**CaMBDa** Handbook

**CaMBDa** (**Ca**lculate **M**etric: **BDa**mage) is a program written in Python 2.7.10 which calculates ***B*Damage** values for the atoms in biomolecular structures, when atomic information is input in PDB format.

NOTE: An historic version of the program written in MATLAB is also available from <https://github.com/td93/B_Damage>

**Please cite:** M. Gerstel, C. M. Deane and E.F. Garman. (2015). J. Synchrotron Radiation. **22**, 201-212 <http://dx.doi.org/doi:10.1107/S1600577515002131>

Contents

[Introduction 3](#_Toc439952845)

[Usage - Inputs 4](#_Toc439952846)

[Required Inputs: 4](#_Toc439952847)

[Optional Inputs: 4](#_Toc439952848)

[Location of Input 5](#_Toc439952849)

[Outputs 6](#_Toc439952850)

[<name>Bdamage.txt: 6](#_Toc439952851)

[<name>UnitCell.pdb: 6](#_Toc439952852)

[Description of Errors output by Python code: 7](#_Toc439952853)

# Introduction

For molecular structures determined using X-ray crystallography each atom in the structure is assigned an ***atomic B factor*** value. This value effectively represents our level of uncertainty about the allocated position of that atom. The most mobile atoms will have the highest B factor values. However an atom's mobility could be due to different factors: increased thermal motion due to absorbed energy from the incident X-ray photons (a sign of *radiation damage*), or it could simply be due to the atom being in a highly flexible region of the protein and it is not surrounded by other 'stationary' atoms. ***B*Damage** is a metric that attempts to deconvolute these factors to give an "effective" B factor value that is just a measure of the damage.

***B*Damage** for a given atom is the ratio of the atom's B factor and the average B factor of atoms within a *similar packing density*. Atoms are grouped into **similar packing density** environments, meaning that atoms in that group have a similar surrounding atomic environment which can be defined in several ways. The default here is that atoms with similar packing densities have a similar number of atoms within a given radius of the atom.

# Usage - Inputs

## Required Inputs:

(**CaMBDa** takes a minimum of one input, which directs the program to where the atomic information can be found.)

There are two options here, if the required input is a structure publicly available on the PDB (<http://www.rcsb.org>) then the four digit PDB accession code, enclosed in inverted commas, is sufficient to run the program. Provided the machine on which the program is running has access to an internet connection, the program will automatically retrieve the required data from the PDB, and download a local copy of the PDB file. The final input should look something like **cambda(‘####’)**

The other option is to provide the program with a file path pointing to a local file in PDB format (allowed file extensions are .pdb and .txt), regardless of whether this file has been deposited in the PDB or not. No internet connection is required to run the program with this kind of input. Once again, this should be enclosed in inverted commas. The final input should look something like **cambda(‘path/to/PDB/file/foo.pdb’)**

## Optional Inputs:

(**CaMBDa** can in addition take a number of optional inputs to vary the output of the programme. Unless you have a specific reason for changing these outputs, leaving them as the default is usually sufficient.)

If the **Packing Density Threshold** used by **CaMBDa** is to be varied, **PDT=##** (where ## is a number in ångströms) can be used. The program uses a default value of 14 (Å)

If the **Bin Size** used by **CaMBDa** is to be varied, **binSize=##** (where ## is a number of atoms) can be used. The program uses a default value of 10 (atoms)

The program can write PDB files for intermediate stages of PDB manipulation. This is a useful tool for debugging, but as a default these are not produced. (Note that the program will always produce a PDB file of the Unit Cell.) There are two options to produce further PDB files: **createAllUnitCellsPDB=True** and **createTrimmedAtomsPDB=True**.

**createAllUnitCellsPDB=True** will make the program write a PDB file which contains all the modelled atoms in a 3x3x3 Unit Cells ‘mini-crystal’.

**createTrimmedAtomsPDB=True** will make the program write a PDB file which contains all the modelled atoms in a cube defined as the orthogonal limits of the asymmetric unit, with an additional margin (equal to the **PDT**).

If a user does wish to produce these PDB files it should be noted that COOT will rarely – if ever – be able to open and view the set of atomic coordinates. PyMOL is the best program to use if these files are to be viewed and used, or alternatively CCP4MG if the filesize is not too great.

Optional inputs can be included using comma delimiters after the required input, within the braces.

An example final input with all optional inputs included would look like: **cambda(‘####’, PDT=##, binSize=##, createAllUnitCellsPDB=bool, createTrimmedAtomsPDB=bool)**

## Location of Input

It is currently the case that the input should be constructed on the line at the bottom of the file ‘**CalculateBdamage.py**’. There is currently a placeholder there in the default program which reads **cambda(‘2BN3’)**. This line can be edited using a Python development environment, or by using a text editor program (such as Notepad++). The program can then be run with this input either by running the script in a Python/iPython terminal (found within the development environment), or via the command line by navigating to the directory containing the downloaded Python script (e.g. C:\GitHub\B\_Damage\Python) then typing **Python CalculateBdamage.py** and pressing the **return** key. In either case, Python needs to be downloaded onto the machine.

# Outputs

(**CaMBDa** sends all output files to a folder called ‘Logfiles’**.** This parent folder will then contain subfolders for and named after each non-unique structure that the program has processed. In this folder (‘<*name>*’) will be a local copy of the input file in PDB format (‘*<name>.*pdb), ‘<*name>*UnitCell.pdb’ and ‘<*name>*Bdamage.txt’. There may also be optional files ‘AllUnitCells.pdb’ and ‘TrimmedAtoms.pdb’ depending on the inputs used when originally running the program.)

## <name>Bdamage.txt:

This file is the main output of the program. It contains a short header describing what each of the column headers means, then a table containing the output data for all the atoms in the asymmetric unit. These data are produced from a run of the program with the PDB structure ‘2BN3’. The data for each atom is stored in a similar manner as it is in a PDB file (in rows) and each column is defined at the top of the file. The column titled **BDAM** contains the B Damage value for each atom.

In addition, the MATLAB version of the program would output a histogram showing the numbers of atoms that are in a similar packing density environment. No such functionality currently exists in the Python release.

## <name>UnitCell.pdb:

This file is generated by PDBCUR. It contains the atomic information for all atoms from the input asymmetric unit, and all other asymmetric units which make up the Unit Cell. For the PDB structure ‘2BN3’, which has space group I2­13, this contains a total of 24 asymmetric units.

# Description of Errors output by Python code:

During execution of the code, a number of exceptions may arise. Here is a list of known exceptions that may arise, and potential reasons that they may occur. If you cannot find an exception you are experiencing in this list, please contact the author at <https://github.com/td93>

HTTP error: Likely that PDB code does not exist

1. Supplied filepath to PDB is neither a .pdb nor .txt file nor a PDB accession code
2. Supplied filepath does not exist
3. Failed to download and save PDB - cause unknown (check internet connection)
4. Failed to copy PDB to a local version. Check that supplied PDB is not in use by another program
5. Failed to generate Unit Cell PDB file. It is likely that PDBCUR failed to run to completion, or experienced an error