DANMARKS TEKNISKE UNIVERSITET

DTU Center for Biological Sequence Analysis

27610 Python and Unix

The Deaminator

Author:
s132330 Vilhelm Krarup Møller

8. April 2016

Contents

1 Abstract

Protein deamidation has been shown to have an important role in the non-enzymatic degradation of proteins. In this report the use of a program designed to estimate the half time of a protein from the amount of ether asparagine or glutamine residues is explained. The program has not been verified with experimental data, but is build on the algorithm's of an earlier program that gave precise estimates of the protein degradation of enzymes tested [3].

2 Introduction

The deamidation rates of proteins can be calculated from the knowledge of the primary and secondary structure of a protein [4]. This program is designed to calculate the half time of any protein with a crystal structure in the PDB database. The program can be used to estimate the shelf life of an enzyme or give an estimate of the amount of functional protein left in your solution after a given period. The program may also be modified to give a prediction of how to raise or lower the deamidation time. This could be relevant for the modification of therapeutically proteins such as insulin to increase the half life in the body.

Deamidation is a non enzymatic reaction that can occur in any polypeptide chain containing amines or aspartate residues. The amines react with the backbone NH2 on the COOH side of ether asparagine, glutamine or aspartate. The reaction is initiated by a nucleophilic attack from the side chain to the NH2 backbone shown in figure 1 [8].

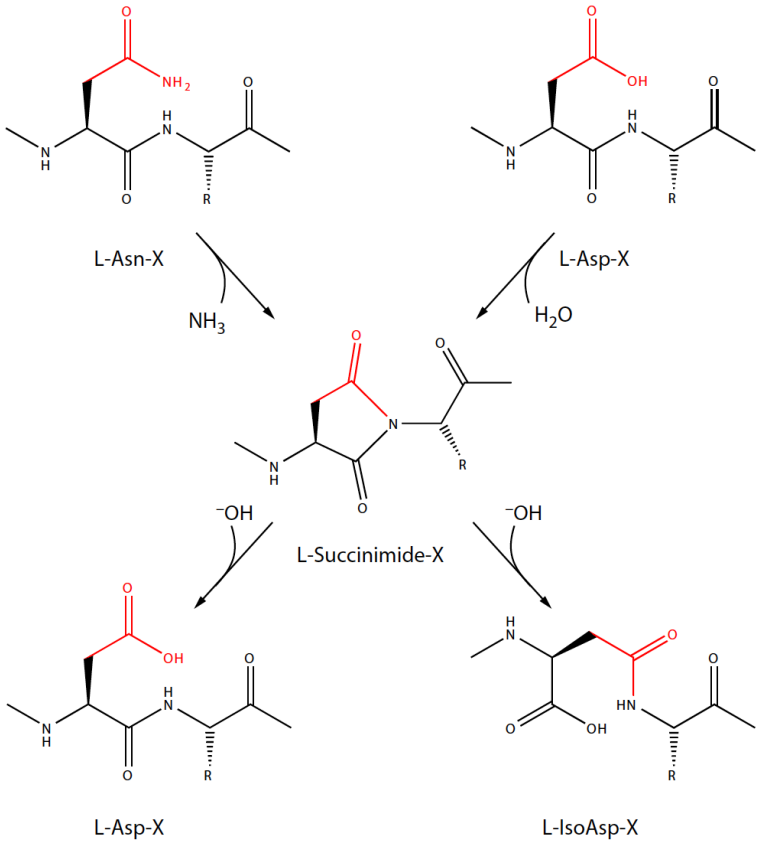

Figure 1) Shows the mechanism of deamidation through the formation of the succinimide intermediate. Only the asparagine, shown to the left is relevant as aspartate has a much lower reaction rate than asparagine [8].

The reaction forms a cyclic succinimide intermediate. The succinimide is hydrolyzed to ether a asparagine or a isoasparagine. The change introduces a negative charge in the protein and in the case of isoasparagine a isomerization. Both of these changes can lead to a structural change that can denature the protein. [8]
The deamidation rates of glutamine and aspartate are much lower than asparagine. [8]. The ratio of the different products of the deamidation reaction is 60-85% isoasparagine and 40-15% [1].
Amino acid deamidation has been know since 1883 [7]. The relevance of this discovery was first recognized in the 1960 when Flatmark showed that the different types of cythochrome c differ only in the level of deamidation. He found that CyI could change to ether CyII CyIII or CIV by the level of deamidation. The study indicated that there was a biological relevance in the deamidation of amines [8].
Building on this Arthor Robinson developed a hypothesis, that the deamidation of any amide was dependent on rate constants given by the primary structure of the amide and the secondary structure position. This hypothesis was named the molecular clock hypothesis. Robinson later validated this hypothesis with experimental data and build a program that calculated the half time of a protein from the primary and secondary structure (www.deamidation.org/). The server has been closed. It is therefor not possible to compare the results of the program discussed in this report with the program made by Robinson [3; 5].

3 Theory

3.1 Protein deamidation and protein half time

Protein deamidation is the removal of the amine residue from ether glutamine (Gln) or asparagine (Asn). The amine residues is replaced by a acid group thereby changing the charge of the amino acid inside the protein. This change has been shown to cause the protein to malfunction and in many instances to be non-enzymatically degraded. [2]
The deamidation reaction occurs when the amino group is withing close proximity to the neighbouring amino acid NH2 backbone. When this occurs a succinimide intermediate is created. When this intermediate is hydrolyzed the residue is deamidated [2]. The reaction can also occur for aspartate and glutamate but the correct alignment is fare less lightly [8] and the effects are not as significant as Asn deamidation.

The half life of a pentapeptide with Gln has been shown to be shorter than the half time of a similar peptide with Asn [5; 6]. The amino acid residue on the NH2 side of the amine carrying amino acid has also shown to be relevant. This is believed to be caused by the size of the side chain. The smaller the chain the greater is the flexibility of the amine side chain. Because of this the chances of deamidation is greater.
The molecular clock hypothesis states that the deamidation rate of a protein is programmable by changing the residues close to any Asn or Gln [8].

3.2 Modeling of protein deamidation

A method to quantify the half time of any protein as been devised by creating a model of the steric hindrance of the neighbouring amino acid adjacent to an amino side chain. The model calculations are build on data from penta peptide sequences. By looking at the short peptides a method to calculate the deamidation rate of asparagine and glutamine could be created from the primary structure of a short peptide
The model was further developed by creating an equation that accounts for the position of any asparagine and its neighbour in the secondary. structure. The effect of deamidation is most apparent when looking at the asparagine residues. Because of this the secondary structure model only accounts for the asparagine residues and not the glutamine and carboxylic side chains [5; 6].

3.2.1 Primary structure model

The model is build on data from different pentha peptides with ether asparagine or glutamine residues placed at the middle position in the peptide (Gly-Xxx-(Asn/Gln)-Yyy-Gly). The residue on the NH2 side(Yyy) of the Asn/Gln was shown to have a 20 fold greater effect than the COOH side(Xxx) of the backbone [5; 6].

As the deamidation rate is increased by grater flexibility of the amide side chain. Robinson created a steric hindrance value for every amino acid placed on the COOH side of a Asn/Gln. The values were found by finding the natural log of the median hydrolysis correction rate constants times $100 \cdot s^{-1}$ for all different canonical 20 amino acid.

$$deamidation\,time = 100 \cdot ln(k)$$

where k is the median hydrolysis correction for a amino acid found experimentally.

The values were normalized to the value of glycine.
Some amino acids have been excluded as they were unable to deamidate the Asn/Gln side chain before the pentha peptide N-termini was deamidated. The exclusion was done if no deamidation had occurred later than 8000 days. The amino acids excluded are proline for both Asn and Gln. Tyrosine and tryptophan was excluded for Gln [5; 6]. A overview of the steric hinracne values is shown in table 1.

Table 1) The steric hindrance values for all amino acid on the COOH side of Asn or Gln are shown. The values were normalized with the Gly value

	Asn	Gln
Yyy	Normalized values to Gly	Normalized values to Gly
Gly	0	0
His	219.5	350.4
Ser	262.0	334.4
Ala	306.8	347.1
Asp	333.7	562.3
Thr	370.6	305.3
Cys	379.8	127.6
Lys	393.1	353.7
Met	396.5	273.5
Glu	401.1	482.0
Arg	400.7	459.5
Phe	411.9	602.5
Tyr	425.1	
Trp	444.0	
Leu	466.3	367.7
Val	538.5	399.7
Ile	561.1	379.0
Gln	40.0	
Asn	40.0	
Pro	500.0	

The deamidation half time of a particular Asn can be found by applying this equation.

$$t_{Asn1/2} = (\frac{ln(2)}{86400}) \cdot e^{\frac{sum}{100}+11.863}$$

$t_{Asn1/2}$ = the deamidation half for a particular Asn.
sum = the steric hindrance value defined by the residues on the COOH side of Asn.
86400 = mean deamidation time in days.

11.863 = a fitted Asn constant.

The deamidation half time of any Gln can be calculated in a similar way. The only difference is the sum and the fitted constant.

$$t_{Gln1/2} = \frac{ln(2)}{86400} \cdot e^{\frac{sum}{100}+18.311}$$

A hydrolysis correction can be made. This is important for the Gln residues and for Asn with long deamidation times.

$$t_{total1/2} = \frac{1}{\frac{1}{8000} + \frac{1}{t_{1/2calculated}}}$$

8000 = the median hydrolysis for the penta peptide GlyXxxAsnProGly. Measured in days. [5; 6]

3.2.2 Secondary structure model

The secondary structure model is also refed to as the 3D model.
The deamidation rate of proteins depends on many other features than can be explained by the primary structure therefor a method to predict the deamidation rates from the secondary structure was created [3; 4]. Studies show that around 60% of the deamidation rates can be predicted using primary structure and 40% by 3D structure. [4]. The method only account for the deamidation of asparagine as this is the amine most lightly to deamidate.

Deamidation coefficient C_D was defined as the deamidation coefficient for every amine in a protein. The total deamidation half time was defined as the sum of all the C_D this total deamidation half time was called I_D, the deamidation index. Both C_D and I_D are measured in days.

The deamidation coefficient was defined as:

$$C_D = 0.01 \cdot t_{1/2} \cdot e^{f(Cm,C_{Sn},S_n)}$$

$t_{1/2}$ = the primary half time calculated was shown above.
C_m = the structural proportionality factor.
C_{Sn} = the 3D structural coefficients for the nth structural observations.
S_n = the nth observation used by C_{Sn}.
$f(Cm, C_{Sn}, S_n)$ = the weighting function for every 3D.
parameter(S_n). The weighting function is given by:

$$f(Cm, C_{Sn}, S_n) = C_m \cdot (C_{S1}S_1 + C_{S2}S_2 + C_{S3}S_3 - \frac{C_{S4,5}S_4}{S_5}$$

$$+C_{S6}S_6 + C_{S7}S_7 + C_{S8}S_8 + C_{S9}S_9 + C_{S10}(1 - S_{10})$$

$$+C_{S11}(5 - S_{11}) + C_{S12}(5 - S_{12})$$

The S_n values are defined by there position in different structures such as α-helixes β-sheets or coils. In general the higher the given S_n value the lower is the chance of deamidation for the individual amine.

Asn in α-helixes region
S_1 is how deep inside the helix the Asn is placed on the NH2 side. $S_1 = 1$ when the Asn is at the first amino acid in the α-helix. If the Asn is further than 3 residues in an α-helix $S_1 = 0$.
S_2 is the distance from helix start on the COOH side. $S_2 = 1$ when the Asn is at the end of a α-helix structure. If

the Asn is further than 3 residues from the α-helix end $S_2 = 0$ or if $S_1 \neq 0$ then $S_2 = 0$.
If the Asn is 4 or more residues from the end of the α-helix then $S_1 = 0$, $S_2 = 0$ and $S_3 = 1$. If ether S_1 or S_2 is different from zero then $S_3 = 0$.
Flexibility in a loop between two anti parallel β-sheets S_4 = the number of residues in the loop. $S_4 = 0$ if the Asn is not flanked by to β-sheets with a length greater than 4 residues on both sides.
S_5 is the number of hydrogen bonds in the flexible region + 1. If $S_5 \neq 0$ then $S_4 \neq 0$.

Hydrogen bonds
S_7 is the number of H-H bonds to the Asn side chain NH2 group. values can be 0,1 or 2.
S_8 is the number of hydrogen bonds to the backbone NH2 on the COOH side of the Asn. This is important for the formation of the succinimide ring.
S_9 number of hydrogen bonds that need to be broken to align the NH2 side chain of Asn to the NH2 on the COOH side of the backbone.

End chain assumptions
Asn placed op to 20 residues from the end of the peptide chain with no α-helix, (β-sheets or disulphide bridge between the Asn and the end of the chain were given a special penalty as they were assumed to have a grater chance of deamidation do to the extra flexibility in that can occur in this part of the protein.
$S_{10} = 1$ if the Asn in 3 or more from any α-helix, (β-sheets or disulphide bridge. If any of these structures are closer than 3, $S_{10} = 0$.
S_{11} is the number of residues on the NH2 side of the Asn with a maximum distance of 5 accepted.
S_{12} is the number of residues on the COOH side of the Asn with a maximum distance of 5.
For the 3D structure calculations the primary steric hindrance values for Asn-Pro: 500, Asn-Asn: 40 and Asn-Gln: 40 was defined.

The total half time of the protein was calculated by:

$$I_D = (\sum (C_D)^{-1})^{-1}$$

$$Deamidation\, half\, time\, of\, protein = 100 \cdot I_D$$

The Deamidation half time is given in days and can range from 1 day to 10,000 days. [3; 4]

4 Design and construction

The program was written in python 3.4.3 using Protein Data Bank files from (www.rcsb.org/pdb/home/home.do). The user can calculate any protein half life by typing a PDB ID into the program. The program feeds the PDB file to the DSSP program (swift.cmbi.ru.nl/gv/dssp/) to retrieve the secondary structure. The program calculates the deamidation of Gln and Asn from the primary structure given by the PDB file and modifies the Asn results using the secondary structure given by DSSP

4.1 Program structure

The program is split into four primary parts based on function. First part loads the PDB file from the PDB website and retrieves the DSSP output file base on the PDB file.
Second part calculates the deamidation half time from the primary structure using both Asn and Gln residues.
Third part defines all the parameters used to calculate the secondary structure information, these values are called S-values.
Fourth part calculates the deamidation half time for the 3D structure using the Asn and finds the I_D value and calculates the half time of the protein. The half time is outputted to the user.

The four parts are split into seven main functions. load_pdb and dssp_import are part one responsible for loading the PDB file and DSSP input. primary_half_time is responsible for the second part, calculating the primary half time. HH_bond, end_chain and helix_S are part three, defines all the parameters used to calculate the secondary structure.
The fourth and final part is the collection of all the S-values and the primary half times to calculate the total half time of the protein and outputs this to the user.

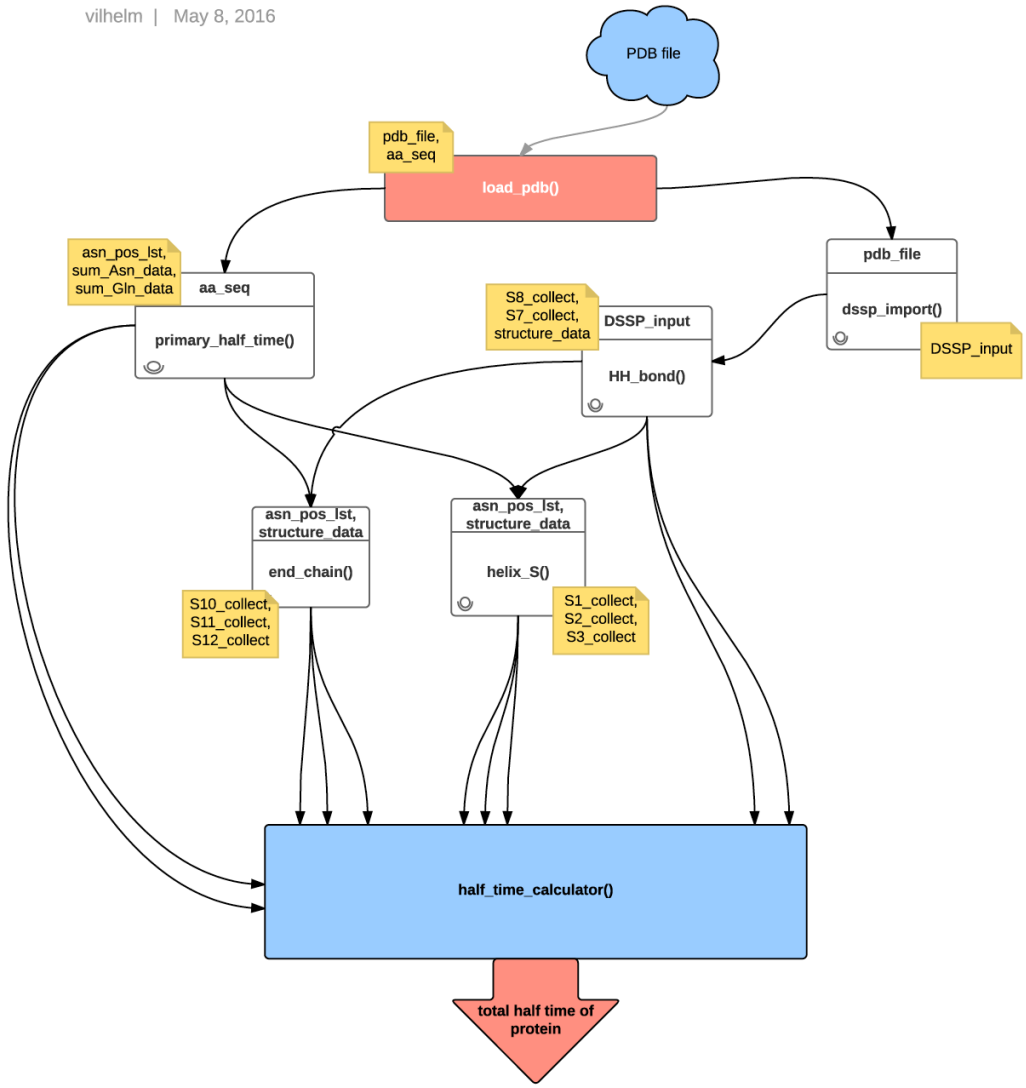

Figure 2) Shows a schematically representation of how the different functions interact with each other. all functions are shown as boxes with there name written in the middle and their input written at the top of the box, with the exception of half_time_calculator which uses all the S values as input. The output of every function is shown in the yellow box beside the function. The PDB file is downloaded from the web, shown as a cloud. The input in every function is shown as in the top of the box. The output of the program is the half time of the protein. This is shown by the red arrow.

A schematic representation of the functions interplay between the different functions is shown in figure 2. All functions created find or calculate values that can be used by the function half_time_calculator this function creates the last function output. This output is used to calculate the half time of the protein.

4.2 Assumptions made in the program

The program was build on a wide range of assumptions. The first was that the protein studied will be similar to the crystal structure obtained. Many proteins change structure when they are crystallized. There is no way to forsee this in the program, but it is important that the program user is aware of this problem.

In the calculation of the primary structure it is assumed that the succinimide intermediate hydrolysis rate is grate enough to assume that all deamidated amines will become aspartate or isoaspartate residues.

All the data used to calculate the primary structure was extracted from penta peptides that were tested at 37°C, pH 7.4 in 0.15 Tris buffer. The proteins may react very different compared to small peptides in Tris buffer. The difference in size is countered by the calculations based on the structural data.

To simplify the program it was assumed that all Asn and Gln in the primary structure would need hydrolysis correction. The assumption seems valid as only a few Asn avoid hydrolysis correction.

In the calculation of S10-S12 it is assumed that the disulphide bridges do not play an important role in the deamidation rate. This assumption needs testing and it would be preferable to include the disulphide bridges.

S_6 was excluded from the program as the constant C_{S6} was estimated to 0 [3].

S_9, S_4 and S_5 was excluded from the program do to a limited time frame. It would be preferable to include these parameters.

5 Results and comparisons

The run time for every function was estimated and the output of the program is shown. The program has not been validated do to a lack of known protein half times.

5.1 Runtime analysis

The program is subdivided into different functions, this has been done to minimize the amount of memory used while running the program, in the hope of optimizing the runtime of the program. The Big O was estimated for every function in the program. The results can be seen in table 2.

Table 2 Shows the Big O estimated for ever function the program.

Function name	Big O equation
pdb load	O(n^5)
dssp import	O($2n$)
primary half time	O(n^2)
HH bond	O($8n^2$)
helix S	(n^8)
end chain	O($12n^2$)
half time calculator	O(gn + 5an)

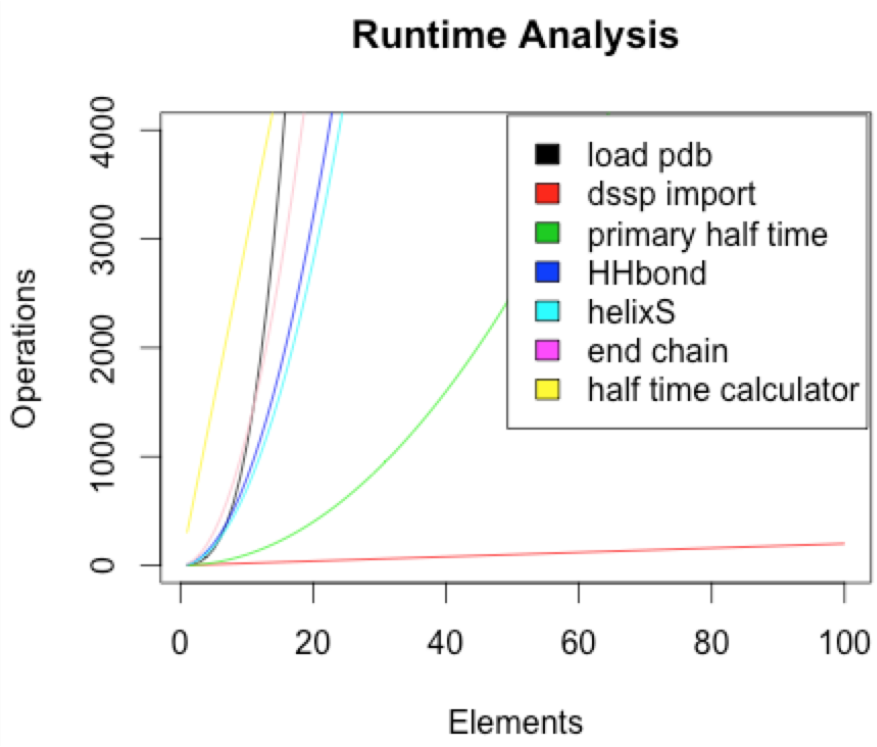

Figure 3 Shows the Big O For every function in the program compared to each other.

The runtime of the different functions can be seen on figure 3. It is clear the most computationally heavy functions are load pdb, HHbond, helix S and end chain.
One of the functions that require the most operations is load pdb as it needs to send a request and extract data from the PDB website. Many of the other functions use the amino acid sequence outputted by the pdb load. The runtime analysis is run on the assumption that half of all amino acid in the sequence are Asn and Gln ($\frac{n}{2} = a = g$). This assumption is made to test the a possible worst case scenario. If n = a the worst case for the functions used to find the penalty values (S-values) for 3D calculation (HHbond, helixS, end chain) will tested. If n = g and a = 0 only pdb load, primary half time and half time calculator will be tested.
The function end chain is in the analysis assumed to run through all the Asn in the sequence this not the case in reality as the function will only run through the last 20 amino acids. This means that the real run time is significantly lower than what is shown on figure 3.

Total Big O of the program
All runtimes estimated for the functions were summed to find the total run time of the program:

$$total\,runtime = O(n^5 + 5 * a * n + g * n + 21 * n^2 + 2 * n)$$

5.2 PDB ID's used for testing

The program was tested on different PDB files, primarily on the small proteins 5BTS and 4ZJ9.

5.3 Validation of assumptions

The program has not been verified on real data as no data made under the same experimental conditions could be found.

In future analysis it will be necessary to find training data. The web program made by Robinson is not available. Comparing this program to Robinsons version would be the optimal way of verify the halftimes without laboratory work.
As there is no comparison to test if the half time estimate is correct there is no way of validating the precision of the output.

6 Program architecture and user manual

An over view of the input output data is given. The problems in the program are discussed and a user manual is presented.

6.1 Input output overview

The program receives the primary input as a PDB file downloaded from the PDB website (www.rcsb.org/pdb). The PDB file is used to extract the sequence information and as a input for the DSSP program (swift.cmbi.ru.nl/gv/dssp/).The DSSP program is called by the function dssp_import.
The output of the program is given as a print statement specifying the half life of the program.
An example of a output is shown below:

Protein half time is estimated to 196.1 days

6.2 Problems and weaknesses

The program can only find a half time if there is any Asn or Gln residues in the protein. This does not fit with reality were any protein will denature at some point.
The experimental data was made in 0.15 Tris at 37°C and pH 7. The natural environment could be different than the experimental conditions. This could lead to incorrect calculations.
The program assumes that there is no deamidation repair system present. In reality a special enzyme exist for this purpose. [2]
The program assumes that there is no denaturing enzymes or chemicals in the solution. This is an important consideration when using the program. The program assumes that every deamidation has an equal impact on the half life of the protein. This assumption is inherently incorrect as some amines will have a greater structural importance than others.

6.3 How to run program

The program requires that python 3 and DSSP are installed.
The program is started by typing the following command in a UNIX terminal:

python3 Deaminator.py

A message will then appear telling the user to type in a PDB ID (ex. 5BTS). Ether lower- or uppercase letter can be used. When this is done the user will receive a message with the estimated half time of the protein. If an error occurs the program will specify the error and terminate.
The program excepts different types of errors. See User manual for more information.

6.4 User manual

The program is build into a seven (7) functions. These are explained in detail below.

6.4.1 PDB load

The user is prompted to write a PDB ID. A request is sent to the PDB website to extract the PDB file for the prompted ID. If the file cannot be retrieved from the website one of three errors will be raised. If a connection or an input output error is raised the program will report this and terminate. If a unknown error occurs then a statement reporting this will be shown, but the program will not terminate.
If the file is successfully found the PDB file will be downloaded onto the computer as a file named PDB file.
The file is opened and tested to insure that the file is not empty.
The file is read line by line and two regexe's are used. One to find the ATOM position and extract amino acid sequence and another to find possible terminal amino acids (search for TER). If a protein contains more polypeptide sequences, then there may occur a terminal sequence inside file. The second regex finds this terminal amino acid and appends it to the amino acid sequence.
Every amino acid is verified to be one of the 20 canonical amino acids used by the ribosome before it is append to the amino acid sequence. If any amino acids are non canonical a print statement will be executed showing the non canonical amino acid and its position. All amino acids are appended the list. If a PDB file contains more than one crystal structure then only the first will be used. This is done by searching for 'ENDMDL ' in the line. If a hit occurs then af flag will be set to stop any further amino acids from being appended.
Simultaneously the positions of every asparagine residues is found.

6.4.2 Primary half time calculation

Using the amino acid sequence the steric hindrance values are found for all amino acids on the COOH side of asparagine or glutamine. The primary half times are calculated for every amine in the sequence and these are appended an asparagine or glutamine list.
As there are many special cases the program is made to account for these. Such as if the last amino acids in the chain is an amine, meaning there is no amino acid on the COOH side. The steric hindrance values is approximated to be the same as if a Asn was placed at the COOH side. [3]
The deamidation rate of Gln is much lower than Asn therefor there is some amino acids that cannot deamidate Gln. These are Pro, Trp, Tyr, Gln, Asn. If any of these amino acids are placed at the COOH side of a Gln then they are excluded from the calculation.

6.4.3 DSSP import

The program sends the PDB file as an input to the DSSP program by calling DSSP on the command terminal. The DSSP output is saved in a file on the computer.

6.4.4 Hydrogen bond penalty

The DSSP output file is used to find the penalty information. This is done using state full parsing. A flag is activated by the header line in the file. The S-values S8 and S7 are found. A simplification of the secondary structure data given by DSSP is also extracted. The simplified structure data is used by the functions helixS and end chain.
If the length of the amino acid sequence derived from the PDB file and the amino acids in the DSSP file do not match a warning will be printed to the screen

6.4.5 Helix penalty

Using the positions of Asn found by the primary half time calculator and the simplified structural data given by hydrogen bond calculator, The helix penalty values can be calculated. The helix penalty values are named S1, S2, S3. If any of the Asn are placed in a helix then the structural data on each side of the Asn is split into a string containing residue structures for the amino acids with in a 3 residue radius. If all residues have a helix structure then S3 = 1.

Else the string is split into two sub strings containing two residue structures, two on each side of the Asn. The length of continuing helix structures is counted for the up and downstream sub strings. The counts are compared and the lowest number is chosen. A bias is introduced if the counts are equal the program will choose $S1 \neq 0, S_2 = 0, S_3 = 0$.

6.4.6 End chain penalty

The Asn that are found in the end of the chain is waited differently as they are assumed to be more lightly to deamidate if they are in a turn. The program finds the Asn placed at 20 residues from the end of the protein. Two sub strings are made upstream and downstream with a length of three residues. If all residues up- or downstream are in turn structures then S10 = 1. Two new sub string with a length of 5 up and downstream are made. The upstream string is reversed and a for loop is used to count the number of residues in turnstructure until an alternative structure is found. This defines S11 and S12.

6.4.7 Final half time calculation

All S-values and steric hindrance values are collected and the I_D and C_D values are calculated. A for loop is used to calculate the 3D deamidation half time of every Asn in the protein and the primary half time calculations of Gln are appended to the list.
If list is not empty the 3D half time is calculated by summing the list and multiplying with 100. The result is given in days.

7 Conclusion

The program is build on the experimental data created by Robinson [3] and the equations to calculate the half time from the deamidation half times for each Asn and Gln in a protein. The gives the user an estimated protein half time. The output given has not been compared to other results, this will be necessary before the program can be used in practice.
A runtime analysis of the program has been made and it is clear that the program is computationally heavy. This is doe to the particular functions that(half time calculator, laod pdb, end chain, HHbond, helixS) se figure 3. Focusing on these functions when optimizing would therefor be preferable.

The program has a wide potential for use usage such as approximating the spoilage time of protein containing foods in sterile containers, predicting the degradation time of therapeutical peptides used in medicine and simulation of the effect of changes to obtain versions of the peptides before experimental work is done.

8 References

[1] DW Aswad, MV Paranandi, and BT Schurter. Isoaspartate in peptides and proteins: formation, significance, and analysis. *Journal of Pharmaceutical and Biomedical Analysis, J. Pharm. Biomed. Anal, J Pharm B, J Pharmaceut Biomed, J Pharmaceut Biomed Anal, J Pharm Biomed Anal, Journal of Pharmaceutical and Biomedical Analysis*, 21(6):1129–1136, 2000.

[2] KJ Reissner and DW Aswad. Deamidation and isoaspartate formation in proteins: unwanted alterations or surreptitious signals? *Cellular and Molecular Life Sciences, Cell. Mol. Life Sci, Cell Mol L, Cell Mol Life Sci, Cmls Cellular and Molecular Life Sciences, Cmls Cell Mol Life Sci, Cellular and Molecular Life Sciences*, 60(7):1281–1295, 2003.

[3] NE Robinson. Protein deamidation. *Proceedings of the National Academy of Sciences of the United States of America, Proc. Natl. Acad. Sci. U. S. a, P Nas Us, P Natl Acad Sci Usa, Proc Nat Acad Sci Usa, Proceedings of the National Academy of Sciences, Proc Natl Acad Sci Usa, Proceedings of the National Academy of Sciences of the United States of Ame, National Academy of Sciences of the United States of America Proceedings, Pnas, Proc Natl Acad Sci U S a, Proc Nat Acad Sci U S a*, 99(8):5283–5288, 2002.

[4] NE Robinson and AB Robinson. Prediction of protein deamidation rates from primary and three-dimensional structure. *Proceedings of the National Academy of Sciences of the United States of America, Proc. Natl. Acad. Sci. U. S. a, P Nas Us, P Natl Acad Sci Usa, Proc Nat Acad Sci Usa, Proceedings of the National Academy of Sciences, Proc Natl Acad Sci Usa, Proceedings of the National Academy of Sciences of the United States of Ame, National Academy of Sciences of the United States of America Proceedings, Pnas, Proc Natl Acad Sci U S a, Proc Nat Acad Sci U S a*, 98(8):4367–4372, 2001.

[5] NE Robinson and AB Robinson. Prediction of primary structure deamidation rates of asparaginyl and glutaminyl peptides through steric and catalytic effects. *Journal of Peptide Research, J. Pept. Res, J Pept Res, Journal of Peptide Research, Journal of Peptide Research : Official Journal of the American Peptide Society*, 63(5):437–448, 2004.

[6] NE Robinson, ZW Robinson, BR Robinson, AL Robinson, JA Robinson, ML Robinson, and AB Robinson. Structure-dependent nonenzymatic deamidation of glutaminyl and asparaginyl pentapeptides. *Journal of Peptide Research, J. Pept. Res, J Pept Res, Journal of Peptide Research, Journal of Peptide Research : Official Journal of the American Peptide Society*, 63(5):426–436, 2004.

[7] E. Schulze and E. Bosshard. Ueber das glutamin. *Berichte Der Deutschen Chemischen Gesellschaft, European Journal of Inorganic Chemistry, Ber. Dtsch. Chem. Ges, Eur J Inorg, Eur J Inorg Chem, Chemische Berichte, Berichte Der Deutschen Chemischen Gesellschaft, Eur. J. Inorg. Chem*, 16(1):312–315, 1883.

[8] Steven J. Weintraub and Benjamin E. Deverman. Chronoregulation by asparagine deamidation. 2007.

9 Appendix

```
#!/usr/bin/python3

# imports
import re
import math
import requests
import sys
from subprocess import call
import os

######################### 1) LOAD FILE #######################################

def load_pdb(pdb_id):
    """ Function outputs amino acid sequence from the PDB ID
    that is requested by the user.
    A comment will be made if the program removes any unknown amino acids
    """
    # if removes any spaces written by the user
    pdb_id = pdb_id.replace(' ', '')
    # test if the length of the input is correct.
    if len(pdb_id) != 4:
        print('The input ', pdb_id, ' is not a PDB ID', '\n')
        sys.exit(1)

    # imports pdb file from the PDB website
    try:
        pdb_web = requests.get('http://www.rcsb.org/pdb/download/downloadFile.do?fileFormat=PDB&
    compression=NO&structureId=' + pdb_id)
        outfile = open('pdb_file.pdb', 'w')
    except requests.ConnectionError as error:
        sys.stdout.write('Error when loading file from the PDB server: '
                         + str(error) + '\n')
        sys.exit(1)
    except IOError as error:
        sys.stdout.write('Cannot write file, reason:' + str(error) + '\n')
        sys.exit(1)
    except:
        sys.stdout.write('Unknown error has occurred when downloading the PDB file from the web
    site')

    # changes the format to txt file and downloads it on the computer
    for char in pdb_web.text:
        if char == '\n':
            outfile.write('\n')
        else:
            outfile.write(char)
    outfile.close()

    # attempt to open the pdb_file
    try:
        infile = open('pdb_file.pdb', 'r')
        # if the PDB file downloaded is empty and error will be raised
        if not os.stat('pdb_file.pdb').st_size:
            sys.stdout.write('The pdb_file downloaded is empty. This may be because no PDB file
    exist for this PDB ID\n')
            sys.exit(1)
    except IOError as error:
        sys.stdout.write('Cannot open file, reason:' + error.strerror + '\n')
        sys.exit(1)

    # used to insure that the sequence only contains the 20 amino acids
    aa_verify = (
    'ALA', 'ARG', 'ASN','ASP', 'CYS', 'GLU', 'GLN',
```

```
    'GLY', 'HIS', 'ILE', 'LEU', 'LYS', 'MET', 'PHE',
    'PRO', 'SER', 'THR', 'TRP', 'TYR', 'VAL'
                )
    re_prot_term = None
    re_aa_pdb = None
    aa_num = 0
    aa_seq = []
    stop_flag = False
    # find all Asn og Gln in sequnece
    for line in infile:
        # find the amino acids in pdb file
        re_aa_pdb = re.search('^ATOM\s+\d+\s+\w.+?\s+(\w+\s\w\s+\d+)\s+.+$', line)
        re_prot_term = re.search('^TER\s+\d*\s*(ASN)\s*\w\s*\d*', line)

        # if there is a terminal Asn on one of the chains in the
        # PDB file, then that is added to the correct position
        if re_prot_term is not None:
            aa_seq.append('ASN')

        if 'ENDMDL   ' in line:
            stop_flag = True

        # splits the amino acids into [amino_acid, chain, amino acid number]
        if re_aa_pdb is not None and not stop_flag:
            aa_pdb = re_aa_pdb.group(1)
            aa_pos = aa_pdb.split()

            # if amino acid number is different from earlier shown and
            # the earlier number is greater than the new number.
            if aa_pos[2] != aa_num:
                aa_num = aa_pos[2]
                # checks that all amino acids are the 20 canonical amino acids
                if aa_pos[0] in aa_verify:
                    aa_seq.append(aa_pos[0])
                else:
                    print('amino acid ', aa_pos[0], 'at position', aa_num,
                          'is non canonical amino acid')
                    aa_seq.append(aa_pos[0])
    infile.close()
    return aa_seq

#####################  2) IMPORT DATA FROM DSSP  ##############################

def dssp_import():
    """ Function runes the DSSP program and saves the result in
    a file named dssp.txt ready to be used by function named HH_bind()
    """
    # sends the PDB file to the DSSP server and retrieves DSSP file
    call('dssp-2.0.4 pdb_file.pdb > dssp.txt', shell = True)

    # opens DSSP file if it can be found
    try:
        dssp_input = open('dssp.txt', 'r')
    except IOError as error:
        sys.stdout.write("Cannot open file, reason:" + error.strerror + "\n")
        sys.exit(1)
    return dssp_input

#######################  3) PRIMARY STRUCTURE  ################################

def prim_calc(next_aa, steric, CONS):
    """ Function calculates the half times of a asparagine or glutamine
    from the primary structure
    Use in primary_half_time()
    """
```

```
126     # make calculation of each amines half time
127     half_time = math.log(2) / 86400 * math.exp(steric / 100 + CONS)
128     # correcting for hydrolysis
129     half_time_hydro = 1 / ((1 / 8000) + (1 / half_time ))
130     return half_time_hydro
131 
132 ##############################################################################
133 
134 def primary_half_time(aa_seq):
135     """ Function finds steric hindrance values for the amino acids that fit the
136     criteria for deamination. A function is called to calculates the
137     deamination half time from the primary structure.
138     The asparagine and glutamine data is collected by
139     this function and append to lists.
140     """
141     ## dictionaries for all defined steric hindrance values for the Asn-Xxx
142     ##  and Gln-Xxx.
143     # P,Y and W are not included for Glutamine as there is no experimental
144     # data available
145     steric_asn_info = {
146                 'GLY':0.0, 'HIS':219.5, 'SER':262.0, 'ALA':306.8,
147                     'ASP':333.7, 'THR':370.6, 'CYS':378.8, 'LYS':393.1,
148                     'MET':396.5, 'GLU':401.1, 'ARG':400.7, 'PHE':411.9,
149                     'TYR':425.1, 'TRP':444.0, 'LEU':466.3, 'VAL':538.5,
150                     'ILE':561.1,
151                     'PRO':500, 'ASN':40, 'GLN':60,
152                     }
153     steric_gln_info = {
154                     'GLY':0.0, 'HIS':350.4, 'SER':334.4, 'ALA':347.1,
155                     'ASP':562.3, 'THR':305.3, 'CYS':127.6, 'LYS':353.7,
156                     'MET':273.5, 'GLU':482.0, 'ARG':459.5, 'PHE':602.0,
157                     'LEU':367.7, 'VAL':399.7,'ILE':379.0
158                 }
159     sum_asn_data, sum_gln_data, asn_pos_lst = [], [], []
160     count = 0
161     AA_LENGTH = len(aa_seq)
162     # finds all Asn and Gln in sequence and appends the amino acids on the
163     # C-terminal of the Asn, Gln
164     for num in range(AA_LENGTH):
165         count += 1
166         # every amino acids in the sequnece is defined as aa
167         aa = aa_seq[num]
168         # the previous amino acid is stored as previous_aa
169         previous_aa = aa_seq[num-1]
170         # finds asparagine residues
171         if aa == 'ASN' and num < AA_LENGTH - 1:
172             # finds amino acid on C-term of Asn and test conditions
173             next_aa = aa_seq[num+1]
174             # inputs the positions of the asparagine residues
175             asn_pos_lst.append(count)
176             # find steric hindrance value from dict of
177             steric_asn = steric_asn_info[next_aa]
178             # specific constant for asparagine deamination
179             ASN_CONS = 11.863
180             # calls function to make primary calculations
181             half_time_hydro = prim_calc(next_aa, steric_asn, ASN_CONS)
182             # appends half time to list
183             sum_asn_data.append(half_time_hydro)
184 
185         # if the terminal amino acid is an asparagine
186         # the value will be calculated
187         elif num == AA_LENGTH - 1 and aa == 'ASN':
188             # find steric hindrance value from dict of
189             steric_asn = steric_asn_info[aa]
190             # specific constant for asparagine deamination
```

```
            ASN_CONS = 11.863
            # calls function to make primary calculations
            half_time_hydro = prim_calc(aa, steric_asn, ASN_CONS)
            # appends half time to list
            sum_asn_data.append(half_time_hydro)

        # finds Glutamine residues
        elif (aa == 'GLN' and previous_aa != 'ASN' and previous_aa != 'GLN' and
                num < AA_LENGTH - 1):
            # finds amino acid on C-term of Gln and test conditions
            next_aa = aa_seq[num+1]
            # different conditions are present due to insufficient data
            exclusion_aa = ['PRO', 'TRP', 'TYR', 'GLN', 'ASN']
            if next_aa not in exclusion_aa:
                # find steric hindrance value from dict
                steric_gln = steric_gln_info[next_aa]
                # specific constant for glutamine deamination
                GLN_CONS = 18.311
                # calls function to make primary calculations
                half_time_hydro = prim_calc(next_aa, steric_gln, GLN_CONS)
                # appends half time to list
                sum_gln_data.append(half_time_hydro)
    return asn_pos_lst, sum_asn_data, sum_gln_data

############################  3) S-VALUE FIND  ################################

def structure_finder(structure):
    """ Function simplifies the DSSP output for the secondary structure.
    The function output is a string consisting of alpha helix = H,
    beta-sheet = B and coil = T.
    Function used in HH_bind()
    """
    if structure == 'H' or structure == 'I' or structure == 'G':
        structure_point = 'H'
    # finds beta-sheet structure
    elif structure == 'B' or structure == 'E':
        structure_point = 'B'
    # finds coil structure
    elif structure == 'T' or structure == 'S' or structure == ' ':
        structure_point = 'T'
    return structure_point

#############################################################################################

def S8_finder(line):
    """ Function finds the S8 value.
    S8 value is the number of interactions with the Asn side chain.
    Function used by HH_bind()
    """
    NH_bind1 = line[38:45].split( maxsplit = 0)
    NH_bind2 = line[63:67].split( maxsplit = 0)

    # find S8. number of H-H to the backbone of the Neighboring  N
    if NH_bind1[0] != '0' or NH_bind2[0] != '0':
        S8 = 1
    else:
        S8 = 0
    return S8

#############################################################################################

def S7_finder(line):
    """ Function finds the S7 value from the DSSP_input.
    The S7 is the number of interactions to the NH2 on the COOH side of Asn
    Function is called in the HH_bond() function.
```

```
    """
    # find S7. number of H-H bonds on Asn NH2 side chain
    N_chain_H_bind = line[25:33].split()
    # if 0 in list there is 1 or 0 H-H
    if '0' in N_chain_H_bind:
        # if S7 = 0 then there is no H-H to amine
        if N_chain_H_bind[0] == '0' and N_chain_H_bind[1] == '0':
            S7 = 0
        else:
            S7 = 1
    # if there is no 0 in list there is two H-H on NH2
    else:
        S7 = 2
    return S7

################################################################################

def HH_bond(dssp_input):
    """ Function reads the DSSP input file and extracts information
    to calculate S7 and S8. The function also outputs a simplification of
    the structure_data
    """
    S7_collect, S8_collect = [], []
    previous_aa = None
    aa = None
    flag_start = False
    structure_data = ''
    amin_search = False
    for line in dssp_input:
        # finds header line
        if '  #  RESIDUE AA STRUCTURE ' in line:
            flag_start = True
        # finds all lines with structural information
        elif flag_start:
            # find all the secondary structures from DSSP
            CHAIN = 11
            # amino acid defined as aa
            aa = line[CHAIN+2]
            # structure information position
            structure = line[CHAIN+5]
            # function that simplifies the DSSP output. The function output is
            # appended to a string
            structure_point = structure_finder(structure)
            structure_data += (structure_point)

            # finds all Asn residues in the sequence
            if aa == 'N':
                # calls function to find the S7 value
                S7 = S7_finder(line)
                S7_collect.append(S7)
                amin_search = True
                # special case for S8 where to asn follow each other.
                # finds all the residues that are on the COOH side
                #  of a asparagine
                if previous_aa == 'N':
                        # Function finds first and second N-H binding
                        # positon for backbone
                        S8 = S8_finder(line)
                        S8_collect.append(S8)

            # all the C-terminal proteins that fit the criteria are
            # found downstream of Asn
            elif amin_search:
                # reset flag to false
                amin_search = False
```

```
                # Function finds first and second N-H binding
                # positon for backbone
                S8 = S8_finder(line)
                S8_collect.append(S8)

            # saves the previous amino acid
            previous_aa = aa

    if len(structure_data) != len(aa_seq):
        print('WORNING: The length of the DSSP import file is not',
         'the same as the length of the amino acid list found in the PDB file\n',
         'This may lead to incorrect results', 'Control the PDB and DSSP files\n',
         'length data from PDB file = ', len(aa_seq),
         ' length data from DSSP = ', len(structure_data), sep='')
    return S7_collect, S8_collect, structure_data

################################################################################

def helix_S(asn_pos_lst, structure_data):
    """ Function finds the asparagine positioned in helix structures.
    All asparagine that are in a helix are given a one zero value
    on ether S1,S2 or S3.
    """
    S1_collect, S2_collect, S3_collect = [], [], []
    # loops through the positions of the asparagine hits, to find the S1-3 values
    for element in asn_pos_lst:
        # finds the asparagines that are in a helix
        if structure_data[element] == 'H':

            # find the structural information of the amino acids adjacent to
            # the asparagine
            sub_string = structure_data[element-2:element+3]
            # if there are only helixes in the sub string then the asparagine is
            # placed deep inside of a helix. therefor S3 = 1 , S1 = 0, S2 = 0
            if 'T' in sub_string or 'B' in sub_string:
                # count the 'H' in the string before and after the hit.
                # if there can only be 2 scenarios for each of the two sub_strings
                before_element = sub_string[:2].count('H')
                after_element = sub_string[-2:].count('H')
                # special case where S1 = 1
                if sub_string[:2] == 'HT' or sub_string[:2] == 'HB':
                    S1 = 1
                    S2 = 0
                # special case where S2 = 1
                elif sub_string[-2:] == 'TH' or sub_string[-2:] == 'BH':
                    S2 = 1
                    S1 = 0
                # if a helix is flanked by other structures
                elif before_element == after_element:
                    S1 = before_element
                    S2 = 0
                # if there are more helix structures upstream than downstream
                elif before_element <= after_element:
                    S1 = before_element + 1
                    S2 = 0
                # if there are more helix structures downstream than upstream
                elif after_element < before_element:
                    S2 = after_element + 1
                    S1 = 0
                S1_collect.append(S1)
                S2_collect.append(S2)
                S3_collect.append(0)
            # then S3 = 1, S1=0, S2=0
            else:
                S1_collect.append(0)
```

```
                S2_collect.append(0)
                S3_collect.append(1)
        # if the asparagine is not placed in a helix then S1,S2,S3 = 0
        else:
            S1_collect.append(0)
            S2_collect.append(0)
            S3_collect.append(0)
    return S1_collect, S2_collect, S3_collect

################################################################################

def S11S12_find(strukture_string):
    """ Function finds a value for ether S11 or S12.
    A op to 5 long string is inputted for upstream or downstream of a Asn.
    Function is called in function end_chain.
    """
    S = 0
    for char in strukture_string:
        if char == 'T':
            S += 1
        else:
            break
    return S

################################################################################

def end_chain(asn_pos_lst, structure_data):
    """ Function calculates S10-S12 from the structural data
    and from a list of Asn positions and a string of the structure_data
    """
    S10_collect, S11_collect, S12_collect = [], [], []
    STRUCTURE_LENGTH = len(structure_data)
    # loops through all the Asn residues found
    for asn_pos in asn_pos_lst:
        # looks at the last 20 amino acids in the peptide chain
        if asn_pos >= STRUCTURE_LENGTH - 20 and structure_data[asn_pos] == 'T':
            # the structure data from the residues +3 and -3 from the Asn
            downstream3 = structure_data[asn_pos:asn_pos+3]
            upstream3 = structure_data[asn_pos:asn_pos+3]

            # if the structures is 3 from the end of the chain and
            # contains only coil structures (T) then S10 = 1
            if ('TTT' in downstream3 or 'TTT' in upstream3):
                S10 = 1
                S10_collect.append(S10)
            # if the chain is further than 3 from helix or beta sheet
            # then S10 = 0 structures
            else:
                S10_collect.append(0)
            # the structure data from +5 and -5 of the Asn
            downstream = structure_data[asn_pos:asn_pos+5]
            upstream_rev = structure_data[asn_pos-1:asn_pos-6:-1]
            # find S11
            # looks at the secondary structures of the Asn and exceeding -5 on N-term
            # by calling function S11S12_find
            S11 = S11S12_find(upstream_rev)
            S11_collect.append(S11)
            # find S12
            # looks at the secondary structures of the Asn and following +5 on C-term
            # by calling function S11S12_find
            S12 = S11S12_find(downstream)
            S12_collect.append(S12)
        # if the Asn is not placed at the end of the chain and not in a coil
        # then S10,S11 and S12 are 0
        else:
```

```
            S10_collect.append(0)
            S11_collect.append(0)
            S12_collect.append(0)
    return S10_collect, S11_collect, S12_collect

########################  4) DEAMIDATION CALCULATOR  ############################

def half_time_calculator(S1_collect, S2_collect, S3_collect, S7_collect, S8_collect, S10_collect,
    S11_collect, S12_collect):
    """Function outputs a list of deamination half times for each glutamine and asparagine and
    a list of with information that can be converted to degradation
    """
    # calculation deamidation for each Gln residues from the primary structure
    ID_lst = [1 / CD for CD in sum_gln_data]

    # calculates the deamination half time for all Asn in the 3D structure
    CD_lst = []
    ASN_LIST_LENGTH = len(asn_pos_lst)
    for element in range(ASN_LIST_LENGTH):
        # if S5 = 0 a division by zero error is avoided by inserting a 1 in the denominator

        f = 0.48 * ( (1.0) * (S1_collect[element]) + (3.1) * (S2_collect[element])
                + 10 * (S3_collect[element])
                + 0.5 * (S7_collect[element]) + 3.2 * (S8_collect[element])
                + 2.0 * (1 - S10_collect[element]) + 0.26 * (5 - S11_collect[element])
                + 0.62 * (5 - S12_collect[element])
                )
        # calculate the individual halftime(CD) of each Asn
        prim_half_time = sum_asn_data[element]
        CD = (0.01) * prim_half_time * math.exp(f)
        CD_lst.append(CD)

        # prepares the individual
        ID_val = 1 / CD
        ID_lst.append(ID_val)
    return ID_lst

################################  MAIN SCRIPT  ##########################################################

# user defined PDB ID
pdb_id = input('Pleas write a PDB ID: ')
# loads PDB file from the web creates a list of the amino acids in the protein
aa_seq = load_pdb(pdb_id)
# caluclates the primary half time
(asn_pos_lst, sum_asn_data, sum_gln_data) = primary_half_time(aa_seq)
# sends a request to the DSSP program on the computer.
# DSSP predicts the secondary structure this is given as output
dssp_input = dssp_import()
# finds the S7-S8 values and finds structal_data
S7_collect, S8_collect, structure_data = HH_bond(dssp_input)
# finds the S values related to helix structe
S1_collect, S2_collect, S3_collect = helix_S(asn_pos_lst, structure_data)
# finds the S values related to the end of the chain
S10_collect, S11_collect, S12_collect = end_chain(asn_pos_lst, structure_data)

LENGTH = len(asn_pos_lst)
if ( len(S1_collect) != LENGTH and len(S2_collect) != LENGTH and
            len(S3_collect) != LENGTH and len(S7_collect) != LENGTH and
            len(S8_collect) != LENGTH and len(S10_collect) != LENGTH and
            len(S11_collect) != LENGTH and len(S12_collect) != LENGTH ):
    sys.stdout.write('The S-value lists are not the same length\n')
    sys.exit(1)

# calculates the prerequisites for the 3D half time of the protein
ID_lst = half_time_calculator(S1_collect, S2_collect, S3_collect, S7_collect, S8_collect,
```

```
    S10_collect, S11_collect, S12_collect)

# the total deamidation half time of total protein:
try:
    # calculates the half time of the protein if ID list is not zero
    half_time_of_total_protein = 100 / sum(ID_lst)

except ZeroDivisionError as error:
    sys.stdout.write('The peptide has no asparagine residues, therefor the half time cannot be calculated\n')
    sys.exit(1)

print('Protein half time is estimated to ', '%.1f' % half_time_of_total_protein, 'days')
```

Listing 1: Python source code