Troubleshooting

> **Note**
>
> Since this information may be updated regularly, please ensure you are viewing the most up-to-date version.

ImportError

In certain cases a failed installation or setup issue can cause you to see the following error message:

```
IMPORTANT: PLEASE READ THIS FOR ADVICE ON HOW TO SOLVE THIS ISSUE!

Importing the numpy c-extensions failed. This error can happen for
different reasons, often due to issues with your setup.
```

The error also has additional information to help you troubleshoot:

- Your Python version
- Your NumPy version

Please check both of these carefully to see if they are what you expect. You may need to check your `PATH` or `PYTHONPATH` environment variables (see Check Environment Variables below).

The following sections list commonly reported issues depending on your setup. If you have an issue/solution that you think should appear please open a NumPy issue so that it will be added.

There are a few commonly reported issues depending on your system/setup. If none of the following tips help you, please be sure to note the following:

- how you installed Python
- how you installed NumPy
- your operating system
- whether or not you have multiple versions of Python installed
- if you built from source, your compiler versions and ideally a build log

when investigating further and asking for support.

Using Python from `conda` (Anaconda)

Please make sure that you have activated your conda environment. See also the conda user-guide. If you use an external editor/development environment it will have to be set up correctly. See below for solutions for some common setups.

Using PyCharm with Anaconda Python

There are fairly common issues when using PyCharm together with Anaconda, please see the PyCharm support

Using VS Code with Anaconda Python (or environments)

A commonly reported issue is related to the environment activation within VSCode. Please see the VSCode support for information on how to correctly set up VSCode with virtual environments or conda.

Using Eclipse/PyDev with Anaconda Python (or environments)

Please see the Anaconda Documentation on how to properly configure Eclipse/PyDev to use Anaconda Python with specific conda environments.

Raspberry Pi

There are sometimes issues reported on Raspberry Pi setups when installing using `pip3 install` (or `pip` install). These will typically mention:

```
libf77blas.so.3: cannot open shared object file: No such file or directory
```

The solution will be to either:

```
sudo apt-get install libatlas-base-dev
```

to install the missing libraries expected by the self-compiled NumPy (ATLAS is a possible provider of linear algebra).

Alternatively use the NumPy provided by Raspbian. In which case run:

```
pip3 uninstall numpy  # remove previously installed version
apt install python3-numpy
```

Debug build on Windows

Rather than building your project in `DEBUG` mode on windows, try building in `RELEASE` mode with debug symbols and no optimization. Full `DEBUG` mode on windows changes the names of the DLLs python expects to find, so if you wish to truly work in `DEBUG` mode you will need to recompile the entire stack of python modules you work with including NumPy

All setups

Occasionally there may be simple issues with old or bad installations of NumPy. In this case you may just try to uninstall and reinstall NumPy. Make sure that NumPy is not found after uninstalling.

Development setup

If you are using a development setup, make sure to run `git clean -xdf` to delete all files not under version control (be careful not to lose any modifications you made, e.g. `site.cfg`). In many cases files from old builds may lead to incorrect builds.

Check environment variables

In general how to set and check your environment variables depends on your system. If you can open a correct python shell, you can also run the following in python:

```
import os
print("PYTHONPATH:", os.environ.get('PYTHONPATH'))
print("PATH:", os.environ.get('PATH'))
```

This may mainly help you if you are not running the python and/or NumPy version you are expecting to run.

C-API incompatibility

If you see an error like:

> RuntimeError: module compiled against API version v1 but this version of numpy is v2

You may have:

- A bad extension “wheel” (binary install) that should use oldest-support-numpy (with manual constraints if necessary) to build their binary packages.

- An environment issue messing with package versions.
- Incompatible package versions somehow enforced manually.
- An extension module compiled locally against a very recent version followed by a NumPy downgrade.
- A compiled extension copied to a different computer with an older NumPy version.

The best thing to do if you see this error is to contact the maintainers of the package that is causing problem so that they can solve the problem properly.

However, while you wait for a solution, a work around that usually works is to upgrade the NumPy version:

```
pip install numpy --upgrade
```

Segfaults or crashes

NumPy tries to use advanced CPU features (SIMD) to speed up operations. If you are getting an “illegal instruction” error or a segfault, one cause could be that the environment claims it can support one or more of these features but actually cannot. This can happen inside a docker image or a VM (qemu, VMWare, ...)

You can use the output of `np.show_runtime()` to show which SIMD features are detected. For instance:

```
>>> np.show_runtime()
WARNING: `threadpoolctl` not found in system! Install it by `pip install \
threadpoolctl`. Once installed, try `np.show_runtime` again for more detaile
build information
[{'simd_extensions': {'baseline': ['SSE', 'SSE2', 'SSE3'],
                      'found': ['SSSE3',
                                'SSE41',
                                'POPCNT',
                                'SSE42',
                                'AVX',
                                'F16C',
                                'FMA3',
                                'AVX2'],
                      'not_found': ['AVX512F',
                                    'AVX512CD',
                                    'AVX512_KNL',
                                    'AVX512_KNM',
                                    'AVX512_SKX',
                                    'AVX512_CLX',
                                    'AVX512_CNL',
                                    'AVX512_ICL']}}]
```

In this case, it shows AVX2 and FMA3 under the `found` section, so you can try disabling them by setting `NPY_DISABLE_CPU_FEATURES="AVX2,FMA3"` in your environment before running python (for cmd.exe on windows):

```
>SET NPY_DISABLE_CPU_FEATURES="AVX2,FMA3"
>python <myprogram.py>
```

By installing threadpoolctl `np.show_runtime()` will show additional information:

```
...
{'architecture': 'Zen',
  'filepath': '/tmp/venv3/lib/python3.9/site-packages/numpy.libs/libopenblas
  'internal_api': 'openblas',
  'num_threads': 24,
  'prefix': 'libopenblas',
  'threading_layer': 'pthreads',
  'user_api': 'blas',
  'version': '0.3.21'}]
```

If you use the wheel from PyPI, it contains code from the OpenBLAS project to speed up matrix operations. This code too can try to use SIMD instructions. It has a different mechanism for choosing which to use, based on a CPU architecture, You can override this architecture by setting `OPENBLAS_CORETYPE`: a minimal value for `x86_64` is `OPENBLAS_CORETYPE=Haswell`. This too needs to be set before running your python (this time for posix):

```
$ OPENBLAS_CORETYPE=Haswell python <myprogram.py>
```