_Rep2

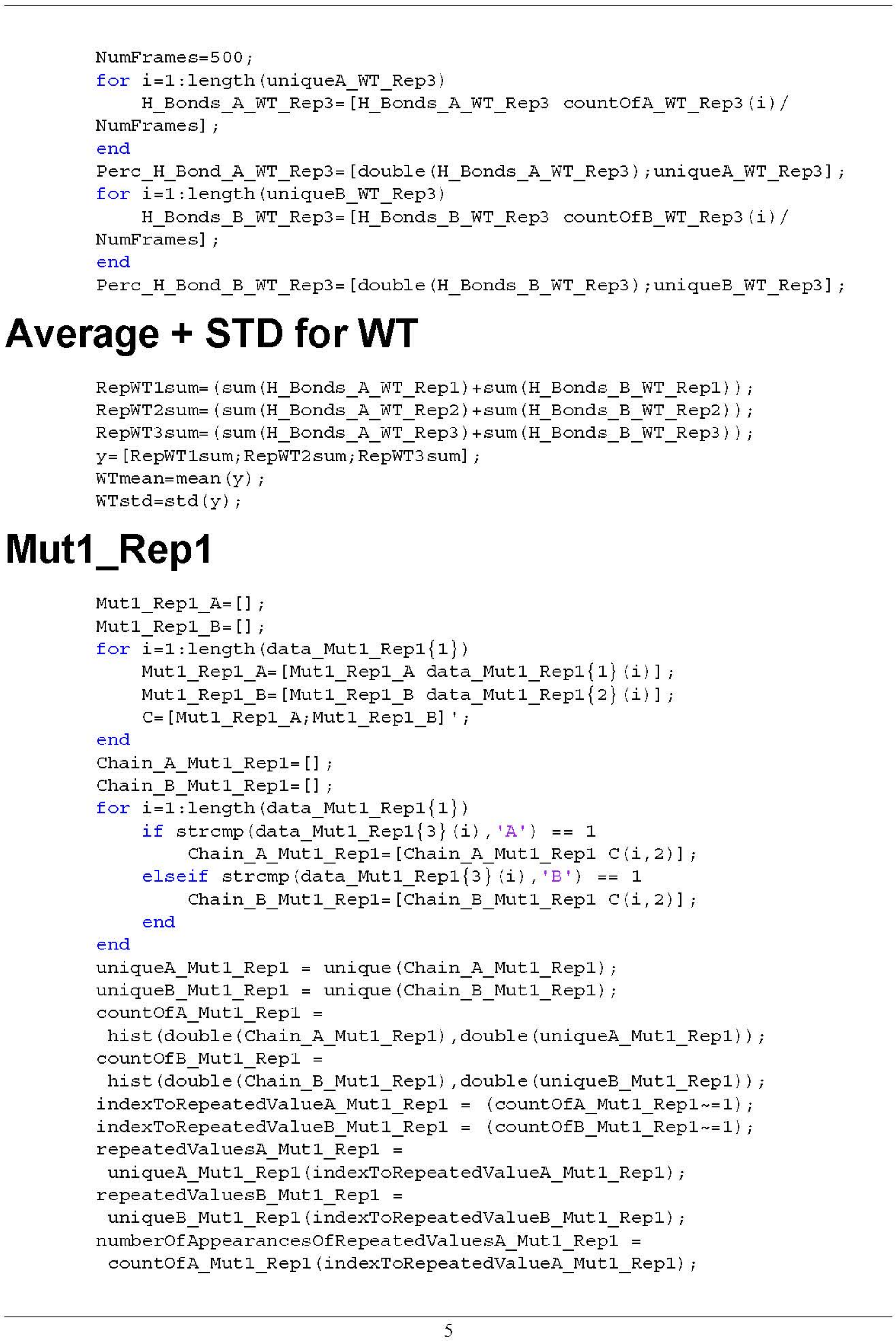
```
WT_Rep2_A=[];
WT_Rep2_B=[];
for i=1:length(data_WT_Rep2{1})
    WT_Rep2_A=[WT_Rep2_A data_WT_Rep2{1}(i)];
    WT_Rep2_B=[WT_Rep2_B data_WT_Rep2{2}(i)];
    C=[WT_Rep2_A;WT_Rep2_B]';
end
Chain_A_WT_Rep2=[];
Chain_B_WT_Rep2=[];
for i=1:length(data_WT_Rep2{1})
    if strcmp(data_WT_Rep2{3}(i),'A') == 1
        Chain_A_WT_Rep2=[Chain_A_WT_Rep2 C(i,2)];
    elseif strcmp(data_WT_Rep2{3}(i),'B') == 1
        Chain_B_WT_Rep2=[Chain_B_WT_Rep2 C(i,2)];
    end
end
uniqueA_WT_Rep2 = unique(Chain_A_WT_Rep2);
uniqueB_WT_Rep2 = unique(Chain_B_WT_Rep2);
countOfA_WT_Rep2 =
 hist(double(Chain_A_WT_Rep2),double(uniqueA_WT_Rep2));
countOfB_WT_Rep2 =
 hist(double(Chain_B_WT_Rep2),double(uniqueB_WT_Rep2));
indexToRepeatedValueA_WT_Rep2 = (countOfA_WT_Rep2~=1);
indexToRepeatedValueB_WT_Rep2 = (countOfB_WT_Rep2~=1);
repeatedValuesA_WT_Rep2 =
 uniqueA_WT_Rep2(indexToRepeatedValueA_WT_Rep2);
repeatedValuesB_WT_Rep2 =
 uniqueB_WT_Rep2(indexToRepeatedValueB_WT_Rep2);
numberOfAppearancesOfRepeatedValuesA_WT_Rep2 =
 countOfA_WT_Rep2(indexToRepeatedValueA_WT_Rep2);
numberOfAppearancesOfRepeatedValuesB_WT_Rep2 =
 countOfB_WT_Rep2(indexToRepeatedValueB_WT_Rep2);
```



```
% Percentage of H-Bonds
H_Bonds_A_WT_Rep2=[];
H_Bonds_B_WT_Rep2=[];
NumFrames=500;
for i=1:length(uniqueA_WT_Rep2)
    H_Bonds_A_WT_Rep2=[H_Bonds_A_WT_Rep2 countOfA_WT_Rep2(i)/
NumFrames];
end
Perc_H_Bond_A_WT_Rep2=[double(H_Bonds_A_WT_Rep2);uniqueA_WT_Rep2];
for i=1:length(uniqueB_WT_Rep2)
    H_Bonds_B_WT_Rep2=[H_Bonds_B_WT_Rep2 countOfB_WT_Rep2(i)/
NumFrames];
end
Perc_H_Bond_B_WT_Rep2=[double(H_Bonds_B_WT_Rep2);uniqueB_WT_Rep2];
```

WT_Rep3

```
WT_Rep3_A=[];
WT_Rep3_B=[];
for i=1:length(data_WT_Rep3{1})
    WT_Rep3_A=[WT_Rep3_A data_WT_Rep3{1}(i)];
    WT_Rep3_B=[WT_Rep3_B data_WT_Rep3{2}(i)];
    C=[WT_Rep3_A;WT_Rep3_B]';
end
Chain_A_WT_Rep3=[];
Chain_B_WT_Rep3=[];
for i=1:length(data_WT_Rep3{1})
    if strcmp(data_WT_Rep3{3}(i),'A') == 1
        Chain_A_WT_Rep3=[Chain_A_WT_Rep3 C(i,2)];
    elseif strcmp(data_WT_Rep3{3}(i),'B') == 1
        Chain_B_WT_Rep3=[Chain_B_WT_Rep3 C(i,2)];
    end
end
uniqueA_WT_Rep3 = unique(Chain_A_WT_Rep3);
uniqueB_WT_Rep3 = unique(Chain_B_WT_Rep3);
countOfA_WT_Rep3 =
 hist(double(Chain_A_WT_Rep3),double(uniqueA_WT_Rep3));
countOfB_WT_Rep3 =
 hist(double(Chain_B_WT_Rep3),double(uniqueB_WT_Rep3));
indexToRepeatedValueA_WT_Rep3 = (countOfA_WT_Rep3~=1);
indexToRepeatedValueB_WT_Rep3 = (countOfB_WT_Rep3~=1);
repeatedValuesA_WT_Rep3 =
 uniqueA_WT_Rep3(indexToRepeatedValueA_WT_Rep3);
repeatedValuesB_WT_Rep3 =
 uniqueB_WT_Rep3(indexToRepeatedValueB_WT_Rep3);
numberOfAppearancesOfRepeatedValuesA_WT_Rep3 =
 countOfA_WT_Rep3(indexToRepeatedValueA_WT_Rep3);
numberOfAppearancesOfRepeatedValuesB_WT_Rep3 =
 countOfB_WT_Rep3(indexToRepeatedValueB_WT_Rep3);

% Percentage of H-Bonds
H_Bonds_A_WT_Rep3=[];
H_Bonds_B_WT_Rep3=[];
```



```
NumFrames=500;
for i=1:length(uniqueA_WT_Rep3)
    H_Bonds_A_WT_Rep3=[H_Bonds_A_WT_Rep3 countOfA_WT_Rep3(i)/
NumFrames];
end
Perc_H_Bond_A_WT_Rep3=[double(H_Bonds_A_WT_Rep3);uniqueA_WT_Rep3];
for i=1:length(uniqueB_WT_Rep3)
    H_Bonds_B_WT_Rep3=[H_Bonds_B_WT_Rep3 countOfB_WT_Rep3(i)/
NumFrames];
end
Perc_H_Bond_B_WT_Rep3=[double(H_Bonds_B_WT_Rep3);uniqueB_WT_Rep3];
```

Average + STD for WT

```
RepWT1sum=(sum(H_Bonds_A_WT_Rep1)+sum(H_Bonds_B_WT_Rep1));
RepWT2sum=(sum(H_Bonds_A_WT_Rep2)+sum(H_Bonds_B_WT_Rep2));
RepWT3sum=(sum(H_Bonds_A_WT_Rep3)+sum(H_Bonds_B_WT_Rep3));
y=[RepWT1sum;RepWT2sum;RepWT3sum];
WTmean=mean(y);
WTstd=std(y);
```

Mut1_Rep1

```
Mut1_Rep1_A=[];
Mut1_Rep1_B=[];
for i=1:length(data_Mut1_Rep1{1})
    Mut1_Rep1_A=[Mut1_Rep1_A data_Mut1_Rep1{1}(i)];
    Mut1_Rep1_B=[Mut1_Rep1_B data_Mut1_Rep1{2}(i)];
    C=[Mut1_Rep1_A;Mut1_Rep1_B]';
end
Chain_A_Mut1_Rep1=[];
Chain_B_Mut1_Rep1=[];
for i=1:length(data_Mut1_Rep1{1})
    if strcmp(data_Mut1_Rep1{3}(i),'A') == 1
        Chain_A_Mut1_Rep1=[Chain_A_Mut1_Rep1 C(i,2)];
    elseif strcmp(data_Mut1_Rep1{3}(i),'B') == 1
        Chain_B_Mut1_Rep1=[Chain_B_Mut1_Rep1 C(i,2)];
    end
end
uniqueA_Mut1_Rep1 = unique(Chain_A_Mut1_Rep1);
uniqueB_Mut1_Rep1 = unique(Chain_B_Mut1_Rep1);
countOfA_Mut1_Rep1 =
 hist(double(Chain_A_Mut1_Rep1),double(uniqueA_Mut1_Rep1));
countOfB_Mut1_Rep1 =
 hist(double(Chain_B_Mut1_Rep1),double(uniqueB_Mut1_Rep1));
indexToRepeatedValueA_Mut1_Rep1 = (countOfA_Mut1_Rep1~=1);
indexToRepeatedValueB_Mut1_Rep1 = (countOfB_Mut1_Rep1~=1);
repeatedValuesA_Mut1_Rep1 =
 uniqueA_Mut1_Rep1(indexToRepeatedValueA_Mut1_Rep1);
repeatedValuesB_Mut1_Rep1 =
 uniqueB_Mut1_Rep1(indexToRepeatedValueB_Mut1_Rep1);
numberOfAppearancesOfRepeatedValuesA_Mut1_Rep1 =
 countOfA_Mut1_Rep1(indexToRepeatedValueA_Mut1_Rep1);
```

```
numberOfAppearancesOfRepeatedValuesB_Mut1_Rep1 =
 countOfB_Mut1_Rep1(indexToRepeatedValueB_Mut1_Rep1);

% Percentage of H-Bonds
H_Bonds_A_Mut1_Rep1=[];
H_Bonds_B_Mut1_Rep1=[];
NumFrames=500;
for i=1:length(uniqueA_Mut1_Rep1)
    H_Bonds_A_Mut1_Rep1=[H_Bonds_A_Mut1_Rep1 countOfA_Mut1_Rep1(i)/
NumFrames];
end
Perc_H_Bond_A_Mut1_Rep1=[double(H_Bonds_A_Mut1_Rep1);uniqueA_Mut1_Rep1];%issue
 wih not displaying decimal
for i=1:length(uniqueB_Mut1_Rep1)
    H_Bonds_B_Mut1_Rep1=[H_Bonds_B_Mut1_Rep1 countOfB_Mut1_Rep1(i)/
NumFrames];
end
Perc_H_Bond_B_Mut1_Rep1=[double(H_Bonds_B_Mut1_Rep1);uniqueB_Mut1_Rep1];%issue
 wih not displaying decimal
```

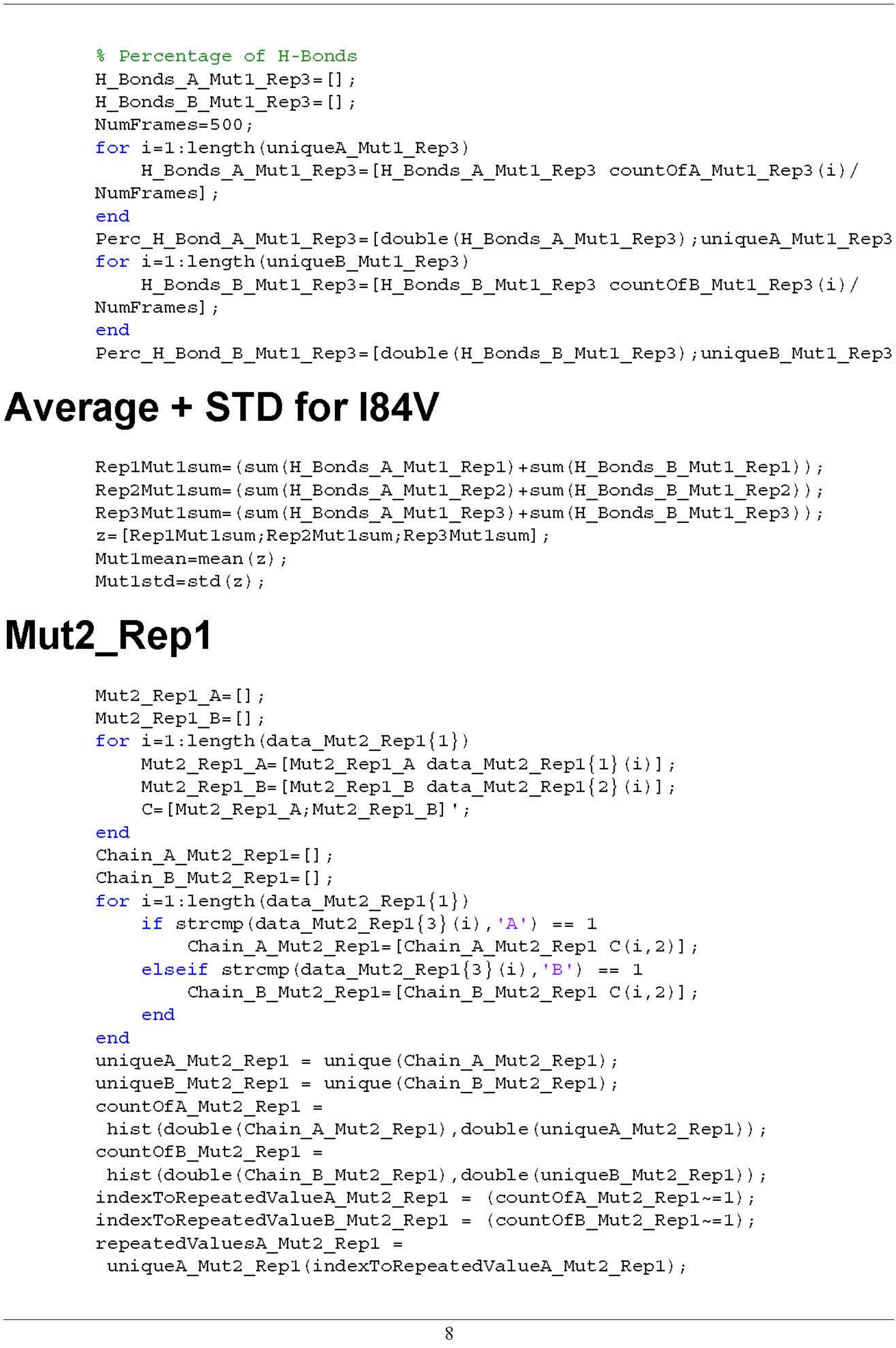
Mut1_Rep2

```
Mut1_Rep2_A=[];
Mut1_Rep2_B=[];
for i=1:length(data_Mut1_Rep2{1})
    Mut1_Rep2_A=[Mut1_Rep2_A data_Mut1_Rep2{1}(i)];
    Mut1_Rep2_B=[Mut1_Rep2_B data_Mut1_Rep2{2}(i)];
    C=[Mut1_Rep2_A;Mut1_Rep2_B]';
end
Chain_A_Mut1_Rep2=[];
Chain_B_Mut1_Rep2=[];
for i=1:length(data_Mut1_Rep2{1})
    if strcmp(data_Mut1_Rep2{3}(i),'A') == 1
        Chain_A_Mut1_Rep2=[Chain_A_Mut1_Rep2 C(i,2)];
    elseif strcmp(data_Mut1_Rep2{3}(i),'B') == 1
        Chain_B_Mut1_Rep2=[Chain_B_Mut1_Rep2 C(i,2)];
    end
end
uniqueA_Mut1_Rep2 = unique(Chain_A_Mut1_Rep2);
uniqueB_Mut1_Rep2 = unique(Chain_B_Mut1_Rep2);
countOfA_Mut1_Rep2 =
 hist(double(Chain_A_Mut1_Rep2),double(uniqueA_Mut1_Rep2));
countOfB_Mut1_Rep2 =
 hist(double(Chain_B_Mut1_Rep2),double(uniqueB_Mut1_Rep2));
indexToRepeatedValueA_Mut1_Rep2 = (countOfA_Mut1_Rep2~=1);
indexToRepeatedValueB_Mut1_Rep2 = (countOfB_Mut1_Rep2~=1);
repeatedValuesA_Mut1_Rep2 =
 uniqueA_Mut1_Rep2(indexToRepeatedValueA_Mut1_Rep2);
repeatedValuesB_Mut1_Rep2 =
 uniqueB_Mut1_Rep2(indexToRepeatedValueB_Mut1_Rep2);
numberOfAppearancesOfRepeatedValuesA_Mut1_Rep2 =
 countOfA_Mut1_Rep2(indexToRepeatedValueA_Mut1_Rep2);
```

```
numberOfAppearancesOfRepeatedValuesB_Mut1_Rep2 =
 countOfB_Mut1_Rep2(indexToRepeatedValueB_Mut1_Rep2);

% Percentage of H-Bonds
H_Bonds_A_Mut1_Rep2=[];
H_Bonds_B_Mut1_Rep2=[];
NumFrames=500;
for i=1:length(uniqueA_Mut1_Rep2)
    H_Bonds_A_Mut1_Rep2=[H_Bonds_A_Mut1_Rep2 countOfA_Mut1_Rep2(i)/
NumFrames];
end
Perc_H_Bond_A_Mut1_Rep2=[double(H_Bonds_A_Mut1_Rep2);uniqueA_Mut1_Rep2];
for i=1:length(uniqueB_Mut1_Rep2)
    H_Bonds_B_Mut1_Rep2=[H_Bonds_B_Mut1_Rep2 countOfB_Mut1_Rep2(i)/
NumFrames];
end
Perc_H_Bond_B_Mut1_Rep2=[double(H_Bonds_B_Mut1_Rep2);uniqueB_Mut1_Rep2];
```

Mut1_Rep3

```
Mut1_Rep3_A=[];
Mut1_Rep3_B=[];
for i=1:length(data_Mut1_Rep3{1})
    Mut1_Rep3_A=[Mut1_Rep3_A data_Mut1_Rep3{1}(i)];
    Mut1_Rep3_B=[Mut1_Rep3_B data_Mut1_Rep3{2}(i)];
    C=[Mut1_Rep3_A;Mut1_Rep3_B]';
end
Chain_A_Mut1_Rep3=[];
Chain_B_Mut1_Rep3=[];
for i=1:length(data_Mut1_Rep3{1})
    if strcmp(data_Mut1_Rep3{3}(i),'A') == 1
        Chain_A_Mut1_Rep3=[Chain_A_Mut1_Rep3 C(i,2)];
    elseif strcmp(data_Mut1_Rep3{3}(i),'B') == 1
        Chain_B_Mut1_Rep3=[Chain_B_Mut1_Rep3 C(i,2)];
    end
end
uniqueA_Mut1_Rep3 = unique(Chain_A_Mut1_Rep3);
uniqueB_Mut1_Rep3 = unique(Chain_B_Mut1_Rep3);
countOfA_Mut1_Rep3 =
 hist(double(Chain_A_Mut1_Rep3),double(uniqueA_Mut1_Rep3));
countOfB_Mut1_Rep3 =
 hist(double(Chain_B_Mut1_Rep3),double(uniqueB_Mut1_Rep3));
indexToRepeatedValueA_Mut1_Rep3 = (countOfA_Mut1_Rep3~=1);
indexToRepeatedValueB_Mut1_Rep3 = (countOfB_Mut1_Rep3~=1);
repeatedValuesA_Mut1_Rep3 =
 uniqueA_Mut1_Rep3(indexToRepeatedValueA_Mut1_Rep3);
repeatedValuesB_Mut1_Rep3 =
 uniqueB_Mut1_Rep3(indexToRepeatedValueB_Mut1_Rep3);
numberOfAppearancesOfRepeatedValuesA_Mut1_Rep3 =
 countOfA_Mut1_Rep3(indexToRepeatedValueA_Mut1_Rep3);
numberOfAppearancesOfRepeatedValuesB_Mut1_Rep3 =
 countOfB_Mut1_Rep3(indexToRepeatedValueB_Mut1_Rep3);
```



```
% Percentage of H-Bonds
H_Bonds_A_Mut1_Rep3=[];
H_Bonds_B_Mut1_Rep3=[];
NumFrames=500;
for i=1:length(uniqueA_Mut1_Rep3)
    H_Bonds_A_Mut1_Rep3=[H_Bonds_A_Mut1_Rep3 countOfA_Mut1_Rep3(i)/
NumFrames];
end
Perc_H_Bond_A_Mut1_Rep3=[double(H_Bonds_A_Mut1_Rep3);uniqueA_Mut1_Rep3];
for i=1:length(uniqueB_Mut1_Rep3)
    H_Bonds_B_Mut1_Rep3=[H_Bonds_B_Mut1_Rep3 countOfB_Mut1_Rep3(i)/
NumFrames];
end
Perc_H_Bond_B_Mut1_Rep3=[double(H_Bonds_B_Mut1_Rep3);uniqueB_Mut1_Rep3];
```

Average + STD for I84V

```
Rep1Mut1sum=(sum(H_Bonds_A_Mut1_Rep1)+sum(H_Bonds_B_Mut1_Rep1));
Rep2Mut1sum=(sum(H_Bonds_A_Mut1_Rep2)+sum(H_Bonds_B_Mut1_Rep2));
Rep3Mut1sum=(sum(H_Bonds_A_Mut1_Rep3)+sum(H_Bonds_B_Mut1_Rep3));
z=[Rep1Mut1sum;Rep2Mut1sum;Rep3Mut1sum];
Mut1mean=mean(z);
Mut1std=std(z);
```

Mut2_Rep1

```
Mut2_Rep1_A=[];
Mut2_Rep1_B=[];
for i=1:length(data_Mut2_Rep1{1})
    Mut2_Rep1_A=[Mut2_Rep1_A data_Mut2_Rep1{1}(i)];
    Mut2_Rep1_B=[Mut2_Rep1_B data_Mut2_Rep1{2}(i)];
    C=[Mut2_Rep1_A;Mut2_Rep1_B]';
end
Chain_A_Mut2_Rep1=[];
Chain_B_Mut2_Rep1=[];
for i=1:length(data_Mut2_Rep1{1})
    if strcmp(data_Mut2_Rep1{3}(i),'A') == 1
        Chain_A_Mut2_Rep1=[Chain_A_Mut2_Rep1 C(i,2)];
    elseif strcmp(data_Mut2_Rep1{3}(i),'B') == 1
        Chain_B_Mut2_Rep1=[Chain_B_Mut2_Rep1 C(i,2)];
    end
end
uniqueA_Mut2_Rep1 = unique(Chain_A_Mut2_Rep1);
uniqueB_Mut2_Rep1 = unique(Chain_B_Mut2_Rep1);
countOfA_Mut2_Rep1 =
 hist(double(Chain_A_Mut2_Rep1),double(uniqueA_Mut2_Rep1));
countOfB_Mut2_Rep1 =
 hist(double(Chain_B_Mut2_Rep1),double(uniqueB_Mut2_Rep1));
indexToRepeatedValueA_Mut2_Rep1 = (countOfA_Mut2_Rep1~=1);
indexToRepeatedValueB_Mut2_Rep1 = (countOfB_Mut2_Rep1~=1);
repeatedValuesA_Mut2_Rep1 =
 uniqueA_Mut2_Rep1(indexToRepeatedValueA_Mut2_Rep1);
```

```
repeatedValuesB_Mut2_Rep1 =
 uniqueB_Mut2_Rep1(indexToRepeatedValueB_Mut2_Rep1);
numberOfAppearancesOfRepeatedValuesA_Mut2_Rep1 =
 countOfA_Mut2_Rep1(indexToRepeatedValueA_Mut2_Rep1);
numberOfAppearancesOfRepeatedValuesB_Mut2_Rep1 =
 countOfB_Mut2_Rep1(indexToRepeatedValueB_Mut2_Rep1);

% Percentage of H-Bonds
H_Bonds_A_Mut2_Rep1=[];
H_Bonds_B_Mut2_Rep1=[];
NumFrames=500;
for i=1:length(uniqueA_Mut2_Rep1)
    H_Bonds_A_Mut2_Rep1=[H_Bonds_A_Mut2_Rep1 countOfA_Mut2_Rep1(i)/
NumFrames];
end
Perc_H_Bond_A_Mut2_Rep1=[double(H_Bonds_A_Mut2_Rep1);uniqueA_Mut2_Rep1];
for i=1:length(uniqueB_Mut2_Rep1)
    H_Bonds_B_Mut2_Rep1=[H_Bonds_B_Mut2_Rep1 countOfB_Mut2_Rep1(i)/
NumFrames];
end
Perc_H_Bond_B_Mut2_Rep1=[double(H_Bonds_B_Mut2_Rep1);uniqueB_Mut2_Rep1];
```

Mut2_Rep3

```
Mut2_Rep3_A=[];
Mut2_Rep3_B=[];
for i=1:length(data_Mut2_Rep3{1})
    Mut2_Rep3_A=[Mut2_Rep3_A data_Mut2_Rep3{1}(i)];
    Mut2_Rep3_B=[Mut2_Rep3_B data_Mut2_Rep3{2}(i)];
    C=[Mut2_Rep3_A;Mut2_Rep3_B]';
end
Chain_A_Mut2_Rep3=[];
Chain_B_Mut2_Rep3=[];
for i=1:length(data_Mut2_Rep3{1})
    if strcmp(data_Mut2_Rep3{3}(i),'A') == 1
        Chain_A_Mut2_Rep3=[Chain_A_Mut2_Rep3 C(i,2)];
    elseif strcmp(data_Mut2_Rep3{3}(i),'B') == 1
        Chain_B_Mut2_Rep3=[Chain_B_Mut2_Rep3 C(i,2)];
    end
end
uniqueA_Mut2_Rep3 = unique(Chain_A_Mut2_Rep3);
uniqueB_Mut2_Rep3 = unique(Chain_B_Mut2_Rep3);
countOfA_Mut2_Rep3 =
 hist(double(Chain_A_Mut2_Rep3),double(uniqueA_Mut2_Rep3));
countOfB_Mut2_Rep3 =
 hist(double(Chain_B_Mut2_Rep3),double(uniqueB_Mut2_Rep3));
indexToRepeatedValueA_Mut2_Rep3 = (countOfA_Mut2_Rep3~=1);
indexToRepeatedValueB_Mut2_Rep3 = (countOfB_Mut2_Rep3~=1);
repeatedValuesA_Mut2_Rep3 =
 uniqueA_Mut2_Rep3(indexToRepeatedValueA_Mut2_Rep3);
repeatedValuesB_Mut2_Rep3 =
 uniqueB_Mut2_Rep3(indexToRepeatedValueB_Mut2_Rep3);
```

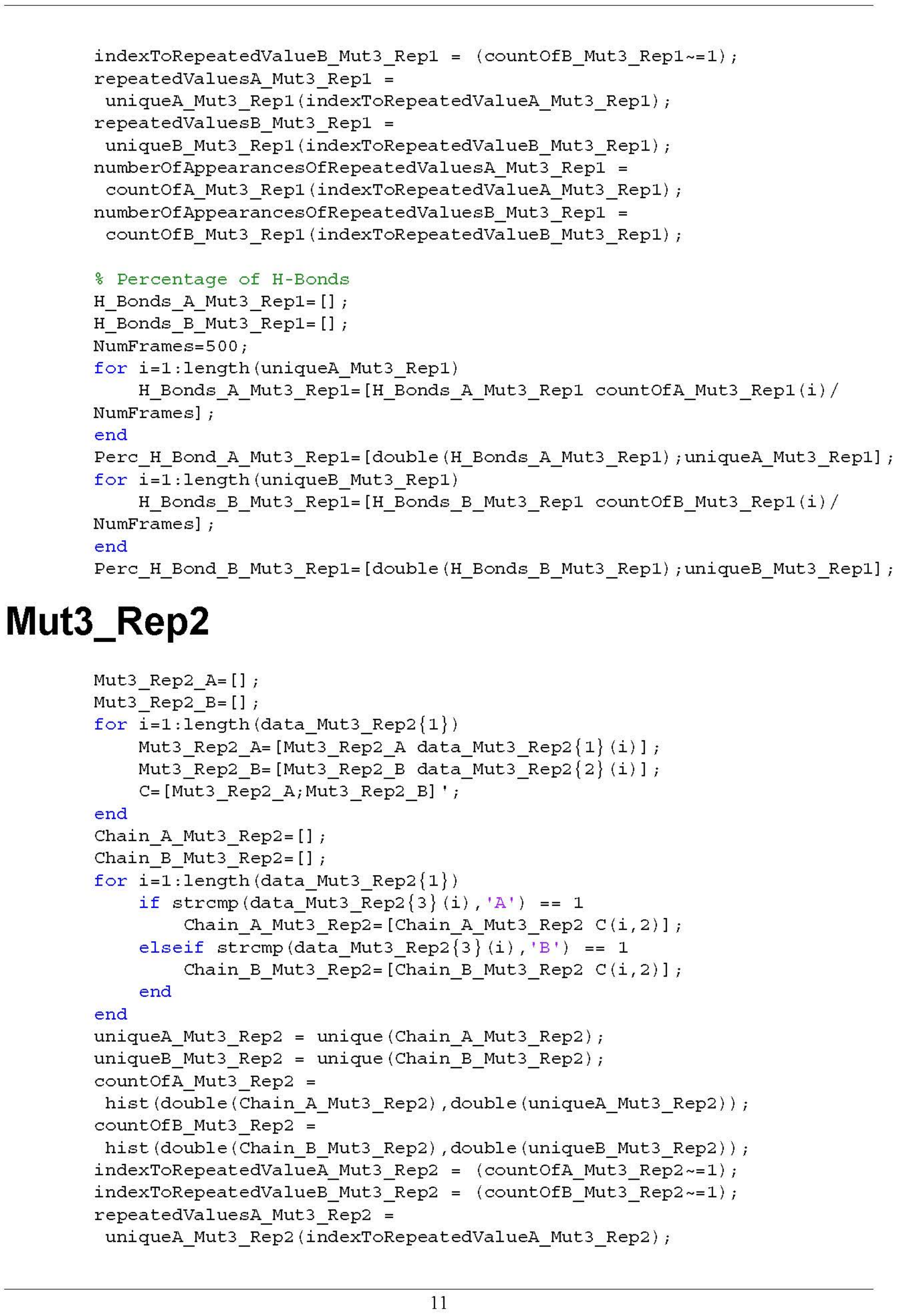
```
numberOfAppearancesOfRepeatedValuesA_Mut2_Rep3 =
 countOfA_Mut2_Rep3(indexToRepeatedValueA_Mut2_Rep3);
numberOfAppearancesOfRepeatedValuesB_Mut2_Rep3 =
 countOfB_Mut2_Rep3(indexToRepeatedValueB_Mut2_Rep3);

% Percentage of H-Bonds
H_Bonds_A_Mut2_Rep3=[];
H_Bonds_B_Mut2_Rep3=[];
NumFrames=500;
for i=1:length(uniqueA_Mut2_Rep3)
    H_Bonds_A_Mut2_Rep3=[H_Bonds_A_Mut2_Rep3 countOfA_Mut2_Rep3(i)/
NumFrames];
end
Perc_H_Bond_A_Mut2_Rep3=[double(H_Bonds_A_Mut2_Rep3);uniqueA_Mut2_Rep3];
for i=1:length(uniqueB_Mut2_Rep3)
    H_Bonds_B_Mut2_Rep3=[H_Bonds_B_Mut2_Rep3 countOfB_Mut2_Rep3(i)/
NumFrames];
end
Perc_H_Bond_B_Mut2_Rep3=[double(H_Bonds_B_Mut2_Rep3);uniqueB_Mut2_Rep3];
```

Average + STD for V82F+I84V

```
Rep1Mut2sum=(sum(H_Bonds_A_Mut2_Rep1)+sum(H_Bonds_B_Mut2_Rep1));
Rep3Mut2sum=(sum(H_Bonds_A_Mut2_Rep3)+sum(H_Bonds_B_Mut2_Rep3));
x=[Rep1Mut2sum;;Rep3Mut2sum];
Mut2mean=mean(x);
Mut2std=std(x);
```

Mut3_Rep1

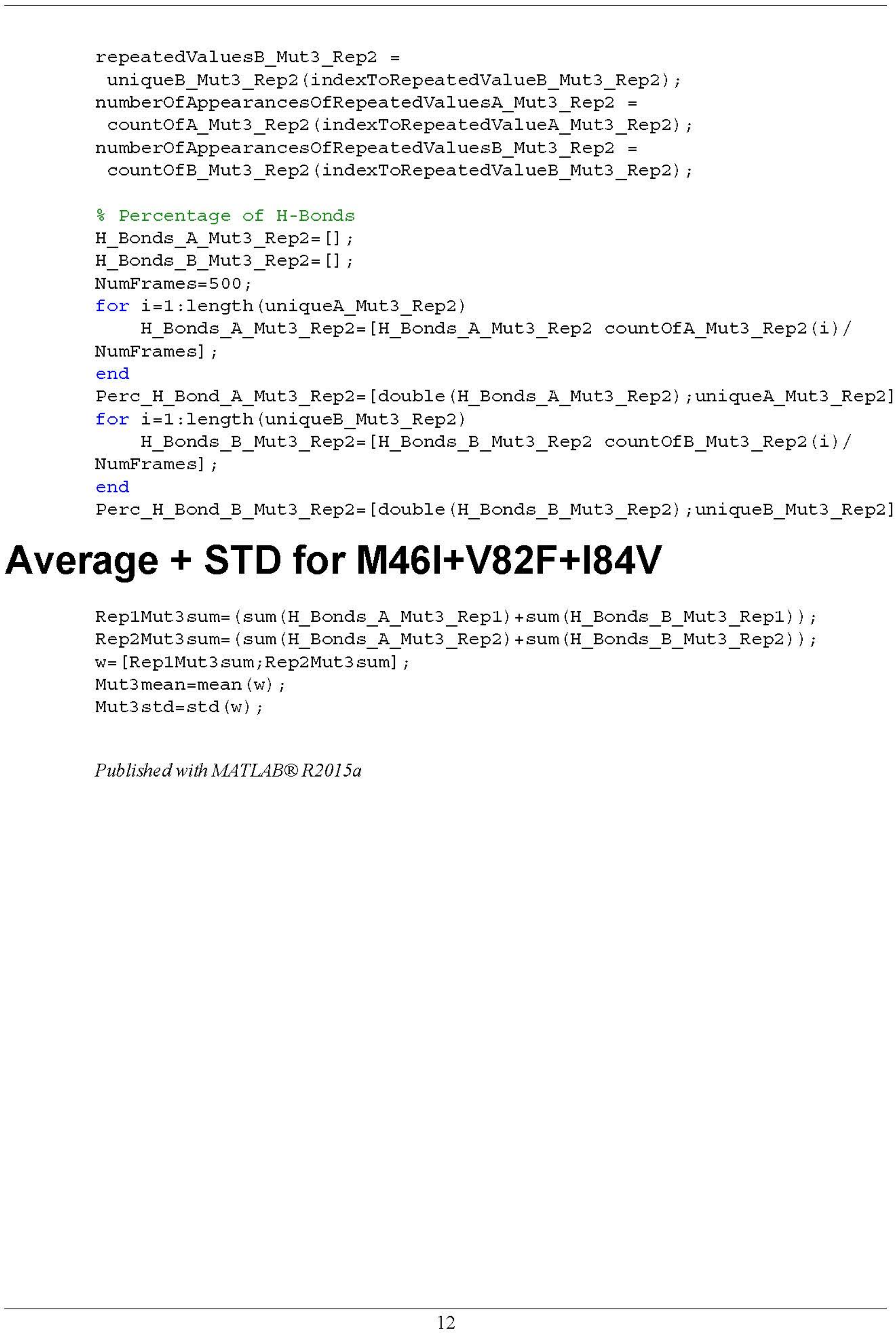
```
Mut3_Rep1_A=[];
Mut3_Rep1_B=[];
for i=1:length(data_Mut3_Rep1{1})
    Mut3_Rep1_A=[Mut3_Rep1_A data_Mut3_Rep1{1}(i)];
    Mut3_Rep1_B=[Mut3_Rep1_B data_Mut3_Rep1{2}(i)];
    C=[Mut3_Rep1_A;Mut3_Rep1_B]';
end
Chain_A_Mut3_Rep1=[];
Chain_B_Mut3_Rep1=[];
for i=1:length(data_Mut3_Rep1{1})
    if strcmp(data_Mut3_Rep1{3}(i),'A') == 1
        Chain_A_Mut3_Rep1=[Chain_A_Mut3_Rep1 C(i,2)];
    elseif strcmp(data_Mut3_Rep1{3}(i),'B') == 1
        Chain_B_Mut3_Rep1=[Chain_B_Mut3_Rep1 C(i,2)];
    end
end
uniqueA_Mut3_Rep1 = unique(Chain_A_Mut3_Rep1);
uniqueB_Mut3_Rep1 = unique(Chain_B_Mut3_Rep1);
countOfA_Mut3_Rep1 =
 hist(double(Chain_A_Mut3_Rep1),double(uniqueA_Mut3_Rep1));
countOfB_Mut3_Rep1 =
 hist(double(Chain_B_Mut3_Rep1),double(uniqueB_Mut3_Rep1));
indexToRepeatedValueA_Mut3_Rep1 = (countOfA_Mut3_Rep1~=1);
```



```
indexToRepeatedValueB_Mut3_Rep1 = (countOfB_Mut3_Rep1~=1);
repeatedValuesA_Mut3_Rep1 =
 uniqueA_Mut3_Rep1(indexToRepeatedValueA_Mut3_Rep1);
repeatedValuesB_Mut3_Rep1 =
 uniqueB_Mut3_Rep1(indexToRepeatedValueB_Mut3_Rep1);
numberOfAppearancesOfRepeatedValuesA_Mut3_Rep1 =
 countOfA_Mut3_Rep1(indexToRepeatedValueA_Mut3_Rep1);
numberOfAppearancesOfRepeatedValuesB_Mut3_Rep1 =
 countOfB_Mut3_Rep1(indexToRepeatedValueB_Mut3_Rep1);

% Percentage of H-Bonds
H_Bonds_A_Mut3_Rep1=[];
H_Bonds_B_Mut3_Rep1=[];
NumFrames=500;
for i=1:length(uniqueA_Mut3_Rep1)
    H_Bonds_A_Mut3_Rep1=[H_Bonds_A_Mut3_Rep1 countOfA_Mut3_Rep1(i)/
NumFrames];
end
Perc_H_Bond_A_Mut3_Rep1=[double(H_Bonds_A_Mut3_Rep1);uniqueA_Mut3_Rep1];
for i=1:length(uniqueB_Mut3_Rep1)
    H_Bonds_B_Mut3_Rep1=[H_Bonds_B_Mut3_Rep1 countOfB_Mut3_Rep1(i)/
NumFrames];
end
Perc_H_Bond_B_Mut3_Rep1=[double(H_Bonds_B_Mut3_Rep1);uniqueB_Mut3_Rep1];
```

Mut3_Rep2

```
Mut3_Rep2_A=[];
Mut3_Rep2_B=[];
for i=1:length(data_Mut3_Rep2{1})
    Mut3_Rep2_A=[Mut3_Rep2_A data_Mut3_Rep2{1}(i)];
    Mut3_Rep2_B=[Mut3_Rep2_B data_Mut3_Rep2{2}(i)];
    C=[Mut3_Rep2_A;Mut3_Rep2_B]';
end
Chain_A_Mut3_Rep2=[];
Chain_B_Mut3_Rep2=[];
for i=1:length(data_Mut3_Rep2{1})
    if strcmp(data_Mut3_Rep2{3}(i),'A') == 1
        Chain_A_Mut3_Rep2=[Chain_A_Mut3_Rep2 C(i,2)];
    elseif strcmp(data_Mut3_Rep2{3}(i),'B') == 1
        Chain_B_Mut3_Rep2=[Chain_B_Mut3_Rep2 C(i,2)];
    end
end
uniqueA_Mut3_Rep2 = unique(Chain_A_Mut3_Rep2);
uniqueB_Mut3_Rep2 = unique(Chain_B_Mut3_Rep2);
countOfA_Mut3_Rep2 =
 hist(double(Chain_A_Mut3_Rep2),double(uniqueA_Mut3_Rep2));
countOfB_Mut3_Rep2 =
 hist(double(Chain_B_Mut3_Rep2),double(uniqueB_Mut3_Rep2));
indexToRepeatedValueA_Mut3_Rep2 = (countOfA_Mut3_Rep2~=1);
indexToRepeatedValueB_Mut3_Rep2 = (countOfB_Mut3_Rep2~=1);
repeatedValuesA_Mut3_Rep2 =
 uniqueA_Mut3_Rep2(indexToRepeatedValueA_Mut3_Rep2);
```



```
repeatedValuesB_Mut3_Rep2 =
 uniqueB_Mut3_Rep2(indexToRepeatedValueB_Mut3_Rep2);
numberOfAppearancesOfRepeatedValuesA_Mut3_Rep2 =
 countOfA_Mut3_Rep2(indexToRepeatedValueA_Mut3_Rep2);
numberOfAppearancesOfRepeatedValuesB_Mut3_Rep2 =
 countOfB_Mut3_Rep2(indexToRepeatedValueB_Mut3_Rep2);

% Percentage of H-Bonds
H_Bonds_A_Mut3_Rep2=[];
H_Bonds_B_Mut3_Rep2=[];
NumFrames=500;
for i=1:length(uniqueA_Mut3_Rep2)
    H_Bonds_A_Mut3_Rep2=[H_Bonds_A_Mut3_Rep2 countOfA_Mut3_Rep2(i)/
NumFrames];
end
Perc_H_Bond_A_Mut3_Rep2=[double(H_Bonds_A_Mut3_Rep2);uniqueA_Mut3_Rep2];
for i=1:length(uniqueB_Mut3_Rep2)
    H_Bonds_B_Mut3_Rep2=[H_Bonds_B_Mut3_Rep2 countOfB_Mut3_Rep2(i)/
NumFrames];
end
Perc_H_Bond_B_Mut3_Rep2=[double(H_Bonds_B_Mut3_Rep2);uniqueB_Mut3_Rep2];
```

Average + STD for M46I+V82F+I84V

```
Rep1Mut3sum=(sum(H_Bonds_A_Mut3_Rep1)+sum(H_Bonds_B_Mut3_Rep1));
Rep2Mut3sum=(sum(H_Bonds_A_Mut3_Rep2)+sum(H_Bonds_B_Mut3_Rep2));
w=[Rep1Mut3sum;Rep2Mut3sum];
Mut3mean=mean(w);
Mut3std=std(w);
```

Published with MATLAB® R2015a

B.6 Alpha-Carbon Distances Script

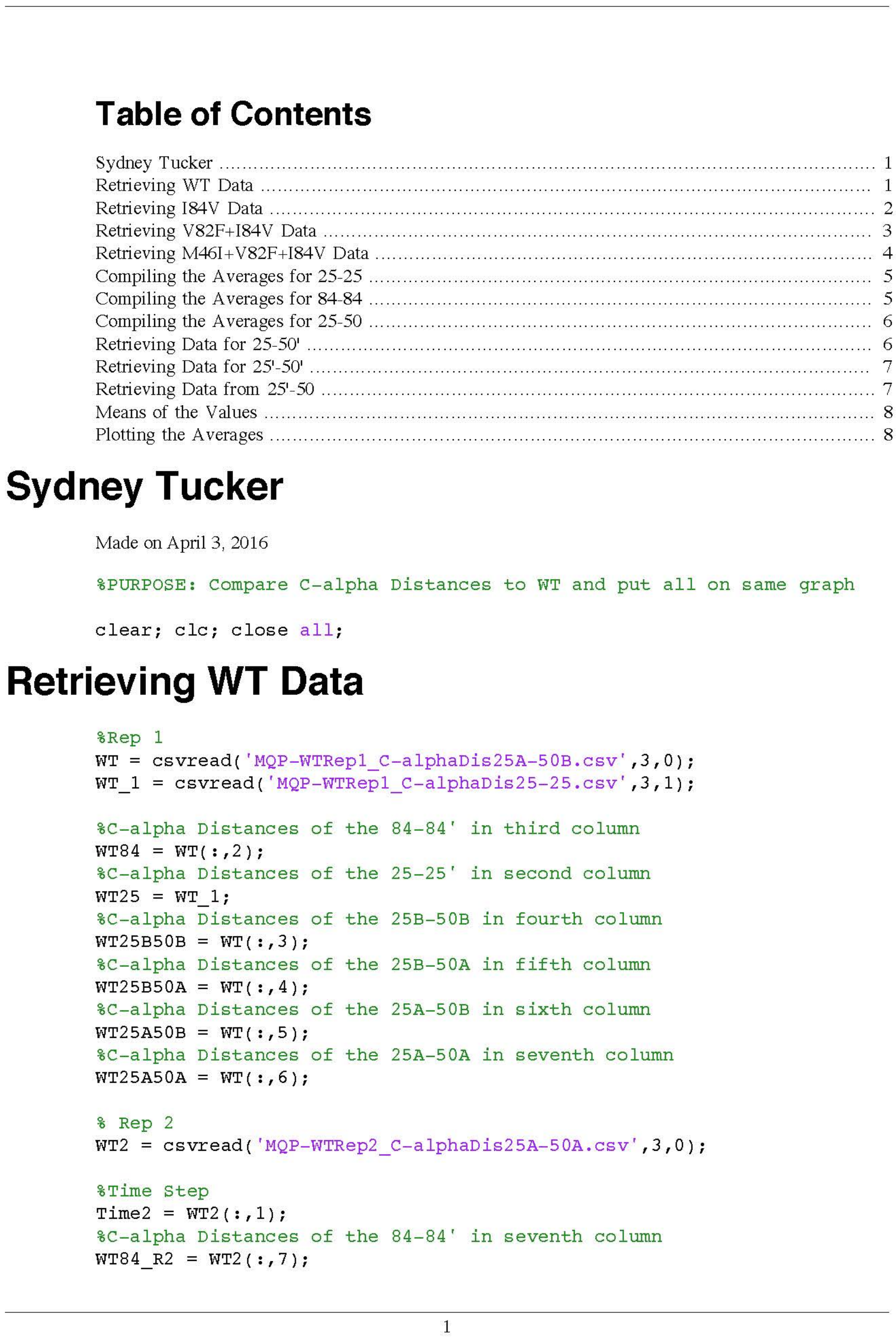


Table of Contents

Sydney Tucker

Made on April 3, 2016

```
%PURPOSE: Compare C-alpha Distances to WT and put all on same graph

clear; clc; close all;
```

Retrieving WT Data

```
%Rep 1
WT = csvread('MQP-WTRep1_C-alphaDis25A-50B.csv',3,0);
WT_1 = csvread('MQP-WTRep1_C-alphaDis25-25.csv',3,1);

%C-alpha Distances of the 84-84' in third column
WT84 = WT(:,2);
%C-alpha Distances of the 25-25' in second column
WT25 = WT_1;
%C-alpha Distances of the 25B-50B in fourth column
WT25B50B = WT(:,3);
%C-alpha Distances of the 25B-50A in fifth column
WT25B50A = WT(:,4);
%C-alpha Distances of the 25A-50B in sixth column
WT25A50B = WT(:,5);
%C-alpha Distances of the 25A-50A in seventh column
WT25A50A = WT(:,6);

% Rep 2
WT2 = csvread('MQP-WTRep2_C-alphaDis25A-50A.csv',3,0);

%Time Step
Time2 = WT2(:,1);
%C-alpha Distances of the 84-84' in seventh column
WT84_R2 = WT2(:,7);
```

```
%C-alpha Distances of the 25-25' in second column
WT25_R2 = WT2(:,2);
%C-alpha Distances of the 25B-50B in third column
WT25B50B_R2 = WT2(:,3);
%C-alpha Distances of the 25B-50A in fourth column
WT25B50A_R2 = WT2(:,4);
%C-alpha Distances of the 25A-50B in fifth column
WT25A50B_R2 = WT2(:,5);
%C-alpha Distances of the 25A-50A in sixth column
WT25A50A_R2 = WT2(:,6);

%Rep 3
WT3 = csvread('MQP-WTRep3_C-alphaDis25A-50A.csv',3,0);
%Time Step
I84V3 = WT3(:,1);
%C-alpha Distances of the 84-84' in third column
WT84_R3 = WT3(:,3);
%C-alpha Distances of the 25-25' in second column
WT25_R3 = WT3(:,2);
%C-alpha Distances of the 25B-50B in fourth column
WT25B50B_R3 = WT3(:,4);
%C-alpha Distances of the 25B-50A in fifth column
WT25B50A_R3 = WT3(:,5);
%C-alpha Distances of the 25A-50B in sixth column
WT25A50B_R3 = WT3(:,6);
%C-alpha Distances of the 25A-50A in seventh column
WT25A50A_R3 = WT3(:,7);
```

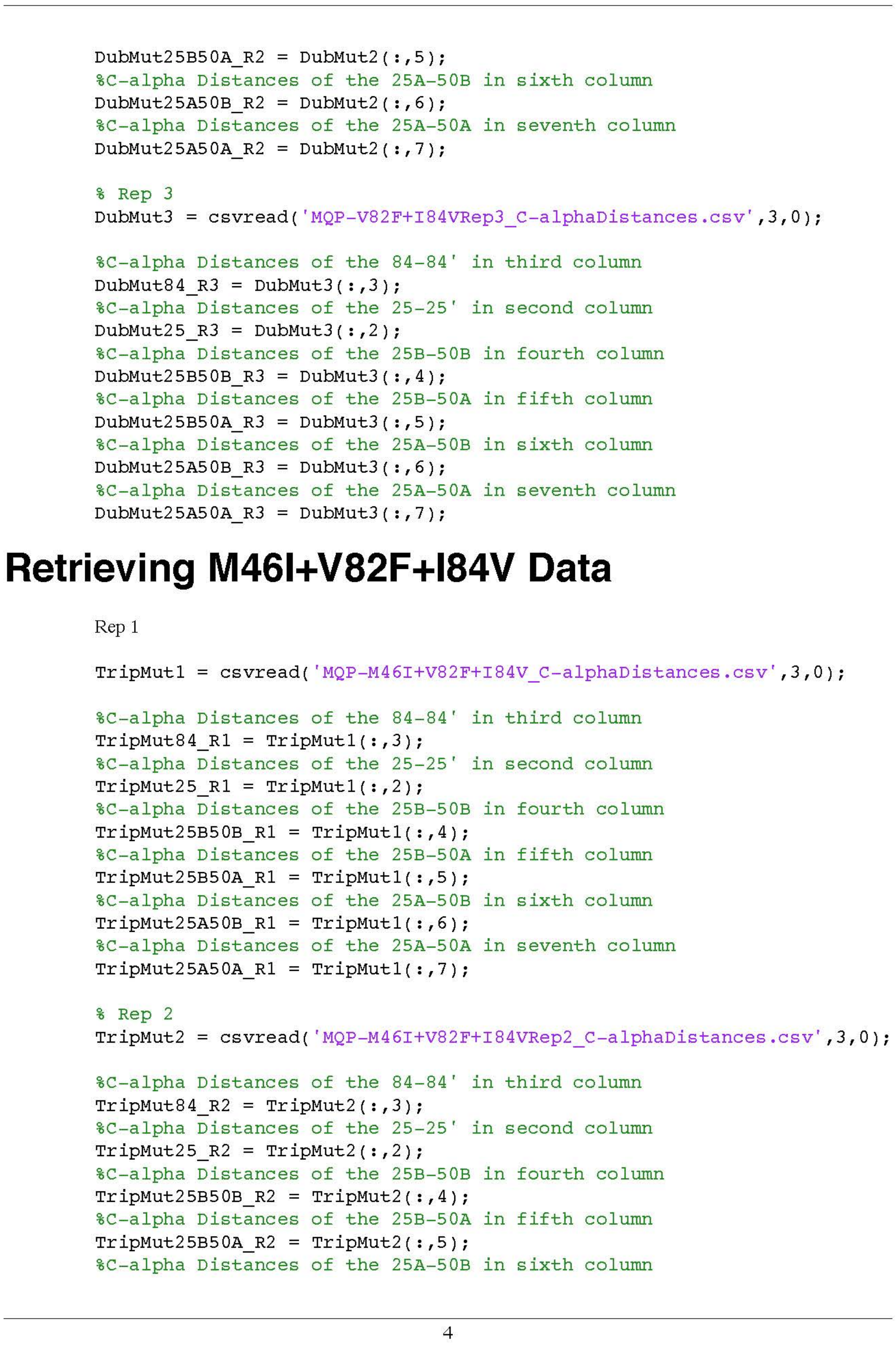
Retrieving I84V Data

```
%Rep 1
I84V = csvread('MQP-I84VRep1_C-alphaDistances.csv',3,0);

%C-alpha Distances of the 84-84' in third column
I84V84_R1 = I84V(:,3);
%C-alpha Distances of the 25-25' in second column
I84V25_R1 = I84V(:,2);
%C-alpha Distances of the 25B-50B in fourth column
I84V25B50B_R1 = I84V(:,4);
%C-alpha Distances of the 25B-50A in fifth column
I84V25B50A_R1 = I84V(:,5);
%C-alpha Distances of the 25A-50B in sixth column
I84V25A50B_R1 = I84V(:,6);
%C-alpha Distances of the 25A-50A in seventh column
I84V25A50A_R1 = I84V(:,7);

% Rep 2
I84V2 = csvread('MQP-I84VRep2_C-alphaDistances.csv',3,0);

%C-alpha Distances of the 84-84' in third column
I84V84_R2 = I84V2(:,3);
%C-alpha Distances of the 25-25' in second column
I84V25_R2 = I84V2(:,2);
```

```
%C-alpha Distances of the 25B-50B in fourth column
I84V25B50B_R2 = I84V2(:,4);
%C-alpha Distances of the 25B-50A in fifth column
I84V25B50A_R2 = I84V2(:,5);
%C-alpha Distances of the 25A-50B in sixth column
I84V25A50B_R2 = I84V2(:,6);
%C-alpha Distances of the 25A-50A in seventh column
I84V25A50A_R2 = I84V2(:,7);

%Rep 3
I84V3 = csvread('MQP-I84VRep3_C-alphaDistances.csv',3,0);

%C-alpha Distances of the 84-84' in third column
I84V84_R3 = I84V3(:,3);
%C-alpha Distances of the 25-25' in second column
I84V25_R3 = I84V3(:,2);
%C-alpha Distances of the 25B-50B in fourth column
I84V25B50B_R3 = I84V3(:,4);
%C-alpha Distances of the 25B-50A in fifth column
I84V25B50A_R3 = I84V3(:,5);
%C-alpha Distances of the 25A-50B in sixth column
I84V25A50B_R3 = I84V3(:,6);
%C-alpha Distances of the 25A-50A in seventh column
I84V25A50A_R3 = I84V3(:,7);
```

Retrieving V82F+I84V Data

```
%Rep 1
DubMut1 = csvread('MQP-V82F+I84V_C-alphaDistances.csv',3,0);

%C-alpha Distances of the 84-84' in third column
DubMut84_R1 = DubMut1(:,3);
%C-alpha Distances of the 25-25' in second column
DubMut25_R1 = DubMut1(:,2);
%C-alpha Distances of the 25B-50B in fourth column
DubMut25B50B_R1 = DubMut1(:,4);
%C-alpha Distances of the 25B-50A in fifth column
DubMut25B50A_R1 = DubMut1(:,5);
%C-alpha Distances of the 25A-50B in sixth column
DubMut25A50B_R1 = DubMut1(:,6);
%C-alpha Distances of the 25A-50A in seventh column
DubMut25A50A_R1 = DubMut1(:,7);

% Rep 2
DubMut2 = csvread('MQP-V82F+I84VRep2_C-alphaDistances.csv',3,0);

%C-alpha Distances of the 84-84' in third column
DubMut84_R2 = DubMut2(:,3);
%C-alpha Distances of the 25-25' in second column
DubMut25_R2 = DubMut2(:,2);
%C-alpha Distances of the 25B-50B in fourth column
DubMut25B50B_R2 = DubMut2(:,4);
%C-alpha Distances of the 25B-50A in fifth column
```



```
DubMut25B50A_R2 = DubMut2(:,5);
%C-alpha Distances of the 25A-50B in sixth column
DubMut25A50B_R2 = DubMut2(:,6);
%C-alpha Distances of the 25A-50A in seventh column
DubMut25A50A_R2 = DubMut2(:,7);

% Rep 3
DubMut3 = csvread('MQP-V82F+I84VRep3_C-alphaDistances.csv',3,0);

%C-alpha Distances of the 84-84' in third column
DubMut84_R3 = DubMut3(:,3);
%C-alpha Distances of the 25-25' in second column
DubMut25_R3 = DubMut3(:,2);
%C-alpha Distances of the 25B-50B in fourth column
DubMut25B50B_R3 = DubMut3(:,4);
%C-alpha Distances of the 25B-50A in fifth column
DubMut25B50A_R3 = DubMut3(:,5);
%C-alpha Distances of the 25A-50B in sixth column
DubMut25A50B_R3 = DubMut3(:,6);
%C-alpha Distances of the 25A-50A in seventh column
DubMut25A50A_R3 = DubMut3(:,7);
```

Retrieving M46I+V82F+I84V Data

Rep 1

```
TripMut1 = csvread('MQP-M46I+V82F+I84V_C-alphaDistances.csv',3,0);

%C-alpha Distances of the 84-84' in third column
TripMut84_R1 = TripMut1(:,3);
%C-alpha Distances of the 25-25' in second column
TripMut25_R1 = TripMut1(:,2);
%C-alpha Distances of the 25B-50B in fourth column
TripMut25B50B_R1 = TripMut1(:,4);
%C-alpha Distances of the 25B-50A in fifth column
TripMut25B50A_R1 = TripMut1(:,5);
%C-alpha Distances of the 25A-50B in sixth column
TripMut25A50B_R1 = TripMut1(:,6);
%C-alpha Distances of the 25A-50A in seventh column
TripMut25A50A_R1 = TripMut1(:,7);

% Rep 2
TripMut2 = csvread('MQP-M46I+V82F+I84VRep2_C-alphaDistances.csv',3,0);

%C-alpha Distances of the 84-84' in third column
TripMut84_R2 = TripMut2(:,3);
%C-alpha Distances of the 25-25' in second column
TripMut25_R2 = TripMut2(:,2);
%C-alpha Distances of the 25B-50B in fourth column
TripMut25B50B_R2 = TripMut2(:,4);
%C-alpha Distances of the 25B-50A in fifth column
TripMut25B50A_R2 = TripMut2(:,5);
%C-alpha Distances of the 25A-50B in sixth column
```

```
TripMut25A50B_R2 = TripMut2(:,6);
%C-alpha Distances of the 25A-50A in seventh column
TripMut25A50A_R2 = TripMut2(:,7);
```

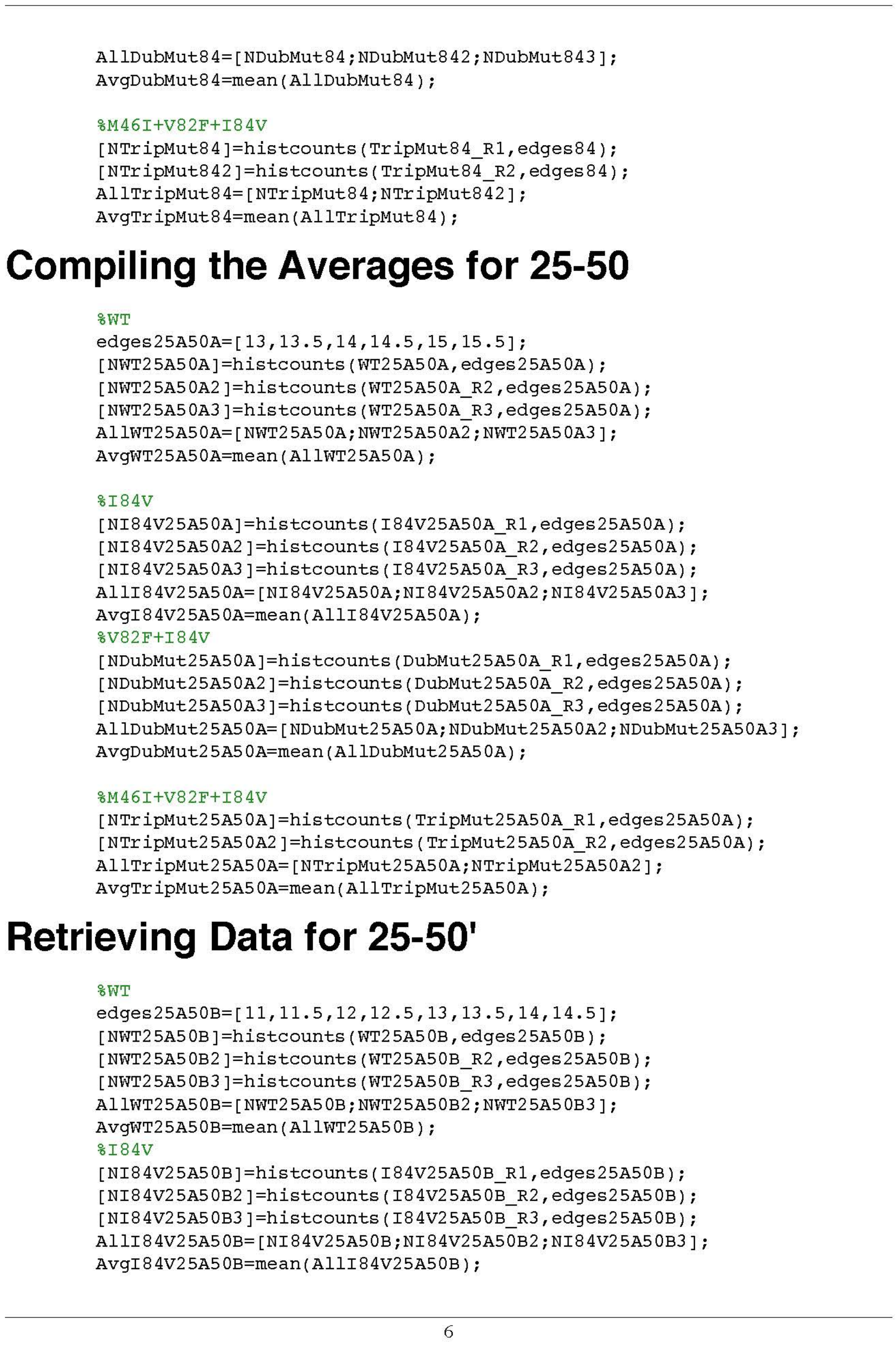
Compiling the Averages for 25-25

WT

```
edges25=[5,5.5,6,6.5,7,7.5];
[NWT25]=histcounts(WT25,edges25);
[NWT252]=histcounts(WT25_R2,edges25);
[NWT253]=histcounts(WT25_R3,edges25);
AllWT25=[NWT25;NWT252;NWT253];
AvgWT25=mean(AllWT25);
%I84V
[NI84V25]=histcounts(I84V25_R1,edges25);
[NI84V252]=histcounts(I84V25_R2,edges25);
[NI84V253]=histcounts(I84V25_R3,edges25);
AllI84V25=[NI84V25;NI84V252;NI84V253];
AvgI84V25=mean(AllI84V25);
%V82F+I84V
[NDubMut25]=histcounts(DubMut25_R1,edges25);
[NDubMut252]=histcounts(DubMut25_R2,edges25);
[NDubMut253]=histcounts(DubMut25_R3,edges25);
AllDubMut25=[NDubMut25;NDubMut252;NDubMut253];
AvgDubMut25=mean(AllDubMut25);
%M46I+V82F+I84V
[NTripMut25]=histcounts(TripMut25_R1,edges25);
[NTripMut252]=histcounts(TripMut25_R2,edges25);
AllTripMut25=[NTripMut25;NTripMut252];
AvgTripMut25=mean(AllTripMut25);
```

Compiling the Averages for 84-84

```
%WT
edges84=[14,14.5,15,15.5,16,16.5,17,17.5];
[NWT84]=histcounts(WT84,edges84);
[NWT842]=histcounts(WT84_R2,edges84);
[NWT843]=histcounts(WT84_R3,edges84);
AllWT84=[NWT84;NWT842;NWT843];
AvgWT84=mean(AllWT84);

%I84V
[NI84V84]=histcounts(I84V84_R1,edges84);
[NI84V842]=histcounts(I84V84_R2,edges84);
[NI84V843]=histcounts(I84V84_R3,edges84);
AllI84V84=[NI84V84;NI84V842;NI84V843];
AvgI84V84=mean(AllI84V84);

%V82F+I84V
[NDubMut84]=histcounts(DubMut84_R1,edges84);
[NDubMut842]=histcounts(DubMut84_R2,edges84);
[NDubMut843]=histcounts(DubMut84_R3,edges84);
```



```
AllDubMut84=[NDubMut84;NDubMut842;NDubMut843];
AvgDubMut84=mean(AllDubMut84);

%M46I+V82F+I84V
[NTripMut84]=histcounts(TripMut84_R1,edges84);
[NTripMut842]=histcounts(TripMut84_R2,edges84);
AllTripMut84=[NTripMut84;NTripMut842];
AvgTripMut84=mean(AllTripMut84);
```

Compiling the Averages for 25-50

```
%WT
edges25A50A=[13,13.5,14,14.5,15,15.5];
[NWT25A50A]=histcounts(WT25A50A,edges25A50A);
[NWT25A50A2]=histcounts(WT25A50A_R2,edges25A50A);
[NWT25A50A3]=histcounts(WT25A50A_R3,edges25A50A);
AllWT25A50A=[NWT25A50A;NWT25A50A2;NWT25A50A3];
AvgWT25A50A=mean(AllWT25A50A);

%I84V
[NI84V25A50A]=histcounts(I84V25A50A_R1,edges25A50A);
[NI84V25A50A2]=histcounts(I84V25A50A_R2,edges25A50A);
[NI84V25A50A3]=histcounts(I84V25A50A_R3,edges25A50A);
AllI84V25A50A=[NI84V25A50A;NI84V25A50A2;NI84V25A50A3];
AvgI84V25A50A=mean(AllI84V25A50A);
%V82F+I84V
[NDubMut25A50A]=histcounts(DubMut25A50A_R1,edges25A50A);
[NDubMut25A50A2]=histcounts(DubMut25A50A_R2,edges25A50A);
[NDubMut25A50A3]=histcounts(DubMut25A50A_R3,edges25A50A);
AllDubMut25A50A=[NDubMut25A50A;NDubMut25A50A2;NDubMut25A50A3];
AvgDubMut25A50A=mean(AllDubMut25A50A);

%M46I+V82F+I84V
[NTripMut25A50A]=histcounts(TripMut25A50A_R1,edges25A50A);
[NTripMut25A50A2]=histcounts(TripMut25A50A_R2,edges25A50A);
AllTripMut25A50A=[NTripMut25A50A;NTripMut25A50A2];
AvgTripMut25A50A=mean(AllTripMut25A50A);
```

Retrieving Data for 25-50'

```
%WT
edges25A50B=[11,11.5,12,12.5,13,13.5,14,14.5];
[NWT25A50B]=histcounts(WT25A50B,edges25A50B);
[NWT25A50B2]=histcounts(WT25A50B_R2,edges25A50B);
[NWT25A50B3]=histcounts(WT25A50B_R3,edges25A50B);
AllWT25A50B=[NWT25A50B;NWT25A50B2;NWT25A50B3];
AvgWT25A50B=mean(AllWT25A50B);
%I84V
[NI84V25A50B]=histcounts(I84V25A50B_R1,edges25A50B);
[NI84V25A50B2]=histcounts(I84V25A50B_R2,edges25A50B);
[NI84V25A50B3]=histcounts(I84V25A50B_R3,edges25A50B);
AllI84V25A50B=[NI84V25A50B;NI84V25A50B2;NI84V25A50B3];
AvgI84V25A50B=mean(AllI84V25A50B);
```

```
%V82F+I84V
[NDubMut25A50B]=histcounts(DubMut25A50B_R1,edges25A50B);
[NDubMut25A50B2]=histcounts(DubMut25A50B_R2,edges25A50B);
[NDubMut25A50B3]=histcounts(DubMut25A50B_R3,edges25A50B);
AllDubMut25A50B=[NDubMut25A50B;NDubMut25A50B2;NDubMut25A50B3];
AvgDubMut25A50B=mean(AllDubMut25A50B);

%M46I+V82F+I84V
[NTripMut25A50B]=histcounts(TripMut25A50B_R1,edges25A50B);
[NTripMut25A50B2]=histcounts(TripMut25A50B_R2,edges25A50B);
AllTripMut25A50B=[NTripMut25A50B;NTripMut25A50B2];
AvgTripMut25A50B=mean(AllTripMut25A50B);
```

Retrieving Data for 25'-50'

```
%WT
edges25B50B=[13,13.5,14,14.5,15,15.5,16];
[NWT25B50B]=histcounts(WT25B50B,edges25B50B);
[NWT25B50B2]=histcounts(WT25B50B_R2,edges25B50B);
[NWT25B50B3]=histcounts(WT25B50B_R3,edges25B50B);
AllWT25B50B=[NWT25B50B;NWT25B50B2;NWT25B50B3];
AvgWT25B50B=mean(AllWT25B50B);

%I84V
[NI84V25B50B]=histcounts(I84V25B50B_R1,edges25B50B);
[NI84V25B50B2]=histcounts(I84V25B50B_R2,edges25B50B);
[NI84V25B50B3]=histcounts(I84V25B50B_R3,edges25B50B);
AllI84V25B50B=[NI84V25B50B;NI84V25B50B2;NI84V25B50B3];
AvgI84V25B50B=mean(AllI84V25B50B);

%V82F+I84V
[NDubMut25B50B]=histcounts(DubMut25B50B_R1,edges25B50B);
[NDubMut25B50B2]=histcounts(DubMut25B50B_R2,edges25B50B);
[NDubMut25B50B3]=histcounts(DubMut25B50B_R3,edges25B50B);
AllDubMut25B50B=[NDubMut25B50B;NDubMut25B50B2;NDubMut25B50B3];
AvgDubMut25B50B=mean(AllDubMut25B50B);

%M46I+V82F+I84V
[NTripMut25B50B]=histcounts(TripMut25B50B_R1,edges25B50B);
[NTripMut25B50B2]=histcounts(TripMut25B50B_R2,edges25B50B);
AllTripMut25B50B=[NTripMut25B50B;NTripMut25B50B2];
AvgTripMut25B50B=mean(AllTripMut25B50B);
```

Retrieving Data from 25'-50

```
%WT
edges25B50A=[11.5,12,12.5,13,13.5,14,14.5];
[NWT25B50A]=histcounts(WT25B50A,edges25B50A);
[NWT25B50A2]=histcounts(WT25B50A_R2,edges25B50A);
[NWT25B50A3]=histcounts(WT25B50A_R3,edges25B50A);
AllWT25B50A=[NWT25B50A;NWT25B50A2;NWT25B50A3];
AvgWT25B50A=mean(AllWT25B50A);
```

```
%I84V
[NI84V25B50A]=histcounts(I84V25B50A_R1,edges25B50A);
[NI84V25B50A2]=histcounts(I84V25B50A_R2,edges25B50A);
[NI84V25B50A3]=histcounts(I84V25B50A_R3,edges25B50A);
AllI84V25B50A=[NI84V25B50A;NI84V25B50A2;NI84V25B50A3];
AvgI84V25B50A=mean(AllI84V25B50A);
%V82F+I84V
[NDubMut25B50A]=histcounts(DubMut25B50A_R1,edges25B50A);
[NDubMut25B50A2]=histcounts(DubMut25B50A_R2,edges25B50A);
[NDubMut25B50A3]=histcounts(DubMut25B50A_R3,edges25B50A);
AllDubMut25B50A=[NDubMut25B50A;NDubMut25B50A2;NDubMut25B50A3];
AvgDubMut25B50A=mean(AllDubMut25B50A);
%M46I+V82F+I84V
[NTripMut25B50A]=histcounts(TripMut25B50A_R1,edges25B50A);
[NTripMut25B50A2]=histcounts(TripMut25B50A_R2,edges25B50A);
AllTripMut25B50A=[NTripMut25B50A;NTripMut25B50A2];
AvgTripMut25B50A=mean(AllTripMut25B50A);
```

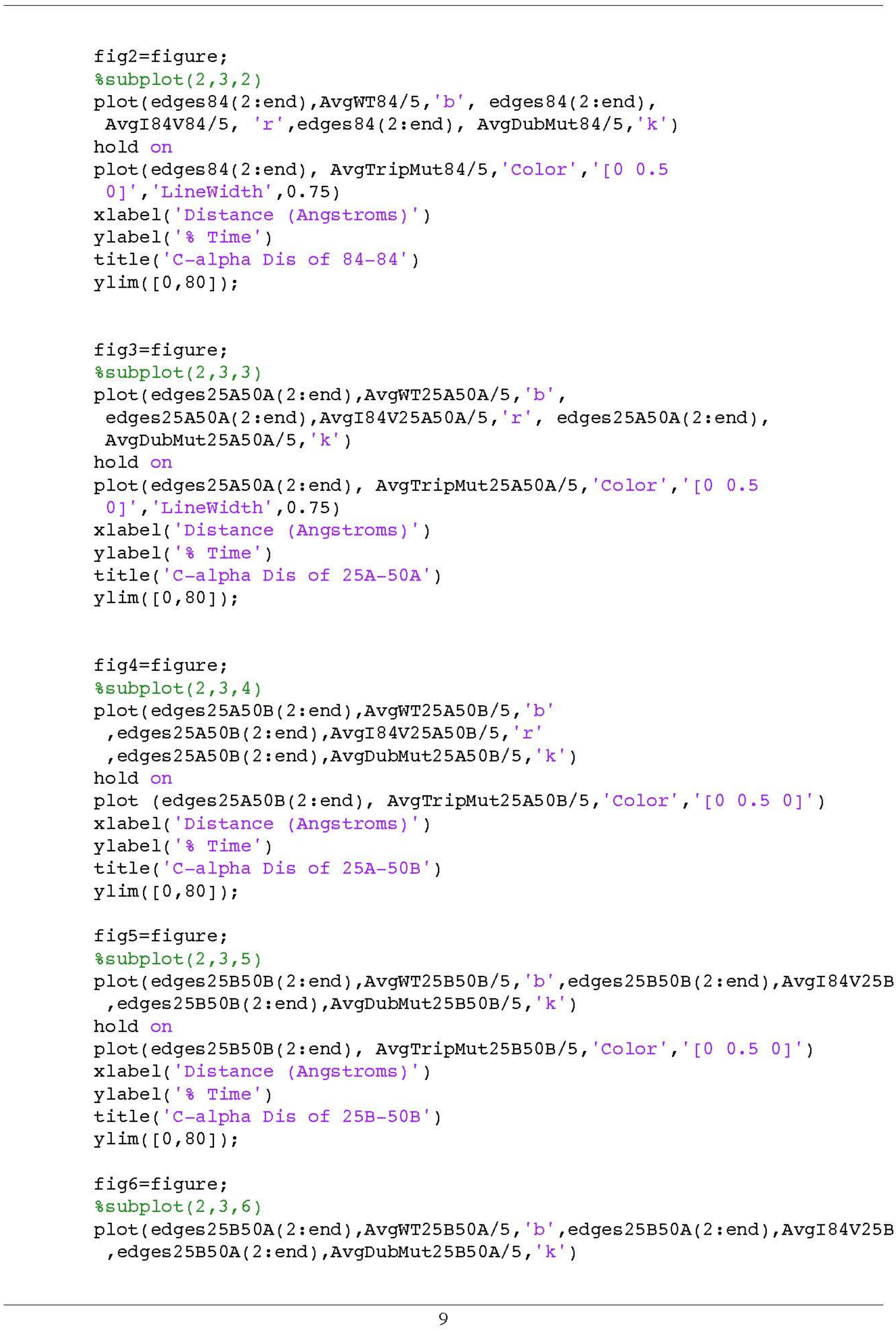
Means of the Values

25-25

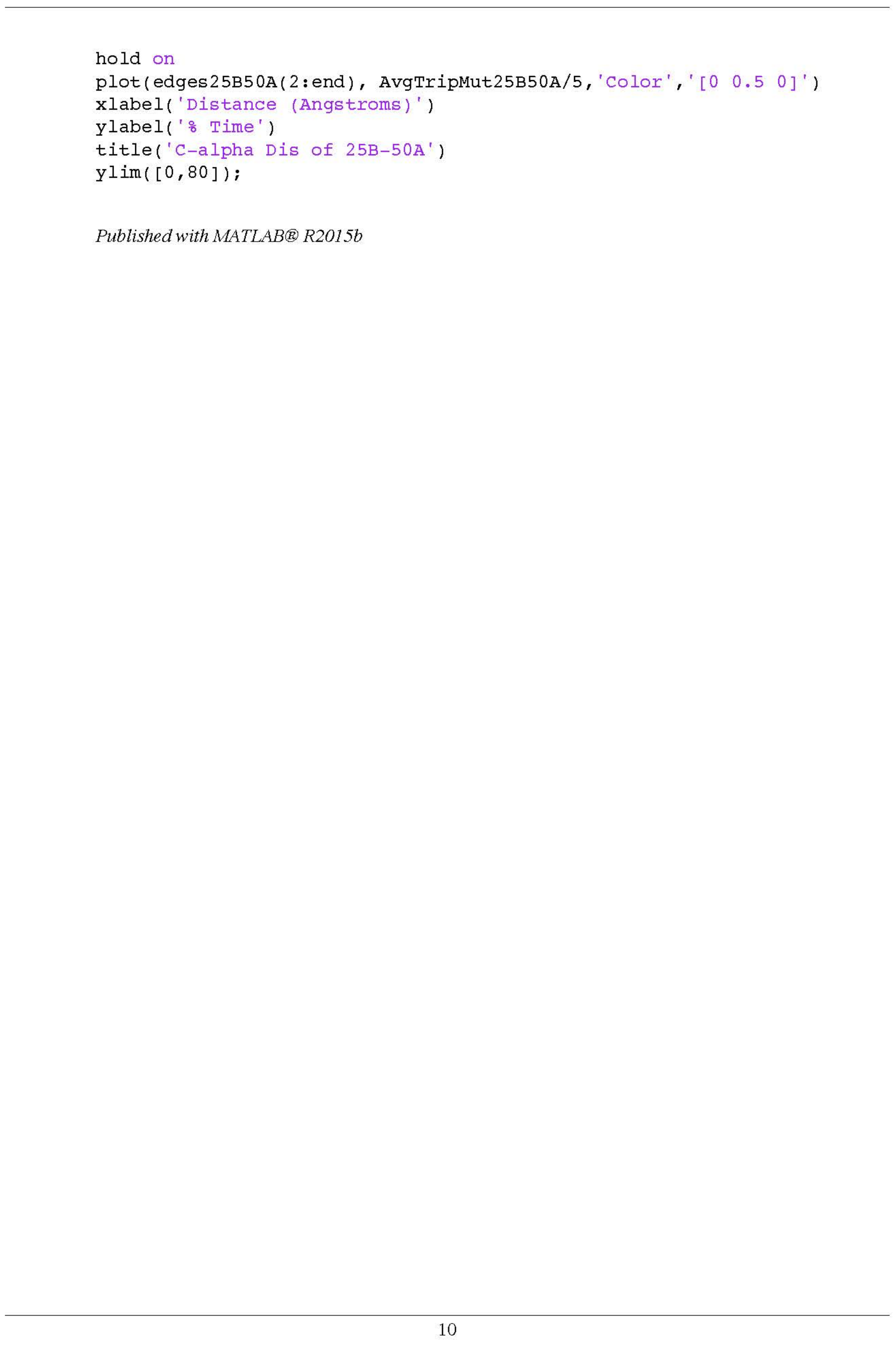
```
AllWT25=[WT25,WT25_R2,WT25_R3];
AllI84V25=[I84V25_R1,I84V25_R2,I84V25_R3];
AllDubMut25=[DubMut25_R1,DubMut25_R2,DubMut25_R3];
AllTripMut25=[TripMut25_R1,TripMut25_R2];
MeanWT25=mean(AllWT25);
MeanI84V25=mean(AllI84V25);
MeanDubMut25=mean(AllDubMut25);
MeanTripMut25=mean(AllTripMut25);

% 84-84
AllWT84=[WT84,WT84_R2,WT84_R3];
AllI84V84=[I84V84_R1,I84V84_R2,I84V84_R3];
AllDubMut84=[DubMut25_R1,DubMut25_R2,DubMut25_R3];
AllTripMut84=[TripMut25_R1,TripMut25_R2];
MeanWT84=mean(AllWT84);
MeanI84V84=mean(AllI84V84);
MeanDubMut84=mean(AllDubMut84);
MeanTripMut84=mean(AllTripMut84);
```

Plotting the Averages

```
fig1=figure;
%subplot(2,3,1)
plot(edges25(2:end),AvgWT25/5,'b',edges25(2:end),AvgI84V25/5,'r',edges25(2:end),Av
hold on
plot(edges25(2:end),AvgTripMut25/5,'Color','[0 0.5 0]')
xlabel('Distance (Angstroms)')
ylabel('% Time')
title('C-alpha Dis of 25-25')
ylim([0,80]);
```



```
fig2=figure;
%subplot(2,3,2)
plot(edges84(2:end),AvgWT84/5,'b', edges84(2:end),
 AvgI84V84/5, 'r',edges84(2:end), AvgDubMut84/5,'k')
hold on
plot(edges84(2:end), AvgTripMut84/5,'Color','[0 0.5
 0]','LineWidth',0.75)
xlabel('Distance (Angstroms)')
ylabel('% Time')
title('C-alpha Dis of 84-84')
ylim([0,80]);


fig3=figure;
%subplot(2,3,3)
plot(edges25A50A(2:end),AvgWT25A50A/5,'b',
 edges25A50A(2:end),AvgI84V25A50A/5,'r', edges25A50A(2:end),
 AvgDubMut25A50A/5,'k')
hold on
plot(edges25A50A(2:end), AvgTripMut25A50A/5,'Color','[0 0.5
 0]','LineWidth',0.75)
xlabel('Distance (Angstroms)')
ylabel('% Time')
title('C-alpha Dis of 25A-50A')
ylim([0,80]);


fig4=figure;
%subplot(2,3,4)
plot(edges25A50B(2:end),AvgWT25A50B/5,'b'
 ,edges25A50B(2:end),AvgI84V25A50B/5,'r'
 ,edges25A50B(2:end),AvgDubMut25A50B/5,'k')
hold on
plot (edges25A50B(2:end), AvgTripMut25A50B/5,'Color','[0 0.5 0]')
xlabel('Distance (Angstroms)')
ylabel('% Time')
title('C-alpha Dis of 25A-50B')
ylim([0,80]);

fig5=figure;
%subplot(2,3,5)
plot(edges25B50B(2:end),AvgWT25B50B/5,'b',edges25B50B(2:end),AvgI84V25B50B/5,'r'
 ,edges25B50B(2:end),AvgDubMut25B50B/5,'k')
hold on
plot(edges25B50B(2:end), AvgTripMut25B50B/5,'Color','[0 0.5 0]')
xlabel('Distance (Angstroms)')
ylabel('% Time')
title('C-alpha Dis of 25B-50B')
ylim([0,80]);

fig6=figure;
%subplot(2,3,6)
plot(edges25B50A(2:end),AvgWT25B50A/5,'b',edges25B50A(2:end),AvgI84V25B50A/5,'r'
 ,edges25B50A(2:end),AvgDubMut25B50A/5,'k')
```



```
hold on
plot(edges25B50A(2:end), AvgTripMut25B50A/5,'Color','[0 0.5 0]')
xlabel('Distance (Angstroms)')
ylabel('% Time')
title('C-alpha Dis of 25B-50A')
ylim([0,80]);
```

Published with MATLAB® R2015b