Universitat Pompeu Fabra Barcelona

PockMan

Tutorial

Wael Badr, Narine Fischer, Radostina Kislava

Master in Bioinformatics for Health Sciences
Universitat Pompeu Fabra
Structural Bioinformatics and Introduction to Python joint project
April 2025

1. What is PockMan?

PockMan is a bioinformatics tool that detects and analyzes potential binding pockets on protein structures. It uses a voxel-based grid projection system to analyze protein surface features and detect potential binding sites. It works with local `.pdb` files or fetches them directly from the RCSB Protein Data Bank using a PDB ID. The tool outputs structural and visual data suitable for downstream analysis in PyMOL or Chimera.

---

2. Installation Instructions

2.1 Clone from GitHub

The project is hosted on GitHub. To work with it, you should first install it locally.

```
> git clone https://github.com/rsk170/PockMan.git
```

```
> cd PockMan
```

2.2 Install the package with pip

This tells pip to build and install the package directly from your current directory in one step.

```
> pip install -e .
```

2.3 Install dependencies

Make sure Python 3 is installed. Recommended dependencies are automatically installed (e.g. `numpy`, `tqdm`, `biopython`, `requests`). In case they are not automatically installed run this on the terminal:

```
> pip install numpy biopython tqdm requests setuptools
```

---

3. Additional commands

3.1 Upgrading the package

To update to the latest version:

```
> pip install --upgrade .
```

3.2 Uninstalling the package

```
> pip uninstall Pockman
```

---

4. Using PockMan: Example Analysis Walkthrough

4.1 Arguments

Pockman allows the user to input the following parameters:

Parameter	Required	Explanation
PDB file	Yes	Path to a local .pdb file or a valid PDB ID (e.g., 2RH1). If a PDB ID has been provided, the program will download the specified PDB files in a folder pdb_files/.
Grid size	No	Sets the resolution of the grid in Angstroms. If not set, it defaults to 1.0
Border	No	Adds extra space around the protein for grid calculation. If not set, it defaults to 5.0.

Diagonals	No	Include diagonal pocket detection for a more comprehensive search. Defaults to False.
Voxel score cut-off	No	Score cut-off to know which voxels take into account for clustering into pockets. Defaults to 4.0
Distance Threshold	No	Maximum distance accepted between voxel and atom to assume that the atom is part of the predicted ligand binding site. Defaults to 4.0

If the user does not provide any value for the optional parameters, we use predetermined default values which give the most optimal output.

These optional parameters allow the user to customize the pocket detection algorithm, especially when working with large or complex proteins.

4.2 Interactive mode (preferred)

To get started with PockMan, navigate to the project directory and simply run in your terminal:

```
> pockman
```

The program will iteratively start asking you to input the required and optional parameters.

Step 1: Select the PDB file to work with

The file can be a local .pdb file or a valid 4-character PDB ID (e.g., 1abc), which the program will fetch from the Protein Data Bank. If a local file is provided, it will be parsed directly. If a PDB ID is provided, PockMan will automatically download the corresponding structure. The pdb files can be downloaded manually from RCSB PDB.

- **Use a local pdb file:**

```
➤ Enter the path to the input PDB file (or a 4-character PDB ID):
1e28.pdb
```

- **Use a PDB ID which will be downloaded automatically:**

```
➤ Enter the path to the input PDB file (or a 4-character PDB ID): 1e28
```

Step 2: Enter the grid size

Enter a number in the allowed range [0.5-3]. If you leave it blank (press “Enter”), the default value of 1 will be used.

```
➤ Enter grid size in Å (default 1.0, allowed 0.5 - 3.0): 1 (or leave blank)
```

In you enter a number outside of the allowed range, you will get an error message:

```
➤ Enter grid size in Å (default 1.0, allowed 0.5 - 3.0): 4

✘  4.0 Å is outside the allowed range [0.5 Å–3.0 Å].

⚠️  The grid size score must be at most 3.0 Å.
```

If your input is not a digit, you will get:

```
✘ Please enter a number.
```

The program will not quit but ask you to try again:

```
➤ Enter grid size in Å (default 1.0, allowed 0.5 - 3.0):
```

Step 3: Enter the border size

Enter a number in the allowed range [2-10]. If you leave it blank (press “Enter”), the default value of 5 will be used.

```
➤ Enter border size in Å (default 5.0, allowed 2.0-10.0): 6 (or leave blank)
```

In you enter a number outside of the allowed range, you will get an error message:

```
➤ Enter border size in Å (default 5.0, allowed 2.0-10.0): 11
```

```
✘  11.0 Å is outside the allowed range [2.0 Å–10.0 Å].

⚠  The border size score must be at most 10.0 Å.
```

If your input is not a digit, you will get:

```
✘ Please enter a number.
```

The program will not quit but ask you to try again:

```
➤ Enter border size in Å (default 5.0, allowed 2.0-10.0):
```

Step 4: Include planar diagonal PSP detection

You must enter “yes” or “no” for this option. If you leave it blank (press “Enter”), the default value will be “no”.

```
➤ Include planar diagonal PSP detection? (yes/no):  yes
```

In you enter something other than “yes”, “no”, “y” or “n”, you will get an error message:

```
➤ Include planar diagonal PSP detection? (yes/no): 28983

✘ Please answer yes or no.
```

The program will not quit but ask you to try again:

```
➤ Include planar diagonal PSP detection? (yes/no):
```

Step 5: Enter voxel score cut-off

The allowed range changes based on whether you are including diagonals or not. If you have included diagonals, the allowed range is [1-12]. If you are not using diagonals, the allowed range is [1-6]. If you leave it blank (press “Enter”), the default value of 4 will be used.

In this case, we have answered “yes” to the previous question about including diagonal detection so we get:

```
➤ Enter the cut-off score desired to select voxels for the pocket
```

```
clustering (default 4, allowed 1-12): 5 (or leave blank)
```

In you enter a number outside of the allowed range, you will get an error message:

```
➤ Enter the cut-off score desired to select voxels for the pocket
clustering (default 4, allowed 1-12): 0

✘  0.0 is outside the allowed range [1–12].

⚠  The cut-off score must be at least 1.
```

If your input is not a digit, you will get:

```
✘ Please enter a number.
```

The program will not quit but ask you to try again:

```
➤ Enter the cut-off score desired to select voxels for the pocket
clustering (default 4, allowed 1-12):
```

Step 6: Enter distance threshold

Enter a number in the allowed range [3-8]. If you leave it blank (press “Enter”), the default value of 4 will be used.

```
➤ Enter the distance threshold to determine closeness between voxel
and atoms (default 4, allowed 3-8): 4 (or leave blank)
```

In you enter a number outside of the allowed range, you will get an error message:

```
➤ Enter the distance threshold to determine closeness between voxel
and atoms (default 4, allowed 3-8): 9

✘  9.0 is outside the allowed range [3–8].

⚠  The distance score must be at most 8.
```

If your input is not a digit, you will get:

```
✘ Please enter a number.
```

The program will not quit but ask you to try again:

```
➤ Enter the distance threshold to determine closeness between voxel and atoms (default 4, allowed 3-8): 9
```

4.3 Command Line Execution

Instead of interactive mode, you could also execute the exact same thing using a command in the following way.

To run PockMan from the command line, use the following command structure:

```
> pockman <pdb_input> [OPTIONS]
```

Optional Arguments

Find the flags for each of the parameters below:

Parameter	**Flag**
Grid size	`--grid_size`
Border	`--border`
Diagonals	`--diagonals`
Voxel score cut-off	`--voxel_score_cut_off`
Distance Threshold	`--distance_threshold`

Use -h, -help to show the help message.

Example command line execution

- **Using a local pdb file:**

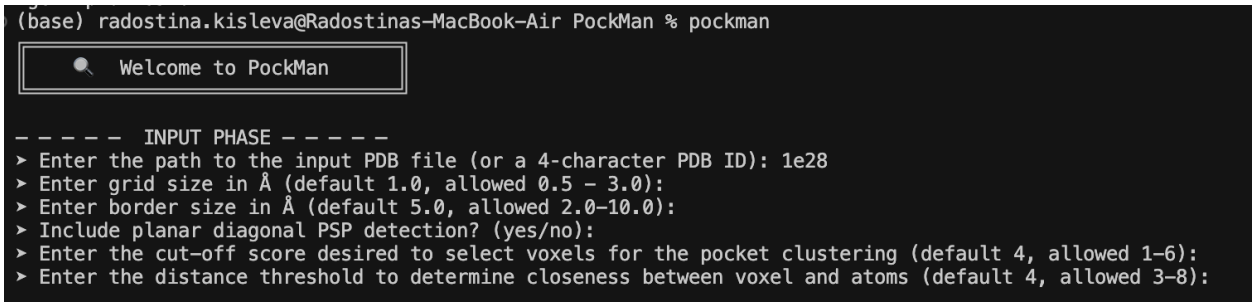
```
> pockman 1a28.pdb --grid_size 1.0 --border 5.0 --diagonals --voxel_score_cut_off 4 --distance_threshold 4
```

- **Using a PDB ID and download it automatically:**

```
> pockman 1a28 --grid_size 1.0 --border 5.0 --diagonals
--voxel_score_cut_off 4 --distance_threshold 4
```

If you don't provide any of the optional flags, the default values described in 4.1 will be used. will be used.

4.4 Input summary



In this example run, this is what we are inputting:

- **Protein structure**: 4-character PDB ID (1e28) which will be downloaded by the program.
- **Grid resolution**: Left blank to use the default value (1).
- **Border size**: Left blank to use the default value (5).
- **Diagonal detection**: Left blank to use the default value (no).
- **Cut-off score:** Left blank to use the default value (4.0).
- **Distance threshold:** Left blank to use the default value (5).

---

5. Output Structure

5.1 Processing Steps

Step 1: PDB download and cleaning

Option A: Local PDB File

If you are using a local PDB file, you will get the following output:

```
- - - - -  PDB FETCH PHASE - - - - -
```

```
🧹Cleaning local PDB file: /Users/radostina.kisleva/Downloads/1e28.pdb

✅ Cleaned PDB file saved to:
/Users/radostina.kisleva/Downloads/1e28_clean.pdb
```

We take the local PDB file and perform “cleaning”. The file is rewritten so that it contains only lines that begin with ATOM, i.e. the standard protein (and nucleic-acid) coordinates. Everything else - HETATM records such as waters, ligands, metal ions, and all remark/header lines, is stripped out, leaving a file with just the protein backbone and side-chain atoms. The cleaned PDB is saved in the same directory as the original PDB but has an added “_cleaned” in the name.

Option B: Download the PDB File

If a local file is not found, PockMan attempts to fetch the PDB entry online:

```
- - - - -  PDB FETCH PHASE - - - - -

📥 Assuming '1e28' is a PDB ID and attempting to download...

✅ Downloaded PDB 1e28 → pdb_files/1e28_raw.pdb

🧹Cleaning local PDB file: pdb_files/1e28.pdb

✅ Cleaned PDB file saved to: pdb_files/1e28.pdb
```

The downloaded file is cleaned as well and saved locally in the `pdb_files/` directory. The original file will be named `1e28_raw.pdb`. During the cleaning, it is rewritten so that it contains only lines that begin with ATOM, i.e. the standard protein (and nucleic-acid) coordinates. Everything else - HETATM records such as waters, ligands, metal ions, and all remark/header lines, is stripped out, leaving a file with just the protein backbone and side-chain atoms.

The following output will appear when you’re using a PDB ID which you have used before. The file will not be downloaded again but since it was already saved on your local machine, the same file will be used:

```
- - - - -  PDB FETCH PHAS - - - - -
```

```
📥 Assuming '1e28' is a PDB ID and attempting to download...

🗄️  PDB file already exists: pdb_files/1e28.pdb
```

Step 2: Analysis step

```
- - - - -  ANALYSIS PHASE - - - - -

🔬 Starting analysis…
```

- Grid projection

```
Protein grid shape: (72, 66, 83)

✅ Protein grid projection completed.
```

A 3D voxel grid is created to represent the protein's spatial volume. The dimensions `(x, y, z)` reflect the size of the protein plus the border. Protein atoms are projected into this grid for structural analysis.

- Pocket detection

```
Detecting pockets:
100%|████████████████████████████████████████████████████████████|
153000/153000 [00:11<00:00, 12946.67it/s]

✅ Protein detection completed.
```

The program evaluates each voxel for pocket likelihood. A progress bar indicates real-time progress so that the user is informed of the time it will take.

- Pocket clustering and scoring

```
✅ Pocket clustering and scoring completed.
```

In this step, Identified pockets are ranked by structural and geometric features.

- Nearby atom detection

```
✅ Nearby atom detection completed.
```

Nearby residues and atoms are associated with each pocket.

Step 3: Output files step

- Binding sites files saved

```
📂 Individual binding sites files saved to:
results/1e28/binding_sites/

📄 General binding sites file saved to:
results/1e28/binding_sites/Ligand_binding_site_1e28.txt
```

The software separates the output into two kinds of files:

1. Individual binding site text files:

For each predicted pocket (or binding site), a separate file is created (e.g., “Ligand_binding_site_{pdb_id}_1.txt”, “Ligand_binding_site_{pdb_id}_2.txt”, etc.)

These files contain the details (such as the list of atoms) that make up each predicted ligand-binding site.

2. General binding site file:

This file (named “Ligand_binding_site_{pdb_id}.txt”) is a combined report that concatenates the individual binding site outputs, giving you an overall summary of all predicted pockets.

The individual files break down the prediction per pocket, while the general file aggregates all the predictions together.

- Chimera files saved

```
📂 Chimera command scripts for  individual pockets saved to: results/1e28/chimera/

📄 Chimera combined command script saved to: results/1e28/chimera/1e28_chimera.cmd
```

The individual Chimera files are per-pocket command scripts. Each one (e.g. "1a28_1_chimera.cmd", "1a28_2_chimera.cmd", etc.) contains commands to load the PDB, display the surface, and color the residues corresponding only to that specific binding site (pocket).

The general Chimera file (e.g. "1a28_chimera.cmd") combines the commands for all pockets into a single script so that when loaded it will set up the visualization for all predicted binding sites at once.

- PyMol files saved

```
📂 PyMOL command scripts for individual pockets saved to: results/1e28/pymol/

📄 PyMOL combined script saved to: results/1e28/pymol/1e28_pymol.pml
```

The individual PyMOL files are per-pocket command scripts. For each predicted binding site (pocket), a separate script (e.g. “{pdb_id}_1_pymol.pml”, “{pdb_id}_2_pymol.pml”, etc.) is generated with commands tailored to visualize just that pocket (for example, setting a specific color, applying a selection for the residues, and showing the surface).

The general PyMOL file (e.g. “{pdb_id}_pymol.pml”) is an aggregate script that combines the commands for all pockets. When you run this combined script in PyMOL, it loads the PDB file and then applies the visualization commands for every pocket sequentially.

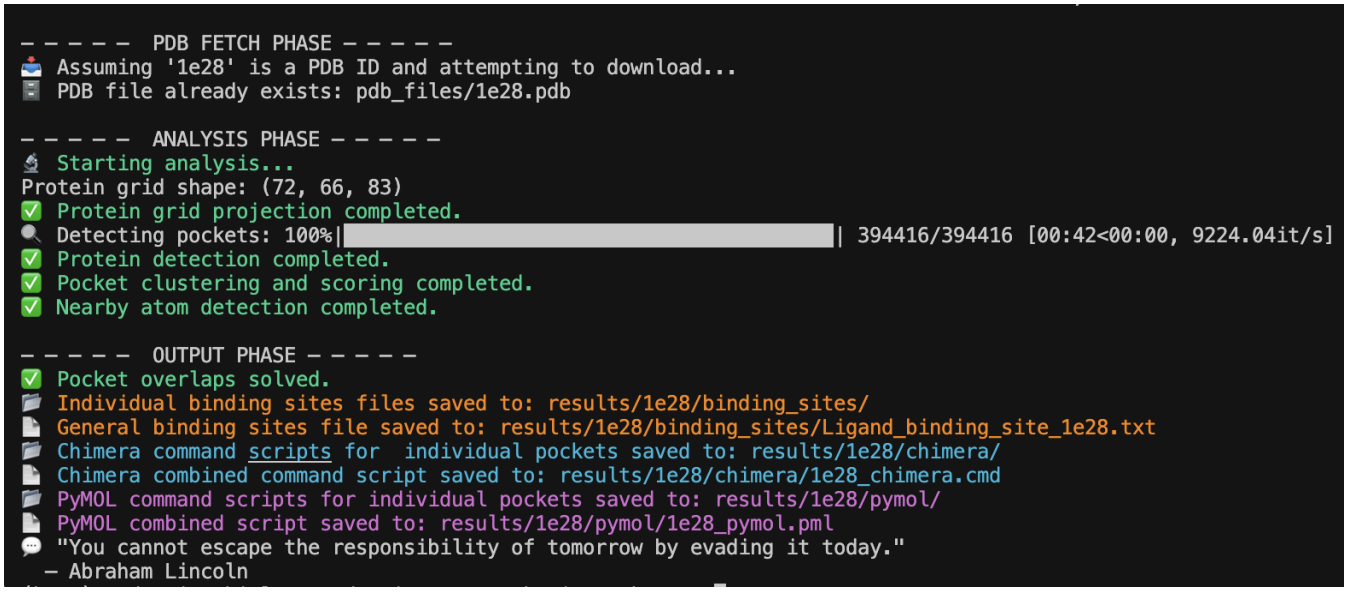
- Motivational quote

```
💬 "You cannot escape the responsibility of tomorrow by evading it today."

   — Abraham Lincoln
```

At the end of each session, PockMan provides a random motivational quote—because science should also lift your spirits. ✨

5.2 Summary



---

6. Output files

The files with results are stored in the “results/” directory of the project. It saves the resulting files for each run that you perform. In the results directory, there are subdirectories for each of the PDB IDs used. Further down, in the directory for each PDB, there are three folders - “chimera”, “pymol” and “binding sites”. Each of them contain the corresponding files.

For example:

results/

├── **1a28/**

│ ├── binding_sites/

│ │ ├── Ligand_binding_site_1a28.txt

│ │ ├── Ligand_binding_site_1a28_1.txt

│ │ ├── Ligand_binding_site_1a28_2.txt

│ │ └── ...

```
│   ├── chimera/
│   │   ├── 1a28_1_chimera.cmd
│   │   ├── 1a28_2_chimera.cmd
│   │   └── 1a28_chimera.cmd
│   └── pymol/
│       ├── 1a28_1_pymol.pml
│       ├── 1a28_2_pymol.pml
│       └── 1a28_pymol.pml
├── 2a28/
│   ├── binding_sites/
│   │   ├── Ligand_binding_site_2a28.txt
│   │   ├── Ligand_binding_site_2a28_1.txt
│   │   └── ...
│   ├── chimera/
│   │   ├── 2a28_1_chimera.cmd
│   │   └── 2a28_chimera.cmd
│   └── pymol/
│       ├── 2a28_1_pymol.pml
│       └── 2a28_pymol.pml
├── 1e28/
│   ├── binding_sites/
│   ├── chimera/
│   └── pymol/
├── 1ppb/
```

```
│   ├── binding_sites/
│   ├── chimera/
│   └── pymol/
└── 1bbp_clean/
    ├── binding_sites/
    ├── chimera/
    └── pymol/
```

To summarize, the following files are generated during a typical run:

File name	**Description**
pdb_files/3htb_raw.pdb	Raw PDB file downloaded from the RCSB PDB.
pdb_files/3htb.pdb	Cleaned PDB file used for analysis (kept only ATOM rows).
results/1e28/binding_sites/Ligand_binding_site_1e28_{binding site number}.txt	For each predicted pocket (or binding site), a separate file is created (e.g., “Ligand_binding_site_1e28_1.txt”, “Ligand_binding_site_1e28_2.txt”, etc.) These files contain the details that make up each predicted ligand-binding site.
results/1e28/binding_sites/Ligand_binding_site_1a28.txt	A combined report that concatenates the individual binding site outputs, giving you an overall summary of all predicted pockets.
results/1e28/chimera/1a28_{binding site number}_chimera.cmd	Per-pocket command scripts. Each one (e.g. "1a28_1_chimera.cmd", "1a28_2_chimera.cmd", etc.) contains

	commands to load the PDB, display the surface, and color the residues corresponding only to that specific binding site (pocket).
results/1e28/chimera/1a28_chimera.cmd	Combines the commands for all pockets into a single script so that when loaded it will set up the visualization for all predicted binding sites at once.
results/1e28/pymol/1e28_{binding site number}_pymol.pml	The individual PyMOL files are per-pocket command scripts. For each predicted binding site (pocket), a separate script (e.g. “{pdb_id}_1_pymol.pml”, “{pdb_id}_2_pymol.pml”, etc.) is generated with commands tailored to visualize just that pocket (for example, setting a specific color, applying a selection for the residues, and showing the surface).
results/1e28/pymol/1e28_pymol.pml	An aggregate script that combines the commands for all pockets. When you run this combined script in PyMOL, it loads the PDB file and then applies the visualization commands for every pocket sequentially.

---

7. Visualizing the Results

PockMan generates two ways of visualizing the results, one with Chimera, and another with PyMOL.

7.1 Chimera Visualisation

For Chimera, PockMan gives command files for visualizing, these can be found as:

`(PDB ID)_chimera.cmd`

`(PDB ID)_(binding site number)_chimera.cmd`

Therefore, two types of documents can be found, a general one for visualizing all ligand-binding sites predicted; and individual files per site. An example of these two files are:

`1e28_chimera.cmd`

`1e28_1_chimera.cmd`

Once these files are obtained, for visualizing them in Chimera:

- Open Chimera
- Go to File → Open…
- Go to the folder with the command files
- Select the one you have to visualize

There is no need to previously load the protein, as the code in the command files is prepared to open it by itself. Afterwards, the command file will load the protein, showing the surface of it, and the predicted sites (or individual predicted site) will be coloured depending on the PockMan score obtained during the analysis.

7.2 PyMOL Visualisation

In PyMOL, PockMan also provides pre-written script files to visualize the predicted binding pockets. These files follow a similar naming convention to the Chimera ones, such as:

`(PDB ID)_pymol.pml` – for visualizing **all predicted sites**

`(PDB ID)_(binding site number)_pymol.pml` – for **individual site visualization**

To use them:

- Open PyMOL.
- Go to `File → Run...` and select the desired `.pml` file.

- The predicted pockets will be automatically highlighted.

Additionally, PyMOL allows interactive inspection of each pocket when loading all predicted sites. By clicking on a predicted site, you can see which residues are involved, making it easy to explore the spatial relationships and validate the predictions in more detail.
