pycf3

Release 2020.12

QuatroPe

Nov 27, 2021

CONTENTS


PyCF3!

CHAPTER

ONE

DESCRIPTION

pycf3 is a Python client for the Cosmicflows-3 Distance-Velocity Calculator, and NAM Distance-Velocity Calculator

CHAPTER

TWO

CODE REPOSITORY & ISSUES

https://github.com/quatrope/pycf3

CHAPTER

THREE

LICENSE

pycf3 is under The BSD 3 License

The BSD 3-clause license allows you almost unlimited freedom with the software so long as you include the BSD copyright and license notice in it (found in Fulltext).

CHAPTER

FOUR

BASIC INSTALL

Execute

```
$ pip install pycf3
```

CHAPTER

FIVE

DEVELOPMENT INSTALL

Clone this repo and install with pip

```
$ git clone https://github.com/quatrope/pycf3.git
$ cd pycf3
$ pip install -e .
```

CHAPTER

SIX

QUICK USAGE

```
>>> import pycf3
>>> cf3 = pycf3.CF3()
>>> result = cf3.calculate_distance(velocity=9000, glon=283, glat=75)
>>> print(result.observed_velocity_)
9000.0
>>> result.observed_distance_
array([136.90134347])
```

For more information, read the tutorial in the documentation.

CHAPTER

SEVEN

CITATION

- If you use the results of this work in your research or other applications, please cite Kourkchi et al. 2020, AJ, 159, 67
- Please acknowledge pycf3 in any research report or publication that requires citation of any author's work. Our suggested acknowledgment is:

  The authors acknowledge the pycf3 project that contributed to the research reported here. <https://pycf3.readthedocs.io/>

7.1 ABOUT THE DATA

All data exposed by pycf3 belongs to the project

> Cosmicflows-3 Distance-Velocity Calculator (http://edd.ifa.hawaii.edu/CF3calculator/) Copyright (C) Cosmicflows Team - The Extragalactic Distance Database (EDD)

Please cite:

> Kourkchi, E., Courtois, H. M., Graziani, R., Hoffman, Y., Pomarede, D., Shaya, E. J., & Tully, R. B. (2020). Cosmicflows-3: Two Distance–Velocity Calculators. The Astronomical Journal, 159(2), 67.

7.1.1 BibText

```
@ARTICLE{2020AJ....159...67K,
    author = {{Kourkchi}, Ehsan and {Courtois}, H{\'e}l{\`e}ne M. and
        {Graziani}, Romain and {Hoffman}, Yehuda and {Pomar{\`e}de}, Daniel and
        {Shaya}, Edward J. and {Tully}, R. Brent},
    title = "{Cosmicflows-3: Two Distance-Velocity Calculators}",
    journal = {\aj},
    keywords = {590, 1146, 902, 1968, Astrophysics - Cosmology and
        Nongalactic Astrophysics, Astrophysics - Astrophysics of Galaxies,
        Astrophysics - Instrumentation and Methods for Astrophysics},
    year = 2020,
    month = feb,
    volume = {159},
    number = {2},
    eid = {67},
    pages = {67},
    doi = {10.3847/1538-3881/ab620e},
    archivePrefix = {arXiv},
```

(continues on next page)

(continued from previous page)

```
    eprint = {1912.07214},
    primaryClass = {astro-ph.CO},
    adsurl = {https://ui.adsabs.harvard.edu/abs/2020AJ....159...67K},
    adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

CHAPTER

EIGHT

SHORT VIDEO DESCRIPTION

CHAPTER

NINE

AUTHORS

- Juan BC - jbcabral@unc.edu.com
- Bruno Sanchez
- Martin Beroiz
- Ehsan Kourkchi.

IATE - CIFASIS

This project is part of the QuatroPe scientific tools.

CHAPTER

TEN

CONTENTS:

10.1 pycf3 tutorial

10.1.1 Interactive Version

Launch Binder for an interactive version of this tutorial!

10.1.2 Introduction

This tutorial is a guide on how to interact with the two Cosmicflows-3 Distance-Velocity Calculators in a Python code. These calculators are featured in Kourkchi et al. 2020, AJ, 159, 67.

pycf3 present an OO API to access the same capabilities of http://edd.ifa.hawaii.edu/NAMcalculator/ and http://edd.ifa.hawaii.edu/CF3calculator/ . Please be gentle with the server.

The first required step is to import the project

[2]:
```
import pycf3
```

10.1.3 Example 1: Sending a request to the NAM D-V calcualtor

First, let's create the **NAM Client**

[3]:
```
nam = pycf3.NAM()
nam
```

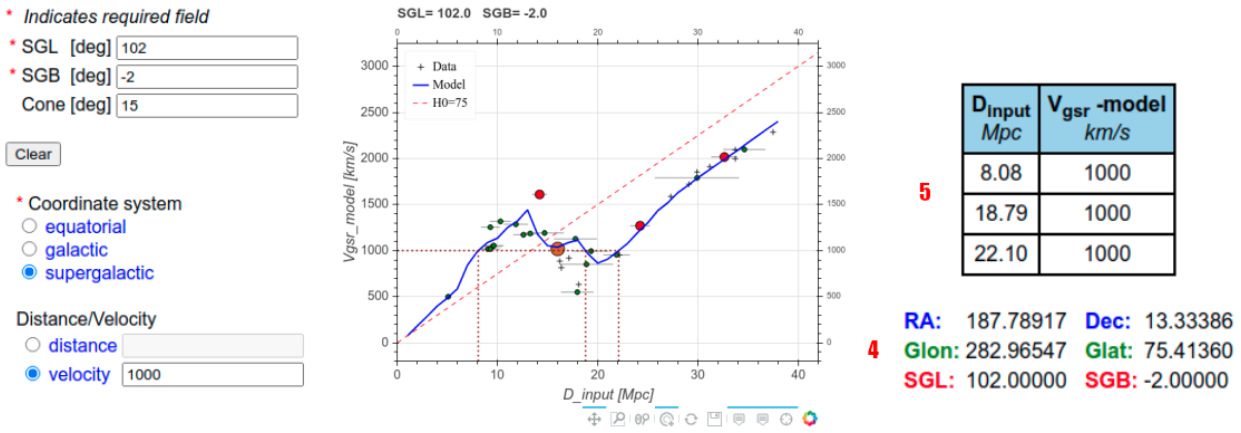
[3]:
```
NAM(calculator='NAM', cache_dir='/home/ehsan/pycf3_data/_cache_', cache_expire=None)
```

Then the basic functionality of the NAM are provided in two different methods

- `calculate_distance(velocity, <coordinates>)` to calculate a **distance** based on a given *velocity*.
- `calculate_velocity(distance, <coordinates>)` to calculate a **velocity** based on a given *distance*.

The `<coordinates>` parameters can be expressed in `ra, dec` (equatorial), `glon, glat` (galactic) or `sgl, sgb` (supergalactic) coordinates. You need to provide both components of a coordinate. Please note that multiple coordinate systems cannot be mixed.

Now, let's assume we want to generate the same query as it appears in the following Figure



D_{input} *Mpc*	V_{gsr} -model *km/s*
8.08	1000
18.79	1000
22.10	1000

Here is how to send the same request in a Python code:

[4]:
```
result = nam.calculate_distance(velocity=1000, sgl=102, sgb=-2)
result
```

[4]:
```
Result - NAM(velocity=1000, sgl=102, sgb=-2)
+----------+-----------------+-----------------------------------------+
| Observed | Distance (Mpc)  | [ 8.08088613 18.78629089 22.09785028] |
|          | Velocity (Km/s) | 1000.0                                  |
+----------+-----------------+-----------------------------------------+
```

Note: You can click `Show/Hide Raw` to check the raw response from the server

The provided coordinates at different coordinate systems can be extracted as follows

[5]:
```
result.calculated_at_
```

[5]:
```
CalculatedAt(ra=187.7891703346409, dec=13.333860121247609, glon=282.9654677357161,␣
↪glat=75.4136002414933, sgl=102.0, sgb=-2.0)
```

and the corresponding **distance/velocity** values are available at:

[6]:
```
result.observed_velocity_
```

[6]:
```
1000.0
```

[7]:
```
result.observed_distance_
```

[7]:
```
array([ 8.08088613, 18.78629089, 22.09785028])
```

In addition, the entire raw response, as it is returned by the **EDD** server, are packaged in the `json_` attribute in the form of a Python dictionary

[8]:
```
result.json_
```

[8]:
```
{'message': 'Success',
 'RA': 187.7891703346409,
 'Dec': 13.333860121247609,
 'Glon': 282.9654677357161,
 'Glat': 75.4136002414933,
 'SGL': 102.0,
 'SGB': -2.0,
```

(continues on next page)

(continued from previous page)

```
 'velocity': 1000.0,
 'distance': [8.08088612690689, 18.786290885088945, 22.097850275812398]}
```

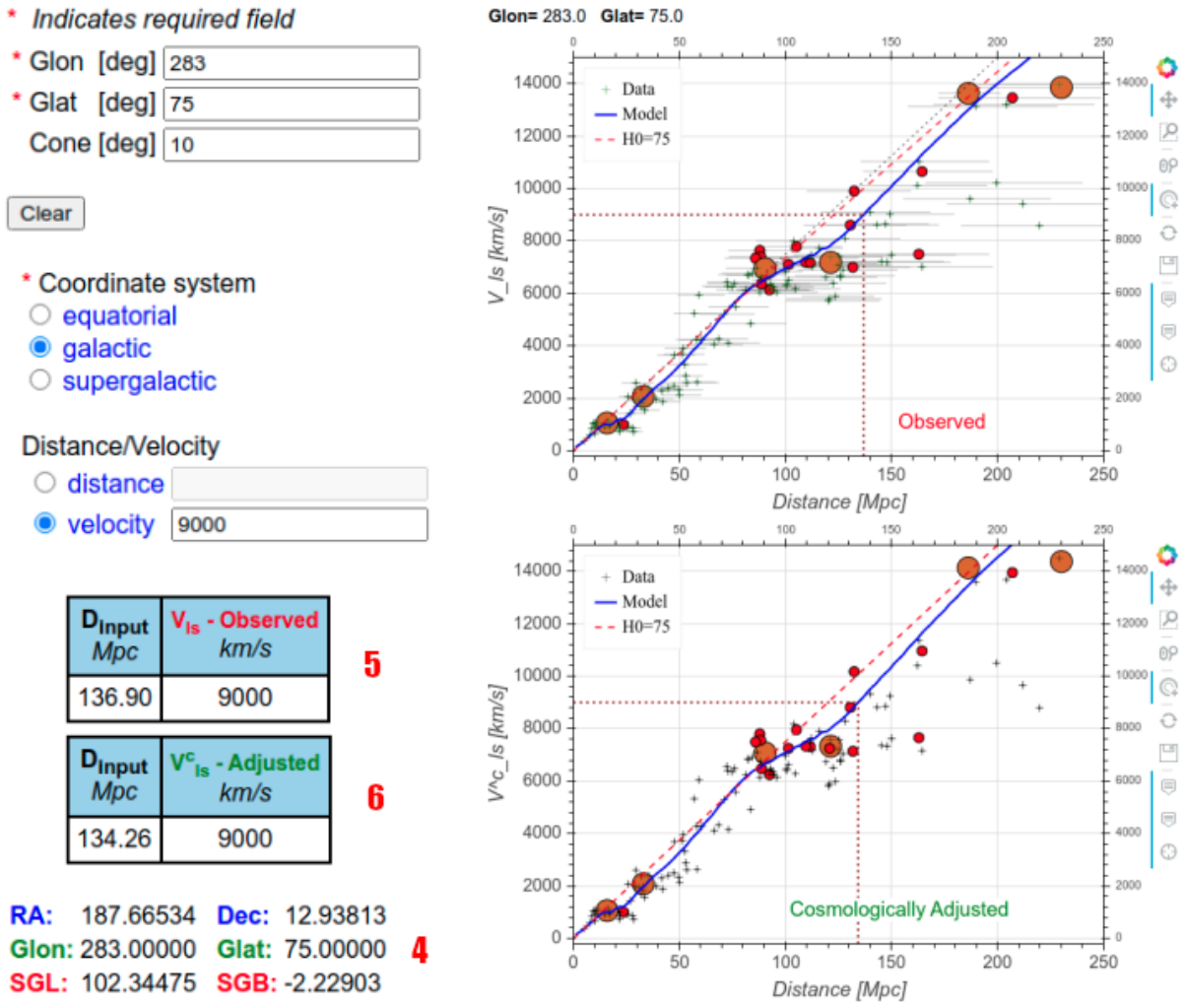
10.1.4 Example 2: How to extract the radial velocity of an object at a given distance in NAM

[9]:
```
nam.calculate_velocity(distance=30, sgl=102, sgb=-2)
```

[9]:
```
Result - NAM(distance=30, sgl=102, sgb=-2)
+----------+-----------------+--------------------+
| Observed | Distance (Mpc)  | [30.]              |
|          | Velocity (Km/s) | 1790.9019256321444 |
+----------+-----------------+--------------------+
```

10.1.5 Example 3: Sending a request to the Cosmicflows-3 D-V calculator (d < 200 Mpc)

We are trying to reproduce this query in Python



As stated before, first we need to create a **Cosmic Flows-3 Client**

[10]:
```
cf3 = pycf3.CF3()
cf3
```

[10]:
```
CF3(calculator='CF3', cache_dir='/home/ehsan/pycf3_data/_cache_', cache_expire=None)
```

Let's calculate the distance

[11]:
```
result = cf3.calculate_distance(velocity=9000, glon=283, glat=75)
result
```

[11]:
```
Result - CF3(velocity=9000, glon=283, glat=75)
+----------+-----------------+----------------+
| Observed | Distance (Mpc)  | [136.90134347] |
|          | Velocity (Km/s) | 9000.0         |
+----------+-----------------+----------------+
| Adjusted | Distance (Mpc)  | [134.26214472] |
|          | Velocity (Km/s) | 9000.0         |
+----------+-----------------+----------------+
```

Similar to NAM, **coordinates** can be accessed with `result.calculated_at_` and **velocity/distance** with `result.observed_velocity_` and `result.observed_distance_` respectively.

In addition, CF3 provides an adjusted version of the **velocity/distance**, that are available in `result.adjusted_distance_` and `result.adjusted_velocity_` . For further details on the adjustments you can visit Here.

[12]:
```
result.adjusted_distance_
```

[12]:
```
array([134.26214472])
```

[13]:
```
result.adjusted_velocity_
```

[13]:
```
9000.0
```

Note: The NAM client also has the *adjusted_* values but all are `None`.

10.1.6 Example 4: How to obtain the radial velocity at a given distance with CF3

[14]:
```
cf3.calculate_velocity(distance=180, ra=187, dec=13)
```

[14]:
```
Result - CF3(distance=180, ra=187, dec=13)
+----------+-----------------+--------------------+
| Observed | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12515.699706446017 |
+----------+-----------------+--------------------+
| Adjusted | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12940.58481990226  |
+----------+-----------------+--------------------+
```

10.1.7 PyCF3 Cache system

By default, any any client instance is created with a cache that prevent to send the same request twice. For example, let's make a similar query twice and measure the calculation time.

Note: We are using the `pycf3.CF3` client as an example, but any other `pycf3` calculator has the same functionalities.

[15]:
```
cf3 = pycf3.CF3()
```

[17]:
```
%%time
cf3.calculate_velocity(distance=180, ra=187, dec=13)
```

```
CPU times: user 6.53 ms, sys: 1.02 ms, total: 7.56 ms
Wall time: 11.6 s
```

[17]:
```
Result - CF3(distance=180, ra=187, dec=13)
+----------+-----------------+--------------------+
| Observed | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12515.699706446017 |
+----------+-----------------+--------------------+
| Adjusted | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12940.58481990226  |
+----------+-----------------+--------------------+
```

[18]:
```
%%time
cf3.calculate_velocity(distance=180, ra=187, dec=13)
```

```
CPU times: user 3.67 ms, sys: 0 ns, total: 3.67 ms
Wall time: 2.86 ms
```

[18]:
```
Result - CF3(distance=180, ra=187, dec=13)
+----------+-----------------+--------------------+
| Observed | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12515.699706446017 |
+----------+-----------------+--------------------+
| Adjusted | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12940.58481990226  |
+----------+-----------------+--------------------+
```

Evidently, the first time execution lasts for about **10 seconds** while the second time execution is of the order of **ms**. This is achieved by storing the results on the local hard drive in order to avoid the successive execution of similar requests (by default a folder called `pycf3_data`, is created in the user home directory)

As expected, if the query is altered by asking for another declination, the process is going to be slow again:

[19]:
```
%%time
cf3.calculate_velocity(distance=180, ra=187, dec=13.5)
```

```
CPU times: user 3.98 ms, sys: 4.03 ms, total: 8.01 ms
Wall time: 12.2 s
```

[19]:
```
Result - CF3(distance=180, ra=187, dec=13.5)
+----------+-----------------+--------------------+
| Observed | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12512.725297090401 |
+----------+-----------------+--------------------+
| Adjusted | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12937.406568949371 |
+----------+-----------------+--------------------+
```

and the execution of the same query is fast again

[20]:
```
%%time
cf3.calculate_velocity(distance=180, ra=187, dec=13.5)
```

```
CPU times: user 1.68 ms, sys: 357 µs, total: 2.04 ms
Wall time: 1.37 ms
```

[20]:
```
Result - CF3(distance=180, ra=187, dec=13.5)
+----------+-----------------+--------------------+
| Observed | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12512.725297090401 |
+----------+-----------------+--------------------+
| Adjusted | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12937.406568949371 |
+----------+-----------------+--------------------+
```

In addition, it is beneficial to recycle the results that are not "too older" than an expiration time. In these cases, you can set how many second your local data will be available by adding the parameter `cache_expire` when you create the *CF3* client.

[21]:
```
cf3 = pycf3.CF3(cache_expire=2)
```

At this point the new cf3 instance shares the same default-cache of the previous one. Now, if we execute any of the previous requests, the process would be fast. As seen, it's only taking a few milliseconds.

[22]:
```
%%time
cf3.calculate_velocity(distance=180, ra=187, dec=13)
```

```
CPU times: user 2.19 ms, sys: 0 ns, total: 2.19 ms
Wall time: 1.65 ms
```

[22]:
```
Result - CF3(distance=180, ra=187, dec=13)
+----------+-----------------+--------------------+
| Observed | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12515.699706446017 |
+----------+-----------------+--------------------+
| Adjusted | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12940.58481990226  |
+----------+-----------------+--------------------+
```

Yo can remove the entire "cached" data by calling the command

[23]:
```
cf3.cache.clear()
```

[23]:
```
2
```

Now we can send the same original request and the result will be available for 2 seconds before it expires.

[24]:
```
%%time
cf3.calculate_velocity(distance=180, ra=187, dec=13.5)

import time
time.sleep(3)  # lets sleep 3 seconds
```

```
CPU times: user 6.2 ms, sys: 3.32 ms, total: 9.52 ms
Wall time: 14.5 s
```

because we waited to long, the next query will be slow again

[25]:
```
%%time
cf3.calculate_velocity(distance=180, ra=187, dec=13.5)
```

```
CPU times: user 6.5 ms, sys: 5.01 ms, total: 11.5 ms
Wall time: 11.7 s
```

[25]:
```
Result - CF3(distance=180, ra=187, dec=13.5)
+----------+-----------------+--------------------+
| Observed | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12512.725297090401 |
+----------+-----------------+--------------------+
| Adjusted | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12937.406568949371 |
+----------+-----------------+--------------------+
```

However, if we don't wait until the cached outcomes are expired, the process would be quick.

[26]:
```
%%time
cf3.calculate_velocity(distance=180, ra=187, dec=13.5)
```

```
CPU times: user 0 ns, sys: 2.3 ms, total: 2.3 ms
Wall time: 1.53 ms
```

[26]:
```
Result - CF3(distance=180, ra=187, dec=13.5)
+----------+-----------------+--------------------+
| Observed | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12512.725297090401 |
+----------+-----------------+--------------------+
| Adjusted | Distance (Mpc)  | [180.]             |
|          | Velocity (Km/s) | 12937.406568949371 |
+----------+-----------------+--------------------+
```

Changing the cache backend

The entire cache backend of pycf3 was created with **DiskCache** (http://www.grantjenks.com/docs/diskcache/)

You can change your cache location (to store different datasets for example) by providing another `diskcache.Cache` or `diskcache.FanoutCache` instance.

```
import diskcache as dcache

cache = dcache.FanoutCache(
    directory="my/cache/directory")

# let make our data valid for 24 hours
cf3 = pycf3.CF3(cache=cache, cache_expire=86400)
```

Finally, to completely deactivate the cache system, pycf3 can be forced to ignore the cache system by setting cache to `NoCache`.

[27]:
```
cf3 = pycf3.CF3(cache=pycf3.NoCache())
cf3
```

[27]:
```
CF3(calculator='CF3', cache_dir='', cache_expire=None)
```

10.1.8 PyCF3 Retry

By default any calculator instance try **3 times** to perform a request. if you want to customize the number of attempts, you need to change the default `session` of the instance.

> Note: We are using the `pycf3.CF3` client as an example, but any other `pycf3` calculator has the same functionalities.

For example if you want to try 2 times:

[28]:
```
session = pycf3.RetrySession(retries=2)
cf3 = pycf3.CF3(session=session)
```

Also if you want to only wait for some arbitrary number of seconds between any request you can add the `timeout=<SECONDS>` to any search request.

[30]:
```
# no more than 5 seconds between every request
cf3.calculate_velocity(distance=180, ra=187, dec=13.5, timeout=5)
```

```
---------------------------------------------------------------------------
timeout                                   Traceback (most recent call last)
~/anaconda3/lib/python3.8/site-packages/urllib3/connectionpool.py in _make_request(self,␣
↪conn, method, url, timeout, chunked, **httplib_request_kw)
    425                     # Otherwise it looks like a bug in the code.
--> 426                     six.raise_from(e, None)
    427         except (SocketTimeout, BaseSSLError, SocketError) as e:

~/anaconda3/lib/python3.8/site-packages/urllib3/packages/six.py in raise_from(value,␣
↪from_value)

~/anaconda3/lib/python3.8/site-packages/urllib3/connectionpool.py in _make_request(self,␣
↪conn, method, url, timeout, chunked, **httplib_request_kw)
    420                 try:
--> 421                     httplib_response = conn.getresponse()
    422                 except BaseException as e:

~/anaconda3/lib/python3.8/http/client.py in getresponse(self)
   1346             try:
-> 1347                 response.begin()
   1348             except ConnectionError:

~/anaconda3/lib/python3.8/http/client.py in begin(self)
    306         while True:
--> 307             version, status, reason = self._read_status()
    308             if status != CONTINUE:

~/anaconda3/lib/python3.8/http/client.py in _read_status(self)
    267     def _read_status(self):
--> 268         line = str(self.fp.readline(_MAXLINE + 1), "iso-8859-1")
    269         if len(line) > _MAXLINE:

~/anaconda3/lib/python3.8/socket.py in readinto(self, b)
    668             try:
--> 669                 return self._sock.recv_into(b)
    670             except timeout:

timeout: timed out

During handling of the above exception, another exception occurred:

ReadTimeoutError                          Traceback (most recent call last)
~/anaconda3/lib/python3.8/site-packages/urllib3/connectionpool.py in urlopen(self,␣
↪method, url, body, headers, retries, redirect, assert_same_host, timeout, pool_timeout,
↪ release_conn, chunked, body_pos, **response_kw)
    669             # Make the request on the httplib connection object.
--> 670             httplib_response = self._make_request(
    671                 conn,

~/anaconda3/lib/python3.8/site-packages/urllib3/connectionpool.py in _make_request(self,␣
↪conn, method, url, timeout, chunked, **httplib_request_kw)
```

(continues on next page)

(continued from previous page)

```
    427         except (SocketTimeout, BaseSSLError, SocketError) as e:
--> 428             self._raise_timeout(err=e, url=url, timeout_value=read_timeout)
    429             raise

~/anaconda3/lib/python3.8/site-packages/urllib3/connectionpool.py in _raise_timeout(self,
↪ err, url, timeout_value)
    334         if isinstance(err, SocketTimeout):
--> 335             raise ReadTimeoutError(
    336                 self, url, "Read timed out. (read timeout=%s)" % timeout_value

ReadTimeoutError: HTTPConnectionPool(host='edd.ifa.hawaii.edu', port=80): Read timed out.
↪ (read timeout=5)

During handling of the above exception, another exception occurred:

MaxRetryError                             Traceback (most recent call last)
~/anaconda3/lib/python3.8/site-packages/requests/adapters.py in send(self, request,␣
↪stream, timeout, verify, cert, proxies)
    438             if not chunked:
--> 439                 resp = conn.urlopen(
    440                     method=request.method,

~/anaconda3/lib/python3.8/site-packages/urllib3/connectionpool.py in urlopen(self,␣
↪method, url, body, headers, retries, redirect, assert_same_host, timeout, pool_timeout,
↪ release_conn, chunked, body_pos, **response_kw)
    753             )
--> 754             return self.urlopen(
    755                 method,

~/anaconda3/lib/python3.8/site-packages/urllib3/connectionpool.py in urlopen(self,␣
↪method, url, body, headers, retries, redirect, assert_same_host, timeout, pool_timeout,
↪ release_conn, chunked, body_pos, **response_kw)
    753             )
--> 754             return self.urlopen(
    755                 method,

~/anaconda3/lib/python3.8/site-packages/urllib3/connectionpool.py in urlopen(self,␣
↪method, url, body, headers, retries, redirect, assert_same_host, timeout, pool_timeout,
↪ release_conn, chunked, body_pos, **response_kw)
    725
--> 726             retries = retries.increment(
    727                 method, url, error=e, _pool=self, _stacktrace=sys.exc_info()[2]

~/anaconda3/lib/python3.8/site-packages/urllib3/util/retry.py in increment(self, method,␣
↪url, response, error, _pool, _stacktrace)
    445         if new_retry.is_exhausted():
--> 446             raise MaxRetryError(_pool, url, error or ResponseError(cause))
    447

MaxRetryError: HTTPConnectionPool(host='edd.ifa.hawaii.edu', port=80): Max retries␣
↪exceeded with url: /CF3calculator/api.php (Caused by ReadTimeoutError(
↪"HTTPConnectionPool(host='edd.ifa.hawaii.edu', port=80): Read timed out. (read␣
↪timeout=5)"))
```

(continues on next page)

(continued from previous page)

```
During handling of the above exception, another exception occurred:

ConnectionError                           Traceback (most recent call last)
<ipython-input-30-a5a25eb08020> in <module>
      1 # no more than 5 seconds between every request
----> 2 cf3.calculate_velocity(distance=180, ra=187, dec=13.5, timeout=5)

/media/Data/Home/PanStarrs/Jan/HI/augment/test_TFRcatal/pycf3/pycf3.py in calculate_
↪velocity(self, distance, ra, dec, glon, glat, sgl, sgb, **get_kwargs)
    902             ra=ra, dec=dec, glon=glon, glat=glat, sgl=sgl, sgb=sgb
    903         )
--> 904         response = self._search(
    905             coordinate_system=coordinate_system,
    906             alpha=alpha,

/media/Data/Home/PanStarrs/Jan/HI/augment/test_TFRcatal/pycf3/pycf3.py in _search(self,␣
↪coordinate_system, alpha, delta, distance, velocity, **get_kwargs)
    727             response = cache.get(key, default=dcache.core.ENOVAL, retry=True)
    728             if response == dcache.core.ENOVAL:
--> 729                 response = self.session.get(
    730                     self.URL, json=payload, **get_kwargs
    731                 )

~/anaconda3/lib/python3.8/site-packages/requests/sessions.py in get(self, url, **kwargs)
    541
    542         kwargs.setdefault('allow_redirects', True)
--> 543         return self.request('GET', url, **kwargs)
    544
    545     def options(self, url, **kwargs):

~/anaconda3/lib/python3.8/site-packages/requests/sessions.py in request(self, method,␣
↪url, params, data, headers, cookies, files, auth, timeout, allow_redirects, proxies,␣
↪hooks, stream, verify, cert, json)
    528         }
    529         send_kwargs.update(settings)
--> 530         resp = self.send(prep, **send_kwargs)
    531
    532         return resp

~/anaconda3/lib/python3.8/site-packages/requests/sessions.py in send(self, request,␣
↪**kwargs)
    641
    642         # Send the request
--> 643         r = adapter.send(request, **kwargs)
    644
    645         # Total elapsed time of the request (approximately)

~/anaconda3/lib/python3.8/site-packages/requests/adapters.py in send(self, request,␣
↪stream, timeout, verify, cert, proxies)
    514                 raise SSLError(e, request=request)
    515
```

(continues on next page)

(continued from previous page)

```
--> 516             raise ConnectionError(e, request=request)
    517
    518         except ClosedPoolError as e:

ConnectionError: HTTPConnectionPool(host='edd.ifa.hawaii.edu', port=80): Max retries␣
↪exceeded with url: /CF3calculator/api.php (Caused by ReadTimeoutError(
↪"HTTPConnectionPool(host='edd.ifa.hawaii.edu', port=80): Read timed out. (read␣
↪timeout=5)"))
```

```
[ ]:
```

10.2 API - pycf3 module

Python client for several cosmic distance calculators.

Calculators:

- Cosmicflows-3 Distance-Velocity Calculator at distances
- Numerical Action Methods model

More information: http://edd.ifa.hawaii.edu/CF3calculator/

For citation check: https://github.com/quatrope/pycf3/blob/master/README.rst

class pycf3.**AbstractClient**(*session: requests.sessions.Session = NOTHING*, *cache: Union[diskcache.core.Cache, diskcache.fanout.FanoutCache] = NOTHING*, *cache_expire: Optional[float] = None*)

Abstract base class for all clients.

Parameters

- **session** (`pycf3.Session` (default: `None`)) – The session to use to send the requests. By default a `pyc3.RetrySession` with 3 retry is created. More info: https://2.python-requests.org, https://urllib3.readthedocs.io/en/latest/reference/urllib3.util.html.
- **cache** (`diskcache.Cache`, `diskcache.Fanout`,) – `pycf3.NoCache` or `None` (default: `None`) Any instance of `diskcache.Cache`, `diskcache.Fanout` or `None` (Default). If it's `None` a `diskcache.Cache` istance is created with the parameter `directory = pycf3.DEFAULT_CACHE_DIR`. More information: http://www.grantjenks.com/docs/diskcache
- **cache_expire** (`float` or None (default=``None``)) – Seconds until item expires (default `None`, no expiry) More information: http://www.grantjenks.com/docs/diskcache

calculate_distance(*velocity*, ***, *ra=None*, *dec=None*, *glon=None*, *glat=None*, *sgl=None*, *sgb=None*, ***get_kwargs*)

Calculate a distance based on the given velocity and location.

The mandatory parameters are `velocity` and a position expressed in two components depending on the chosen coordinate system:

- `ra` and `dec` for an equatorial system.
- `glon` and `glat` for a galactic system.
- `sgl` and `sgb` for a supergalactic system.

Coordinates cannot be mixed between systems, and must be expressed in J2000 as 360° decimal.

The returned distance(s) are expressed in Mpc, and potentially can be more than one value.

Parameters

- **velocity** (int or float) – Model velocity in km/s.
- **ra** (int or float (optional)) – Right ascension. If you provide ra you need to provide also dec.
- **dec** (int or float (optional)) – Declination. dec must be >= -90 and <= 90. If you provide dec you need to provide also ra.
- **glon** (int or float (optional)) – Galactic longitude. If you provide glon you need to provide also glat.
- **glat** (int or float (optional)) – Galactic latitude. glat must be >= -90 and <= 90. If you provide glat you need to provide also glon.
- **sgl** (int or float (optional)) – Super-galactic longitude. If you provide sgl you need to provide also sgb.
- **sgb** (int or float (optional)) – Super-galactic latitude. sgb must be >= -90 and <= 90. If you provide sgb you need to provide also sgl.
- **get_kwargs** – Optional arguments that request.get takes.

Returns Result object that automatically parses the entire model returned by the remote calculator.

Return type *pycf3.Result*

calculate_velocity(*distance*, *, *ra=None*, *dec=None*, *glon=None*, *glat=None*, *sgl=None*, *sgb=None*, ***get_kwargs*)

Calculate a velocity based on the given distance and location.

The mandatory parameters are distance and a position expressed in two components depending on the chosen coordinate system:

- ra and dec for an equatorial system.
- glon and glat for a galactic system.
- sgl and sgb for a supergalactic system.

Coordinates cannot be mixed between systems, and must be expressed in J2000 as 360° decimal.

The returned velocity are expressed in Km/s.

Parameters

- **distance** (int or float) – Distance(s) in Mpc.
- **ra** (int or float (optional)) – Right ascension. If you provide ra you need to provide also dec.
- **dec** (int or float (optional)) – Declination. dec must be >= -90 and <= 90. If you provide dec you need to provide also ra.
- **glon** (int or float (optional)) – Galactic longitude. If you provide glon you need to provide also glat.
- **glat** (int or float (optional)) – Galactic latitude. glat must be >= -90 and <= 90. If you provide glat you need to provide also glon.

- **sgl** (int or float (optional)) – Super-galactic longitude. If you provide sgl you need to provide also sgb.
- **sgb** (int or float (optional)) – Super-galactic latitude. sgb must be >= -90 and <= 90. If you provide sgb you need to provide also sgl.
- **get_kwargs** – Optional arguments that request.get takes.

Returns Result object that automatically parses the entire model returned by the remote calculator.

Return type *pycf3.Result*

equatorial_search(*ra=187.78917*, *dec=13.33386*, *distance=None*, *velocity=None*, ***get_kwargs*)
Search by equatorial coordinates.

The coordinates are expressed in J2000 as 360° decimal.

Deprecated since version 2020.12: "Use *calculate_velocity* or *calculate_distance* instead"

Parameters

- **ra** (int or float (default: 187.78917)) – Right ascension.
- **dec** (int or float (default: 13.33386)) – Declination. dec must be >= -90 and <= 90
- **distance** (int, float or None (default: None)) – Distance(s) in Mpc.
- **velocity** (int, float or None (default: None)) – Velocity in km/s. The returned distance potentially can be more than ine value.
- **get_kwargs** – Optional arguments that request.get takes.

Returns Result object that automatically parses the entire model returned by the remote calculator.

Return type *pycf3.Result*

galactic_search(*glon=282.96547*, *glat=75.4136*, *distance=None*, *velocity=None*, ***get_kwargs*)
Search by galactic coordinates.

The coordinates are expressed in J2000 as 360° decimal.

Deprecated since version 2020.12: "Use *calculate_velocity* or *calculate_distance* instead"

Parameters

- **glon** (int or float (default: 282.96547)) – Galactic longitude.
- **glat** (int or float (default: 75.41360)) – Galactic latitude. dec must be >= -90 and <= 90
- **distance** (int, float or None (default: None)) – Distance(s) in Mpc.
- **velocity** (int, float or None (default: None)) – Velocity in km/s. The returned distance potentially can be more than ine value.
- **get_kwargs** – Optional arguments that request.get takes.

Returns Result object that automatically parses the entire model returned by the remote calculator.

Return type *pycf3.Result*

supergalactic_search(*sgl=102.0*, *sgb=- 2.0*, *distance=None*, *velocity=None*, ***get_kwargs*)
Search super-galactic coordinates.

The coordinates are expressed in J2000 as 360° decimal.

Deprecated since version 2020.12: "Use *calculate_velocity* or *calculate_distance* instead"

Parameters

- **sgl** (int or float (default: 102)) – Super-galactic longitude.
- **sgb** (int or float (default: -2)) – Super-galactic latitude. dec must be >= -90 and <= 90
- **distance** (int, float or None (default: None)) – Distance(s) in Mpc.
- **velocity** (int, float or None (default: None)) – Velocity in km/s. The returned distance potentially can be more than ine value.
- **get_kwargs** – Optional arguments that request.get takes.

Returns Result object that automatically parses the entire model returned by the remote calculator.

Return type *pycf3.Result*

class pycf3.**CF3**(*session: requests.sessions.Session = NOTHING*, *cache: Union[diskcache.core.Cache, diskcache.fanout.FanoutCache] = NOTHING*, *cache_expire: Optional[float] = None*)

Client for the *Cosmicflows-3 Distance-Velocity Calculator*[3].

It computes expectation distances or velocities based on smoothed velocity field from the Wiener filter model of Graziani et al. 2019[4].

More information: http://edd.ifa.hawaii.edu/CF3calculator/ .

Parameters

- **session** (pycf3.Session (default: None)) – The session to use to send the requests. By default a pyc3.RetrySession with 3 retry is created. More info: https://2.python-requests.org, https://urllib3.readthedocs.io/en/latest/reference/urllib3.util.html.
- **cache** (diskcache.Cache, diskcache.Fanout,) – pycf3.NoCache or None (default: None) Any instance of diskcache.Cache, diskcache.Fanout or None (Default). If it's None a diskcache.Cache istance is created with the parameter directory = pycf3.DEFAULT_CACHE_DIR. More information: http://www.grantjenks.com/docs/diskcache
- **cache_expire** (float or None (default=``None``)) – Seconds until item expires (default None, no expiry) More information: http://www.grantjenks.com/docs/diskcache

References

MAX_DISTANCE = 200

Maximum distance to adjust the velocity.

MAX_VELOCITY = 15000

Maximum velocity to adjust the distance.

exception pycf3.**CFDeprecationWarning**

Custom class to inform that some functionality is in desuse.

exception pycf3.**MixedCoordinateSystemError**

Raised when the parameters are from different coordinates systems.

[3] Kourkchi, E., Courtois, H. M., Graziani, R., Hoffman, Y., Pomarede, D., Shaya, E. J., & Tully, R. B. (2020). Cosmicflows-3: Two Distance-Velocity Calculators. The Astronomical Journal, 159(2), 67.

[4] Graziani, R., Courtois, H. M., Lavaux, G., Hoffman, Y., Tully, R. B., Copin, Y., & Pomarède, D. (2019). The peculiar velocity field up to z 0.05 by forward-modelling Cosmicflows-3 data. Monthly Notices of the Royal Astronomical Society, 488(4), 5438-5451.

class pycf3.**NAM**(*session: requests.sessions.Session = NOTHING*, *cache: Union[diskcache.core.Cache, diskcache.fanout.FanoutCache] = NOTHING*, *cache_expire: Optional[float] = None*)

Client for the *NAM Distance-Velocity Calculator*[1].

Compute expectation distances or velocities based on smoothed velocity field from the Numerical Action Methods model of Shaya et al. 2017.[2].

More information: http://edd.ifa.hawaii.edu/NAMcalculator.

Parameters

- **session** (pycf3.Session (default: None)) – The session to use to send the requests. By default a pyc3.RetrySession with 3 retry is created. More info: https://2.python-requests.org, https://urllib3.readthedocs.io/en/latest/reference/urllib3.util.html.
- **cache** (diskcache.Cache, diskcache.Fanout,) – pycf3.NoCache or None (default: None) Any instance of diskcache.Cache, diskcache.Fanout or None (Default). If it's None a diskcache.Cache istance is created with the parameter directory = pycf3.DEFAULT_CACHE_DIR. More information: http://www.grantjenks.com/docs/diskcache
- **cache_expire** (float or None (default=``None``)) – Seconds until item expires (default None, no expiry) More information: http://www.grantjenks.com/docs/diskcache

References

MAX_DISTANCE = 38

Maximum distance to adjust the velocity.

MAX_VELOCITY = 2400

Maximum velocity to adjust the distance.

class pycf3.**NoCache**

Implements a minimalist no-cache for disk-cache.

expire(*now=None*, *retry=False*)

Return 0.

get(*key*, *default=None*, **args*, ***kwargs*)

Return the default.

set(*key*, *value*, **args*, ***kwargs*)

Return True.

class pycf3.**Result**(*calculator*, *url*, *coordinate*, *calculated_by*, *alpha*, *delta*, *distance*, *velocity*, *response_*)

Parsed result.

Parameters

- **calculator** (str) – The used calculator.
- **url** (str) – The url of the calculator.
- **coordinate** (Coordinate) – Coordinate system used to create this result.
- **alpha** (int or float) – α value for the coordinate system.
- **delta** (int or float) – δ value for the coordinate system.
- **distance** (int, float or None) – Distance used to calculate the velocity in Mpc.

[1] Kourkchi, E., Courtois, H. M., Graziani, R., Hoffman, Y., Pomarede, D., Shaya, E. J., & Tully, R. B. (2020). Cosmicflows-3: Two Distance-Velocity Calculators. The Astronomical Journal, 159(2), 67.

[2] Shaya, E. J., Tully, R. B., Hoffman, Y., & Pomarède, D. (2017). Action dynamics of the local supercluster. The Astrophysical Journal, 850(2), 207.

- **velocity** (int, float or None) – Velocity used to calculate the distance in Km/s.

response_
Original response object create by the *requests* library. More information: https://2.python-requests.org

Type requests.Response

observed_distance_
Observed distances.

Type numpy.ndarray

observed_velocity_
Observed velocity

Type float

adjusted_distance_
Cosmologically adjusted distances.

Type numpy.ndarray

adjusted_velocity_
Cosmologically adjusted velocity, V_{ls}^{c}.

Type float

calculated_at_
Coordinates in all the three supported systems.

Type pycf3.CalculatedAt

property json_
Proxy to response_.json().

property search_at_
Proxy to response.calculated_at_.

Deprecated since version 2020.12: "Use *calculated_at* instead"

property search_by
Proxy to response.calculated_by.

Deprecated since version 2020.12: "Use *calculated_by* instead"

class pycf3.**RetrySession**(*retries=3*, *backoff_factor=0.3*, *status_forcelist=(500, 502, 504)*, ***session_options*)
Session with retry.

Parameters

- **retries** (int (default: 3)) – Total number of retries to allow. It's a good idea to set this to some sensibly-high value to account for unexpected edge cases and avoid infinite retry loops. Set to 0 to fail on the first retry.
- **backoff_factor** (float (default: 0.3)) – A backoff factor to apply between attempts after the second try (most errors are resolved immediately by a second try without a delay). urllib3 will sleep for:

  ```
  {backoff factor} * (2 ** ({number of total retries} - 1))
  ```

  seconds. If the backoff_factor is 0.1, then sleep() will sleep for [0.0s, 0.2s, 0.4s, ...] between retries. It will never be longer than urllib3.Retry.BACKOFF_MAX.
- **status_forcelist** (iterable (default: 500, 502, 504)) – A set of integer HTTP status codes that we should force a retry on. A retry is initiated if the request method is in

method_whitelist and the response status code is in status_forcelist. By default, this is `500, 502, 504`.

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX