

← → ↺ 🏠 🌐 pypi.org/project/biochemHH/0.0.3/ ★ 📄 📁 🌐 ⋮

biochemHH 0.0.3

 [Latest release](#)

`pip install biochemHH==0.0.3` 

Released: Apr 16, 2026

Biochemistry & molecular biology helper tools

Navigation

-  Project description
-  Release history
-  Download files

Verified details

These details have been [verified by PyPI](#)

Maintainers

 [hsinmei](#)

Unverified details

These details have **not** been verified by PyPI

Meta

- **License:** GNU General Public License v2 (GPLv2) (GPL-2.0-only)

Project description

biochemHH

biochemHH is a Python package developed for experimental protein engineering.

It provides tools for:

- Batch-mode DNA cloning primer design, prioritizing primer reuse
- Protein-ligand interaction analysis
- Editing and processing of PDB, CIF, and GenBank (GB) files

It can be used for:

- Studying of an underinvestigated protein (**module Homolog**)
- Preparing PDB inputs for AI protein design tools (**module StructureHH**)
- Analyzing PDB outputs from AI protein design tools (**module StructureHH**)
- Constructing plasmids for experimental protein screening (**module GBparser**)

Quick Start (Installation & Accessing Example Files)

biochemHH is designed for wet lab scientists who may be new to programming. Ready-to-run example code blocks are provided to get started quickly.

Figure 1.

The PyPI project page of biochemHH (<https://pypi.org/project/biochemHH/>), an open-source Python package developed for computational protein engineering. The package provides modular tools for DNA cloning design, protein structure processing, and protein analysis, accompanied by runnable example scripts for users.

```

import os
from biochemHH import GBparser
from biochemHH.GBparser import reverse_complement
from biochemHH.GBparser import preferred_codon
from biochemHH.GBparser import preferred_codon_split
from biochemHH.GBparser import Read_gb
from biochemHH import Homolog
from biochemHH import StructureHH
import numpy as np

if __name__ == '__main__':

    # Set your own working directory (wd). Input and output files will be located in the working directory.

    # { ..... set working directory
    wd = r'C:\Users\ees26\Desktop\input'    # Replace this with your working directory path without Chinese characters
    os.chdir(wd)
    # }

    # { ..... Replace this empty block with a code block from an example script
    # }

```

Figure 2.

The installation includes a template file *playground.py* that handles package imports. Users can set their working directory and then copy and paste code blocks from the example files into this script to run them directly.

A.

This example file consist of the following parts:

1. Primer design

- Analyze a forward primer OR a primer pair
- Design primer pairs
- Analyze an oligo's secondary structure propensity
- Pick overlapping regions for primer walk
- Manually pick the annealing regions for golden gate cloning

2. Codon choice

- (supported species: 'ecoli', 'ecoliK', 'bacillus', 'yeast', 'pichia', 'mouse', 'hamster', 'human')
- Codon optimization
 - Codon diagnosis
 - Choose a degenerate codon
 - Create a dictionary of preferred codon

3. GB editing tools (BATCH)

- Remove primer annotations from all GB files
- Edit & annotate all GB files containing the target_seq
- Edit & annotate all GB files containing the target_seq in certain context
- Simple string replacement applied to all GB files

4. GB editing tools (SINGLE)

- Duplicate a GB file given a new filename
- Reverse the sequence
- Reindex the sequence
- Remove features
- Add features
- Display by range
- Display by feature
- Sequence editing by index
- Sequence editing by variation of an annotated CDS/gene

5. SEQ to Notation interconversion

- SEQ to Notation
- Notation to SEQ

6. Snippets useful in cloning design

- Check if the sequence contain BsaI site
- Update list_0 with list_1
- Batch-mode definition update
- Full GB display

B.

This example file consist of the following parts:

1. Print notes

2. Cloning design given an edited gb file

- Example 2a: design based on the indices of mutation sites or connecting points

3. GB file editing & cloning design

- Example 3a: mutate BsaI site on a vector
- Example 3b: subclone a gene onto a vector
- Example 3c: construct a fusion protein involving primer walk
- Example 3d: subclone a gene onto a vector & mutate BsaI site simultaneously

C.

This example file consist of the following parts:

0. Print notes

1. Cloning design (batch-mode)

- Example 1a: alanine scanning → [Demonstrated in figure 6](#)
- Example 1b: saturated mutagenesis → [Demonstrated in figure 7](#)
- Example 1c: cloning design based on a list of notations → [Demonstrated in figure 8 & 9](#)

D.

This example file consist of the following parts:

1. Edit a single PDB(CIF) file

- Duplicate a structure
- Extract certain residue (ligand) from a structure
- Substitute the nucleobase
- Rename chains
- Delete chains
- Edit residue: re-assign, rename, and keep all atoms
- Edit residue: remove certain atoms
- Edit chain: retain a list of residues
- Edit chain: remove a list of residues
- Duplicate a residue
- ATOM to HETATM (some AI tools only consider HETATM as ligand)
- Rotate atoms along an axis
- Confined random backbone rotation

2. Align & merge two PDB(CIF) files

- Chimera example 1, alignment by oligonucleotide
- Chimera example 2, alignment by amino acid
- Chimera example 3, alignment by atoms

E.

This example file contains the following code blocks:

- Print the indices of a certain amino acid
- Detect cysteine (reduced) and disulfide bond (oxidized)
- Load a structure and create a PyMOL script for visualization
- Shell-wise analysis of residue-level interactions → [Demonstrated in figure 4 & 5](#)

Figure 3.

The installation includes five example files, each containing a table of contents followed by the corresponding code blocks. Shown here are the tables of contents of

(A) "examples_DNA.py",

(B) "examples_Cloning_(advanced).py",

(C) "examples_Cloning_(batch-mode).py",

(D) "examples_PDB(CIF)_editing.py", and

(E) "examples_Protein_analyses.py".

A.

```
# { ..... Shell-wise analysis of residue-level interactions
filename = '5AXN.cif'
target_chain = 'A'
target_resi = [304]
coi = [] # To analyze against residues from all chains instead, set coi = [].
shell = 1
analyze_potential = False
D, d, n, N = StructureHH.Shell_analyses(wd, filename,
                                       target_chain=target_chain,
                                       target_resi=target_resi,
                                       coi=coi,
                                       shell=shell,
                                       analyze_potential=analyze_potential,
                                       potential_cutoff=5,
                                       combine_categories= None
                                       )
```

B.

```
create folder C:/Users/hsinmei/My Drive/YD script/Input 1/Input/input_Bond_matrices
writing bond matrix, please wait
5AXN_bond.csv created

..... 5AXN_shell_analyses
t           = "5AXN.cif"
target_chain = "A"
target_resi  = [304]
coi          = []
shell       = 1
analyze_potential = False
potential_cutoff = 5

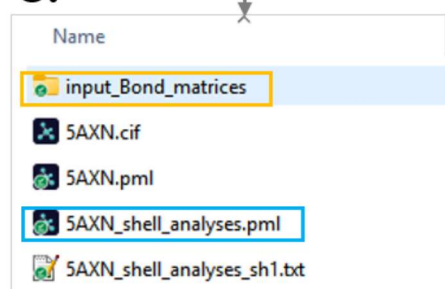
shell1:
  chain A [19, 114, 133, 303]
  chain B [83, 86]
  chain P [2, 72]

..... Paste the following line in PYMOL command prompt
@C:\Users\hsinmei\My Drive\YD script\Input 1\Input\5AXN_shell_analyses.pml

The run took 8.794899499975145 s
```

This folder is empty.

C.



D.

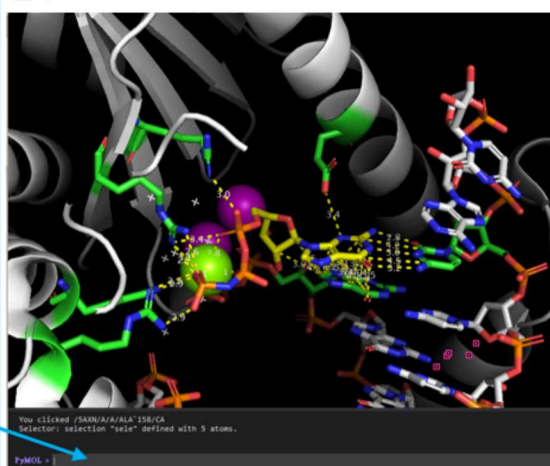


Figure 4. Demonstration of shell-wise analysis of residue-level interactions.

- (A) **Example code** for downloading *5AXN.cif* from RCSB database if it is not present in the working directory, generating an all-to-all bond matrix, identifying first-shell residues around the target residue (target_chain = "A"; target_resi = [304]), and creating PML scripts.
- (B) **Standard output after execution** (~9 s). The identified first-shell residues and a PyMOL command are printed to the console.
- (C) **Working directory after execution**, showing the generated *input_Bond_matrices* subfolder and PML scripts.
- (D) **Running the PyMOL command** loads the PML script and visualizes the interactions (yellow, target residue; green, first-shell residues).

A.

```
# { ..... Shell-wise analysis of residue-level interactions
filename = '5AXN.cif'
target_chain = 'A'
target_resi = [304]
coi = [] # To analyze against residues from all chains instead, set coi = [].
shell = 2
analyze_potential = False
D, d, n, N = StructureHH.Shell_analyses(wd, filename,
                                       target_chain=target_chain,
                                       target_resi=target_resi,
                                       coi=coi,
                                       shell=shell,
                                       analyze_potential=analyze_potential,
                                       potential_cutoff=5,
                                       combine_categories=None
                                       )
```

B.

```
5AXN bond_matrix available

..... SAXN_shell_analyses
t                = "5AXN.cif"
target_chain     = "A"
target_resi      = [304]
coi              = []
shell           = 2
analyze_potential = False
potential_cutoff = 5

shell1:
  chain A [19, 114, 133, 303]
  chain B [83, 86]
  chain P [2, 72]
shell2:
  chain A [24, 25, 26, 67, 70, 112, 126, 129, 130, 131, 135, 136, 137, 302, 401, 403, 404, 406]
  chain B [5, 8, 11, 80, 84, 87, 88, 89, 90]
  chain P [3, 71]

..... Paste the following line in PYMOL command prompt
@C:\Users\hsinmei\My Drive\YD script\Input 1\Input\5AXN_shell_analyses.pml

The run took 0.2626667999429628 s
```

C.

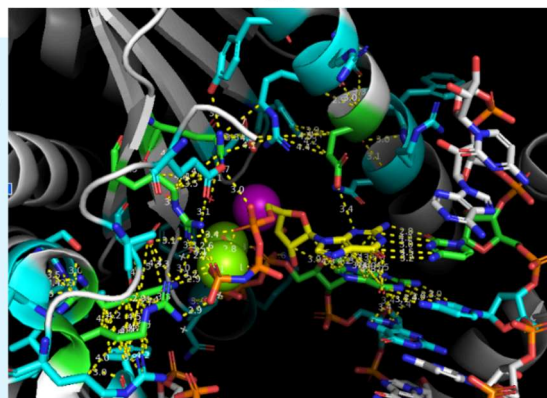


Figure 5. Subsequent shell-wise residue-level interaction analysis.

(A) Example code for extending the analysis to the second shell.

(B) Standard output after execution. The algorithm automatically detects and loads the 5AXN bond matrix stored in *input_Bond_matrices*, reducing the execution time to ~0.26 s.

(C) Running the PyMOL command loads the PML script and visualizes the interactions (yellow, target residue; green, first-shell residues; cyan, second-shell residues).

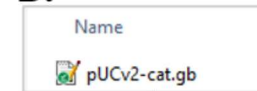
A.

```
# { ..... Example 1a: alanine scanning
GBparser.cloning_design_wrapper_1(
    base_filename='pUCv2-cat',          # template plasmid GB file
    feature_select='cat',                # gene name
    mutation_codon_no=[42, 45],         # residues to be mutated
    replacement_codon=preferred_codon('A', species='ecoli'), # mutate to Ala(A)
    plasmid_prefix='plasmid-',
    oligo_prefix='P',
    N_match_cutoff=2, YR_match_cutoff=4, GC_threshold=0,
    verbose=False)
GBparser.update_list('Oligo')
GBparser.update_list('Plasmid')
```

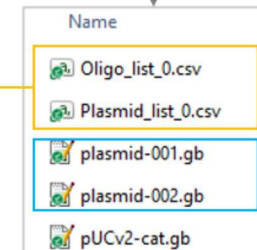
C.

===== Overview			
----- plasmid-001			
pUCv2-cat_T42A			
Primer pair	Template	Product (bp)	
P02, P01	pUCv2-cat	3044	
----- plasmid-002			
pUCv2-cat_L45A			
Primer pair	Template	Product (bp)	
P04, P03	pUCv2-cat	3044	

B.



D.



E.

Plasmid_list_0.csv

	A	B
1	Name	Definition
2	plasmid-001	pUCv2-cat_T42A
3	plasmid-002	pUCv2-cat_L45A

Oligo_list_0.csv

	A	B	C	D
1	Name	Length	Bases	Note
2	P01	27	aaggtcttcaTATCCAGCTGAACGGTCT	designed on 30-JUL-2026 for construction of plasmid-001
3	P02	43	aaggtctctgGATATTGCAGCCTTTTAAAGACCGTAAAGAAAA	designed on 30-JUL-2026 for construction of plasmid-001
4	P03	26	aaggtctcaAGGCCGTAATATCCAGC	designed on 30-JUL-2026 for construction of plasmid-002
5	P04	42	aaggtctcgGCTTTTGCAAGACCGTAAAGAAAAATAAGCAC	designed on 30-JUL-2026 for construction of plasmid-002

Figure 6. Demonstration of alanine scanning.

- (A) Example code** for independently mutating residues 40 and 45 (mutation_codon_no) of the *cat* gene (feature_select) in plasmid pUCv2-cat (base_filename) to alanine.
- (B) Before execution, the working directory should contain** the corresponding plasmid GenBank file *pUCv2-cat.gb* with the *cat* gene properly annotated.
- (C) Standard output after execution**, providing cloning instructions for each designed plasmid, for example, the blue-framed instruction indicates: "Create plasmid-001 by Golden Gate assembly of a 3044-bp PCR product amplified from pUCv2-cat using primers P02 and P01."
- (D) Working directory after execution.** If *plasmid_list_0.csv* and *oligo_list_0.csv* files do not exist, they are created automatically. GenBank (GB) files are generated for the designed plasmids.
- (E) The designed plasmids and cloning primers (oligos) are stored in CSV format.**

A.

```
# { ..... Example 1b: saturated mutagenesis
GBparser.cloning_design_wrapper_1(
    base_filename='pUCv2-cat',          # template plasmid GB file
    feature_select='cat',                # gene name
    mutation_codon_no=[45],              # residue to be mutated
    replacement_codon=preferred_codon_split('HKRS', species='ecoli'),
    plasmid_prefix='plasmid-',
    oligo_prefix='P',
    N_match_cutoff=2, YR_match_cutoff=4, GC_threshold=0,
    verbose=False)
GBparser.update_list('Oligo')
GBparser.update_list('Plasmid')
```

B.

Overview			
----- plasmid-003			
pUCv2-cat_L45H			
Primer pair	Template	Product (bp)	
P05, P03	pUCv2-cat	3044	
----- plasmid-004			
pUCv2-cat_L45K			
Primer pair	Template	Product (bp)	
P06, P03	pUCv2-cat	3044	
----- plasmid-005			
pUCv2-cat_L45R			
Primer pair	Template	Product (bp)	
P07, P03	pUCv2-cat	3044	
----- plasmid-006			
pUCv2-cat_L45S			
Primer pair	Template	Product (bp)	
P08, P03	pUCv2-cat	3044	

C.

Name
Oligo_list_0.csv
Plasmid_list_0.csv
plasmid-001.gb
plasmid-002.gb
plasmid-003.gb
plasmid-004.gb
plasmid-005.gb
plasmid-006.gb
pUCv2-cat.gb

D.

Plasmid_list_0.csv		Oligo_list_0.csv			
	A	B	A	B	C
1	Name	Definition	1	Name	Length
2	plasmid-001	pUCv2-cat_T42A	2	P01	27
3	plasmid-002	pUCv2-cat_L45A	3	P02	43
4	plasmid-003	pUCv2-cat_L45H	4	P03	26
5	plasmid-004	pUCv2-cat_L45K	5	P04	42
6	plasmid-005	pUCv2-cat_L45R	6	P05	42
7	plasmid-006	pUCv2-cat_L45S	7	P06	42
			8	P07	42
			9	P08	42

Figure 7. Subsequent saturated mutagenesis.

- (A) **Example code** for independently mutating residue 45 to histidine, lysine, arginine, and serine (abbr. H, K, R, S respectively). By specifying plasmid_prefix and oligo_prefix, newly designed plasmids and oligos are indexed sequentially following the last existing plasmid or oligo with the same prefix.
- (B) **Standard output after execution.** The algorithm automatically detects and reuses primer P03, previously designed and stored in *oligo_list_0.csv*.
- (C) **Working directory after execution.** If *plasmid_list_0.csv* and *oligo_list_0.csv* files already exist, they are updated. GenBank (GB) files are generated for the designed plasmids.
- (D) **The newly designed plasmids and oligos are appended to the existing CSV lists.**

A.

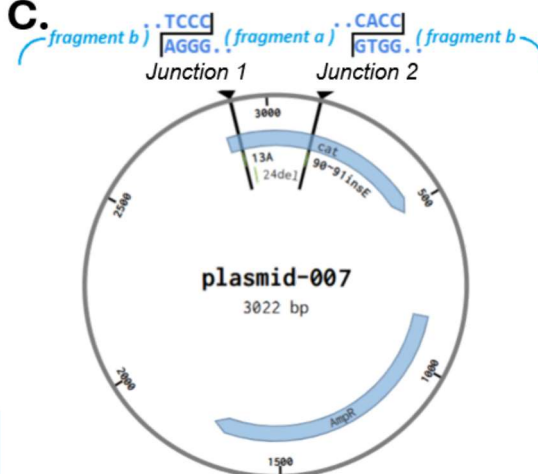
```
# { ..... Example 1c: cloning design based on a list of notations
notations = ['_13A_24del_90~91insE',
            '_44~45insAET_91~93del_140~142delinsAAG']

GBparser.cloning_design_wrapper_1(
    base_filename='pUCv2-cat',
    feature_select='cat',
    notations=notations,
    species='ecoli',
    plasmid_prefix='plasmid-',
    oligo_prefix='Q',
    N_match_cutoff=2, YR_match_cutoff=4, GC_threshold=0,
    verbose=False)
GBparser.update_list('Oligo')
GBparser.update_list('Plasmid')
```

B.

```
===== Overview
----- plasmid-007
pUCv2-cat_13A_24del_90~91insE
Primer pair Template Product (bp)
Q04, Q01 pUCv2-cat 2822 (fragment b)
Q02, Q03 pUCv2-cat 244 (fragment a)
----- plasmid-008
pUCv2-cat_44~45insAET_91~93del_140~142delinsAAG
Primer pair Template Product (bp)
Q08, P03 pUCv2-cat 2759
Q05, Q03 pUCv2-cat 169
Q06, Q07 pUCv2-cat 160
```

C.



D.

```
----- Golden gate annealing region
Selected design
combo: ['TCCC', 'CACC'],
overhangs: ['TCCC', 'CACC', 'GGGA', 'GTTG'],
overhangs_YR: ['YYYY', 'YRYY', 'RRRR', 'RRYR'],
max_N_match: 2,
max_YR_match: 3,
gc_content: [0.75, 0.75],
score: 6,
```

Figure 8.

Subsequent cloning design based on a list of mutation notations.

- (A) Example code for designing two plasmids.** The first encodes a mutated CAT protein with a glutamate inserted between residues 90 and 91 (*90~91insE*), deletion of residue 24 (*24del*), and substitution of residue 13 with alanine (*13A*). The second encodes a mutated CAT protein with residues 140–142 replaced by alanine–alanine–glycine (*140~142delinsAAG*), deletion of residues 91–93 (*91~93del*), and insertion of alanine–glutamate–threonine between residues 44 and 45 (*44~45insAET*). `oligo_prefix` is reset to 'Q' to distinguish oligos designed in this run.
- (B) Standard output after execution.** The algorithm automatically detects and reuses primers stored in `oligo_list_0.csv` (P03) as well as primers generated in the same run (Q03)
- (C) Plasmid-007 is assembled from two PCR fragments** joined at junction 1 and junction 2. See Figure 9 for details.
- (D) Assembly accuracy is determined by the orthogonality of the annealing ends (overhangs).** In this example, the maximum pairwise identity among the four overhangs is 2 (`max_N_match`). To account for ligase tolerance of noncanonical pyrimidine–purine pairing, nucleotide sequences (A/T/C/G) are converted to Y/R representation and analyzed similarly (`max_YR_match`).

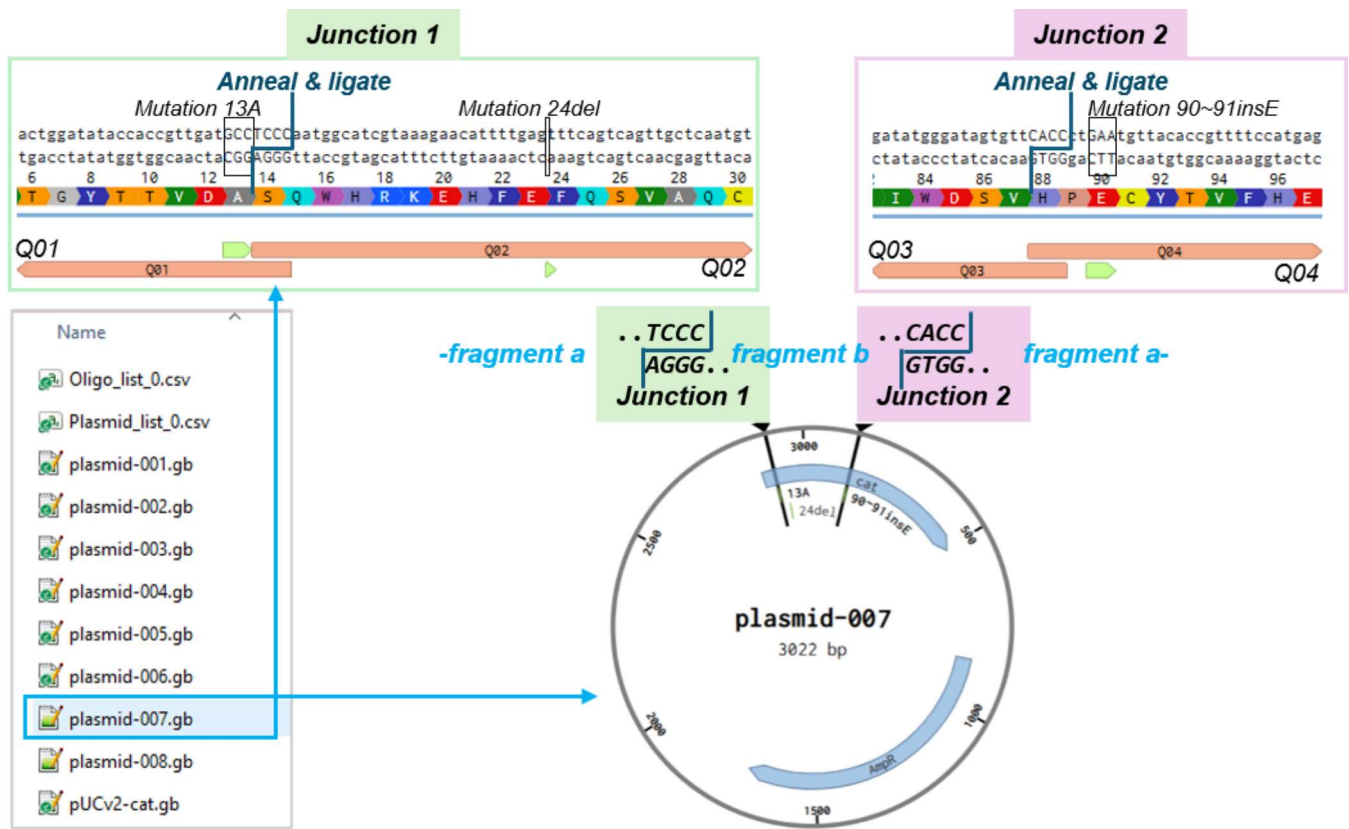


Figure 9. Assemble plasmid-007 from two PCR fragments.

GenBank (GB) files can be visualized using various sequence visualization software (here, Benchling). Because mutations are introduced by PCR primers, they are placed at fragment junctions. The three mutation sites (13, 24, and 90–91) are grouped into two mutation regions (13–24 and 90–91), each assigned a fragment junction. Consequently, plasmid-007 is assembled from two PCR fragments.