pyvinecopulib

website up License MIT Build Status failing DOI 10.5281/zenodo.14841456

Introduction

What are vine copulas?

Vine copulas are a flexible class of dependence models consisting of bivariate building blocks (see e.g., Aas et al., 2009). You can find a comprehensive list of publications and other materials on vine-copula.org.

What is pyvinecopulib?

pyvinecopulib is the python interface to vinecopulib, a header-only C++ library for vine copula models based on Eigen. It provides high-performance implementations of the core features of the popular VineCopula R library, in particular inference algorithms for both vine copula and bivariate copula models. Advantages over VineCopula are

- a stand-alone C++ library with interfaces to both R and Python,
- a sleaker and more modern API,
- shorter runtimes and lower memory consumption, especially in high dimensions,
- nonparametric and multi-parameter families.

License

pyvinecopulib is provided under an MIT license that can be found in the LICENSE file. By using, distributing, or contributing to this project, you agree to the terms and conditions of this license.

Contact

If you have any questions regarding the library, feel free to open an issue or send a mail to info@vinecopulib.org.

Installation

With pip

The latest release can be installed using `pip`:

```
pip install pyvinecopulib
```

With conda

Similarly, it can be installed with `conda`:

```
conda install conda-forge::pyvinecopulib
```

Or with `mamba`:

```
mamba install conda-forge::pyvinecopulib
```

From source

Start by cloning this repository, noting the `--recursive` option which is needed for the `vinecopulib` and `wdm` submodules:

```
git clone --recursive https://github.com/vinecopulib/pyvinecopulib.git
cd pyvinecopulib
```

The main build time prerequisites are:

- scikit-build-core (>=0.4.3),
- nanobind (>=2.7.0),
- a compiler with C++17 support.

To install from source, `Eigen` and `Boost` also need to be available, and CMake will try to find suitable versions automatically.

The recommended way to install `pyvinecopulib` from source is to use `conda` or `mamba`. A reproducible environment, also including requirements for the `pyvinecopulib`'s development and documentation, can be created using:

```
python scripts/generate_requirements.py --format yml # from pyvinecopulib's root
mamba env create -f environment.yml
mamba activate pyvinecopulib
```

Alternatively, you can specify manually the location of `Eigen` and `Boost` using the environment variables `EIGEN3_INCLUDE_DIR` and `Boost_INCLUDE_DIR` respectively. On Linux, you can install the required packages and set the environment variables as follows:

```
sudo apt-get install libeigen3-dev libboost-all-dev
export Boost_INCLUDE_DIR=/usr/include
export EIGEN3_INCLUDE_DIR=/usr/include/eigen3
```

Finally, you can build and install `pyvinecopulib` using `pip`:

```
pip install .
```

Stubs and documentation can then be generated using the custom scripts:

```
python scripts/generate_metadata.py --env pyvinecopulib
```

Or use the Makefile for convenience:

```
make metadata    # Generate all (stubs, docstrings, examples)
make stubs       # Generate type stubs only
make docstrings  # Generate C++ docstrings only
```

Note that the `generate_requirements.py` script can also be used to generate a `requirements.txt` file for use with `pip` via the `--format` option:

```
python scripts/generate_requirements.py --format txt
```

Building the documentation

Documentation for the example project is generated using Sphinx and the "Read the Docs" theme. The following command generates HTML-based reference documentation; for other formats please refer to the Sphinx manual:

```
cd docs
python serve_sphinx.py
```

Development

This project includes comprehensive development tools including pre-commit hooks and a Makefile to streamline development workflow.

Quick Development Setup

1. **Clone and setup environment**:

```
git clone --recursive https://github.com/vinecopulib/pyvinecopulib.git
cd pyvinecopulib
make env-conda                 # Create conda environment
conda activate pyvinecopulib   # Activate environment
```

2. **Setup development tools**:

```
make dev-setup                 # Install dependencies and pre-commit hooks
```

3. **Development workflow**:

```
make quick-check               # Run fast checks (lint, type-check, test)
make check-all                 # Run comprehensive checks before commit
```

Development Commands

Use `make help` to see all available commands. Key commands include:

Command	Description
`make install-dev`	Install development dependencies
`make test`	Run all tests

Command	Description
`make test-fast`	Run tests without coverage
`make test-examples`	Run example notebooks
`make lint`	Run code linting with ruff
`make format`	Format code with ruff
`make type-check`	Run type checking with mypy
`make docs`	Build documentation
`make docs-serve`	Serve documentation locally
`make clean`	Clean build artifacts
`make stubs`	Generate type stubs (custom script)
`make docstrings`	Generate C++ docstrings
`make metadata`	Generate all metadata (stubs, docstrings, examples)
`make examples`	Process and execute example notebooks
`make clear-cache`	Clear Python cache files

Pre-commit Hooks

Pre-commit hooks automatically run code quality checks before each commit:

- **Ruff**: Python linting and code formatting
- **MyPy**: Type checking with project configuration
- **Clang-format**: C++ code formatting (src/ directory only)
- **CMake-format**: CMake file formatting
- **General hooks**: Trailing whitespace, YAML/TOML validation, etc.

Install hooks with:

```
make pre-commit-install
```

Run manually on all files:

```
make pre-commit
```

Development Workflow

1. **Start new feature/fix**:

   ```
   git checkout -b feature/my-feature
   ```

2. **During development** (run frequently):

   ```
   make quick-check                  # Fast feedback loop
   ```

3. **Before committing**:

```
make check-all                    # Comprehensive quality checks
git add .
git commit -m "Add new feature"   # Pre-commit hooks run automatically
```

Code Style Guidelines

- **Python**: Follow PEP 8, enforced by ruff
- **C++**: Follow Google style guide, enforced by clang-format
- **Type hints**: Required for all Python code
- **Documentation**: Use docstrings for all public functions

Testing

- **All tests**: `make test`
- **Fast tests**: `make test-fast` (for quick development feedback)
- **Example notebooks**: `make test-examples`
- **Performance benchmarks**: `make benchmark`

Environment Management

The project uses conda for environment management. The Makefile automatically detects conda environments:

```
make env-conda                      # Create new environment
conda activate pyvinecopulib        # Activate environment
make env-update                     # Update existing environment
make update-deps                    # Update dependency files
```

Release Process

Before releasing, run comprehensive checks:

```
make release-check
```

This ensures all tests pass, documentation builds correctly, and examples work.

Troubleshooting

- **Build issues**: `make debug-build`
- **Installation issues**: `make debug-install`
- **Project status**: `make status`
- **Clean everything**: `make git-clean` (⚠️ destructive)

Development Tips

- Use `make quick-check` frequently during development for fast feedback
- Pre-commit hooks automatically fix many formatting issues
- Run `make check-all` before pushing changes to ensure quality

- Use `make metadata` to regenerate stubs and docstrings after C++ changes

- The project uses custom scripts in `scripts/` for stub generation (not nanobind’s default)

- Keep commits focused and write clear commit messages
- Add tests for new functionality